Trabajo Fin de Máster

Técnicas de aprendizaje supervisado aplicadas a la resolución de problemas de optimización

Sofía Rodríguez Ballesteros

Máster en Técnicas Estadísticas

Curso 2021-2022

Propuesta de Trabajo Fin de Máster

Título en galego: Técnicas de aprendizaxe supervisada aplicadas á resolución de problemas de optimización.
Título en español: Técnicas de aprendizaje supervisado aplicadas a la resolución de problemas de optimización.
English title: Supervised learning techniques for solving optimization problems.
Modalidad: Modalidad B
Autora: Sofía Rodríguez Ballesteros, Universidad de Santiago de Compostela.
Directores: Beatriz Pateiro López, Universidad de Santiago de Compostela; Brais González Rodríguez, Universidad de Santiago de Compostela
Tutor: Julio González Díaz, Instituto Tecnológico de Matemática Industrial (ITMATI)
Breve resumen del trabajo: En problemas de optimización cuya resolución se basa en técnicas de ramificación y acotación, las decisiones tomadas en dicho proceso de ramificación y acotación pueden resultar cruciales para la resolución eficiente del problema. Por un lado, es necesario determinar la variable por la cual se ramifica. Ramificar por una variable que no simplifique ninguno de los subproblemas que genera, lleva a incrementar de forma poco eficiente el tamaño del árbol de búsqueda. Por otro lado, también debemos seleccionar el nodo a resolver, lo cual se relaciona con la forma en que se va recorriendo el árbol. La incorporación de técnicas de aprendizaje automático en este contexto tiene como objetivo que los algoritmos "aprendan" sobre su mejor configuración. Con este trabajo se pretende que el alumno o la alumna haga una revisión de las principales técnicas de aprendizaje supervisado que pueden ser de utilidad en este contexto, evaluando su rendimiento a través de ejemplos prácticos.

Doña Beatriz Pateiro López, de la Universidad de Santiago de Compostela, don Brais González Rodríguez, de la Universidad de Santiago de Compostela y don Julio González Díaz del Instituto Tecnológico de Matemática Industrial (ITMATI), informan que el Trabajo Fin de Máster titulado

Técnicas de aprendizaje supervisado aplicadas a la resolución de problemas de optimización

fue realizado bajo su dirección por doña Sofía Rodríguez Ballesteros para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago de Compostela, a 2 de febrero de 2022.

La directora:

PATEIRO LOPEZ BEATRIZ - 46898344X

Firmado digitalmente por PATEIRO LOPEZ BEATRIZ - 46898344X
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-46898344X, givenName=BEATRIZ, sn=PATEIRO LOPEZ, cn=PATEIRO LOPEZ BEATRIZ - 46898344X
Fecha: 2022.02.02 11:23:12 +01'00'

Doña Beatriz Pateiro López

El director:

GONZALEZ RODRIGUEZ BRAIS - 44661382M

Firmado digitalmente por GONZALEZ RODRIGUEZ BRAIS - 44661382M
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-44661382M, givenName=BRAIS, sn=GONZALEZ RODRIGUEZ, cn=GONZALEZ RODRIGUEZ BRAIS - 44661382M
Fecha: 2022.02.02 12:58:36 +01'00'

Don Brais González Rodríguez

El tutor:

GONZALEZ DIAZ JULIO - 33332679K

Firmado digitalmente por GONZALEZ DIAZ JULIO - 33332679K
Fecha: 2022.02.02 14:17:26 +01'00'

Don Julio González Díaz

La autora:

RODRIGUEZ BALLESTEROS, SOFIA (FIRMA)

Firmado digitalmente por RODRIGUEZ BALLESTEROS, SOFIA (FIRMA)
Fecha: 2022.02.02 11:08:49 +01'00'

Doña Sofía Rodríguez Ballesteros

Índice general

Agradecimientos

En primer lugar, me gustaría dar las gracias a la Universidad de Santiago de Compostela y, en particular, a todo el personal docente de la Facultad de Matemáticas, que me ha acompañado a lo largo de estos seis años de formación y que ha contribuido a mi desarrollo, tanto personal como académico.

Gracias a mis directores, Beatriz Pateiro y Brais González, que han sido un apoyo constante a lo largo del desarrollo de este proyecto. Su completa disponibilidad y dedicación ha conseguido que el ambiente de trabajo haya sido cómodo e idóneo para completar esta memoria. No tengo más que palabras de agradecimiento por su trato cercano y me siento afortunada de haber elegido este proyecto y finalizar mis estudios con tan buenas sensaciones.

Me gustaría agradecer, de forma especial, el apoyo que he recibido a lo largo del máster por parte de mi tutor, Julio González. Desde el primer momento he podido contar con su ayuda y consejos, sintiéndome arropada a la hora de tomar decisiones importantes. Tanto por aconsejarme cursar este máster, como por aceptarme en este proyecto, gracias, pues no podría estar más satisfecha con el resultado final.

Finalmente, me gustaría dar las gracias a mis padres, a mis abuelos y a mi preciada hermana Diana, pues sabiendo mejor que nadie mis carencias y debilidades, han conseguido impulsarme siempre hacia delante con su cariño y comprensión. Y por supuesto, agradecer a mis amigos más cercanos por haberme acompañado todo este tiempo y ayudarme a crecer como persona, pudiendo hacer todo esto posible.

Resumen

Resumen en español

El interés en la optimización polinómica, que busca ir un paso más allá de los problemas de programación convexa, radica en la excesiva complejidad que supone la obtención de un óptimo global para los problemas de programación no lineal. En este contexto se ha desarrollado un nuevo optimizador global basado, esencialmente, en un algoritmo de ramificación y acotación, y que recibe el nombre de `RAPOSa`. Repasando brevemente los conceptos básicos de programación polinómica y describiendo en detalle esta técnica, nos encontraremos ante una cuestión fundamental: deberemos determinar la variable por la cual se ramifica y, por lo tanto, el criterio de ramificación que consideraremos. Es aquí donde entran en juego las diferentes técnicas de aprendizaje supervisado, que presentaremos, primero de forma teórica y, posteriormente, en el marco de nuestro problema práctico. De este modo, llevaremos a cabo un proceso de aprendizaje sobre los diferentes criterios de ramificación para que, posteriormente, sean los propios modelos ajustados aquellos que consigan predecir el criterio óptimo dado un nuevo problema de programación polinómica. Finalizaremos esta memoria con un análisis exhaustivo de los resultados obtenidos para las diferentes metodologías consideradas, evaluando su desempeño mediante una medida de rendimiento convenientemente seleccionada de acuerdo con las características de nuestro problema.

English abstract

The interest in polynomial optimization, which seeks to go one step beyond convex programming problems, lies in the excessive complexity involved in obtaining a global optimum for nonlinear programming problems. In this context, a new global optimizer has been developed, essentially based on a branch and bound algorithm, which is called `RAPOSa`. Briefly reviewing the basic concepts of polynomial programming and describing this technique in detail, we will be faced with a fundamental question: the branching variable and, therefore, the branching criterion to be considered, needs to be determined. This is where different supervised learning techniques come into play, which we will present, first theoretically, and then in the framework of our practical problem. In this way, we will carry out a learning process on the different branching criteria so that, later on, it will be the adjusted models themselves that manage to predict the optimal criterion given a new polynomial programming problem. We will conclude this report with an exhaustive analysis of the results obtained for the different methodologies considered, evaluating their performance by means of a performance measure conveniently selected, according to the characteristics of our problem.

Capítulo 1

RAPOSa: un nuevo optimizador global

En este primer capítulo de la memoria introduciremos el solver `RAPOSa` (**R**eformulation **A**lgorithm for **P**olynomial **O**ptimization - **Sa**ntiago). En particular, se trata de un optimizador global diseñado específicamente para la resolución de problemas de programación polinómica con variables acotadas. `RAPOSa` ha sido implementado completamente en el lenguaje de programación `C++` y se basa en el algoritmo RLT (*Reformulation-Linearization Technique*), introducido en Sherali and Tuncbilek (1991) y posteriormente mejorado en Sherali et al. (2012a), Sherali et al. (2012b), Dalkiran and Sherali (2013).

En particular, no se trata de una herramienta de código abierto, aunque sí que se encuentra disponible gratuitamente para su uso en Linux, Windows y MacOS. Cabe destacar que `RAPOSa` nos permite resolver problemas directamente desde `AMPL` y a partir de archivos `.nl` obtenidos en `AMPL`, `Pyomo` o `JuMP`. También puede ejecutarse desde el servidor NEOS [1] y en ella se combina el uso de solvers lineales y no lineales, algunos de los cuales precisan de una licencia para su uso. Toda la información detallada relativa a `RAPOSa` puede encontrarse en `http://www.itmati.com/RAPOSa/index.html`.

Con el objetivo de contextualizar y poder comprender este nuevo optimizador global, deberemos introducir una serie de conceptos fundamentales a lo largo de este primer capítulo. Así, en las sucesivas secciones vamos a centrarnos en los problemas de programación polinómica, así como en el algoritmo RLT, diseñado específicamente para su resolución. Estos problemas van a aparecer con frecuencia en la práctica y, en particular, han sido estudiados durante el convenio de prácticas entre la Universidad de Santiago de Compostela (USC) y el ITMATI, centro con el cual se colaboró durante el desarrollo de esta memoria. Los conjuntos de problemas utilizados específicamente en este trabajo serán descritos en mayor detalle en el segundo capítulo a lo largo de la Sección 3.1.

1.1. Optimización polinómica

La tarea de encontrar el óptimo global en un problema de programación no lineal, exceptuando aquellos de pequeño tamaño, es complicada. A lo largo de los años se han desarrollado diferentes técnicas de optimización con dicho propósito. Así, disponemos de algoritmos eficientes que garantizan la obtención de una solución global en el caso de los problemas de programación lineal, cuadrática y de programación convexa en general. En cuanto a la programación no lineal, se han desarrollado varios métodos de optimización, la mayoría de los cuales únicamente son capaces de obtener óptimos locales. La investigación en este campo nos lleva a la conclusión de que los problemas de optimización no lineal

[1] Ver Czyzyk et al. (1998), Dolan (2001) y Gropp and Moré (1997)

son excesivamente complejos como para obtener su solución óptima global.

Partiendo de esta motivación, introduciremos en esta sección los problemas de programación polinómica. Así, nos encontramos ante problemas de programación matemática cuya función objetivo es un polinomio, el cual se encuentra restringido a un conjunto definido, asimismo, por polinomios. En particular, consideraremos problemas con variables de decisión continuas.

Presentamos a continuación la formulación matemática de este tipo de problemas. Cabe destacar que hemos seguido la notación de Sherali and Tuncbilek (1991), utilizada también en González-Rodríguez (2017). En particular, puede consultarse esta última referencia para obtener una explicación más detallada de los conceptos abordados tanto en esta sección como, en general, a lo largo de todo este primer capítulo.

De esta forma, cualquier problema de programación polinómica, que denotaremos por PP(Ω), se puede escribir de la siguiente forma:

$$\begin{array}{rl} \text{minimizar} & \phi_0(\mathbf{x}) \\ \text{sujeto a} & \phi_r(\mathbf{x}) \geq \beta_r,\ r = 1, \ldots, R_1 \\ & \phi_r(\mathbf{x}) = \beta_r,\ r = R_1 + 1, \ldots, R \\ & \mathbf{x} \in \Omega \subset \mathbb{R}^n \end{array} \tag{1.1}$$

donde:

- $\phi_r(\mathbf{x}) = \sum_{t \in T_r} \alpha_{rt} \prod_{j \in J_{rt}} x_j$ es un polinomio de grado δ_r. En particular, para $r = 0, 1, \ldots, R$, tenemos que T_r es un conjunto de índices y $\alpha_{rt} \in \mathbb{R}$ son los coeficientes asociados a cada monomio $(\prod_{j \in J_{rt}} x_j)$. En este contexto vamos a permitir que aparezcan índices repetidos en cada conjunto J_{rt}. Así, por ejemplo, si $J_{rt} = \{1, 2, 2, 3\}$ tendríamos que el correspondiente monomio sería $x_1 x_2^2 x_3$.
- $\Omega = \{\mathbf{x} : 0 \leq l_j \leq x_j \leq u_j < \infty \text{ para todo } j \in \{1, \ldots, n\}\} \subset \mathbb{R}^n$ es un hiperrectángulo conteniendo a la región factible de nuestro problema PP(Ω), donde los l_j y los u_j se denominan cotas inferiores y cotas superiores, respectivamente.

Consideremos el conjunto $N = \{1, \ldots, n\}$. Diremos que δ es el grado del problema si es el mayor de los grados de los polinomios que aparecen en la formulación de nuestro problema PP(Ω).

Por otra parte, vamos a detenernos en los conjuntos J_{rt} con $t \in T_r$ y $r = 0, 1, \ldots, R$, que aparecían en la formulación de nuestro problema PP(Ω), y describirlos en mayor detalle. En primer lugar, tenemos que estos conjuntos J_{rt} reciben el nombre de multiconjuntos. Un multiconjunto es un par (S, p), donde S es un conjunto y $p : S \longrightarrow \mathbb{R}$ es una función que asigna a cada elemento de S su multiplicidad. Haciendo un abuso de notación, denotaremos mediante (N, δ) el multiconjunto de las variables (N, p) para las cuales $p(i) = \delta$, para cada $i \in \mathbb{R}$. Para cada multiconjunto (N, p), su cardinal se define por $|(N, p)| = \sum_{i \in N} p(i)$. Denotando el conjunto de las δ réplicas de N por $\bar{N} = \{N, \ldots, N\}$, 1164tendremos que $J_{rt} \subseteq \bar{N}$, con $1 \leq |J_{rt}| \leq \delta$, para $t \in T_r$, $r = 0, 1, \ldots, R$.

Como puede deducirse de la definición del problema de programación polinómica anterior, estamos suponiendo que nuestra región factible está contenida en $\mathbb{R}^n$. Si la denotamos por K, podremos considerar cualquier conjunto $\mathrm{K} \neq \mathbb{R}^n$, siempre que se trate de un conjunto compacto. Por otra parte, no asumiremos ninguna hipótesis previa en términos de conexidad o convexidad.

De esta forma hemos realizado un planteamiento de nuestro problema mediante una modelización bastante general que incluye, a su vez, otros problemas de optimización como casos particulares. Así, tendremos que los problemas de programación lineal serán un caso particular de nuestro problema

PP(Ω) cuando ambas, la función objetivo y el conjunto de restricciones, sean lineales. Asimismo, estaremos ante un problema de programación cuadrático, si la función objetivo es un polinomio de segundo grado y la región factible está constituida por un sistema de ecuaciones lineales. También serán casos particulares los problemas de programación lineal cero-uno y los problemas de programación lineal en enteros mixta.

Algunos casos particulares del problema PP(Ω), como es el caso $n = 1$, pueden encontrarse estudiados en detalle en el Capítulo 2 de Lasserre (2015). Por su parte, cabe destacar que el caso multivariante con $\mathrm{K} = \mathbb{R}^n$ es un problema NP-duro, cuando trabajamos con un polinomio ϕ_0 en n variables y de grado mayor o igual a cuatro. En general, el caso multivariante va a diferir radicalmente del caso univariante, pues en $\mathbb{R}^n$ no podemos expresar todos los polinomios no negativos como suma de cuadrados de otros polinomios.

1.2. Reformulation-Linearization Technique (RLT)

El problema PP(Ω) pertenece a la amplia clase los problemas de optimización con restricciones. Numerosos métodos han sido desarrollados tratando de abordarlo, como es el caso de aquellos propuestos por Horst (1990) y Horst and Tuy (1990), que incluyen técnicas de ramificación y acotación, aproximación externa y combinaciones de las mismas. Por su parte, algunos casos particulares de este problema han recibido especial atención; tal es el caso de su minimización bajo la hipótesis de concavidad o cuando la función objetivo es bilineal.

En esta sección introduciremos y desarrollaremos el algoritmo "Reformulation-Linearization Technique" (RLT). Este método nos va a permitir obtener un óptimo global para nuestro problema PP(Ω) mediante la relajación lineal del mismo. Así, pasaremos de trabajar con un problema claramente no lineal a ir resolviendo sus relajaciones lineales, y todo esto enmarcado dentro de un algoritmo de ramificación y acotación similar al empleado en la resolución de los problemas de programación entera.

A lo largo de los años, numerosos investigadores han utilizado esta técnica para resolver casos particulares del problema PP(Ω). Podemos destacar a Sherali and Adams (1989) que trabajaron con problemas de programación polinómica entera cero-uno y problemas de programación polinómica en enteros mixta. En particular, nos vamos a centrar en la versión más generalizada de este método, la cual nos va a permitir resolver problemas de programación polinómica continuos. La idea general consistirá en particionar el hiperrectángulo Ω, que contiene a la región factible de nuestro problema, obteniendo hiperrectángulos más pequeños a medida que avanza el algoritmo. Así, la elección de una partición adecuada será fundamental para asegurar la convergencia al óptimo del problema.

1.2.1. Relajación lineal del problema

Para construir la relajación lineal de nuestro problema PP(Ω) debemos introducir una serie de conceptos fundamentales, que estarán presentes en el algoritmo RLT. Estos elementos serán los *bounding factors* y las variables RLT.

Dado un problema de programación polinómica PP(Ω), definido como en (1.1), denotaremos por *bounding factors* a las desigualdades de la forma $(x_j - l_j) \geq 0$ y $(u_j - x_j) \geq 0$, con $j \in N$. Haciendo uso de estas desigualdades, se podrán generar nuevas restricciones mediante productos de las mismas. En concreto, multiplicando δ *bounding factors* diferentes entre sí. Estas restricciones serán de la forma

$$F_\delta(J_1, J_2) \equiv \prod_{j \in J_1} (x_j - l_j) \prod_{j \in J_2} (u_j - x_j) \geq 0, \tag{1.2}$$

donde $(J_1 \cup J_2) \subset (N, \delta), |J_1 \cup J_2| = \delta$. Podemos observar que el algoritmo RLT utiliza el grado de nuestro ploblema, δ, para fijar el número de *bounding factors* en cada producto y, a partir de ahí, genera todas las restricciones posibles. De forma general, hablaremos de *bound factors* cuando queramos referirnos al conjunto dado por todas estas restricciones. Nótese que todos los puntos de Ω satisfacen los *bound factors*.

Por otra parte, hablaremos de variables RLT y las definiremos para cada multiconjunto $J \subset (N, \delta)$ tal que $2 \leq |J| \leq \delta$ como

$$X_J = \prod_{j \in J} x_j, \tag{1.3}$$

donde se asume que los índices de J son una sucesión creciente (no estrictamente), y donde $X_{\{j\}} \equiv x_j$ para todo $j \in N$, con $X_\emptyset \equiv 1$. Notemos que cada multiconjunto J puede identificarse con un monomio. Así, como ya vimos antes, el multiconjunto $J = \{1, 2, 2, 3\}$ hace referencia al monomio $x_1 x_2^2 x_3$. Por lo tanto, podremos asociar a cada uno de estos monomios diferentes *bounding factors* con $J = J_1 \cup J_2$, dependiendo de qué variables utilicemos junto con las cotas inferiores y cuáles con las cotas superiores, en la ecuación (1.2). Asimismo, el multiconjunto $J = \{1, 2, 2, 3\}$ va a definir la variable RLT X_{1223}.

A continuación vamos a formular la relajación lineal de nuestro problema PP(Ω), obtenida mediante el algoritmo RLT. Para ello, deberemos sustituir los polinomios de grado mayor que 1 presentes en (1.1) por sus linealizaciones, esto es, por sus correspondientes variables RLT. Denotaremos la linealización por $[\cdot]_L$. Además, incluiremos los *bound factors* (1.2) linealizados, para así obtener una relajación lineal más fuerte. El problema linealizado, que denotaremos por LP(Ω) tiene la siguiente forma:

$$\begin{aligned}
\text{minimizar} \quad & [\phi_0(\mathbf{x})]_L \\
\text{sujeto a} \quad & [\phi_r(\mathbf{x})]_L \geq \beta_r, \qquad r = 1, \ldots, R_1 \\
& [\phi_r(\mathbf{x})]_L = \beta_r, \qquad r = R_1 + 1, \ldots, R \\
& [F_\delta(J_1, J_2)]_L \geq 0, \;\; J_1 \cup J_2 \subset (N, \delta), \;\; |J_1 \cup J_2| = \delta \\
& \mathbf{x} \in \Omega \subset \mathbb{R}^n
\end{aligned} \tag{1.4}$$

Claramente, si añadimos como restricciones las variables definidas en (1.3), tendremos que el problema lineal LP(Ω), es equivalente a nuestro problema PP(Ω) original. Se puede obtener una información más detallada de estos conceptos, así como del desarrollo del algoritmo que se incluirá en el siguiente apartado, en Sherali and Tuncbilek (1991).

1.2.2. Algoritmo RLT

El siguiente paso del método RLT será incorporar nuestro problema LP(Ω) en un algoritmo de ramificación y acotación para así obtener un óptimo global de nuestro problema PP(Ω). Este procedimiento supondrá particionar el hiperrectánculo Ω en un conjunto de hiperrectángulos más pequeños asociados a cada uno de los nodos del árbol de ramificación y acotación. Como ya se comentó, la elección de una partición adecuada de nuestro hiperrectángulo Ω será fundamental en el desarrollo de este proceso. En concreto, descompondremos Ω en dos hiperrectángulos basándonos en la variable de ramificación, que será seleccionada mediante una regla de ramificación.

A continuación comentaremos cómo procede el algoritmo RLT de forma general. En primer lugar, se resolverá el problema relajado (1.4) para obtener una cota inferior de nuestro problema original PP(Ω). Posteriormente, se procederá con el método de ramificación y acotación para obtener la solución global de este último. Puesto que toda solución del problema relajado LP(Ω) que satisfaga las restricciones de (1.3) es una solución factible del problema PP(Ω), tendremos una regla de ramificación basada en la violación de dichas restricciones.

Así, sea $\Omega' \subset \Omega$ y supongamos que $(\overline{\mathbf{X}}, \overline{\mathbf{x}})$ es una solución óptima del problema relajado LP(Ω'). La idea será ramificar, en caso de existir, por la variable del problema relajado que está causando una mayor infactibilidad en el problema original. De esta forma obtenemos dos nuevos subproblemas en los cuales no vuelve a aparecer este óptimo. La regla de ramificación nos indica que debemos ramificar por la variable x_p donde:

$$p \in \arg\max_{j \in N} \{\theta_j\}, \text{ siendo}$$

$$\theta_j = \max_{t \in \{1,\dots,\delta-1\}} \max_{\substack{J \subset \bar{N} \\ |J|=t}} \{|\bar{X}_{J \cup j} - \bar{x}_j \bar{X}_J|\} \text{ para todo } j \in N. \tag{1.5}$$

Claramente, si $\theta_j = 0$ para todo $j \in N$, tendremos que $\overline{\mathbf{x}}$ es factible para PP(Ω') (de hecho, es factible para PP(Ω)) y no será necesario particionar el conjunto Ω'.

A medida que crece el árbol de búsqueda, el algoritmo RLT va obteniendo e incrementando el valor de las cotas inferiores para la solución óptima del problema de minimización, mediante la resolución de los problemas relajados. A su vez, obtiene cotas superiores para esta solución óptima cuando alguna de las soluciones obtenidas para las relajaciones lineales es factible en el problema polinómico original PP(Ω). Así, este método va calculando las diferencias absolutas y relativas entre las cotas inferiores y superiores, lo que resulta en los *gaps* absolutos y relativos. Finalmente, el objetivo del algoritmo será cerrar estos *gaps* y así encontrar la solución óptima del problema polinómico. Puede consultarse el Apéndice A, si se desea obtener una información más detallada acerca del algoritmo RLT.

1.3. Mejoras en la implementación de RAPOSa y RLT

Una vez introducido el algoritmo RLT, punto de partida en el desarrollo del solver `RAPOSa`, y antes de finalizar este primer capítulo, presentaremos una serie de criterios de ramificación (*branching criteria*), incluyendo el comentado en la sección anterior. Hasta ahora hemos seguido la regla de ramificación propuesta en Sherali and Tuncbilek (1991), escogiendo como variable de ramificación aquella que resulte en una mayor violación de las restricciones (1.3). De forma más precisa, la expresión (1.5) nos indica la variable por la que deberemos ramificar, dada una solución de la relajación lineal de nuestro problema de programación polinómica.

En particular, la modificación del criterio de ramificación empleado en el algoritmo RLT formará parte de una serie de mejoras en la implementación básica de esta técnica que se pueden encontrar en González-Rodríguez et al. (2020). Con el objetivo de estudiar su impacto en el desempeño de la técnica RLT, tanto individual como colectivo, se utiliza como referencia una versión básica de `RAPOSa`, sobre la que se agregan las diferentes mejoras. Cabe destacar que en este mismo artículo ya se había refinado esta configuración básica, consiguiendo una mayor robustez y permitiéndonos evaluar con mayor precisión cómo afectan las nuevas modificaciones a la misma.

Finalmente, concluiremos la sección definiendo una medida para la selección del criterio de ramificación óptimo dado un problema de programación polinómica, que será utilizada durante el Capítulo 3. Tendremos que el estudio computacional realizado en este capítulo de la memoria se va a basar fundamentalmente en esta medida, a la que denotaremos por "ritmo". En concreto, será la base sobre la que construiremos nuestra variable respuesta (*output*), objetivo de nuestro problema de aprendizaje, y que también se comentará en detalle a continuación.

1.3.1. Criterios de ramificación

Comenzaremos definiendo los diferentes criterios de ramificación considerados en el desarrollo de esta memoria. Como bien hemos comentado, partimos de la configuración básica de `RAPOSa` considerando

como regla de ramificación la propuesta en la Sección 1.2.2 de este primer capítulo, que denominaremos criterio del máximo. Ahora bien, recientemente en Sherali et al. (2012a), se utiliza un criterio más sofisticado para la selección de esta variable de ramificación, sustituyendo el máximo de la expresión (1.5) por sumas y ponderando, para cada variable, la violación de las restricciones (1.3) mediante unos pesos determinados. Siguiendo un enfoque parecido, podemos definir θ_j como sigue:

$$\theta_j = \sum_{t\in\{1,\dots,\delta-1\}} \sum_{\substack{J\subset\bar{N}\\|J|=t}} w(j,J)|\overline{X}_{J\cup\{j\}} - \overline{x}_j\overline{X}_j|, \tag{1.6}$$

donde los $w(j,J)$ representan los pesos asociados a cada una de las posibles variables de ramificación, y que dependiendo de como se definan, nos proporcionarán diferentes criterios de decisión. A continuación, describiremos tres de las reglas seleccionadas para nuestro estudio, en función de los pesos utilizados en la expresión (1.6) anterior.

- Pesos constantes. Seleccionaremos $w(j,J) = 1$ para todo $j \in N$ y para todo J. De esta forma, obtenemos la expresión
$$\theta_j = \sum_{t\in\{1,\dots,\delta-1\}} \sum_{\substack{J\subset\bar{N}\\|J|=t}} |\overline{X}_{J\cup\{j\}} - \overline{x}_j\overline{X}_j|,$$
que se corresponderá con el criterio (1.5), considerado en la configuración básica de `RAPOSa`, al sustituir las sumas por máximos. En general, si conservamos las sumas, tendremos una nueva regla de ramificación que denotaremos por criterio de la suma.

- Valores duales (*dual values*). Dado un multiconjunto $J \subset \bar{N}$, podemos considerar los valores duales asociados a las restricciones del problema que contienen a J. Así, podríamos definir los pesos, $w(j,J)$, como la suma de los valores absolutos de dichos valores duales, obteniendo un nuevo criterio de ramificación.

- Rango de la variable (*variable range*). Definiremos el rango de una variable como la diferencia entre su cota superior e inferior. Así, podemos tomar $w(j,J)$ como el cociente entre el rango de la variable x_j en el nodo actual (el nodo hijo en el que nos encontremos durante el algoritmo de ramificación y acotación) y el rango de la misma en el nodo padre. De esta forma, asignaremos una mayor prioridad a aquellas variables cuyo rango se haya reducido en menor medida en el avance del algoritmo.

Los otros dos criterios considerados van a consistir en calcular, para cada posible variable de ramificación, la centralidad de vector propio (*eigenvector centrality* o *eigencentrality*) de dos grafos. De forma general, tendremos que se trata de una medida de centralidad utilizada para cuantificar el nivel de influencia de un nodo en un grafo. Así, se asignarán valores de influencia relativos a cada uno de los nodos de la red basándose en el supuesto de que las conexiones entre los nodos de alta influencia contribuyen más a la importancia del nodo en cuestión que las conexiones con los nodos de baja influencia. Valores altos de esta medida de centralidad significan que un nodo está conectado con muchos nodos que a su vez tienen una influencia elevada en el grafo.

En mayor detalle, supongamos que disponemos de un grafo $G(E,V)$ no orientado, consistente en un conjunto de vértices V y un conjunto de aristas E. Sea $\mathbf{A}$ la matriz de adyacencia para este grafo, la cual se define como $a_{ij} = 1$ si los vértices i y j están conectados mediante una arista, y como $a_{ij} = 0$ en caso contrario. Como A es una matriz simétrica, tendremos que sus autovalores serán reales, sus autovectores serán ortogonales y será diagonalizable (ver Golub and Loon (1983)). La centralidad de vector propio, $\mathbf{x}$, se puede describir entonces de dos formas equivalentes, como una ecuación matricial y como una suma:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \quad \lambda x_i = \sum_{j=1}^{n} a_{ij}x_j \quad i = 1,\dots,n.$$

Por lo tanto, comprobamos que la importancia de un vértice, cuantificada a través de esta medida de centralidad, será proporcional a la suma de las medidas de centralidad de los nodos con los que está conectado. Por su parte, λ será el mayor autovalor de la matriz A y n el número total de nodos en el grafo.

Por último, describiremos brevemente cómo se definen los dos grafos que estamos considerando para la selección de los dos criterios de ramificación, teniendo en cuenta la anterior medida de centralidad:

- El primer grafo recibe el nombre de intersección variable-variable (*intersection variable-variable*) y consiste en definir un grafo con tantos nodos como variables tiene el problema y poner aristas entre dos nodos cuando las variables correspondientes aparezcan en el mismo monomio.
- El segundo grafo se denota por intersección monomio-restricción (*intersection monomial-constraint*). Se basa en considerar un grafo donde tenemos tantos nodos como monomios y restricciones. En particular, definiremos una arista entre un monomio y una restricción cuando dicho monomio aparece en la restricción correspondiente.

1.3.2. Selección del criterio óptimo: KPI

El objetivo final de esta memoria será aprender sobre los criterios de ramificación definidos en el apartado anterior. En concreto, dispondremos de una serie de conjuntos de problemas de programación polinómica, los cuales serán resueltos haciendo uso del solver `RAPOSa`. Como ya hemos comentado, la implementación de esta técnica lleva asociada la incorporación del problema en un algoritmo de ramificación y acotación. En el avance de este algoritmo nos encontramos con el problema de selección de la variable de ramificación y, por ende, la elección del criterio de ramificación.

Ahora bien, no estableceremos una de las reglas de ramificación definidas previamente de forma manual. El interés de esta memoria será conseguir que el método seleccione aquel criterio más adecuado, dependiendo del problema que esté minimizando en ese momento. Es decir, estudiaremos el desempeño de los diferentes criterios sobre un gran número de problemas de optimización polinómica con el objetivo de que el método aprenda sobre los mismos y sea capaz de seleccionar el criterio óptimo cuando se enfrente a un problema nuevo, no incluido en el proceso de entrenamiento. De esta forma, enmarcamos nuestro problema en el contexto de los problemas de aprendizaje supervisado, que serán descritos en detalle, junto con algunas de las principales técnicas para su resolución, en el Capítulo 2.

Necesitaremos, por lo tanto, evaluar el desempeño de los diferentes criterios de ramificación en el avance del algoritmo. Para ello, definiremos una serie de medidas de rendimiento o KPIs (*Key Performance Indicator*):

- *Gap* óptimo (*optimality gap*). Diferencia entre la cota superior y la cota inferior que exista, en el momento de detención del algoritmo, para la función objetivo.
- Número de iteraciones. Contabiliza el número de veces que `RAPOSa` ramificó (en concreto, será el número de ramificaciones realizadas por el algoritmo de ramificación y acotación). Por otra parte, definimos el *iteration worst gap*. En primer lugar, para cada criterio de ramificación, se comprueba el *gap* óptimo obtenido y se devuelve su valor para la regla de ramificación que presenta una peor actuación. Finalmente, esta medida nos proporciona el número de iteraciones que necesitaron cada uno de los criterios para alcanzar dicho *gap*.
- Tiempo, en segundos, empleado en la ejecución. En este caso, podemos definir también el *time worst gap*, que es análogo al *iteration worst gap*, pero, en lugar de devolvernos el número de iteraciones necesarias para alcanzar el peor *gap* por cada uno de los criterios de ramificación, nos devuelve el tiempo empleado en llegar a dicho *gap*.
- Profundidad. Al trabajar con un algoritmo de ramificación y acotación, se genera un árbol en el que podemos medir su profundidad, esto es, el número de hojas construidas.

Todas las medidas presentadas hasta ahora aparecen con frecuencia en la literatura, es decir, se utilizan habitualmente para medir el desempeño de criterios o incluso de métodos de resolución para problemas de optimización. Ahora bien, en este trabajo no utilizaremos ninguna de las anteriores medidas para cuantificar la mejor actuación de un criterio frente a los demás, sino que definiremos una nueva, denominada "ritmo", que será nuestra medida de rendimiento o KPI.

La motivación en la construcción de esta nueva KPI surge de la deficiencia presente en las anteriores medidas para recoger información en términos de *gap* y tiempo de ejecución, es decir, para combinar ambas medidas en el proceso de selección del criterio de ramificación óptimo. Por esta razón, definimos una nueva medida que nos proporcionará más información en nuestro proceso de aprendizaje. En concreto, definiremos nuestra KPI como el siguiente cociente:

$$\mathrm{KPI} = \frac{t}{R}, \tag{1.7}$$

donde t es el tiempo de ejecución y R representa la reducción de la cota inferior para la función objetivo de nuestro problema de programación polinómica. Así, si denotamos la cota inferior de la función objetivo al comienzo del algoritmo de ramificación y acotación por LB_i, y la cota inferior obtenida al detenerse el mismo por LB_f, tendríamos que su incremento viene dado por

$$R = \mathrm{LB}_f - \mathrm{LB}_i. \tag{1.8}$$

Cabe destacar que hemos utilizado cotas inferiores, ya que no se dispone de la cota superior inicial en muchos de los problemas considerados. Es decir, es preferible recoger la información relativa al *gap* utilizando cotas inferiores en lugar de considerar simplemente el incremento del mismo.

Con esto, hemos definido una nueva medida capaz de unificar la información correspondiente al desempeño de nuestros criterios de ramificación en términos de *gap* y tiempos de ejecución. Cabe destacar que la medida de la profundidad del árbol no será prioritaria y, por lo tanto, no nos hemos centrado en ella. Ahora bien, el siguiente paso, como se verá teóricamente en el próximo capítulo, será definir la variable respuesta, es decir, la variable de interés para nuestro problema de aprendizaje supervisado. Esta variable, obtenida para cada criterio a partir de la medida KPI, viene dada por la siguiente expresión:

$$Y = \frac{\mathrm{KPI}_{\mathrm{best}}}{\mathrm{KPI}} = \frac{t^{\mathrm{best}}/(\mathrm{LB}_f^{\mathrm{best}} - \mathrm{LB}_i^{\mathrm{best}})}{t/(\mathrm{LB}_f - \mathrm{LB}_i)} = \frac{t^{\mathrm{best}}/R^*}{t/R}, \tag{1.9}$$

donde observamos que R^* coincide con el mejor valor de la expresión (1.8), esto es, la mayor reducción de la cota inferior para la función objetivo de nuestro problema. En particular, tendremos que $Y \in [0, 1]$, lo cual será una gran ventaja a la hora de representar e interpretar los resultados obtenidos en el proceso de aprendizaje. En concreto, tendremos que cuanto más próximo a uno esté este valor para un determinado criterio de ramificación, mejor será la actuación del mismo.

Capítulo 2

Técnicas de aprendizaje supervisado

En este capítulo de la memoria se llevará a cabo una revisión teórica de algunas de las técnicas de aprendizaje supervisado más comúnmente utilizadas tanto en problemas de clasificación como de regresión. En particular, serán importantes para comprender los resultados prácticos del capítulo 3.

Comenzaremos hablando de la denominada ciencia de datos (*data science*; también conocida como *science of learning*). Esta área de conocimiento juega un papel fundamental en el campo de la estadística, la minería de datos (*data mining*) y la inteligencia artificial, así como en determinadas ramas de la ingeniería y otras disciplinas. La idea fundamental será "dejar hablar a los datos", esto es, aprender de ellos. Típicamente, dispondremos de una o varias variables de interés, también denominadas variables respuesta (*outputs, dependent variables*), que podrán ser cuantitativas o cualitativas, y las cuales querremos predecir basándonos en un conjunto de variables explicativas, predictores o, también denominados, atributos (*inputs, independent variables, features*). En concreto, dispondremos de un conjunto de observaciones de nuestras variables respuesta y de una serie de atributos, obtenidos a partir de un conjunto de objetos, que en nuestro caso serán los problemas de programación polinómica. Este será nuestro conjunto de datos de entrenamiento (*training set*). A partir de este conjunto de datos seremos capaces de construir un modelo (*learner*) con el que realizar predicciones de las variables de interés sobre nuevas observaciones. En concreto, diremos que un modelo adecuado será aquel que realice buenas predicciones sobre las variables respuesta.

El escenario descrito anteriormente se corresponde con un problema de aprendizaje supervisado (*supervised learning problem*), pues disponemos de una o varias variables respuesta en las que radica el interés de nuestro proceso de aprendizaje. En los problemas de aprendizaje no supervisado (*unsupervised learning problems*) únicamente dispondremos del conjunto de atributos, pero no consideraremos ninguna variable de interés. Así, el objetivo será describir el conjunto de datos y tratar de organizarlo mediante grupos o *clusters*. En este caso, debido a la naturaleza de nuestro problema práctico, únicamente nos centraremos en el aprendizaje supervisado. En efecto, durante la Sección 1.3.2 ya introducimos nuestra variable de interés, obtenida para cada criterio de ramificación a partir de nuestra medida de rendimiento o KPI, denominada "ritmo".

En las sucesivas secciones nos centraremos en describir diferentes técnicas de aprendizaje supervisado, que posteriormente serán utilizadas en el proceso de aprendizaje para llevar a cabo la predicción de nuestra variable respuesta. En particular, nos vamos a encontrar ante un problema de regresión, puesto que nuestra respuesta es cuantitativa. Cabe destacar que algunos de los métodos estudiados también podrán utilizarse para la resolución de problemas de clasificación, donde la respuesta será cualitativa. Para obtener más detalles sobre el tipo de variables con las que podremos trabajar, así como la terminología que seguiremos en el desarrollo del capítulo, puede consultarse la Sección 2.2 de Hastie et al. (2009).

2.1. Modelos aditivos generalizados (GAMs)

Los modelos de regresión juegan un papel fundamental en el análisis de datos, realizando predicciones y proporcionando reglas de clasificación. Asimismo, serán una herramienta fundamental para el estudio y comprensión de la importancia de los predictores.

El modelo de regresión más extendido y conocido es el lineal (*linear model*, LM). En este modelo de mínimos cuadrados ordinarios o lineales disponemos de una variable respuesta Y, que es el objeto de nuestro estudio y que, además, asumimos que sigue una distribución normal de media μ y varianza σ^2, condicionada a X. Así, dado un conjunto de atributos, tendremos un modelo de la forma:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon, \text{ donde } \epsilon \sim \mathcal{N}(0, \sigma^2).$$

A pesar de tratarse de un modelo muy simple, no suele ajustarse a la realidad de los datos. En muchas situaciones, los efectos de las variables explicativas no van a ser lineales. Este será el escenario en el que nos encontraremos en nuestro problema y, en concreto, el motivo por el cual deberemos trabajar con métodos más flexibles que nos permitan incluir y caracterizar los efectos no lineales de los predictores. Es aquí donde aparecen los llamados modelos aditivos generalizados (*generalized additive models*, GAMs).

Comenzaremos definiendo lo que se entiende por modelo aditivo generalizado, siguiendo la notación de Hastie et al. (2009). En primer lugar, tenemos que dichos modelos se enmarcan en el contexto de los modelos semiparamétricos de regresión, cuyo principal objetivo es resolver el problema del desastre de dimensionalidad que originaría una suavización con múltiples variables explicativas. Así, son el resultado de extender un modelo lineal múltiple permitiendo que cada elemento del modelo sea una función no lineal de un predictor y manteniendo la aditividad. De forma más precisa, tendremos que el modelo aditivo generalizado presenta la siguiente formulación:

$$E(Y|X) = E(Y|X_1, X_2, \ldots, X_p) = \alpha + f_1(X_1) + f_2(X_2) \ldots + f_p(X_p) = f(X), \tag{2.1}$$

donde $X_1, X_2, \ldots, X_p$ representan los predictores e Y es la variable de interés. Por su parte, las funciones f_j, $j = 1, 2, \ldots, p$, son funciones no lineales y suaves (*smooth functions*), que no se definen previamente y recogen los efectos no paramétricos de los predictores. Por lo tanto, tenemos que los modelos aditivos generalizados son una combinación lineal de funciones no lineales y, al igual que ocurre con los modelos lineales múltiples, se pueden incorporar tanto predictores continuos como cualitativos.

Dentro de este tipo de modelos también nos encontramos con los modelos parcialmente lineales y los modelos aditivos. Utilizaremos los modelos aditivos cuando la variable respuesta sea continua y, en concreto, cuando se pueda asumir que sigue una distribución normal. Por su parte, los modelos aditivos generalizados se aplican a una variable respuesta que se supone que sigue un modelo de distribución conocido distinto de la normal. En general, la media condicionada, $\mu(X)$, de la variable respuesta Y está relacionada con una combinación lineal de los predictores a través de la llamada función de enlace (*link*), g:

$$g[\mu(X)] = \alpha + f_1(X_1) + f_2(X_2) \ldots + f_p(X_p)$$

Algunos ejemplos de funciones de enlace son los siguientes:

- $g(\mu) = \mu$ es la función identidad, utilizada para los modelos lineales y aditivos con respuesta gaussiana.
- $g(\mu) = \text{logit}(\mu)$ para el caso logístico o $g(\mu) = \text{probit}(\mu)$, para modelar probabilidades binomiales. La función probit es la inversa de la función de distribución gaussiana: $\text{probit}(\mu) = \psi{-}1(\mu)$.
- $g(\mu) = \log(\mu)$ para modelos lineales y aditivos cuya respuesta sigue una distribución Poisson (respuesta de modo recuento).

Todas estas funciones de enlace pertenecen a la llamada familia exponencial, que a su vez incluye a las distribuciones gamma y binomial negativa.

A continuación, vamos a centrarnos en cómo modelar las funciones suaves f_j. Como bien hemos comentado, sería extremadamente restrictivo suponer que la verdadera función $f(X)$, con $f = (f_1, \ldots, f_p)$, es lineal en X. En la mayoría de los problemas de regresión, $f(X) = \mathbb{E}(Y|X)$ será una función no lineal y, a mayores, no aditiva de X. Por lo tanto, aproximar $f(X)$ por una función lineal será conveniente e incluso necesario en determinadas ocasiones (principalmente por la fácil interpretabilidad del modelo lineal). Como se puede ver en el Capítulo 5 de Hastie et al. (2009), ajustando cada uno de estos términos mediante expansiones de bases de funciones, obtendríamos un modelo que podría ajustarse simplemente mediante el método de mínimos cuadrados. En particular, dado el vector de atributos X podemos definir la m-ésima transformación de X como la función $h_m : \mathbb{R}^p \longrightarrow \mathbb{R}$ de forma que

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X), \ m = 1, \ldots, M. \tag{2.2}$$

Así, obtenemos una expansión lineal mediante bases de funciones en X. Siguiendo esta metodología, una vez que determinemos las funciones de la base h_m, estaremos ante un modelo lineal en estas nuevas variables y podremos ajustarlo con los métodos habituales. Algunos ejemplos de funciones h_m que se utilizan con frecuencia son los siguientes:

- $h_m(X) = X_m$, $m = 1, \ldots, p$, que nos devuelve el modelo lineal original.
- $h_m(X) = X_j^2$ o $h_m(X) = X_j X_k$, nos permite introducir interacciones entre nuestros atributos a través de términos polinómicos, obteniendo así expansiones de Taylor de mayor orden.
- $h_m(X) = \log(X_j)$, $\sqrt{X_j}$, ..., permiten introducir transformaciones no lineales en determinados atributos.

En nuestro caso particular, ajustaremos cada una de estas funciones no paramétricamente mediante un suavizado del gráfico de dispersión (*scatterplot smoothing*), el cual consiste en resaltar la tendencia subyacente de los datos, y proporcionaremos un algoritmo que nos permita estimar simultáneamente las p funciones. Existen numerosos métodos para realizar este suavizado, incluyendo splines, regresión tipo núcleo, *loess* o su versión ponderada, *lowess*. Nos centraremos en los splines y, en concreto, cabe destacar los splines de suavización, los splines de regresión y los splines de penalización, los cuales describiremos detalladamente a continuación.

2.1.1. Efectos no paramétricos: estimación mediante splines

La estimación mediante splines es una metodología que tiene como objetivo recoger la tendencia subyacente de los datos mediante el ajuste de una curva que pase cerca de los puntos muestrales y al mismo tiempo sea suave. De forma general, podemos definir un spline como una función polinómica a trozos. Así, dependiendo de si el grado del polinomio en cada trozo es uno, dos o tres, obtendremos splines lineales, cuadráticos o cúbicos, respectivamente. En concreto, el spline de grado cero consistirá en una función polinómica a trozos para la que hemos considerado polinomios de grado cero (funciones constantes) en cada uno de los intervalos definidos por los nodos de la partición. Así, este spline quedará totalmente determinado conociendo el valor constante que toma en cada subintervalo. En general, los splines dependerán del grado del polinomio, el número de nodos y la localización de los nodos.

La elección habitual consiste en seleccionar polinomios a trozos de grado tres, esto es, splines cúbicos. Es más, no existe ninguna razón que nos lleve a considerar polinomios de grado mayor, a menos que estemos interesados en sus derivadas suavizadas. Por lo tanto, en la práctica, solo nos centraremos en los splines de grado tres, como mucho. Si consideramos un conjunto de nodos $\xi_1, \ldots, \xi_K$,

y suponemos que están situados en el intervalo $[a, b]$ de la forma $a < \xi_1 < \xi_2 < \ldots < \xi_K < b$, entonces podemos representar un spline cúbico como una función de la forma

$$g(\xi) = d_i(\xi - \xi_i)^3 + c_i(\xi - \xi_i)^2 + b_i(\xi - \xi_i) + a_i, \text{ para } \xi \in [\xi_i, \xi_{i+1}]$$

Claramente se trata de un polinomio de grado tres en cada subintervalo. Asimismo, pediremos que la función g y sus dos primeras derivadas sean continuas en los nodos, lo cual se traduce en una serie de restricciones sobre los coeficientes de los polinomios en intervalos consecutivos. Si además consideramos que la segunda y tercera derivada de g son nulas en a y b (nulas en los subintervalos extremos), entonces estaremos ante un spline cúbico natural.

Siguiendo con la notación de 2.2, podemos introducir la conocida como base exponencial truncada, donde las funciones de la base h_m presentan la siguiente forma:

$$h_j(X) = X^{j-1}, j = 1, \ldots, M,$$
$$h_{M+l}(X) = (X - \xi_l)_+^{M-1}, l = 1, \ldots, K.$$

En concreto, diremos que M es el orden del spline, siendo $M = 4$ en el caso de los splines cúbicos. De esta forma, tenemos que el orden M se obtiene sumando uno al grado del spline. En la práctica, los órdenes más habituales serán $M = 1, 2$ y 4.

Splines de regresión

Si recordamos la definición de spline presentada anteriormente, vemos que partimos de un conjunto de nodos prefijados. Este tipo de splines también se conocen como splines de regresión (*regression splines*). En ellos será fundamental seleccionar el orden del spline, el número de nodos y su posición. En particular, este método consiste en escoger una base de funciones de tipo spline, y proyectar los datos sobre dicha base por mínimos cuadrados. De esta forma, se trata de un método de ajuste no paramétrico fácil de entender conceptualmente, rápido de calcular y próximo a los conceptos de regresión paramétrica.

En general cuando hablamos de bases de funciones de tipo spline, nos estaremos refiriendo a la base de B-splines definida en de Boor (1978). No obstante, el conjunto de splines es un espacio vectorial de dimensión el número de grados de libertad (K + grado + 1, donde K es el número de nodos considerados, sin contar los nodos extremos y el grado se corresponde con el de la función polinómica en cada subintervalo) y, por lo tanto, admite muchos otros sistemas de bases, que dependiendo de la naturaleza de los datos muestrales, pueden ser más o menos adecuados (por ejemplo, la base exponencial truncada). En particular, cualquier función spline S se puede expresar como

$$S(x) = \sum_{k=1}^{K+orden} c_k B_k(x, \tau),$$

siendo $B_k(x, \tau)$ la k-ésima función de la base de B-splines definida sobre el conjunto de nodos τ, y evaluada en el punto x. Por su parte, tendremos que c_k son los coeficientes correspondientes a la función spline S en la base. Para obtener información detallada sobre la base de B-splines, se puede consultar el Apéndice del Capítulo 5 del Hastie et al. (2009).

En particular, tendremos que los splines de regresión no presentarán problemas en la frontera, como ocurre con otros métodos de suavizado (suavizadores de tipo núcleo), en los que el extender la curva ajustada fuera del dominio de los datos hace que esta tienda hacia cero. Además, la base de B-splines se construye con el propósito de que los elementos de la base se solapen lo menos posible, tratando de reducir la colinealidad entre ellos. En de Boor (1978) se presenta un algoritmo para el cálculo de la base de B-splines, fácil de implementar en `R`. En concreto, la función `bs` del paquete **splines** nos permitirá obtener la base de B-splines en una rejilla de valores fijos de x, una vez fijados los nodos.

Splines de suavización

En este apartado nos centraremos en el método de splines de suavización (*smoothing splines*) que evita el problema de seleccionar los nodos a priori mediante el uso de un conjunto máximo de nodos. La complejidad del ajuste, por su parte, será controlada mediante penalización. Así, consideremos el problema de minimizar la siguiente función objetivo mixta, que combina la suma residual de cuadrados con una penalización por rugosidad (curvatura):

$$RRS(f,\lambda) = \sum_{i=1}^{N}\{y_i - g(x_i)\}^2 + \lambda \int (f''(t))^2 dt, \tag{2.3}$$

donde λ es el parámetro de suavización. Así, tendremos que el primer término mide si el ajuste recoge adecuadamente la tendencia subyacente de los datos, mientras que el segundo término penaliza la curvatura de la función. Por su parte, λ controla el balance entre el sesgo y la varianza de la curva ajustada. En particular, hay dos casos especiales:

- $\lambda = 0$: f podrá ser cualquier función que interpolará las observaciones.
- $\lambda = \infty$: la segunda derivada se hace 0, por lo que tendremos un ajuste lineal.

Sea X un predictor, entonces vamos a denotar por $\mathbf{x}$ al vector de las N observaciones de dicha variable explicativa. En particular, se puede demostrar que el problema de minimización (2.3) tiene una única solución, que es el spline cúbico natural con nodos en los valores observados x_i, $i = 1, \ldots, N$. Así, podemos representar el spline cúbico natural, solución del problema, como sigue:

$$f(x) = \sum_{i=1}^{N} N_i(x)\theta_i,$$

donde los $N_i(x)$ representan el conjunto N-dimensional de funciones básicas que permiten representar la familia de los splines naturales.

Regresión spline penalizada

Finalmente, concluiremos este primer apartado relativo a la estimación spline comentando la regresión spline penalizada (*P-Splines*). Así, como hemos visto hasta ahora, disponemos de dos enfoques para llevar a cabo la estimación de la función de regresión mediante splines: los splines de suavización y los splines de regresión. En particular, los splines de suavización utilizan tantos parámetros como observaciones, lo que hace que su implementación no sea eficiente cuando el número de datos es muy elevado. Por otra parte, los splines de regresión pueden ser ajustados mediante mínimos cuadrados una vez que se ha seleccionado el número de nodos y su localización. Sin embargo, no existe un criterio automático para la realización de dichos procesos, lo que nos conduce a algoritmos de selección de nodos bastante complicados.

Teniendo en cuenta estos aspectos, surgen los splines con penalizaciones, combinando ambos enfoques de forma que se utilicen menos parámetros que en los splines de suavización, pero que al mismo tiempo la selección del número de nodos y su localización no sea tan determinante como en los splines de regresión. Así, si en el problema de minimización de la función objetivo mixta (2.3) no imponemos ninguna restricción a la curva f, obtendremos como resultado el spline de suavización (cúbico). Ahora bien, si restringimos la función f a una expresión finita dentro de una base, como la base de B-splines, obtendremos los splines penalizados.

Como ya hemos comentado en 2.2, tenemos que si f se restringe a una base, entonces podrá expresarse como sigue:

$$f(x) = \sum_{m=1}^{M} \beta_m b_m(x)$$

donde b_m sería la m-ésima función de la base, evaluada sobre el punto x, y β_m son los coeficientes correspondientes a la función f en dicha base. Así, tendremos que:

$$S(g) = (Y - X\beta)'(Y - X\beta) + \lambda\beta' K\beta$$

siendo K una matriz de penalización, que dependerá de la base. Minimizando esta expresión, tendremos que la solución del problema viene dada por $\hat{\beta} = (X'X + \lambda K)^{-1}X'Y$, de forma que se conserva el lenguaje de los modelos paramétricos. Con esto, podremos obtener predicciones mediante $\hat{Y} = X\hat{\beta}$. En particular, el parámetro de suavización λ se escogerá por un criterio de validación cruzada o validación cruzada generalizada.

Además de considerar una base de splines, también podríamos pensar en restringir la función f a una familia paramétrica o a otras bases, para obtener el spline de penalización. A grandes rasgos, tendremos que se trata de suavizadores muy sencillos de implementar y utilizar, y que permiten una gran flexibilidad en la modelización. El único inconveniente de considerar una base de splines para restringir la función f, es que tendremos que fijar el número de nodos y su localización. Aun así, mediante esta selección de los nodos podremos adaptar la base a las características previstas de la función de regresión.

2.2. Boosting

La metodología boosting es una de las herramientas de aprendizaje más poderosas introducidas en los últimos veinte años. En un principio, fue diseñado para su uso en problemas de clasificación, sin embargo, puede extenderse a problemas de regresión. La idea fundamental de este método es ajustar, de forma secuencial, múltiples modelos sencillos, que predicen ligeramente mejor que lo esperado por el mero azar (predictores débiles o *weak learners*, en inglés). De esta forma, cada nuevo modelo utilizará información procedente del construido en la etapa anterior, mejorando así las predicciones en cada iteración. Los árboles de decisión pequeños, esto es, construidos con poca profundidad, resultan muy convenientes para esta técnica, ya que no consiguen obtener buenas predicciones. Además, diferenciándose del método de *bagging* (Sección 8.7 de Hastie et al. (2009)), tenemos que el boosting no hará uso de la técnica bootstrap para generar numerosas muestras de entrenamiento distintas, sino que la diferencia entre los árboles se deberá a la modificación en la importancia o peso de las observaciones a lo largo del proceso.

Las primeras ideas de este método fueron desarrolladas por Valiant (1984) y Kearns and Valiant (1994), los cuales no encontraron una implementación efectiva del mismo. A continuación vamos a introducir el algoritmo AdaBoost.M1, presentado por Freund and Schapire (1997), que logra implementar esta técnica en el contexto de los problemas de clasificación. Para ello, consideraremos un problema con dos categorías y variable respuesta codificada como $Y \in \{-1, 1\}$. Dado un vector de variables predictoras, X, un clasificador $G(X)$ nos devolverá como predicción uno de los dos valores, 1 o -1. La tasa de errores de clasificación en la muestra de entrenamiento viene dada por

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

y la tasa de error esperada para las predicciones futuras será $\mathbb{E}_{XY}\mathbb{I}(Y \neq G(X))$. Como ya comentamos, en el método boosting modificaremos secuencialmente el conjunto de observaciones, generando una sucesión de predictores débiles, $G_m(x), m = 1, 2, \ldots, M$, para, finalmente, combinar todas las predicciones realizadas a lo largo del proceso y obtener la clasificación final,

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right).$$

Los $\alpha_1, \alpha_2, \ldots, \alpha_M$ se corresponden con los pesos asociados a cada uno de los modelos ajustados a lo largo de las M iteraciones del proceso. Su objetivo es asignar una mayor influencia a aquellos $G_m(x)$ que proporcionen una clasificación más exacta. Por su parte, la modificación del conjunto de datos se consigue mediante el uso de pesos, $w_1, w_2, \ldots, w_N$, en las observaciones del conjunto de entrenamiento $(x_i, y_i), i = 1, 2, \ldots, N$. Inicialmente, los pesos para cada observación serán los habituales, $w_i = 1/N$. En las sucesivas iteraciones, $m = 2, 3, \ldots, M$, se modificará el peso de cada observación, individualmente, y se repetirá el algoritmo de clasificación en el nuevo conjunto de datos ponderado. De esta forma, a medida que avanza el método, las observaciones difíciles de clasificar correctamente recibirán una influencia cada vez mayor.

El algoritmo AdaBoost.M1 se trata de un método "discreto", en el sentido de que tenemos una respuesta dicotómica. Sin embargo, puede modificarse adecuadamente, transformando nuestros clasificadores, $G_m(x)$, de manera que devuelvan como predicciones valores continuos (ver "*Real AdaBoost*" en Friedman et al. (2000)). Por su parte, se puede consultar en detalle este algoritmo, descrito en la Sección 10.1 de Hastie et al. (2009), así como el resto del capítulo, si se desea profundizar en la metodología boosting. En lo que respecta a esta memoria, únicamente nos vamos a centrar en el algoritmo de *gradient boosting*, también conocido como potenciación del gradiente, que desarrollaremos en el siguiente apartado, y que es fundamental en el campo del aprendizaje supervisado cuando nos encontremos ante problemas de regresión.

2.2.1. *Gradient boosting*

Este algoritmo fue desarrollado por Friedman (2001) y pertenece a la familia de los métodos iterativos de descenso del gradiente. Cabe destacar que este método permitió extender la técnica de boosting a problemas de regresión, así como establecer una conexión entre esta herramienta de aprendizaje y otros campos próximos, como son los modelos aditivos o la regresión logística. La idea fundamental de este método será encontrar un modelo aditivo que minimice una determinada función de pérdida utilizando predictores débiles; en nuestro caso concreto, árboles de poca profundidad.

De forma general, tendremos que la pérdida ocasionada al utilizar $f(x)$ para predecir y en los datos de entrenamiento viene dada por

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)), \tag{2.4}$$

donde $f(x)$ es nuestro predictor débil y se restringe a una suma de árboles. Así, el objetivo será minimizar $L(f)$ con respecto a f. Ignorando la restricción impuesta a $f(x)$, tendremos que minimizar (2.4) puede verse como un problema de optimización numérica

$$\hat{\mathbf{f}} = \arg \min_{\mathbf{f}} L(\mathbf{f}), \tag{2.5}$$

donde los "parámetros" $\mathbf{f} \in \mathbb{R}^N$ son los valores que toma la función de aproximación, $f(x_i)$, en cada uno de los N datos de entrenamiento,

$$\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T.$$

Utilizando procedimientos de optimización numérica para resolver (2.5), obtenemos la siguiente solución del problema de minimización

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \ \mathbf{h}_m \in \mathbb{R}^N,$$

donde $\mathbf{f}_0 = \mathbf{h}_0$ sería una solución inicial, y cada $\mathbf{f}_m$ se obtiene a partir del vector de "parámetros" $\mathbf{f}_{m-1}$, correspondiente a la suma de los valores obtenidos en las etapas anteriores. En concreto, podríamos

seleccionar $\mathbf{h}_m = -\rho_m \mathbf{g}_m$, donde ρ_m es un escalar y $\mathbf{g}_m \in \mathbb{R}^N$ es el gradiente de $L(f)$ evaluado en $\mathbf{f} = \mathbf{f}_{m-1}$. Las componentes de dicho gradiente vienen dadas por:

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}. \tag{2.6}$$

Sin embargo, este gradiente está únicamente definido sobre los datos del conjunto de entrenamiento, x_i, mientras que nuestro objetivo con este algoritmo es generalizar $f_M(x)$ a un nuevo conjunto de observaciones, no representado por nuestra muestra de entrenamiento. Así, la solución que ofrece el algoritmo *gradient boosting* consiste en introducir un árbol, $T(x; \Theta_m)$, en la m-ésima iteración, de forma que sus predicciones estén lo más cerca posible de la expresión del gradiente (2.6), con signo negativo. Siguiendo esta idea, y considerando el error cuadrático como medida de esta "proximidad", obtenemos que

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2. \tag{2.7}$$

Esto es, ajustamos el árbol T con los valores obtenidos a partir del gradiente negativo, mediante mínimos cuadrados. En la Tabla 10.2 de la Sección 10.10 de Hastie et al. (2009), encontramos un resumen con los gradientes obtenidos para las principales funciones de pérdida. En concreto, al considerar como función de pérdida la suma residual de cuadrados, $L(f) = \frac{1}{2}(y_i - f(x_i))^2$, obtenemos que el gradiente cambiado de signo coincide, precisamente, con los residuos, $-g_{im} = y_i - f_{m-1}(x_i) = r_{im}$. De esta forma, (2.7) será equivalente al método boosting por mínimos cuadrados habitual.

A continuación, veamos el algoritmo 1 para un problema de regresión considerando como función de pérdida la suma residual de cuadrados. En concreto, se tratará de un proceso iterativo que dependerá de tres parámetros fundamentales: el número de iteraciones, M, el parámetro de regularización, λ y el número de cortes de cada árbol, d. Además, comprobamos como la optimización se realiza en las sucesivas etapas, y no globalmente.

Algoritmo 1 Algoritmo *Gradient Tree boosting*

1. Seleccionar los valores de los tres parámetros, M, λ y d, e inicializar el algoritmo estableciendo una primera predicción constante, con valor 0. Así, para $i = 1, 2, \ldots, N$, podemos obtener los residuos de las observaciones del conjunto de entrenamiento, x_i, $r_{i0} = y_i$.

2. Para $m = 1, \ldots, M$:

   a. Ajustamos un árbol de regresión, $T(x; \Theta_m)$, con d cortes y utilizando los valores obtenidos a partir del gradiente negativo, sabiendo que se corresponden con los residuos, r_{im}.

   b. Calculamos la versión regularizada del árbol, $\lambda T(x; \Theta_m)$.

   c. Actualizamos los residuos,
   $$r_{im} \longleftarrow r_{im} - \lambda T(x; \Theta_m).$$

3. Finalmente, obtenemos nuestro modelo boosting,
$$\hat{f}(x) = \sum_{m=1}^{M} \lambda T(x; \Theta_m).$$

En la Sección 10.1 de Hastie et al. (2009) podemos encontrar una versión más general de este algoritmo. Asimismo, cabe destacar la importancia de la selección óptima de los tres parámetros del

algoritmo. En concreto, tenemos que un elevado número de árboles podría conducirnos a problemas de sobreajuste, lo cual diferencia a este método del *bagging* y de los bosques aleatorios, donde la independencia de los árboles evita este problema.

Finalmente, cabe destacar el algoritmo *stochastic gradient boosting*, propuesto por Friedman (2001) e inspirado por la técnica *bagging* de Breiman. Se trata de una variante del método *gradient boosting*, en la que, durante la etapa 2.*a* del algoritmo anterior, no se considera toda la muestra de entrenamiento para realizar el ajuste del árbol de regresión, sino que se selecciona al azar un subconjunto. Esto incorpora un nuevo parámetro al método, correspondiente a la fracción utilizada de los datos. Así, aunque esta variante ofrece importantes ventajas, también puede resultar más complicada en términos de ajuste de los parámetros.

2.3. Bosques aleatorios

Los bosques aleatorios (*random forests*, ver Breiman (2001)) se tratan de una modificación de la metodología *bagging* (*bootstrap aggregation*, ver Sección 8.7 de Hastie et al. (2009)), conocida como empaquetado, para el caso de árboles de decisión. Este último método consiste en la utilización de bootstrap para generar numerosas muestras de entrenamiento y, de esta manera, entrenar un modelo distinto con cada una de ellas. De este modo podremos hacer predicciones obteniendo tantas como modelos, esto es, tantas predicciones como muestras de entrenamiento. Finalmente, se realizará un promedio de todas las predicciones obtenidas, reduciendo considerablemente la varianza y simplificando la solución. En concreto, tenemos que cada árbol generado en el proceso de *bagging* está idénticamente distribuido (i.d.), a diferencia de lo que ocurría en el método boosting, de manera que el valor esperado del promedio de B árboles será el mismo que el valor esperado de cada uno de ellos. Por lo tanto, el sesgo permanecerá constante, y solo podremos conseguir una mejora a través de la reducción de la varianza.

El promedio de B variables aleatorias i.i.d., cada una de ellas con varianza σ^2, tiene varianza $\frac{1}{B}\sigma^2$. Si las variables son i.d. (idénticamente distribuidas, pero no necesariamente independientes), con una correlación positiva ρ dos a dos, entonces la varianza de la media será

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{2.8}$$

A medida que aumentamos el número de árboles, B, el segundo término desaparece, de manera que el beneficio obtenido con el promedio dependerá de la correlación existente entre los pares de árboles. Partiendo de esta idea, el método de bosques aleatorios trata de obtener una reducción de varianza mayor, pero sin empeorar en términos de sesgo. Así, se introduce una aleatoriedad en las variables (y no solo en las observaciones), pues antes de realizar cada uno de los cortes, se seleccionan al azar $m < p$ predictores, de entre todas las p variables predictoras, que serán los candidatos para el corte. De esta manera se consigue una decorrelación de los árboles, y un desempeño similar al del método boosting, convirtiéndose ambas técnicas en las principales opciones a la hora de afrontar problemas de aprendizaje supervisado.

Una vez obtenidos B árboles, $\{T(x;\Theta_b)\}_{b=1}^{B}$, en problemas de regresión, las predicciones vendrán dadas por

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b). \tag{2.9}$$

El término Θ_b caracteriza el b-ésimo árbol obtenido con bosques aleatorios en términos de predictores seleccionados, puntos de corte en cada nodo, y valores obtenidos en los nodos terminales. Intuitivamente, reduciendo m conseguiremos disminuir la correlación entre los pares de árboles en el

ensemble[1] y, por (2.8), conseguiremos una reducción de la varianza en el promedio.

Por último, cabe destacar que en problemas de regresión es recomendable utilizar por defecto un valor de $\lfloor p/3 \rfloor$ para m, y fijar un mínimo de cinco observaciones en los nodos terminales. Aun así, en la práctica nos encontramos con que los valores óptimos para estos parámetros dependerán del problema con el que estemos trabajando, y por lo tanto, debe llevarse a cabo un proceso de tuneado de los mismos.

2.3.1. Muestras Out-Of-Bag (OOB)

En el método *bagging* y, por lo tanto, en bosques aleatorios, resulta posible estimar el error de test sin necesidad de recurrir a la validación cruzada. El hecho de que los árboles se ajusten empleando muestras generadas por bootstrapping conlleva a que cada ajuste solo utilice, aproximadamente, dos tercios de los datos. En detalle, tenemos que en un muestreo por bootstrapping, si el tamaño de los datos de entrenamiento es N, cada observación tiene una probabilidad de ser elegida de $\frac{1}{N}$. Por lo tanto, la probabilidad de no ser elegida en todo el proceso es aproximadamente un tercio. A este tercio restante se le conoce como out-of-bag.

De esta forma, para cada observación $z_i = (x_i, y_i)$, construiremos la función de predicciones como en (2.9), pero únicamente ponderando aquellos árboles en los que no aparecía dicha observación. Siguiendo este proceso, se pueden obtener las predicciones para las N observaciones y con ellas calcular el error cuadrático medio OOB, para problemas de regresión. Como la variable respuesta de cada observación se predice utilizando únicamente los árboles en cuyo ajuste no participó dicha observación, el error OOB sirve como estimación del error de test. De hecho, si el número de árboles es suficientemente alto, el error OOB es prácticamente equivalente al error de validación cruzada dejando uno fuera (*leave-one-out cross-validation error*). Por lo tanto, el error OOB estimado será prácticamente igual al obtenido mediante validación cruzada con N iteraciones y, una vez que este error se estabiliza, finaliza el proceso de entrenamiento.

En bosques aleatorios el número de árboles no es un hiperparámetro crítico en cuanto que, añadir árboles, solo puede conseguir mejorar el resultado. Esto nos viene a decir que no se va a producir sobreajuste por exceso de árboles. Sin embargo, añadir árboles una vez que la mejora se estabiliza supone un gasto computacional innecesario. En particular, tenemos que el número de muestras bootstrap que se consideran (número de árboles que se construyen) se trata de una aproximación por el método de Monte Carlo, por lo que se estudiará como un problema de convergencia del error OOB al aumentar el número de árboles. Si aparentemente hay convergencia con unos pocos cientos de árboles, no va a variar mucho el nivel de error al aumentar este número. Por otra parte, si el número de árboles es demasiado pequeño puede que se obtengan pocas (o incluso ninguna) predicciones OOB para alguna de las observaciones de la muestra de entrenamiento.

2.3.2. Importancia de los predictores

Si bien es cierto que el proceso de *bagging* y, por lo tanto, de bosques aleatorios, consigue mejorar la capacidad predictiva en comparación a los modelos basados en un único árbol, esto tiene un coste asociado, la interpretabilidad del modelo se reduce. Al tratarse de una combinación de múltiples árboles, no es posible obtener una representación gráfica sencilla del modelo y no es inmediato identificar de forma visual qué predictores son más importantes. Sin embargo, se han desarrollado nuevas estrategias para cuantificar la importancia de los predictores que hacen de estos modelos una herramienta muy potente, no solo para predecir, sino también para el análisis exploratorio.

[1] Los métodos de *ensemble* combinan múltiples modelos en uno nuevo con el objetivo de lograr un equilibrio entre sesgo y varianza, consiguiendo así mejores predicciones que cualquiera de los modelos individuales originales.

En particular, en el caso de bosques aleatorios, podremos construir gráficos de importancia de las variables explicativas de igual forma a como se procede en los modelos de *gradient boosting* (ver la Sección 10.13 de Hastie et al. (2009)). Así, para cada árbol del bosque, tendremos que en cada etapa la mejora en el criterio de ramificación se corresponderá con la importancia asociada a la variable de ramificación, y se irá acumulando a lo largo de todos los árboles generados de forma separada para cada variable.

Los bosques aleatorios también utilizan las muestras out-of-bag para determinar una nueva medida de importancia de los predictores. En particular, tratarán de medir la capacidad predictora que presentará cada variable. Así, tras la construcción de B árboles, se obtienen las predicciones a partir de las muestras OOB y, en particular, se calcula la exactitud de la predicción. A continuación, para cada predictor j, se realiza una permutación de dicha variable en todos los árboles del modelo, manteniendo el resto constante. Posteriormente, se recalcula la métrica de error y se obtiene el incremento en la exactitud debido a la permutación del predictor. Así, si la variable permutada estaba contribuyendo al modelo, es de esperar que este aumente su error, ya que se pierde la información que proporcionaba dicha variable. Por su parte, el incremento puede resultar negativo cuando la variable no contribuía inicialmente al modelo y al reorganizarla aleatoriamente, por mero azar, se consigue mejorar ligeramente el modelo resultante. En general, una variable en estas condiciones se considerará con una importancia próxima a cero.

2.4. Regresión cuantil

La regresión cuantil está emergiendo gradualmente como una metodología estadística unificada para la estimación de modelos en los que aparecen funciones de regresión cuantil, en las que el cuantil de la variable respuesta, Y, se encuentra condicionado a cierto valor de la variable o variables explicativas, X. Así, si consideramos el cuantil 0.5, estaremos ante la regresión mediana (la mediana de Y condicionada a X), mientras que si trabajamos con el cuantil 0.25, estudiaremos el efecto de la variable o variables explicativas sobre el primer cuartil de la distribución de Y.

Al complementar el enfoque clásico de la regresión en media, mediante la estimación de la recta por el método de mínimos cuadrados, la regresión cuantil ofrece una estrategia sistemática que nos permite estudiar cómo las covariables influyen en la posición, la escala y la forma de la función de distribución de la variable respuesta, no solo en la parte central de dicha distribución, sino en otras posiciones inferiores o superiores. Si bien la metodología es diferente a la empleada en regresión por mínimos cuadrados ordinarios, la interpretación final es muy similar, ya que se obtienen una serie de coeficientes que estiman el efecto que tiene cada predictor sobre un cuantil específico de la variable respuesta.

Comparándola con la regresión en media, tendremos que la regresión cuantil es una metodología flexible, capaz de adaptarse a condiciones generales de heterocedasticidad y ausencia de normalidad. Al poder realizar regresión sobre cualquier parte de la distribución somos capaces de conocer la influencia de los predictores desde el mínimo al máximo rango de la variable respuesta. Esto es especialmente útil en modelos de regresión heterocedásticos, ya que no tendremos un único ratio de cambio (pendiente) que represente bien a toda la variable respuesta a la vez. De forma general, tendremos que se trata de un procedimiento robusto frente a observaciones atípicas, a diferencia de la regresión en media, en la que un valor extremo de los datos va a provocar un cambio en la selección de los valores centrales.

La referencia fundamental en la que nos apoyaremos para el desarrollo de esta sección es el libro de Koenker (2005), en el que el autor ha estudiado exhaustivamente el tema de la regresión cuantil. Cabe destacar que esta monografía es el primer tratamiento en profundidad del tema, que abarca modelos lineales y no lineales, paramétricos y no paramétricos. Asimismo, se ilustran una serie de metodologías

con diversas aplicaciones en economía, biología, ecología y finanzas. Omitiremos los detalles de muchas de las técnicas de regresión cuantil, pudiendo consultarse los detalles metodológicos en el libro de Koenker (2005).

2.4.1. La función de pérdida cuantílica

Definiremos un cuantil como la inversa de la función de distribución. Así, diremos que el cuantil τ (donde $\tau \in (0,1)$) de la distribución de una variable respuesta Y será el punto y tal que

$$F(y) = \mathbb{P}(Y \leq y) = \tau. \tag{2.10}$$

En concreto, para que la expresión (2.10) no presente ningún problema, deberemos trabajar con variables respuesta con función de distribución F continua y estrictamente creciente. En estas condiciones, tendremos que la relación entre los valores de y y las probabilidades τ es biyectiva. Supondremos entonces que nos encontramos en este escenario.

A continuación, vamos a extender los cuantiles al contexto de la regresión. Comencemos recordando el método clásico de regresión en media, en la cual la estimación de la recta de regresión se realizaba por el método de mínimos cuadrados. Este criterio surge del hecho de que, sin variables explicativas, la media cumple que

$$\mathbb{E}(Y) = \arg\min_y \mathbb{E}\{(Y-y)^2\},$$

es decir, la media es el valor que minimiza la desviación cuadrática media. Por lo tanto, dado un conjunto de observaciones de nuestra variable explicativa, $\{y_1, \ldots, y_N\}$, tendremos que

$$\overline{y} = \arg\min_y \frac{1}{N}\sum_{i=1}^{N}(y_i - y)^2.$$

Si trabajamos con la mediana, ya hemos comentado que se trata del punto que ocupa la posición central en la muestra de observaciones ordenadas, esto es, hará mínima la desviación absoluta. Podemos expresarlo como sigue

$$\text{Mediana}(Y) = \arg\min_y \mathbb{E}(|Y-y|) \tag{2.11}$$

y, análogamente, dado un conjunto $\{y_1, \ldots, y_N\}$ de observacioens de la variable respuesta Y, tendremos que

$$\text{Mediana muestral}(Y) = \arg\min_y \frac{1}{N}\sum_{i=1}^{N}|y_i - y|.$$

En general, tendremos que el cuantil τ de Y, $Q_Y(\tau)$, se puede definir como

$$Q_Y(\tau) = \arg\min_y \mathbb{E}\{p_\tau(Y-y)\},$$

donde p_τ denota la función de pérdida cuantílica,

$$p_\tau(z) = (\tau - 1)z\mathbb{I}(z<0) + \tau z\mathbb{I}(z \geq 0),$$

que nos permite extender la expresión (2.11) a cualquier cuantil. En particular, retomando el conjunto de observaciones de nuestra variable respuesta Y, tenemos la siguiente estimación de $Q_Y(\tau)$:

$$\hat{Q}_Y(\tau) = \arg\min_y \frac{1}{N}\sum_{i=1}^{N} p_\tau(y_i - y).$$

Para finalizar este apartado, podemos trasladar todas estas ideas al marco de la regresión. Así, la recta de regresión cuantil podría estimarse como la recta de mínima pérdida cuantílica (pérdida absoluta, en el caso de la mediana). Por lo tanto, dada una muestra de observaciones, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, tendremos un modelo de la forma

$$y_i = x_i^T \beta + \epsilon_i,$$

donde los errores verifican que $\mathbb{P}(\epsilon_i \leq 0|X) = \tau$ (el cuantil de orden τ del error es cero). Entonces, podemos estimar el parámetro β como

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

siendo p_τ la función de pérdida cuantílica. En lo que respecta a la notación seguida durante este apartado, recordemos que puede consultarse la Sección 2.2 de Hastie et al. (2009). Así, como norma general, los vectores no se escribirán en negrita, excepto cuando tengan N componentes. Así, podremos diferenciar un vector de atributos de longitud p, x_i, de la i-ésima observación del vector $\mathbf{x}_j$ de longitud N, que contiene todas las observaciones de la variable explicativa X_j.

2.4.2. Regresión cuantil aditiva

Durante el Capítulo 3 presentaremos diversos resultados prácticos y, entre ellos, nos encontraremos con el uso de los métodos de boosting y bosques aleatorios aplicados a problemas de regresión cuantil, así como el uso de modelos de regresión cuantil aditivos generalizados (*quantile generalized additive models*, QGAM). Nos centraremos en estos últimos. En la Sección 2.1 ya introducimos los modelos aditivos generalizados. En ellos, se sobreentiende que la regresión realizada es la habitual, esto es, regresión en media. A continuación, sin entrar demasiado detalle, extenderemos los conceptos presentados en el apartado anterior y los adaptaremos a estos modelos generales. Para obtener información detallada sobre estos modelos, pueden consultarse los artículos de Fasiolo et al. (2021b) y Fasiolo et al. (2021a).

Así, supongamos que nos encontramos con un modelo aditivo generalizado que presenta una expresión como en (2.1). Por lo tanto, disponemos de p funciones no lineales y suaves no definidas previamente y que recogen los efectos no paramétricos de las variables explicativas. En concreto, para $j = 1, \ldots, p$, podemos representar estos términos suaves como

$$f_j(x) = \sum_{m=1}^{M} \beta_{jm} b_{jm}(x_j),$$

donde los β_{jm} son coeficientes desconocidos y los $b_{jm}(x_j)$ se corresponden a una base de funciones de splines conocida. La dimensión de la base de splines, M, se selecciona de manera que sea suficientemente generosa como para evitar un sobresuavizado. Sin embargo, es la penalización β_j, la encargada de controlar la verdadera complejidad de las funciones f_j. Más concretamente, dada una muestra de observaciones, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, supongamos que tenemos un modelo de la forma

$$y_i = x_i^T \beta + \epsilon_i,$$

donde x_i es la fila i-ésima de la matriz de diseño $\mathbf{X}$, de dimensión $N \times p$, que contiene la base de funciones de splines evaluada en x_i. En este contexto, tendremos que minimizar la siguiente función de pérdida cuantílica penalizada:

$$\frac{1}{\sigma} \sum_{i=1}^{N} p_\tau(y_i - x_i^T \beta) + \frac{1}{2} \sum_{j=1}^{M} \gamma_j \beta^T \mathbf{S}_j \beta, \tag{2.12}$$

donde $\gamma = \{\gamma_1, \ldots, \gamma_M\}$ es un vector de parámetros de suavizado positivos y $\frac{1}{\sigma} > 0$ es la denominada "tasa de aprendizaje", que determina el balance entre la pérdida y la penalización añadida.

Por su parte, las $\mathbf{S}_j$ son matrices semidefinidas positivas cuyo objetivo es penalizar la curvatura del correspondiente efecto no paramétrico de las funciones f_j. En concreto, seleccionaremos las matrices $\mathbf{S}_j$ convenientemente de acuerdo con el tipo de penalización empleada para los términos de suavizado. Cabe destacar que la selección del vector γ y el valor de σ serán uno de los retos principales en este método de regresión cuantil aditiva.

Capítulo 3

Resultados prácticos

Como ya hemos comentado en el primer capítulo, el objetivo último de esta memoria será seleccionar el criterio de ramificación óptimo a utilizar por el algoritmo RLT, en el que se basa el solver `RAPOSa`. Esto es, dado un problema de programación polinómica, debemos ser capaces de elegir el mejor criterio de ramificación de entre todos los considerados, basándonos en la nueva medida de rendimiento o KPI, "ritmo", definida en la Sección 1.3.2. Para ello, en el capítulo 2 hemos introducido una serie de técnicas de aprendizaje supervisado, las cuales nos permitirán llevar a cabo esta tarea.

Claramente, nuestro interés radica en demostrar que este proceso de aprendizaje va a proporcionar mejores resultados que la selección manual de alguno de los criterios de ramificación comentados a lo largo del trabajo. Es decir, queremos beneficiarnos de este aprendizaje, consiguiendo un mejor desempeño en el algoritmo de `RAPOSa`. Por esta razón, en este tercer capítulo presentaremos una serie de resultados prácticos, donde compararemos el rendimiento obtenido haciendo uso de las técnicas de aprendizaje incluidas en el capítulo anterior, medido en términos de KPI, frente a los resultados obtenidos fijando cada uno de los criterios. En particular, también compararemos estas técnicas entre sí, determinando cuál de ellas proporciona un mayor beneficio en términos de aprendizaje.

Por último, y previamente a todo lo anterior, deberemos introducir la batería de problemas de programación polinómica utilizada en el estudio computacional, así como las variables predictoras obtenidas a partir de la misma. Comenzaremos presentando nuestra fuente de datos en el siguiente apartado y, posteriormente, nos centraremos en los atributos, describiéndolos brevemente y analizando cómo se relacionan entre sí.

3.1. Base de datos: muestra de entrenamiento y test

En lo que respecta a la base de datos empleada para la obtención de los resultados prácticos, trabajaremos con tres conjuntos de problemas distintos. El primero de ellos ha sido obtenido de Dalkiran and Sherali (2016) y contiene 180 problemas de programación polinómica generados aleatoriamente y que se diferencian en el grado, número de variables y densidad. Por su parte, el segundo conjunto de datos procede de la referencia clásica `MINLPLib` (Bussieck et al. (2003)), que se trata de una batería de problemas de programación no lineal en enteros mixta. En concreto, vamos a seleccionar solo aquellos problemas de programación polinómica con variables acotadas y variables continuas, resultando en un total de 168 observaciones. El tercer conjunto de datos procede de otra referencia conocida, `QPLIB` (Furini et al. (2018)), una librería que contiene una colección de diversos problemas de programación cuadrática. En particular, hemos realizado una selección similar a la anterior, resultando en un total de 63 problemas. A lo largo del trabajo, nos referiremos a estos tres conjuntos como DS-TS, MINLPLib-TS y QPLIB-TS, respectivamente. Asimismo, trabajaremos con un total de 354 problemas, en lugar

de los 411 presentes en nuestra base de datos, pues hemos eliminado aquellos problemas que no tienen restricciones[1] y los que se resolvían en una iteración (ya que no se llega a utilizar el criterio de ramificación).

Si bien es cierto que ya disponemos de una batería de problemas para llevar a cabo nuestra tarea de selección del criterio de ramificación óptimo, debemos recordar la metodología seguida en los problemas de aprendizaje supervisado y, en general, el procedimiento habitual en aprendizaje automático. Aceptando que nuestro conjunto de observaciones es lo suficientemente grande, procederemos con su partición en dos conjuntos disjuntos, conocidos como muestra de entrenamiento y de test. En particular, seleccionaremos el 70 % de los datos como muestra de entrenamiento y el 30 % restante como muestra de test.

El conjunto de problemas para el entrenamiento se elige de forma aleatoria manteniendo la proporción de los mismos en las diferentes "familias"[2] disponibles en nuestra base de datos. Fijando un número mínimo de 10 observaciones por "familia", obtendríamos la clasificación que se recoge en la Figura 3.1.

Battery | prefix	d2	d3	d4	d5	d6	ex	ka	po	st	wa	qplib	other
DS-TS	30	28	30	29	30							29
MINLPLib-TS						26	32	23	16	26		8
QPLIB-TS											55	

Figura 3.1: Fuente de datos con los problemas de programación polinómica agrupados en función de su "familia" y la batería a la que pertenecen.

Atendiendo a los resultados recogidos en la tabla, observamos que tenemos un total de 12 "familias" diferentes, siendo las cinco primeras columnas correspondientes a problemas del conjunto DS-TS, las cinco siguientes a MINLPLib-TS y la onceava correspondiente a la batería QPLIB-TS. Por último, tenemos una columna denotada por `other`, donde se recogen problemas de los conjuntos DS-TS y MINLPLib-TS, que no pertenecen a ninguna de las "familias" anteriores, pero que tampoco alcanzan un mínimo de 10 observaciones como para conformar una propia.

Cabe destacar que la razón de agrupar los problemas siguiendo este criterio se debe a la similitud en las propiedades de los problemas pertenecientes a una misma "familia". De esta forma, al seleccionar un 70 % de los datos para obtener la muestra de entrenamiento, mantenemos la proporción de las observaciones en cada una de las clases. Con esto, conseguiremos beneficiar al proceso de aprendizaje, evitando que el conjunto de entrenamiento esté formado por problemas muy parecidos entre sí, lo cual derivaría en un modelo sobreajustado, cuya capacidad predictiva decaería considerablemente al considerar una batería de problemas con características distintas a los anteriores. Además, para una valoración más precisa de los resultados obtenidos, realizaremos 10 particiones de la batería de problemas según el procedimiento expuesto y, representaremos los resultados de las diferentes metodologías, sobre los conjuntos de test correspondientes a cada una de dichas 10 particiones.

[1] La definición de algunas variables explicativas va a depender de las restricciones del problema, por ejemplo, en algunos casos dividiremos por el número de restricciones.

[2] Entendemos por "familia" los dos primeros dígitos o letras que figuran en el nombre de cada problema en su batería correspondiente. En concreto, tendremos que para los problemas de la batería DS-TS, estos dígitos indican el grado del problema (por ejemplo, `d2` sería grado 2), mientras que para la batería MINLPLib-TS tenemos dos letras que diferencian las clases de problemas presentes en este conjunto de datos.

3.1.1. Atributos

En este apartado introduciremos nuestro conjunto de variables predictoras, utilizadas en el proceso de aprendizaje. En el Cuadro 3.1 hemos recogido los nombres empleados en `R` para referirnos a cada uno de nuestros atributos, así como una breve descripción de los mismos.

Variable	**Breve descripción**
`degree`	Grado del problema de programación polinómica
`n.constr`	Número de restricciones del problema de programación polinómica
`n.equatily`	Número de restricciones de igualdad dividido por el número total de restricciones
`n.linear`	Número de restricciones lineales dividido por el número total de restricciones
`n.quadratic`	Número de restricciones cuadráticas dividido por el número total de restricciones
`n.var`	Número de variables del problema de programación polinómica
`n.varcons`	Número de variables del problema dividido por el número de restricciones más uno (correspondiente a la función objetivo)
`n.vardegree`	Número de variables del problema dividido por el grado del mismo
`range.average`	Media de los rangos (diferencia entre la cota superior y la inferior) de las variables
`range.var`	Varianza de los rangos de las variables
`range.q50`	Mediana de los rangos de las variables
`n.monom`	Número de monomios presentes en el problema de programación polinómica
`monom.cons`	Número de monomios presentes en el problema dividido por el número de restricciones más uno (pues se tiene en cuenta la función objetivo)
`monom.div`	Número de monomios lineales en el problema dividido por el número total de monomios
`quad.monom`	Número de monomios cuadráticos en el problema dividido por el total de monomios
`diff.linear`	Número de monomios lineales distintos en el problema dividido por el total de monomios distintos
`diff.quadratic`	Número de monomios cuadrados distintos en el problema dividido por el total de monomios distintos existentes
`diff.monom`	Numero de monomios distintos en el problema dividido por el total total de restricciones más uno (asociado a la función objetivo)
`average.perc`	Media del porcentaje de monomios en cada restricción y en la función objetivo, esto es, para cada restricción (y la función objetivo) calculamos el porcentaje de monomios existentes en función del número total de monomios y, a continuación, calculamos la media de dichos valores

Continúa en la página siguiente.

Variable	Breve descripción
`coeff.average`	Media de los coeficientes presentes en el problema de programación polinómica
`coeff.var`	Varianza de todos los coeficientes del problema de programación polinómica
`density`	Densidad del problema de programación polinómica, esto es, el número de monomios distintos presentes en el problema dividido por el número total de monomios que podrían aparecer en el problema teniendo en cuenta su grado y número de variables
`density.var`	Varianza de la densidad de las variables. La densidad de cada variable es el número de monomios distintos en los que dicha variable aparece dividido por el número total de monomios diferentes que aparecen en el problema
`cons.aver`	Media del número de veces que aparece cada variable en las restricciones y en la función objetivo, esto es, para cada variable calculamos el número de restricciones en las que está presente (y sumamos uno a ese valor si aparece en la función objetivo) y hacemos la media
`cons.var`	Varianza del número de veces que aparece cada variable en las restricciones y en la función objetivo, esto es, para cada variable calculamos el número de restricciones en las que está presente (y sumamos uno a ese valor si aparece en la función objetivo) y calculamos la varianza
`perc.one`	Porcentaje de variables que no están presentes en ningún monomio de grado mayor que uno
`perc.two`	Porcentaje de variables que no están presentes en ningún monomio de grado mayor que dos
`i.density`	Densidad de nuestro primer grafo, denotado por intersección variable-variable
`a.density`	Densidad del segundo grafo, denominado intersección monomio-restricción
`i.transit`	Transitividad del grafo intersección variable-variable
`a.transit`	Transitividad del grafo intersección monomio-restricción
`i.mod`	Modularidad del grafo intersección variable-variable
`a.mod`	Modularidad del grafo intersección monomio-restricción
`i.tree`	Anchura del árbol (*treewidth*) en el grafo intersección variable-variable
`a.tree`	Anchura del árbol (*treewidth*) en el grafo intersección monomio-restricción

Cuadro 3.1: Descripción de las variables predictoras empleadas en nuestro problema de aprendizaje.

Así, dispondremos de un total de 36 variables explicativas, asociadas a las diferentes características de los problemas de programación polinómica presentes en nuestra base de datos. Nos encontraremos con predictores relacionados con el número de restricciones, variables o monomios de los problemas, así como otros que dependerán exclusivamente de las propiedades de los grafos. En concreto, tendremos que este es el caso de las últimas 8 variables recogidas en el Cuadro anterior. Puesto que disponemos de dos grafos, vamos a definir conjuntamente estas propiedades:

- Densidad. En teoría de grafos, se define como la proporción de aristas que posee un grafo. De esta forma, un grafo denso sería aquel que posee un número de aristas cercano a las que tendría si estuviese completo. Por la contra, un grafo disperso es aquel que posee un número de aristas muy bajo, cercanas a las que tendría si fuera un grafo vacío.

- Transitividad. Se basa el número relativo de triángulos en el grafo, comparado con el número total de ternas de nodos. En concreto, tendremos que

$$T = \frac{3 \times \text{número de triángulos en el grafo}}{\text{número de ternas en el grafo}}.$$

  Observamos que el numerador está multiplicado por 3, pues cada triángulo contribuye a la formación de tres ternas distintas, centradas en cada uno de los vértices del triángulo. Con esta definición, tenemos que $T \in [0, 1]$, obteniendo un grafo con todas sus aristas cuando $T = 1$.

- Modularidad. Se trata de una medida estructural de los grafos que indica la capacidad que posee una red para dividirse en módulos o *clusters*. En concreto, se corresponderá con la fracción de las aristas que caen dentro de los grupos dados, menos la fracción esperada si las aristas se distribuyesen al azar.

- Anchura del árbol. En un grafo no dirigido, será un número entero que especifica, de manera informal, lo lejos que está el grafo de ser un árbol. Esto es, el conjunto de vértices de mayor tamaño que puede obtenerse en una descomposición en árbol del grafo.

Esta amplia gama de atributos va a provocar que existan grandes diferencias entre los mismos, sobre todo en el rango de valores en los que se mueve cada variable. En la Figura 3.2 mostramos algunos ejemplos particulares donde pueden apreciarse estas diferencias. En particular, tendremos que todas las variables serán numéricas, pero que, claramente, no están medidas en la misma escala. Por ejemplo, nos encontramos con la variable `perc.one` cuyo intervalo muestral es $[0, 1]$, mucho más pequeño que el de la variable `n.var`, que presenta una escala de valores más alta. Además, apreciamos la existencia de valores atípicos. En concreto, las variables `range.average` y `range.q50` presentan una mediana de 2.1 en ambos casos, pero sus medias son muy elevadas, debido a que los valores máximos alcanzan el valor de 10^6. Esto tiene como consecuencia que el rango muestral (diferencia entre el valor máximo y el valor mínimo) sea significativamente diferente de unas variables a otras.

Atributo | Características	Clase	Mínimo	Mediana	Media	Máximo
n.var	numérica	1	16	67.64	1039
perc.one	numérica	0	0.25	0.4359	1
range.average	numérica	0.9	2.1	65769	1e+06
range.q50	numérica	0.7	2.1	67018.6	1e+06
monom.cons	numérica	1.667	13.208	119.017	10150
i.mod	numérica	0	0.03061	0.29007	0.96459

Figura 3.2: Información sobre el tipo de variable (numérica o categórica), valor mínimo y máximo, mediana y media, de seis atributos pertenecientes a nuestra base de datos.

Esta diferencia en la escala de las variables va a suponer un problema y, en concreto, afectará a la variabilidad de las mismas. En nuestro caso particular, comenzaremos realizando una transformación de los datos, con el objetivo de reducir los intervalos muestrales y, de esta forma, conseguir que nuestros

predictores tengan una escala parecida. En concreto, utilizaremos la familia de transformaciones Box-Cox, definidas como una función continua que varía con respecto a la potencia λ. Dada la variable explicativa X_j, tendremos la siguiente transformación:

$$\begin{cases} \dfrac{X_j^\lambda - 1}{\lambda}, & \text{si } \lambda \neq 0 \\ \log(X_j), & \text{si } \lambda = 0 \end{cases}$$

Estas transformaciones deben aplicarse a variables que toman valores positivos. Sin embargo, en nuestro conjunto de atributos nos encontramos con que muchos de los valores observados son cero, razón por la cual la transformación se aplicará a la variable Z, tal que $Z = X_j + c$, siendo c una constante que garantice que $Z > 0$. Cabe destacar que, dentro de esta familia de transformaciones, nos encontramos con la transformación mediante el logaritmo neperiano para la estabilización de la varianza, la cual será utilizada en varias ocasiones.

Atributo | Características	Clase	Mínimo	Mediana	Media	Máximo
n.var	numérica	0	2.773	3.018	6.946
perc.one	numérica	0	0.25	0.4359	1
range.average	numérica	-0.1335	0.7497	2.9788	13.8155
range.q50	numérica	-0.3488	0.7419	2.8928	13.8155
monom.cons	numérica	0.5108	2.5808	2.9663	9.2252
i.mod	numérica	0	0.03061	0.29007	0.96459

Figura 3.3: Tabla con las seis variables predictoras anteriores tras realizar las transformaciones de Box-Cox para estabilizar la varianza.

En la Figura 3.3 podemos observar cómo se modifica la escala de las variables recogidas en la figura anterior. En concreto, tenemos que el rango de valores en los que se mueven las diferentes variables es mucho más parecido y parece que han desaparecido los datos atípicos. Cabe destacar que únicamente estamos recogiendo la información de seis de nuestras 36 variables explicativas, sin embargo, se ha corroborado esta afirmación realizando un estudio detallado del conjunto de las mismas.

Una vez llevada a cabo la transformación de nuestros datos vamos a estudiar la correlación existente entre nuestras variables predictoras. En la Figura 3.4 se recoge la correlación entre los diferentes atributos dos a dos. En concreto, solo se muestran las 25 correlaciones más importantes, utilizando el color azul para representar a las correlaciones positivas, y el color rojo para indicar la existencia de correlación negativa. Así pues, tenemos que las variables `range.average` y `range.q50` presentan una fuerte correlación positiva, con valor 1, seguidas de `coef.average` y `coeff.var`. Si nos fijamos en la descripción de estas variables, realizada en el Cuadro 3.1, comprobamos que es normal obtener correlaciones elevadas, debido a la naturaleza de las mismas. Por ejemplo, tenemos que las primeras representan la media y la mediana de los rangos de las variables, respectivamente, medidas que están estrechamente relacionadas entre sí. Asimismo, comprobamos que las variables `cons.average` y `a.mod` presentan una correlación muy negativa, en torno al -0.9, que se corresponde con la primera barra en color rojo.

Figura 3.4: Correlaciones entre las variables explicativas dos a dos. El gráfico únicamente recoge las 25 primeras correlaciones, ordenadas de mayor a menor importancia.

Llegamos a este punto, está claro que nuestros atributos presentan correlaciones elevadas. Por lo tanto, llevaremos a cabo un Análisis de Componentes Principales (ACP), sobre nuestro conjunto de variables predictoras, con el objetivo de trasformar este conjunto original de variables correlacionadas en otro conjunto de nuevas variables incorreladas entre sí (sin información repetida o redundante), llamado conjunto de componentes principales. Estas nuevas variables serán combinación lineal de las anteriores y se van construyendo según el orden de importancia en cuanto a la variabilidad total que recogen de la muestra. Así, en nuestro caso, vamos a buscar $d < 36$ variables que recojan la mayor parte de la información o variabilidad de los datos, consiguiendo a la vez una reducción de la dimensión. Claramente, si las variables originales no estuviesen interrelacionadas de partida, no tendría sentido realizar un análisis de componentes principales.

Así, comencemos definiendo la primera componente principal, Z_1, de un vector aleatorio p-dimensional $X = (X_1, \ldots, X_p)'$ con vector de medias $\mu = \mathbb{E}(X)$ y matriz de covarianzas $\mathbf{\Sigma} = \mathbb{E}((X - \mu)(X - \mu)')$:

$$Z_1 = v_1'X = v_{11}X_1 + \ldots + v_{p1}X_d \text{ con } v_1 = (v_{11}, \ldots, v_{p1})' \in \mathbb{R}^p,$$

$$\text{Var}(Z_1) = \text{máx}\{\text{Var}(v'X) : v \in \mathbb{R}^p, v'v = 1\}.$$

Por lo tanto, se trata de una variable aleatoria obtenida como la combinación lineal normalizada con mayor varianza de entre todas las existentes entre las variables de X. En particular, tenemos que $\text{Var}(Z_1) = \lambda_1$, donde λ_1 es el mayor autovalor de la matriz de covarianzas $\mathbf{\Sigma}$ y v_1 un autovector asociado a λ_1 de norma uno. Por otra parte, definimos la segunda componente principal de x como la variable aleatoria, Z_2,

$$Z_2 = v_2'X = v_{12}X_1 + \ldots + v_{d2}X_p \text{ con } v_2 = (v_{12}, \ldots, v_{p2})' \in \mathbb{R}^p,$$

$$\text{Var}(Z_2) = \text{máx}\{\text{Var}(v'X) : v \in \mathbb{R}^p, v'v = 1, v'v_1 = 0\}$$

obtenida como la combinación lineal de mayor varianza de las variables de X formada por vectores unitarios ortogonales a v_1. En particular, $\text{Var}(Z_2) = \lambda_2$, donde λ_2 es el mayor autovalor de la matriz de covarianzas $\mathbf{\Sigma}$ y v_2 es un autovector asociado a λ_2 de norma uno y ortogonal a v_1. Por último, cabe destacar que la condición de ortogonalidad entre los vectores v_1 y v_2 es equivalente a la incorrelación

de las componentes Z_1 y Z_2. Por lo tanto, la segunda componente principal podría definirse como la combinación lineal de mayor varianza de las variables de X entre aquellas combinaciones lineales normalizadas e incorrelacionadas con la primera componente principal.

Continuando con este proceso, podemos definir las d componentes principales de X como las variables aleatorias $(Z_1, \ldots, Z_d)$ tales que

$$Z_1 = v_1'X, \ldots, Z_d = v_p'X, \quad v_1, \ldots, v_p \in \mathbb{R}^p.$$

con

$$\begin{aligned}
&\mathrm{Var}(Z_1) = \text{máx}\{\mathrm{Var}(v'X) : v \in \mathbb{R}^p, v'v = 1\},\\
&\mathrm{Var}(Z_2) = \text{máx}\{\mathrm{Var}(v'X) : v \in \mathbb{R}^p, v'v = 1, v'v_1 = 0\},\\
&\vdots\\
&\mathrm{Var}(Z_p) = \text{máx}\{\mathrm{Var}(v'X) : v \in \mathbb{R}^p, v'v = 1, v'v_1 = 0, \ldots, v_{p-1}'v = 0\}.
\end{aligned}$$

Análogamente, para $j \in \{1, \ldots, d\}$ tenemos que $\mathrm{Var}(Z_j) = \lambda_j$, siendo $\lambda_1 \geq \ldots \geq \lambda_p \geq 0$ los p autovalores ordenados de la matriz de covarianzas $\mathbf{\Sigma}$ y $v_1, \ldots, v_p$ sus autovectores asociados normalizados. Por lo tanto, $\{v_1, \ldots, v_p\}$ es una base ortonormal de autovectores. Además, $\mathrm{Cov}(Z_j, Z_k) = 0$ si $j \neq k$ y $\mathrm{Var}(Z_j) = \lambda_j$ para cada $j \in \{1, \ldots, p\}$. Podemos escribir:

$$Z = \mathbf{V}'X$$

con $Z = (Z_1, \ldots, Z_p)'$ y $\mathbf{V} = (v_1, \ldots, v_p)$ la matriz cuyas columnas son los autovectores de $\mathbf{\Sigma}$. Así, $\mathrm{Cov}(Z, Z) = \mathbf{V}'\mathbf{\Sigma}\mathbf{V}$ y se deduce que el proceso de extracción de las componentes principales se reduce a la diagonalización de la matriz de covarianzas del vector aleatorio X.

Por lo tanto, observamos que un vector aleatorio general, puede descomponerse en sus componentes principales, únicamente con la condición de que exista la matriz de covarianzas. Ahora bien, a pesar de haber transformado nuestros datos, seguimos teniendo diferencias en la escala en la que están medidas nuestras variables predictoras. Esto va a traducirse en un problema importante para el Análisis de Componentes Principales, ya que los resultados que obtengamos van a depender de esta escala original. Así, un cambio en la escala de una variable, manteniendo fijas las demás, va a traducirse en una modificación de las componentes principales, llegando incluso a cambiar la interpretación de las mismas. Este problema se puede solventar aplicando el Análisis de Componentes Principales a las variables estandarizadas, lo cual equivale a trabajar con la matriz de correlaciones, en lugar de la matriz de covarianzas. De esta manera, en los resultados prácticos, tendremos en cuenta esta consideración para llevar a cabo los cálculos relativos a la construcción de las componentes principales.

Por último, cabe destacar que la construcción de las componentes principales solo será necesaria para ajustar el modelo de regresión cuantil aditivo generalizado (QGAM). En el caso de los bosques aleatorios y el método boosting trabajaremos con el conjunto total de los atributos, una vez realizada la transformación Box-Cox. En particular, tendremos que tanto en la metodología de bosques aleatorios como en el método boosting estamos utilizando árboles de decisión, los cuales no son modelos lineales, por lo que la presencia de correlación entre las variables no será un problema. Además, en bosques aleatorios ya sabemos que se incorpora una aleatoriedad en la selección de los predictores en cada uno de los cortes. Esta misma aleatoriedad se introduce en el método *stochastic gradient boosting*, que será el utilizado en esta memoria, seleccionando al azar un subconjunto de la muestra de entrenamiento.

3.2. Implementación en R

En lo que respecta a las ejecuciones de `RAPOSa`, tendremos que todas ellas han sido realizadas en el superordenador Finisterrae II, que es la denominación genérica de las distintas generaciones

de superordenadores del Centro de Supercomputación de Galicia (CESGA), integradas en la instalación Científico Técnica Singular (ICTS), Red Española de Supercomputación (RES). En particular, se ha ejecutado cada instancia con seis configuraciones diferentes de `RAPOSa`, basadas en los seis criterios de ramificación definidos en la Sección 1.3.1. El tiempo límite en cada ejecución fue fijado a 10 minutos.

Por otro lado, la parte correspondiente al aprendizaje, en la que reside nuestro interés, ha sido desarrollada en el lenguaje de programación `R` (ver `R Core Team` (2021)). En concreto, se han implementado tres técnicas de aprendizaje supervisado, ya introducidas en el capítulo teórico: modelos de regresión cuantil aditivos generalizados (QGAM) y regresión cuantil combinada con los métodos de bosques aleatorios y boosting. En los siguientes apartados presentaremos estos métodos desde un enfoque más práctico, incluyendo las funciones de `R` utilizadas, así como los principales parámetros asociados a los mismos.

El objetivo principal será comparar el desempeño de estos métodos de aprendizaje con los resultados obtenidos con las seis configuraciones de `RAPOSa`, asociadas a cada uno de los criterios de ramificación. Representaremos diagramas de cajas basados en nuestra medida de rendimiento o KPI, "ritmo", así como de una serie de gráficos obtenidos con el lenguaje de programación `Python`, que nos permitirán analizar el rendimiento de software (*performance profiles*, ver Dolan and Moré (2002)). Asimismo, nos apoyaremos en tablas, recogiendo numéricamente los valores de nuestra medida "ritmo" y comparándolos entre las diferentes ejecuciones realizadas.

Comenzaremos introduciendo los resultados obtenidos para las seis configuraciones de `RAPOSa`. Recordemos que estamos trabajando con una versión básica y robusta de este solver, pero como bien se comentó en la Sección 1.3, puede consultarse González-Rodríguez et al. (2020) si se desea modificar la implementación básica de esta técnica, teniendo en cuenta diferentes mejoras. Como ya hemos comentado, vamos a seleccionar 10 submuestras para el aprendizaje, lo cual nos permitirá realizar una valoración más precisa de los resultados obtenidos. De esta forma, en el Cuadro 3.2 hemos recogido los resultados obtenidos para cada una de las submuestras siguiendo dos criterios diferentes.

	Submuestra	1	2	3	4	5	6	7	8	9	10
Optimo (test)	Media geométrica	0.469	1.020	1.414	0.886	1.460	1.353	0.989	0.831	1.484	1.719
Óptimo	Media geométrica	0.241	0.550	1.013	0.574	0.920	0.868	0.527	0.616	1.002	1.130

Cuadro 3.2: Tabla con las medias geométricas de la medida de rendimiento "ritmo" en los problemas que forman parte de la muestra de test de cada una de las 10 particiones realizadas.

En primer lugar, tenemos la fila denominada `Óptimo (test)`, que consiste en elegir, para todas las submuestras, el criterio de ramificación que presenta en promedio un mejor desempeño, medido en términos de "ritmo". Así, resolvemos todos los problemas de cada una de las 10 submuestras con las seis configuraciones de `RAPOSa` (cada una de ellas asociada a un criterio de ramificación) y calculamos la media geométrica de nuestra medida "ritmo", a partir de los valores obtenidos para cada problema. Aquella configuración que obtenga, en promedio, un menor valor de esta medida a lo largo de las 10 submuestras, será la seleccionada y resolveremos todos los problemas seleccionando el mismo criterio de ramificación. En este caso, el ganador es el criterio dual. Por otra parte, tenemos la segunda fila, denominada `Óptimo`, que consiste en elegir, para cada problema de la submuestra, el criterio de ramificación que consiga un menor valor de "ritmo". Así, estamos en el mejor escenario posible, pues cada problema se resuelve con el criterio óptimo, en términos de nuestra KPI. Finalmente, se calcula la media geométrica de los valores de la medida "ritmo" obtenidos para cada problema de la submuestra.

En el Cuadro 3.3, se recogen los promedios obtenidos a partir de las medias geométricas de cada una de las columnas del Cuadro 3.2 anterior. En concreto, estamos promediando los resultados obtenidos para cada una de las 10 submuestras consideradas.

Promedio óptimo (test)	Promedio óptimo	Margen de mejora
1.163	0.744	0.360

Cuadro 3.3: Promedio de resultados y margen de mejora para la configuración básica de `RAPOSa`.

Asimismo, se incluye el margen de mejora existente entre elegir el mejor criterio de cada vez, y el mejor criterio en promedio, para todos los problemas de las 10 submuestras. Es decir, realizamos el siguiente cociente:

$$\frac{1.163 - 0.744}{1.163} = 0.360.$$

En los sucesivos apartados, vamos a tomar como referencia este valor para determinar la mejora obtenida con los métodos de aprendizaje supervisado. De esta forma, consideraremos las mismas 10 submuestras, así como todos los resultados obtenidos para las seis configuraciones de `RAPOSa`. Esta información será utilizada en el proceso de aprendizaje, para cada uno de los métodos comentados anteriormente: QGAM, boosting y bosques aleatorios. En particular, entrenaremos nuestros modelos con el objetivo de predecir el criterio de ramificación óptimo que debemos seleccionar para cada problema de programación polinómica perteneciente a las diferentes submuestras. De igual forma, calcularemos la media geométrica de nuestra medida "ritmo" para cada submuestra, siguiendo el mismo esquema que en el Cuadro 3.2. Estas medias geométricas se promediarán y se compararán los resultados obtenidos con respecto al margen de mejora recogido en el Cuadro 3.3.

3.2.1. QGAM

Comenzamos comentando el modelo de regresión cuantil aditivo generalizado (QGAM) y los resultados obtenidos en el proceso de aprendizaje. En primer lugar, tenemos que el método ha sido ajustado utilizando la función `qgam` de `R`. Para ello, hemos realizado previamente un Análisis de Componentes Principales, donde se han construido un total de 15 componentes, a partir del conjunto de las 36 variables predictoras originales. En concreto, hemos incluido estas componentes en el modelo a través de términos suaves, `s()`, que dependen de la base de splines utilizada y del número de funciones empleadas en la base. En cuanto a la base de splines, hemos seleccionado la base de splines cúbicos, `bs = 'cr'`, con un total de `k=7` splines conformando la misma. También se ha probado a ajustar el modelo utilizando otras opciones, sin embargo, los mejores resultados se han obtenido para esta configuración. Además, no es conveniente aumentar el número de funciones de la base de splines, pues el tiempo computacional necesario para ajustar el modelo aumenta considerablemente. Por otra parte, la función `qgam` nos permite especificar el cuantil con el que se llevará a cabo la regresión. En nuestro caso, vamos a elegir el cuantil 0.3 para obtener todos los resultados de este apartado.

En lo que respecta a nuestra variable respuesta, se tratará de una variable continua definida a partir de nuestra medida de rendimiento o KPI. A pesar de ya haber sido introducida en la Sección 1.3.2 del primer capítulo, para facilitar la lectura, recordaremos su expresión:

$$Y = \frac{\mathrm{KPI}_{\mathrm{best}}}{\mathrm{KPI}} = \frac{t^{\mathrm{best}}/(\mathrm{LB}_f^{\mathrm{best}} - \mathrm{LB}_i^{\mathrm{best}})}{t/(\mathrm{LB}_f - \mathrm{LB}_i)} = \frac{t^{\mathrm{best}}/R^*}{t/R}.$$

De esta forma, tendremos que esta variable tomará valores en el intervalo $[0, 1]$. Asimismo, sus observaciones, $y_{ij}, i = 1, \ldots, N, j = 1, \ldots 6$, constituirán una matriz de tamaño $N \times 6$, esto es, una columna por cada criterio y una fila por cada problema de nuestra muestra de entrenamiento. Puesto

que no vamos a trabajar con respuesta múltiple, se ajustará un modelo de regresión cuantil aditivo generalizado para cada una de las columnas, $y_{i1}, \ldots, y_{i6}$ con $i = 1, \ldots, N$, de la matriz anterior, es decir, ajustaremos un modelo para cada uno de los criterios de ramificación. A continuación, obtendremos las predicciones $\hat{y}_{i1}, \ldots, \hat{y}_{i6}$ de todos los problemas de la muestra de entrenamiento, para cada uno de los criterios, y seleccionaremos para cada problema el mejor valor obtenido, esto es, el más próximo a 1. Por lo tanto, dado un problema $i \in \{1, \ldots, N\}$, calcularemos

$$\hat{y}_i = \underset{j=1,\ldots,6}{\text{máx}}\{\hat{y}_{ij}\},$$

obteniendo así la predicción utilizando este modelo. Como ya hemos comentado, esta predicción estará asociada a un determinado criterio de ramificación, que será el criterio óptimo que deberá utilizarse para la resolución del problema, según nuestro modelo. En concreto, dado el criterio $j \in \{1, \ldots, 6\}$ seleccionado, devolveremos el ratio asociado, esto es, el valor y_{ij} de nuestra matriz de partida. De esta forma, podremos determinar cómo de bien ha realizado las predicciones el modelo, siendo el caso $y_{ij} = 1$ el mejor escenario, ya que habríamos seleccionado el mejor criterio de ramificación.

Hasta ahora únicamente nos hemos centrado en la muestra de entrenamiento, sin embargo, nuestro interés radica en las predicciones realizadas sobre la muestra de test. Claramente, el modelo realizará buenas predicciones al utilizar los datos de entrenamiento, pues se ha ajustado utilizando la información correspondiente a los mismos. Sin embargo, queremos conseguir que se obtengan buenas predicciones cuando reciba nueva información, esto es, un problema no perteneciente al conjunto de entrenamiento. Denotemos por y_{lj}, con $l = 1, \ldots, L$ y $j = 1, \ldots, 6$, las observaciones de nuestra variable respuesta para los L problemas pertenecientes al conjunto de test, y para cada uno de los seis criterios de ramificación. En concreto, esta matriz de observaciones será el punto de referencia para comprobar la precisión de las predicciones obtenidas con nuestro modelo.

De esta forma, partiendo de las observaciones de los atributos correspondientes a los problemas de la muestra de test y dada la base de 15 componentes principales construida anteriormente, obtendremos los valores asociados en esa base para el conjunto de test y realizamos las predicciones utilizando estos resultados. En particular, obtendremos $\hat{y}_{lj}$ y, para cada problema l del conjunto de test, calcularemos

$$\hat{y}_l = \underset{j=1,\ldots,6}{\text{máx}}\{\hat{y}_{lj}\},$$

del mismo modo que con la muestra de entrenamiento. Así, obtenemos el criterio óptimo predicho mediante nuestro modelo aditivo generalizado. Finalmente, seleccionamos el valor y_{lj} correspondiente de la matriz de observaciones de nuestra variable respuesta, que será el ratio obtenido tras nuestro proceso de aprendizaje. De igual forma, podríamos recuperar el valor de la medida "ritmo" a partir de esta variable respuesta. En el Cuadro 3.4, hemos incluido las medias geométricas de nuestra medida "ritmo", tras realizar todo este proceso de aprendizaje, para cada una de las submuestras. Para ello, hemos construido a partir de los datos de cada submuestra los conjuntos de entrenamiento y de test, y hemos realizado el ajuste del modelo y obtenido las predicciones como se ha explicado a lo largo de la sección. En concreto, hemos recuperado los valores de "ritmo" a partir de los ratios, $y_{lj}, l = 1, \ldots, L, j = 1, \ldots, 6$, seleccionados en el proceso de aprendizaje. Estos valores se obtienen para cada uno de los problemas de la muestra de test y se calcula su media geométrica dentro de cada una de las submuestras.

Submuestra	1	2	3	4	5	6	7	8	9	10	Promedio
Media geométrica	0.281	0.649	1.146	0.651	1.058	1.04	0.574	0.742	1.146	1.354	0.864

Cuadro 3.4: Tabla de resultados con la media geométrica de nuestra medida "ritmo", para todos los problemas pertenecientes a cada una de las 10 submuestras, junto con el promedio de dichos valores.

Por otra parte, calculamos el promedio de las medias geométricas obtenidas, el cual se va a utilizar para comparar los nuevos resultados con el margen de mejora obtenido en el Cuadro 3.3, esto es, el margen que existía entre elegir siempre el mejor criterio y elegir, para las 10 submuestras, el criterio que presentaba un mejor desempeño en promedio. Nuestro objetivo será que el método de aprendizaje seleccione siempre el mejor criterio para la resolución de cada problema, por lo tanto, calcularemos el margen de mejora sustituyendo el criterio óptimo anterior, por el criterio de decisión que proporciona nuestro modelo QGAM. En el Cuadro 3.5 se recoge este cálculo, así como el porcentaje de mejora.

Margen de mejora	Nuestra mejora	Porcentaje de mejora
0.3599	0.2566	0.7129

Cuadro 3.5: Margen de mejora para la configuración básica de `RAPOSa`, junto con la mejora obtenida utilizando el modelo de regresión cuantil aditivo generalizado.

Por lo tanto, comprobamos que la mejora no es tan grande como la que se obtiene al elegir siempre el criterio óptimo. Esto es natural, ya que si fuese así, estaríamos en la situación en que nuestro modelo nunca se equivocaría en la predicción, pues estaríamos eligiendo siempre el ratio correspondiente al mejor criterio. Es decir, para cada problema l, con $l = 1, \ldots, L$, se tendría que el criterio j seleccionado (donde $\hat{y}_{lj}$ es máximo para $j = 1, \ldots, 6$) cumpliría que $y_{lj} = 1$. A pesar de ello, hemos conseguido una mejora del 71.29 % con respecto a seleccionar siempre el criterio dual, que se correspondía con el criterio óptimo, en promedio, dentro de cada submuestra.

A continuación, hemos incluido dos gráficos adicionales que recogen información acerca del desempeño del método. En primer lugar, en la Figura 3.5, hemos representado un diagrama de cajas, correspondiente a los ratios, y_{lj}, seleccionados para los problemas pertenecientes a la submuestra 2.

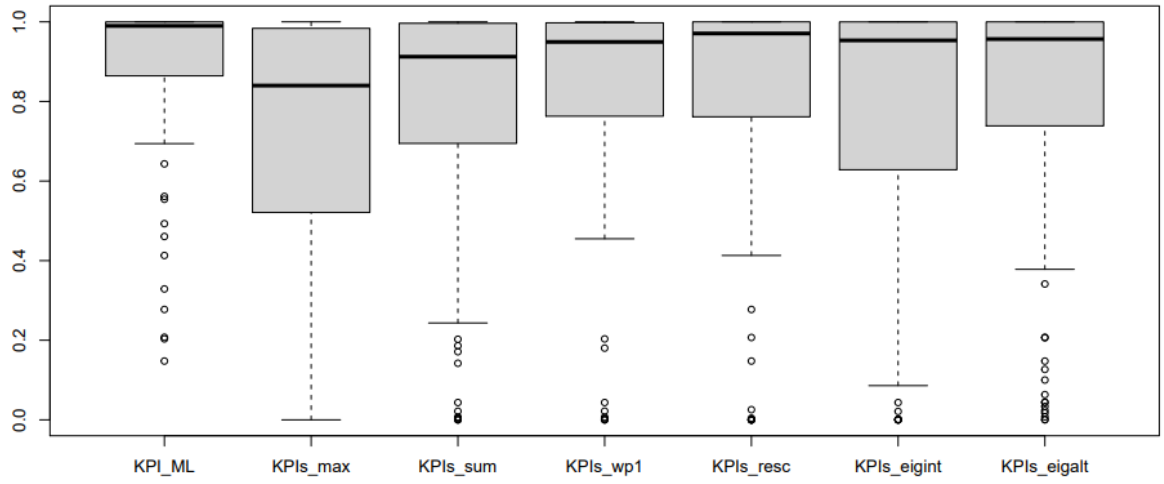


Figura 3.5: Diagrama de cajas con valores de nuestra variable respuesta para cada uno de los criterios, incluyendo el valor obtenido a partir del método QGAM.

La razón por la que se ha escogido esta submuestra, es que nos permite visualizar cómo nuestro método de aprendizaje presenta un mejor desempeño, para este conjunto de problemas, que la elección manual de uno de los seis criterios. Claramente, los resultados van variando a lo largo de las submuestras, pues hemos construido los diferentes conjuntos de entrenamiento y test de forma aleatoria. Así,

habrá gráficos de cajas donde nuestro criterio no ofrecerá los mejores resultados. Sin embargo, como comprobamos con los resultados de la tabla anterior, en promedio, ofrece una mejora significativa. Por último, podemos observar cómo existen problemas para los que la predicción del criterio de ramificación óptimo ha sido mala, esto es, no se ha seleccionado el mejor criterio y, además, nos hemos equivocado por mucho. Una ventaja de utilizar una variable continua como respuesta es que nos permite saber "por cuánto" nos estamos equivocando, ya que no es lo mismo seleccionar el segundo mejor criterio, que elegir el peor de todos.

El segundo gráfico se recoge en la Figura 3.6, y se trata de un *performance profile* obtenido en el lenguaje de programación `Python`. Expliquemos en detalle cómo se interpreta. En primer lugar, tenemos que en el eje horizontal aparece representada nuestra variable respuesta, y_{lj}, pero invertida. De esta forma, conseguimos que, en lugar de moverse en el intervalo $[0, 1]$, nuestro cociente tome valores mayores o iguales a 1, siendo el mejor valor este último. Por otra parte, en el eje vertical tenemos, para cada criterio, el porcentaje de problemas que presentan un ratio menor o igual que el ratio correspondiente en el eje horizontal. De esta forma, obtenemos un gráfico del rendimiento de los criterios siempre creciente. En particular, estamos representando los resultados del aprendizaje obtenidos para la primera submuestra. Claramente, observamos cómo estos resultados son mejores que para el resto de criterios. En concreto, para nuestro modelo QGAM, obtenemos que casi todos los problemas presentan un ratio inferior o igual a 2, mientras que para el resto de criterios, seguimos teniendo que, para un ratio igual a 8, aproximadamente un 10 % de los problemas presentan un valor superior de esta medida.

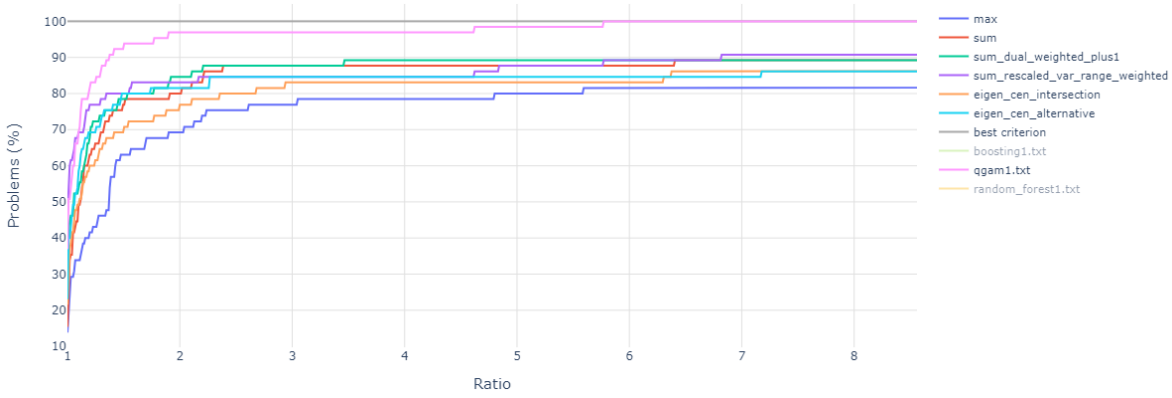
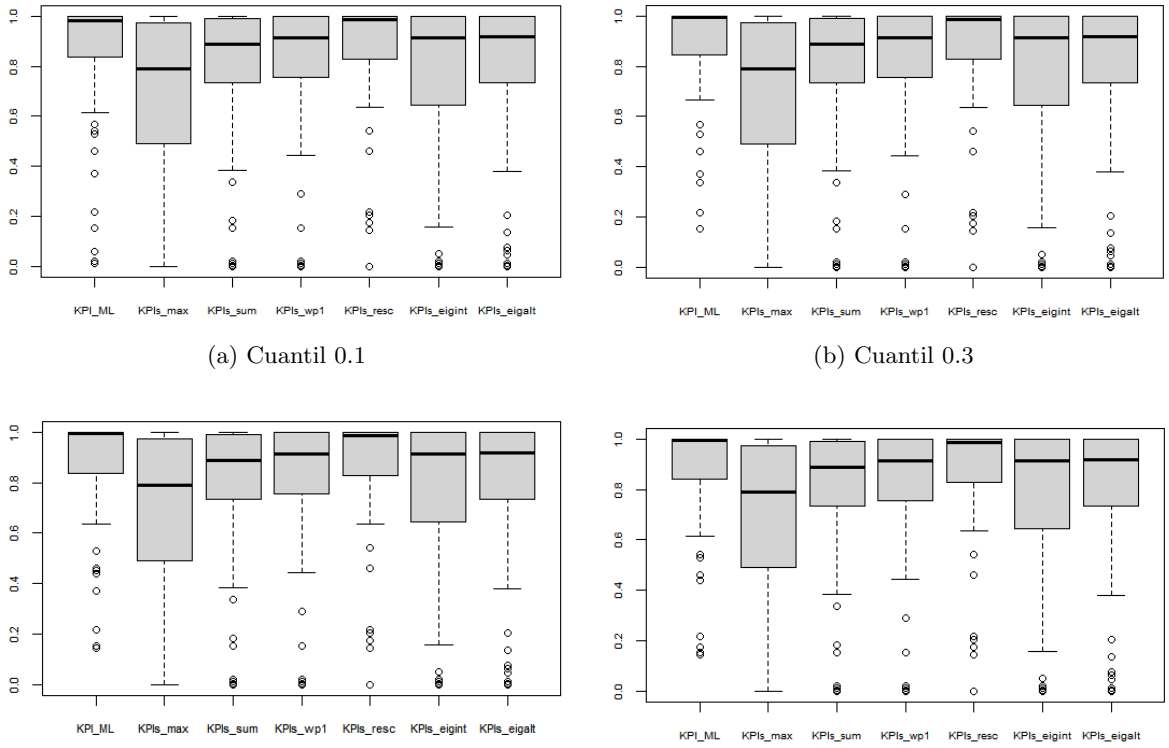


Figura 3.6: Análisis del rendimiento de software (*performance profile*) para la primera submuestra de problemas, comparando los resultados obtenidos para las seis configuraciones de `RAPOSa` y las predicciones obtenidas mediante el modelo de regresión cuantil aditivo generalizado.

Finalmente, solo queda comentar la elección del cuantil utilizado para llevar a cabo la regresión. Recordamos que, para cada uno de los criterios de ramificación establecidos, nuestras variables respuesta se calculan a partir de la medida de rendimiento o KPI y toman valores en el intervalo $[0, 1]$, de forma que cuanto más próximo a uno esté este valor para un determinado criterio, mejor será la actuación del mismo. A partir del análisis descriptivo de dichas variables, se observa un comportamiento asimétrico de las mismas y valores atípicos, lo que hace que los modelos de regresión basados en la media condicional no sean adecuados en esta situación. En este contexto, la regresión cuantil se presenta como una metodología más adecuada, al no hacer ninguna suposición sobre la distribución de la variable respuesta y ser más robusta ante la presencia de atípicos. En la Figura 3.7 se recogen cuatro diagramas de cajas obtenidos tras ajustar el modelo QGAM con cuatro cuantiles diferentes, $0.1, 0.3, 0.5, 0.8$, y para los mismos valores del resto de parámetros.



(a) Cuantil 0.1

(b) Cuantil 0.3

(c) Cuantil 0.5

(d) Cuantil 0.8

Figura 3.7: Diagramas de cajas obtenidos mediante el modelo de regresión cuantil aditivo generalizado (QGAM), considerando cuatro cuantiles diferentes.

Si bien los niveles evaluados proporcionan conclusiones similares, es con el $\tau = 0.3$ con el que se obtienen los mejores resultados y el que se ha seleccionado como valor de referencia. Además, se han realizado numerosas pruebas, comprobando los resultados obtenidos para cada una de las diez submuestras y, en general, se observa un mejor desempeño del método con esta elección.

3.2.2. Regresión cuantil con bosques aleatorios

A continuación, nos centraremos en el método de bosques aleatorios y, en particular, en la función `ranger` de `R` con la que se han obtenido los diferentes resultados. En esta metodología y, en concreto, para esta función, tendremos una amplia variedad de parámetros que podrán tunearse con el objetivo de obtener predicciones más precisas. Por esta misma razón, a diferencia de lo ocurrido en el método anterior, vamos a disponer de un mayor número de modelos ajustados en función de estos parámetros. Así, tendremos que la función `ranger` consigue implementar el método de bosques aleatorios (Breiman (2001)), siendo especialmente adecuada para bases de datos de gran dimensión. Permite resolver problemas tanto de clasificación como de regresión (Meinshausen (2006)) y, en concreto, problemas de regresión cuantil, obteniendo lo que se conoce como *quantile regression forest* y que será el método implementado en nuestro problema.

Al igual que ocurría en el modelo QGAM, seguiremos trabajando con el cuantil 0.3, pues es el que ha ofrecido, en líneas generales, unos mejores resultados en las diferentes pruebas realizadas. Por otra parte, vamos a tener tres parámetros fundamentales, los cuales ya han sido comentados en el segundo capítulo de teoría: el número de árboles considerados, B, el número de predictores seleccionados al azar en cada uno de los cortes, m, y el número mínimo de observaciones en los nodos terminales.

Estos parámetros se corresponderán, respectivamente, con `num.trees`, `mtry` y `min.node.size`, en la función de `R`. Por otra parte, recordemos que los valores recomendados para estos parámetros en los problemas de regresión eran $\lfloor p/3 \rfloor$ para m, 11 predictores en nuestro caso, y fijar un mínimo de cinco observaciones en los nodos terminales. Asimismo, el valor por defecto para el número de árboles, B, es 500. Sin embargo, ya se observa convergencia del error en las muestras OOB para 300 árboles, por lo que utilizaremos este valor.

Con todo esto, vamos a considerar tres configuraciones diferentes de este modelo, y obtener las predicciones para las 10 submuestras de test, con cada una de ellas. En concreto, denotaremos por `default` a la configuración que utiliza los valores por defecto. En lo que respecta a las otras dos configuraciones, una incluye un tuneado de los parámetros `mtry` y `min.node.size`, y otra tunea, a mayores, el parámetro `num.trees`. En concreto, seleccionaremos el número de predictores y el mínimo numero de observaciones en los nodos terminales utilizando una función `R`, que implementa este proceso de selección de los parámetros óptimos en función del error absoluto medio. Por otra parte, la elección óptima del número de árboles se ha realizado implementando una función que, a partir de los valores obtenidos para los otros dos parámetros, estudia la convergencia del error en las muestras OOB, en relación con el número de árboles que se construyen en el modelo. Por supuesto, este tuneado de parámetros se efectúa para cada modelo ajustado. Como ya describimos en el apartado anterior, se está ajustando un modelo para cada columna, $y_{i1}, \ldots, y_{i6}$ con $i = 1, \ldots, N$, de la matriz de observaciones de la variable respuesta (para los datos de entrenamiento). Es decir, ajustaremos un modelo para cada uno de los criterios de ramificación.

	Random forest (tuning)	Random forest (tuning + tree)	Random forest (default)	Random forest (selected)
Submuestra	Media geométrica	Media geométrica	Media geométrica	Media geométrica
1	0.327	0.326	0.291	0.287
2	0.624	0.626	0.619	0.623
3	1.172	1.182	1.139	1.140
4	0.660	0.682	0.649	0.695
5	1.096	1.097	1.100	1.010
6	1.063	1.065	1.065	1.056
7	0.662	0.673	0.577	0.580
8	0.725	0.717	0.702	0.704
9	1.268	1.248	1.136	1.122
10	1,337	1.498	1.298	1.274
Promedio	0.893	0.911	0.858	0.849

Cuadro 3.6: Tabla de resultados con la media geométrica de nuestra medida "ritmo", para todos los problemas pertenecientes a cada una de las 10 submuestras, junto con el promedio de dichos valores.

En el Cuadro 3.6 hemos recogido, para esta metodología, las medias geométricas de nuestra medida "ritmo", tras realizar el proceso de aprendizaje, para cada una de las submuestras. Asimismo,

en la última fila incluimos el promedio de los resultados obtenidos. Cabe destacar que hemos añadido una cuarta columna con una configuración adicional, denotada por `selected`, que utiliza valores prefijados de los parámetros. En concreto, `num.tree = 300`, `mtry = 16` y `min.node.size = 10`. Estos valores han sido seleccionados convenientemente, tras realizar numerosas pruebas y, atendiendo a los resultados recogidos en la tabla, parece ser la configuración que ofrece un mejor resultado, en lo que respecta a nuestro proceso de aprendizaje sobre los criterios de ramificación. Por su parte, los modelos que tuneaban dos y tres de los hiperparámetros no obtienen un desempeño tan bueno. Así, puede que desde el punto de vista estadístico sean mejores modelos, sin embargo, desde un enfoque más próximo al aprendizaje automático, nos interesa aquel modelo que se beneficie más del proceso de aprendizaje y sea capaz de seleccionar el mejor criterio de cade vez o, al menos, uno de los mejores.

Por lo tanto, continuaremos trabajando con la configuración denotada por `selected`. El promedio obtenido va a utilizarse, al igual que hacíamos antes, para calcular un nuevo margen de mejora y compararlo con el obtenido en el Cuadro 3.3. En concreto, obtendremos este margen sustituyendo el criterio óptimo por el criterio de decisión que proporciona nuestro modelo de regresión cuantil para bosque aleatorios. En el Cuadro 3.7 se recoge este cálculo, así como el porcentaje de mejora.

Margen de mejora	Nuestra mejora	Porcentaje de mejora
0.3599	0.2697	0.7495

Cuadro 3.7: Margen de mejora para la configuración básica de `RAPOSa`, junto con la mejora obtenida utilizando el método de regresión cuantil para bosques aleatorios (*quantile regression forest*).

Observamos que el porcentaje de mejora es del 74.95 %, mayor que el conseguido con el modelo de regresión cuantil aditivo generalizado. Asimismo, hemos recogido en la Figura 3.8, un diagrama de cajas correspondiente a los ratios, y_{lj}, seleccionados para los problemas pertenecientes a la segunda submuestra, en la que se aprecia, claramente, la mejora obtenida con el proceso de aprendizaje, en términos de nuestra medida de rendimiento o KPI.

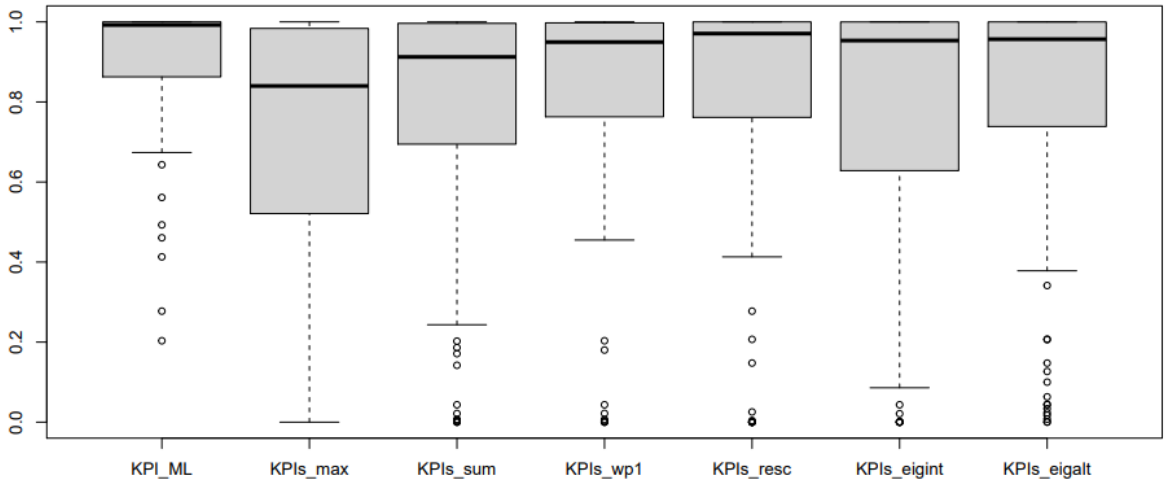


Figura 3.8: Diagrama de cajas con valores de nuestra variable respuesta para cada uno de los criterios, incluyendo el valor obtenido a partir del modelo *quantile regression forest*.

En mayor detalle, observamos que el segundo cuartil, Q_2, esto es, la mediana de los valores de los ratios, se encuentra sobre el valor 1, para nuestro método de bosques aleatorios. Por otra parte,

atendiendo al primer cuartil, Q_1, tenemos que se encuentra en torno al valor 0.7. Así, solo un 25 % de los ratios son menores que este valor, siendo estos, valores atípicos. Por lo tanto, si comparamos este resultado con los primeros cuartiles obtenidos para los otros seis criterios de ramificación, observamos cómo se obtienen valores mucho más pequeños, siendo el mejor de ellos el que se consigue con el criterio dual, `KPIs_wp1`, con un valor de aproximadamente 0.5. Por lo tanto, aunque no elijamos siempre el mejor criterio, obtenemos mejores predicciones que las conseguidas fijando manualmente uno de los criterios disponibles. En general, se obtienen las mismas conclusiones a lo largo de las 10 submuestras, aunque los resultados varían ligeramente en algunas de ellas. Sin embargo, como ya concluimos a partir de los resultados recogidos en el Cuadro 3.7, en promedio la mejora es significativa.

Finalmente, recogemos en la Figura 3.9 un *performance profile* obtenido en el lenguaje de programación `Python` y cuya interpretación ya fue explicada para el modelo QGAM. De esta forma, volvemos a representar los resultados del aprendizaje obtenidos para la submuestra 1 y observamos cómo mejoran con respecto al resto de criterios de ramificación. En concreto, para el método de bosques aleatorios, obtenemos que un 90 % de los problemas de la muestra de test presentan un ratio ligeramente mayor que 1. Asimismo, para un valor de 2, prácticamente abarcamos el total de problemas, mientras que, para el resto de criterios, seguimos teniendo que con un ratio igual a 8, aproximadamente un 10 % de los problemas siguen teniendo un valor superior de esta medida. Finalmente, cabe destacar que se comprueban resultados similares para el resto de submuestras.

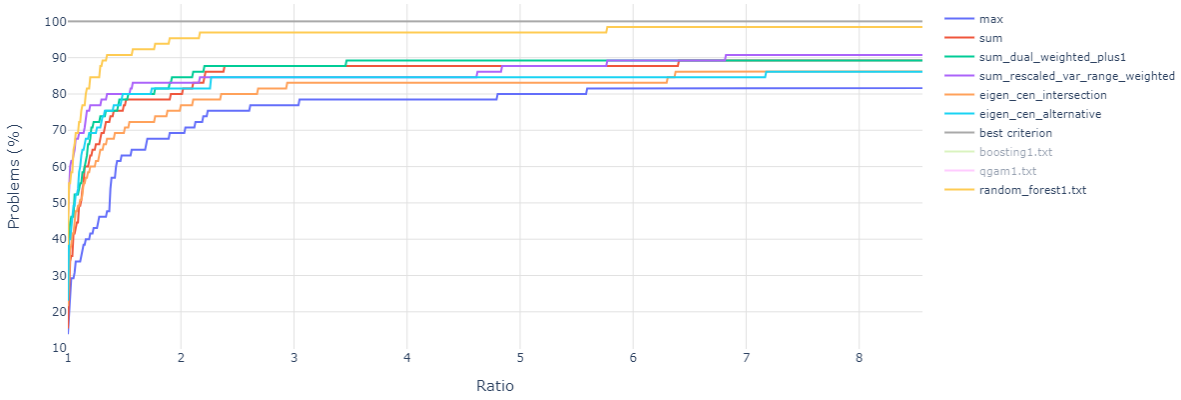


Figura 3.9: Análisis del rendimiento de software (*performance profile*) para la primera submuestra de problemas, comparando los resultados obtenidos para las seis configuraciones de `RAPOSa` y las predicciones obtenidas mediante el método de regresión cuantil para bosques aleatorios.

3.2.3. Regresión cuantil con la metodología boosting

Finalmente, comentaremos la metodología boosting y, en concreto, el algoritmo *stochastic gradient boosting*, que se implementará haciendo uso de la función de `R`, `gbm`. Esta función nos va a permitir barajar diferentes opciones en términos de los parámetros del modelo, así como realizar regresión cuantil, al igual que en los métodos anteriores. En cuanto a la selección del cuantil utilizado, volveremos a trabajar con el cuantil 0.3, por las mismas razones que se expusieron anteriormente. Por otra parte, vamos a centrarnos en el tuneado de los parámetros del modelo. Si recordamos el capítulo teórico, tenemos tres parámetros fundamentales en esta metodología: el número de iteraciones, M, el parámetro de regularización, λ, y el número de cortes de cada árbol, d. En concreto, se corresponderán con las siguientes opciones de nuestra función `gbm`:

- `n.trees`. Valor entero que representa el número de árboles que deberá ajustar el método. Por lo tanto, se corresponde con el número de iteraciones realizadas por el algoritmo *stochastic gradient boosting*.

- `shrinkage`. Es el parámetro de regularización, $\lambda \in (0, 1)$. Puede interpretarse como una medida de la proporción de aprendizaje, esto es, la velocidad a la que aprende el método. Utilizaremos el valor por defecto de 0.1.
- `interaction.depth`. Valor entero que indica la mayor profundidad que puede tener cada árbol, esto es, el número de cortes. Así, un valor 1 se traduce en un modelo aditivo, mientras que un valor 2 permite interacciones de orden dos en el modelo. En concreto, permitiremos árboles de profundidad 2.

Recordemos que, en este método, un número elevado de árboles puede conducirnos a problemas de sobreajuste. Por lo tanto, utilizaremos la función `gbm.perf` de `R` que estima el número óptimo de iteraciones que debe realizar el algoritmo. En concreto, esta función debe partir de un modelo "base", ajustado previamente. Así, tomaremos, por ejemplo, `n.tree = 250` para ajustar este primer modelo, junto con el resto de parámetros, fijados a los valores comentados anteriormente. Por último, cabe destacar que hemos seleccionado un número mínimo de 10 observaciones en los nodos terminales y 5 grupos para validación cruzada.

De esta forma, construiremos a partir de los datos de cada submuestra los conjuntos de entrenamiento y de test, y ajustaremos el modelo a los datos de entrenamiento, mediante el algoritmo *stochastic gradient boosting*. En concreto, como ya sabemos, ajustaremos un modelo para cada una de las columnas de la matriz de valores de la variable respuesta, asociadas a cada criterio de ramificación. Además, primero se ajustará el modelo "base" para, posteriormente, utilizar la función `gbm.perf` anterior, y así tunear el número de iteraciones. En el Cuadro 3.8, hemos incluido las medias geométricas de nuestra medida "ritmo", obtenida a partir de las predicciones de los ratios, $y_{lj}, l = 1, \ldots, L, j = 1, \ldots, 6$, seleccionados para cada uno de los problemas de las diez submuestras de test. Además, también obtenemos el promedio de estos valores.

Submuestra	1	2	3	4	5	6	7	8	9	10	Promedio
Media geométrica	0.278	0.649	1.162	0.683	1.044	1.089	0.586	0.706	1.156	1.281	0.863

Cuadro 3.8: Tabla de resultados con la media geométrica de nuestra medida "ritmo", para todos los problemas pertenecientes a cada una de las 10 submuestras, junto con el promedio de dichos valores.

De igual forma que para los métodos anteriores, vamos a utilizar el promedio obtenido para calcular el nuevo margen de mejora, sustituyendo el criterio óptimo por el criterio de decisión que proporciona nuestro modelo boosting. En concreto, compararemos nuestra nueva mejora, con el margen que habíamos obtenido al comienzo de la sección. En el Cuadro 3.9 se recoge este cálculo, junto con el porcentaje de mejora.

Margen de mejora	Nuestra mejora	Porcentaje de mejora
0.3599	0.2573	0.7150

Cuadro 3.9: Margen de mejora para la configuración básica de `RAPOSa`, junto con la mejora obtenida utilizando el algoritmo *stochastic gradient boosting*.

Obtenemos una mejora del 71.50 %, ligeramente mayor que la conseguida con el modelo de regresión cuantil aditivo generalizado, pero inferior a la del modelo de bosques aleatorios. Además, recordemos que en la metodología de bosques aleatorios no llevábamos a cabo ninguna búsqueda óptima de los parámetros del modelo, pues habíamos preseleccionado unos valores adecuados a priori. Sin embargo,

en el método boosting debemos utilizar una función adicional para llevar a cabo este tuneado, así como ajustar, previamente, un modelo "base" sobre el que se realiza la búsqueda.

A continuación, recogemos en la Figura 3.10, un diagrama de cajas con los valores de los ratios, y_{lj}, seleccionados para los problemas pertenecientes a la segunda submuestra. Si nos fijamos en el criterio de ramificación obtenido con el aprendizaje, tenemos que la mediana, Q_2, se encuentra sobre el valor 1. Asimismo, tenemos un primer cuartir, Q_1, por encima del valor 0.6. Por lo tanto, el 75 % de los ratios se encuentran por encima de este valor. Además, no parece haber demasiados valores atípicos.

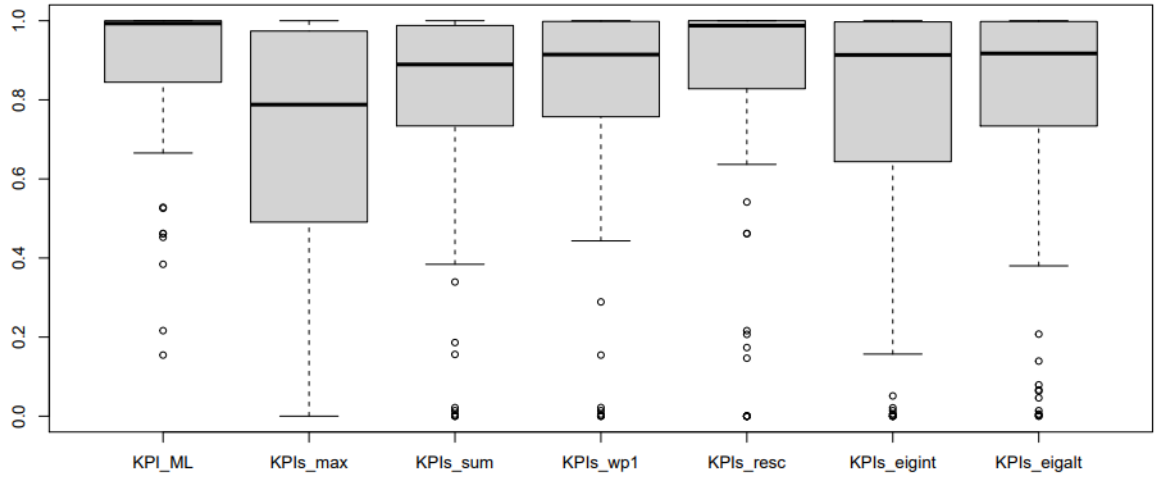


Figura 3.10: Diagrama de cajas con valores de nuestra variable respuesta para cada uno de los criterios, incluyendo el valor obtenido a partir del método boosting.

En concreto, obtenemos que para esta submuestra, el criterio del rango, `KPIs_resc`, tiene un desempeño similar al anterior, mientras que el resto de criterios ya presentan peores valores. Por último, tenemos que los resultados son similares para las demás submuestras, en términos de nuestro nuevo criterio construido.

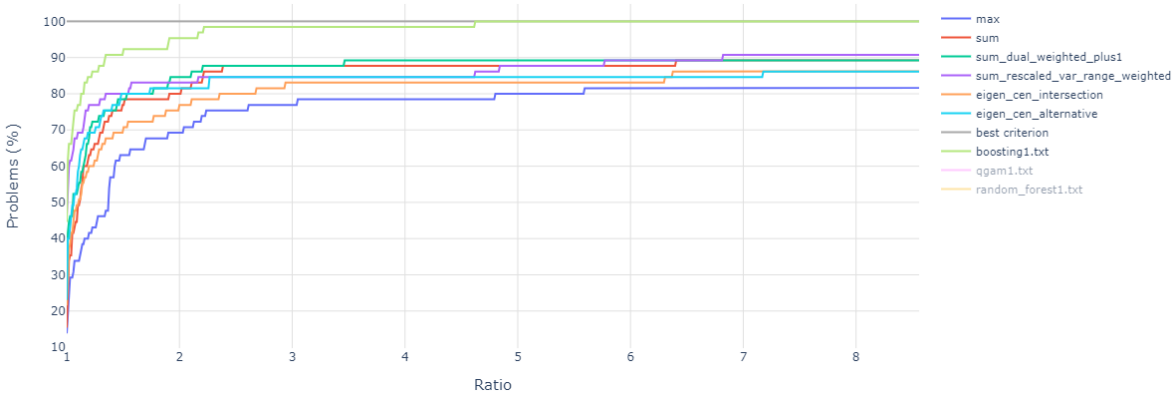


Figura 3.11: Análisis del rendimiento de software (*performance profile*) para la primera submuestra de problemas, comparando los resultados obtenidos para las seis configuraciones de `RAPOSa` y las predicciones obtenidas mediante el método *stochastic gradient boosting*.

Al igual que para los otros dos métodos, vamos a representar en la Figura 3.11 el *performance profile* con el porcentaje de problemas en función de los ratios, calculados como la inversa de nuestra variable respuesta. En concreto, se obtienen muy buenos resultados, ya que todos los problemas presentan un ratio menor que 5, en el caso de nuestro modelo de aprendizaje. Además, ya observamos que la gran mayoría se sitúa por debajo del valor 2. En lo que respecta al resto de criterios, los resultados son los mismos que antes, por lo que ya sabemos que se alcanzan valores muy elevados del ratio y, por lo tanto, un peor desempeño.

3.2.4. Importancia de los predictores

En esta última sección, vamos a incluir una serie de gráficos relativos a la importancia de los predictores, calculada a partir de los modelos de bosques aleatorios y boosting, construidos anteriormente. Comenzaremos mostrando los resultados para el primero de ambos modelos. En concreto, en la función `ranger` hemos incluido la opción `importance = ‘‘impurity’’`, que calcula la importancia Gini, también conocida como impureza de media decreciente. Esta medida calcula la importancia de cada atributo como la suma del número de divisiones (en el conjunto total de árboles) que incluyen al mismo, esto es, el número de veces que la variable es seleccionada al azar como candidata para el corte, proporcionalmente al número de muestras que divide.

En la Figura 3.12 hemos representado dos gráficos de importancia, correspondientes a los dos primeros criterios de ramificación, el criterio del máximo y el criterio de la suma. En concreto, estamos considerando los problemas de entrenamiento pertenecientes a la primera submuestra. Por otra parte, el orden de las variables ha sido seleccionado manualmente, de forma que coincida para los dos métodos que vamos a comentar. Así, no vamos a utilizar los gráficos “estándar” de importancia de los predictores, donde las variables explicativas aparecen colocadas en orden decreciente de importancia. Aun así, hemos tomado como referencia los resultados obtenidos para bosques aleatorios, tratando de reproducir dicho esquema.

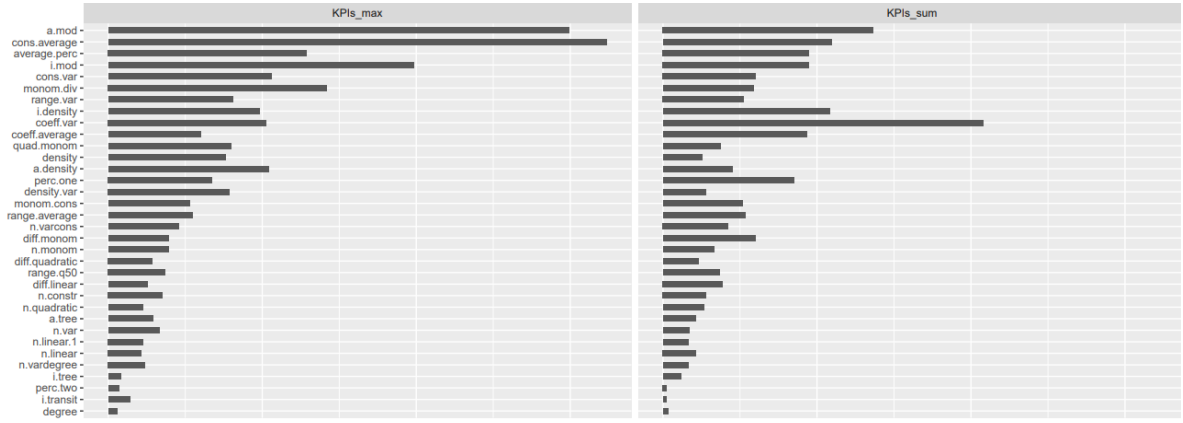


Figura 3.12: Importancia de las variables predictoras utilizando la metodología de bosques aleatorios, para los criterios de ramificación del máximo y de la suma. Estamos considerando únicamente 34 variables explicativas del conjunto total de atributos.

Si nos centramos en el gráfico de la izquierda, observamos que las variables `a.mod` y `cons.average` son las que presentan una mayor importancia para el criterio del máximo, seguidas de la variable `i.mod`. Por su parte, en el gráfico de la derecha correspondiente al criterio de la suma, observamos que la variable de mayor importancia es `coeff.var`, seguida de `a.mod`. Sin embargo, el nivel de importancia de esta última se ha reducido en gran medida, al igual que el del resto de predictores. Por lo tanto, comprobamos que los valores de importancia varían entre ambos criterios. Asimismo, aunque no se

representa, también se aprecian diferencias significativas en los cuatro criterios restantes.

Ahora bien, no nos sorprende que existan diferencias en la importancia de los predictores entre los seis criterios de ramificación, pues se definen de forma independiente entre sí. Sin embargo, estos valores deberían conservarse a lo largo de las 10 submuestras. Para reflejar este comportamiento, vamos a centrarnos en el criterio de la suma y mostraremos, en la Figura 3.13, los resultados obtenidos para la segunda, tercera y cuarta submuestra. Así, parece que se conservan, a grandes rasgos, los valores de importancia obtenidos en la Figura 3.12. En concreto, tenemos que las variables de mayor importancia para estas cuatro submuestras siguen siendo `a.mod` y `cons.average`. Por último, cabe destacar que no se muestra la ordenación de los atributos, pues estamos considerando la misma que en el gráfico anterior.

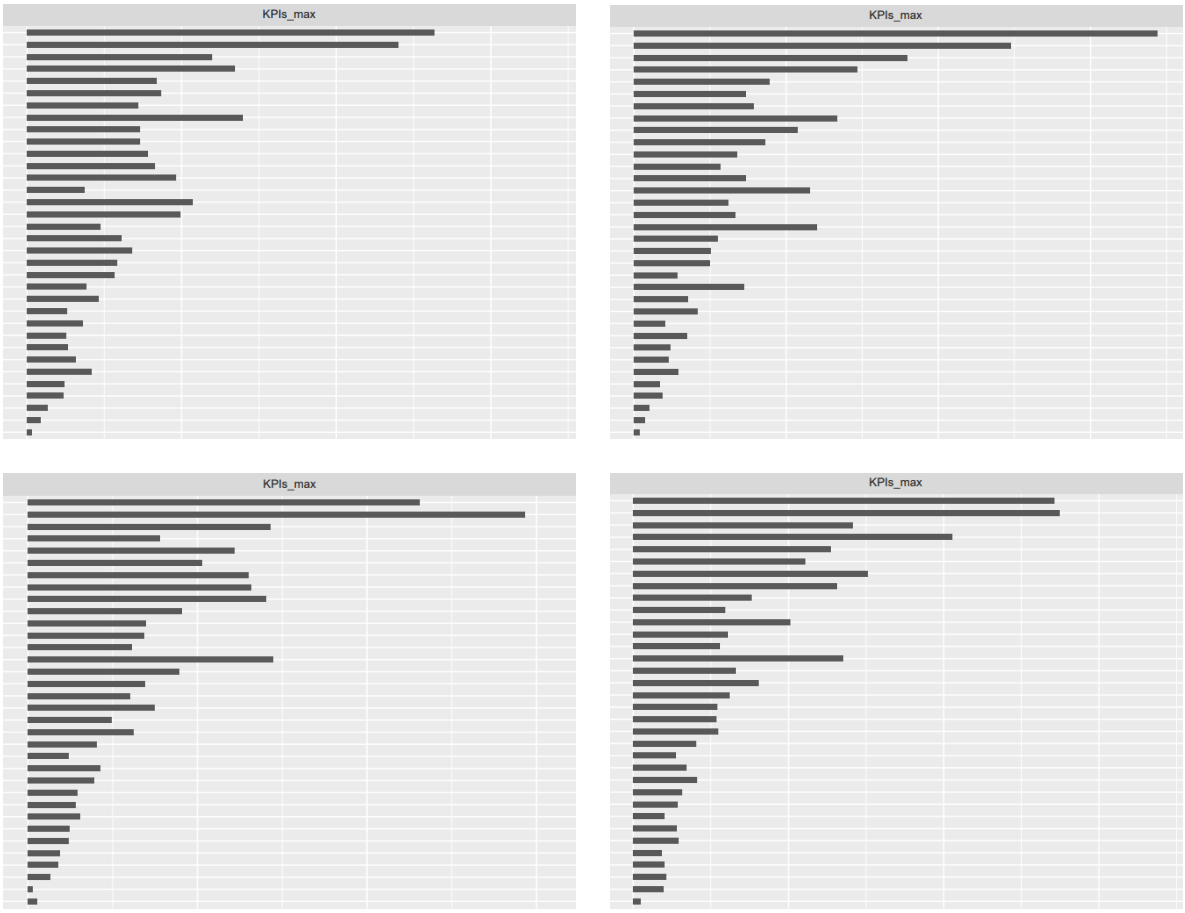


Figura 3.13: Importancia de las variables predictoras utilizando la metodología de bosques aleatorios para diferentes submuestras de entrenamiento, y considerando el criterio del máximo. Se sigue manteniendo la ordenación de las variables predictoras considerada anteriormente.

Por otra parte, vamos a comentar los resultados de importancia de los predictores obtenidos a partir del método boosting y, en concreto, mediante el algoritmo *stochastic gradient boosting*. En este caso, obtendremos las medidas de importancia a partir de la función `summary()` de `R`, sobre cada modelo ajustado. Estos valores de importancia se calculan siguiendo la técnica descrita en Breiman (2001) y que, en concreto, ya se comentó en la Sección 2.3.2 del segundo capítulo. Así, la importancia de un predictor estará representada por el aumento medio del error de predicción provocado por la permutación del mismo. De esta forma, estamos utilizando una medida de importancia distinta a la

empleada en el método de bosques aleatorios, lo que provocará diferencias en los resultados obtenidos. En la Figura 3.14 se recogen de nuevo los valores de importancia de los predictores, correspondientes a los dos primeros criterios de ramificación y a los problemas de entrenamiento de la primera submuestra. Como ya se podía intuir, en este caso no vamos a apreciar ningún orden decreciente en la importancia de estas variables. Asimismo, no se observa ningún atributo que destaque en importancia sobre el resto, en contraste con lo que ocurría para el modelo anterior.

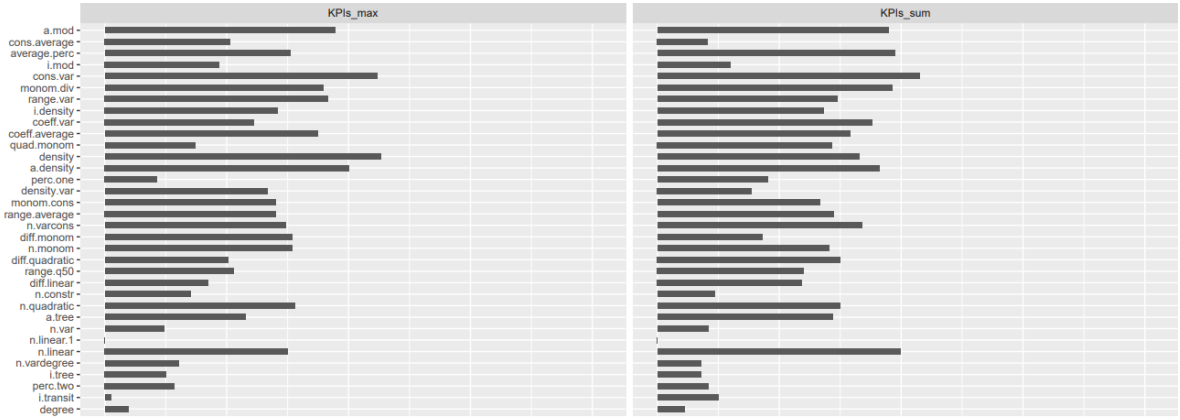


Figura 3.14: Importancia de las variables predictoras utilizando el método *stochastic gradient boosting*, para los dos primeros criterios de ramificación, esto es, el criterio del máximo y el criterio de la suma. En el eje vertical observamos las 34 variables explicativas consideradas en el proceso de aprendizaje.

Finalmente, cabe destacar que se han considerado los resultados obtenidos para las 10 submuestras disponibles, tanto para este método como para el modelo de bosques aleatorios, comprobando que la importancia de los predictores se conserva, para cada uno de los criterios, entre las diferentes submuestras. De esta forma, podemos garantizar que nuestros modelos ofrecen resultados consistentes para el conjunto total de problemas de nuestra base de datos.

Capítulo 4

Conclusiones

Concluiremos esta memoria resumiendo todos los resultados obtenidos en el capítulo anterior, determinando así cuál es el modelo que ofrece un mejor resultado, de entre los tres métodos ajustados. En el Cuadro 4.1 hemos recogido estos resultados, mostrando las medias geométricas de nuestra medida "ritmo" para cada una de las diez submuestras de test disponibles, así como su promedio, calculado en la última fila. A la vista de los resultados observados en el cuadro, podemos concluir que el modelo de bosques aleatorios es el que ofrece un mejor resultado, seguido del método boosting.

	Óptimo (test)	Criterio óptimo	Boosting	QGAM	Random forest
Submuestra	Media geométrica	Media geométrica	Media geométrica	Media geométrica	Media geométrica
1	0.469	0.241	0.278	0.281	0.287
2	1.020	0.550	0.649	0.649	0.623
3	1.414	1.013	1.162	1.146	1.140
4	0.886	0.574	0.683	0.651	0.695
5	1.460	0.920	1.044	1.058	1.010
6	1.353	0.868	1.089	1.040	1.056
7	0.989	0.527	0.586	0.574	0.580
8	0.831	0.616	0.706	0.742	0.704
9	1.484	1.002	1.156	1.146	1.122
10	1.719	1.130	1.281	1.354	1.274
Promedio	1.163	0.744	0.863	0.864	0.849

Cuadro 4.1: Tabla con un resumen de los resultados obtenidos en el capítulo anterior. En concreto, tenemos una fila por cada una de las 10 submuestras consideradas y la media geométrica de nuestra medida de rendimiento, "ritmo", obtenida al seleccionar el mejor criterio, en promedio, para todas las submuestras (dual); seleccionando, para cada problema, siempre el mejor criterio; y seleccionando el mejor criterio a partir de nuestros tres modelos de aprendizaje.

En concreto, obteníamos una mejora del 74.95 % para esta metodología, frente al 71.29 % de mejora obtenida con el modelo de regresión cuantil aditivo generalizado y el 71.50 % conseguido por el algoritmo *stochastic gradient boosting*. Asimismo, cabe destacar que la metodología de bosques aleatorios es la que presenta una implementación más rápida, mientras que, por el contrario, el modelo QGAM precisa de un tiempo de ejecución elevado. Es más, podemos encontrarnos con problemas en este sentido al aumentar el número de funciones consideradas en la base de splines. En lo que respecta al método boosting, su tiempo de ejecución no es tan elevado, pero debemos realizar un tuneado de los parámetros para ajustar cada uno de los seis modelos, sin obtener mejoras significativas. De hecho, vemos que el desempeño de estos dos últimos métodos es similar.

A continuación, recogemos en la Figura 4.1 el *performance profile* con los resultados para la primera submuestra de test, mostrando conjuntamente los resultados obtenidos para los diferentes modelos. Puesto que hemos obtenido un rendimiento similar por parte de los tres métodos, comprobamos que sus gráficas no presentan diferencias significativas, siendo todas ellas claramente mejores que las obtenidas para las seis configuraciones de `RAPOSa`.

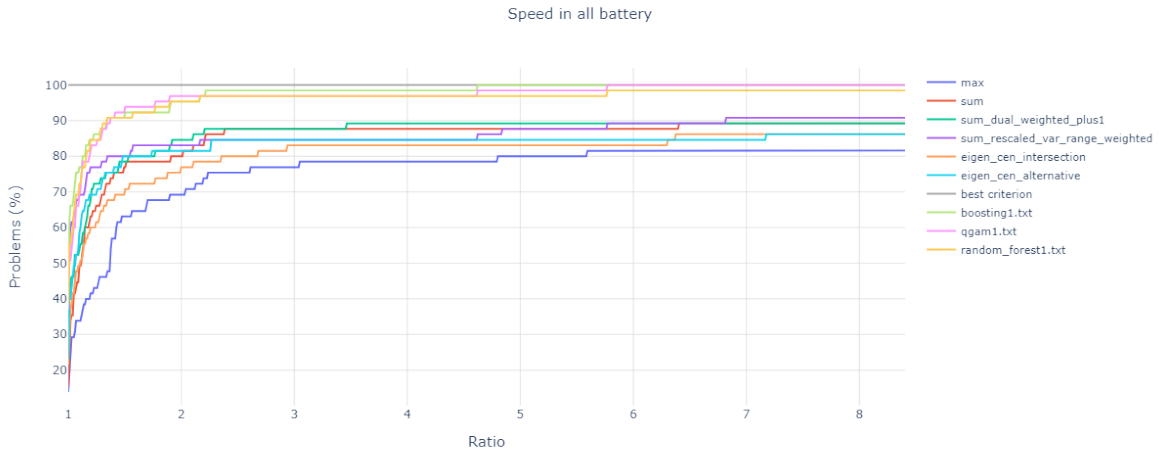


Figura 4.1: Análisis del rendimiento de software (*performance profile*), para la primera submuestra de test, comparando los resultados disponibles para las seis configuraciones de `RAPOSa`, y las predicciones obtenidas a partir de los tres modelos ajustados en el capítulo anterior: QGAM, bosques aleatorios y boosting.

Por lo tanto, en base a todos los resultados comentados, tenemos que los tres modelos consiguen beneficiarse del proceso de aprendizaje y proporcionar mejores resultados que la elección manual de un criterio de ramificación para la resolución de todos los problemas de programación polinómica. En concreto, nos decantaremos por el modelo de regresión cuantil con bosques aleatorios, ya que su desempeño es ligeramente mejor que el de los otros dos modelos. A continuación, utilizaremos este modelo seleccionado (bosques aleatorios) para proporcionar una serie de resultados adicionales que pueden derivarse del mismo y que aportarán más argumentos a favor de su elección como el mejor modelo para este proceso de aprendizaje.

4.1. Evaluación de la precisión de las predicciones

Como ya se comentó en la Sección 2.3.1 del segundo capítulo, podemos evaluar el desempeño de nuestro modelo de regresión cuantil con bosques aleatorios utilizando las muestras OOB. Así, esta

metodología genera por boostrapping B muestras de entrenamiento y ajusta, con cada una de ellas, un modelo distinto. Sin embargo, tendremos que, aproximadamente, un tercio de los problemas generados estarán repetidos, por lo tanto, solo utilizaremos dos tercios de los problemas de cada muestra de entrenamiento para construir el modelo. Como ya vimos, hemos denominado por out-of-bag a una observación que no participó en el ajuste de un determinado árbol. De esta forma, obtenemos las muestras OOB, las cuales están formadas por los problemas no utilizamos para entrenar un modelo y que, por lo tanto, podrán emplearse a modo de muestras de test para evaluar la precisión de las predicciones de cada modelo.

En nuestro caso, no nos centraremos en obtener el error cuadrático medio OOB, sino que aprovecharemos la ventaja que supone disponer de muestras de test con las que realizar predicciones. Así, en lugar de dividir nuestra base de datos en 10 submuestras y construir para cada una de ellas un conjunto de entrenamiento y otro de test, vamos a considerar el total de nuestros problemas y utilizar las muestras OOB para realizar las predicciones a partir del modelo construido. En concreto, ajustaremos un modelo por cada uno de los seis criterios de ramificación, tal y como se describió en el capítulo anterior. Además, los resultados obtenidos también serán análogos, diferenciándose, únicamente, en el hecho de que ahora solo disponemos de una submuestra.

Por lo tanto, comenzamos mostrando en el Cuadro 4.2 los resultados obtenidos para las seis configuraciones de `RAPOSa` y para nuestro método de aprendizaje. En concreto, la primera columna, `Óptimo (test)`, consiste en resolver todos los problemas con el mismo criterio (dual), siendo este el que presenta un mejor valor de la medida "ritmo", en promedio; y, a continuación, efectuar la media geométrica de estos valores seleccionados. En lo que respecta a la segunda columna, `Óptimo`, consiste en elegir, para cada problema, siempre el mejor criterio para, posteriormente, efectuar la media geométrica de los valores de "ritmo" obtenidos. Por último, calculamos, mediante las observaciones out-of-bag, las predicciones de la variable respuesta, Y, para cada uno de nuestros problemas. Utilizando estas predicciones, podemos recuperar la medida "ritmo", asociada al criterio óptimo, y efectuar la media geométrica de estos valores. El resultado de este proceso se recoge en la última columna, `OOB`, que se muestra en el cuadro.

Óptimo (test)	Óptimo	OOB
1.059	0.624	0.748

Cuadro 4.2: Tabla de resultados con la media geométrica de nuestra medida de rendimiento, "ritmo", obtenida al seleccionar el mejor criterio, en promedio, para nuestra base de datos (dual); seleccionando, para cada problema, siempre el mejor criterio; o utilizando las muestras OOB para obtener las predicciones mediante el método de bosques aleatorios.

Por otra parte, en el Cuadro 4.3, recogemos el margen de mejora existente entre elegir el mejor criterio de cada vez y el criterio que presenta mejores resultados, en promedio (dual), para el conjunto de problemas de nuestra base de datos, de la misma forma que se calculaba en capítulo anterior.

Margen de mejora	Nuestra mejora	Porcentaje de mejora
0.410	0.294	0.716

Cuadro 4.3: Margen de mejora para la configuración básica de `RAPOSa`, junto con la mejora obtenida utilizando las muestras OOB en la metodología de bosques aleatorios.

Asimismo, se incluye nuestra mejora, sustituyendo el mejor criterio por el que nos proporciona nuestro modelo de aprendizaje. Es decir, realizamos el siguiente cociente:

$$\frac{1.059 - 0.746}{1.059} = 0.294,$$

Obtenemos un porcentaje de mejora del 71.6 %. En concreto, esta mejora ha disminuido con respecto al modelo de bosques aleatorios ajustado anteriormente, para el que habíamos obtenido un porcentaje del 74.95 %. Sin embargo, sigue siendo una mejora aceptable y competitiva con los demás modelos ajustados, pues es ligeramente superior a los porcentajes obtenidos para el método boosting, 71.50 %, y el modelo QGAM, 71.29 %. De todos modos, debemos tener en cuenta que el conjunto de entrenamiento para los modelos construidos en el tercer capítulo es diferente al considerado para este último. Al trabajar con árboles aleatorios y las muestras OOB, estamos considerando el conjunto total de nuestros problemas para entrenar el modelo, lo cual no ocurría antes, pues los datos pertenecientes a las muestras de test no participaban en el ajuste del modelo. Luego, estamos proporcionamos una mayor información a este último modelo.

A continuación, incluiremos una serie de gráficos para estudiar el desempeño de este modelo en relación a las seis configuraciones de `RAPOSa`. Trabajaremos con el mismo tipo de gráficos que en el capítulo anterior. Así, comenzaremos incluyendo en la Figura 4.2 un *performance profile* con los resultados del aprendizaje obtenidos para los problemas de nuestra base de datos. En concreto, obtenemos que casi el total de los problemas presentan un ratio inferior a 2. Por su parte, los resultados obtenidos para el resto de criterios son similares a los del capítulo anterior y, por lo tanto, peores que con el aprendizaje.

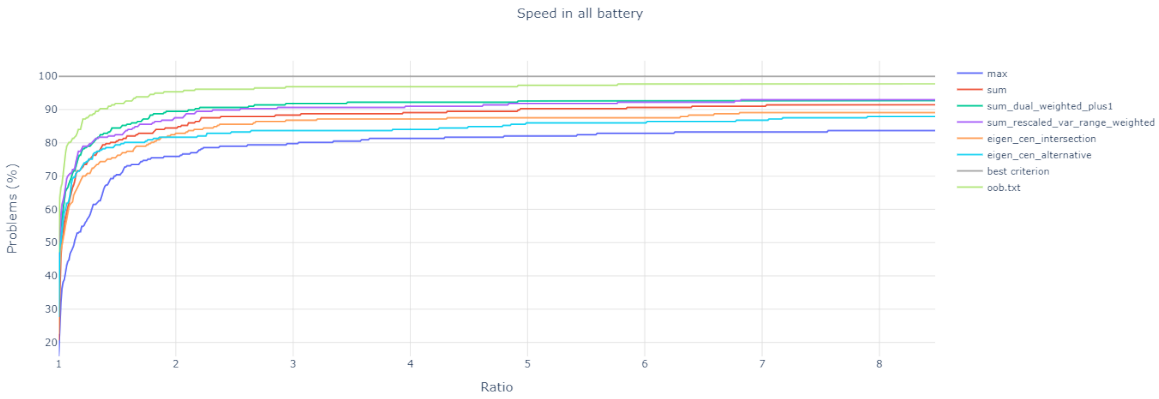


Figura 4.2: Análisis del rendimiento de software (*performance profile*), sobre el conjunto total de problemas, comparando los resultados obtenidos para las seis configuraciones de `RAPOSa`, y las predicciones obtenidas a partir de las muestras OOB, con el método de bosques aleatorios.

En la Figura 4.3 mostramos el gráfico correspondiente al diagrama de cajas de nuestra variable respuesta en función al criterio utilizado para la ramificación. Sin entrar en demasiado detalle, podemos concluir a partir de ambos gráficos que nuestro modelo de bosques aleatorios presenta un mejor desempeño que cualquiera de las seis configuraciones de `RAPOSa`. Por lo tanto, será una mejor opción a la hora de seleccionar el criterio de ramificación óptimo.

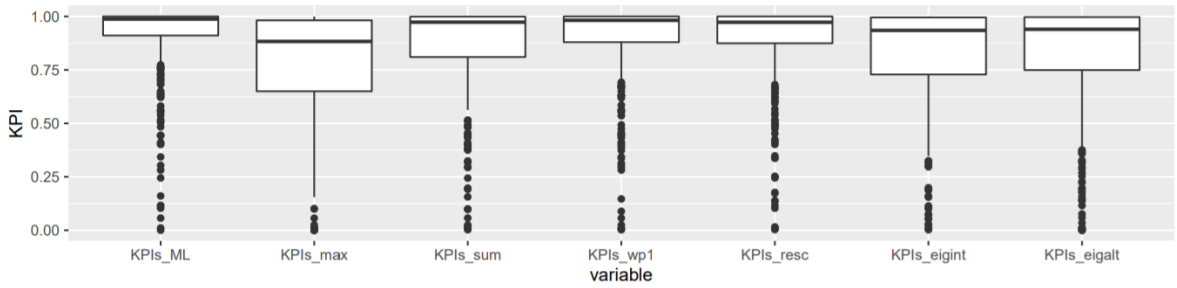


Figura 4.3: Diagrama de cajas con valores de nuestra variable respuesta para cada uno de los criterios, incluyendo los valores obtenidos a partir de las muestras OOB.

A mayores, incluimos la Figura 4.4, que corrobora la afirmación anterior. En este diagrama se vuelven a recoger los valores de nuestra variable respuesta en función del criterio de ramificación considerado. Sin embargo, vamos a añadir una escala de colores, de verde a rojo, asignados a los seis criterios disponibles, siguiendo un orden decreciente de optimalidad. De esta forma, el color verde oscuro se corresponderá con el mejor criterio para la resolución de un determinado problema, mientras que el color rojo será el peor criterio posible. Por último, se incluye tanto el porcentaje de problemas, como el número de ellos resueltos por cada criterio de ramificación.

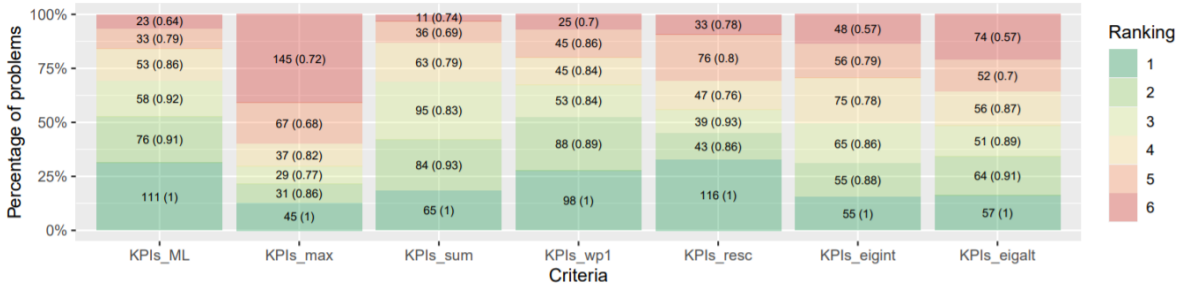


Figura 4.4: Diagrama del porcentaje de problemas en función de los valores de la variable respuesta, Y. Por lo tanto, incluimos una columna por cada criterio de ramificación, añadiendo una al principio, `KPIs_ML`, que representa las predicciones obtenidas a partir de las muestras OOB con el método de bosques aleatorios.

Comentemos en mayor detalle los resultados recogidos en el gráfico anterior, por ejemplo, para el criterio obtenido tras el proceso de aprendizaje, `KPIs_ML`. En concreto, observamos que hay 111 problemas para los cuales el criterio seleccionado por nuestro modelo es el mejor posible, esto es, nuestra variable respuesta toma el valor 1. Fijémonos en que los valores que toma la variable respuesta, en cada sección coloreada, aparecen entre paréntesis. Por otra parte, la variable respuesta toma el valor promedio 0.91 para 76 de nuestros problemas, para los cuales estaremos seleccionando el segundo mejor criterio (zona coloreada en verde claro). Así, el 50 % de los problemas se van a resolver seleccionando, o bien el mejor criterio, o bien el segundo mejor. Finalmente, tenemos que hemos seleccionado el peor criterio para 23 de nuestros problemas, tomando la variable respuesta un valor medio de 0.64.

Si comparamos estos resultados, por ejemplo, con la tercera columna, que se corresponde con el criterio de la suma, `KPIs_sum`, comprobamos cómo solo 65 problemas han sido resueltos eligiendo el mejor criterio, mientras que seleccionamos el segundo mejor en 84. Asimismo, esto no abarca todavía el 50 % del total de la base de datos, es decir, en esta primera mitad de los datos también se asignará el tercer mejor criterio a algunos de los problemas. En concreto, observamos que hay un total

de 95 problemas para los que se elegirá este tercer mejor criterio, con un valor de nuestra medida de rendimiento medio de 0.83. Por lo tanto, la eficacia de este criterio será menor. Asimismo, podemos extender este comentario al resto de criterios disponibles, justificando así la mejora obtenida con el proceso de aprendizaje.

Por último, nos vamos a detener en la importancia de los predictores. En la Figura 4.5 hemos recogido dos de los gráficos de importancia para los criterios del máximo y de la suma. En concreto, vamos a continuar con la ordenación de los atributos propuesta en el capítulo anterior, que había sido elegida convenientemente para tratar de reproducir el orden decreciente de importancia de las variables predictoras. Si nos fijamos en la gráfica de la izquierda, que se corresponde con el criterio del máximo, comprobamos que se conserva este comportamiento, siendo las variables más importante `a.mod` y `cons.average`. En cuanto al criterio de la suma, ya se aprecia una mayor diferencia.

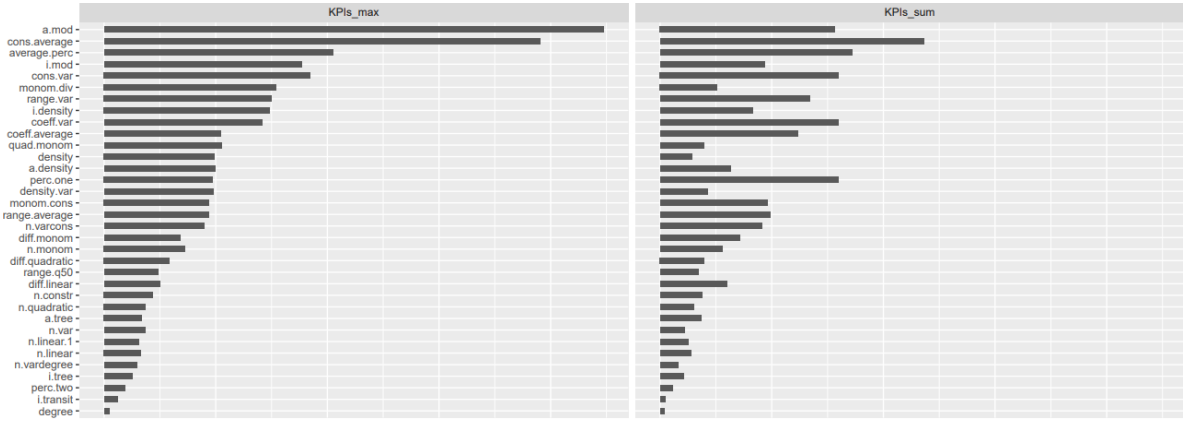
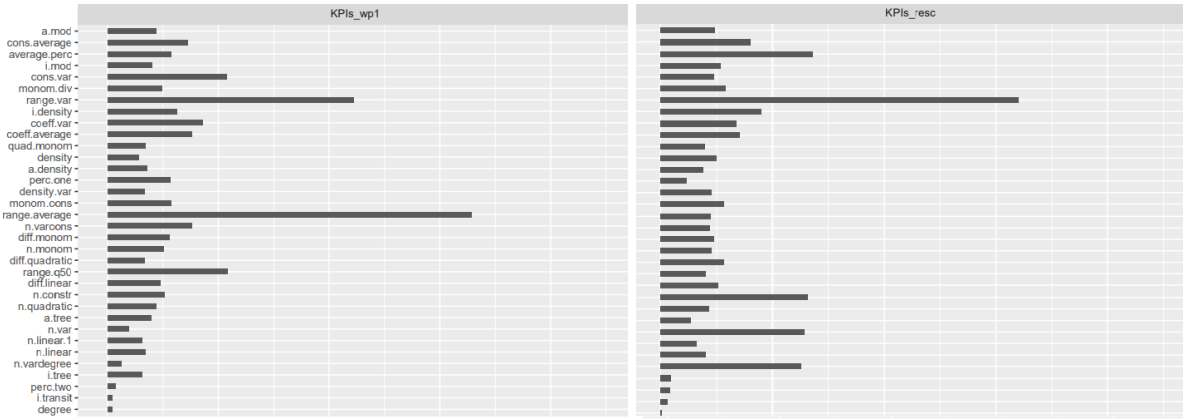


Figura 4.5: Importancia de las variables predictoras, utilizando el método de bosques aleatorios para las muestras OOB. En concreto, mostramos los resultados obtenidos para los criterios del máximo y de la suma.

Adicionalmente, podemos incluir los gráficos de importancia correspondientes a los otros cuatro criterios de ramificación. Como ya habíamos comentado en el capítulo anterior, la importancia de las variables explicativas va a modificarse dependiendo del criterio que estemos considerando. En la Figura 4.6 hemos recogido estos cuatro gráficos, conservando la ordenación anterior de los atributos. A grandes rasgos, se comprueba gráficamente cómo distan enormemente de los incluidos en la Figura 4.5, y entre ellos mismos. En concreto, las variables `a.mod` y `cons.average` únicamente muestran un nivel de importancia significativo para el último criterio, `KPIs_eigalt`.



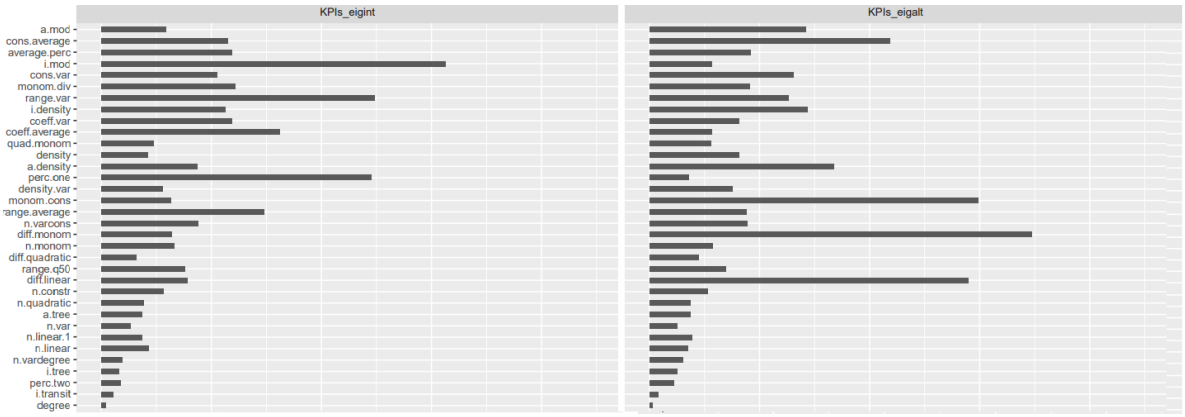


Figura 4.6: Importancia de las variables predictoras utilizando la metodología de bosques aleatorios, para los criterios de valores duales, rango de la variable y la centralidad de vector propio (calculada para los dos grafos).

4.2. Trabajo futuro

En resumen, a lo largo de esta memoria hemos implementado diferentes metodologías de aprendizaje supervisado con el objetivo de predecir el criterio de ramificación óptimo para la resolución de los problemas de programación polinómica de nuestra base de datos. De esta forma, hemos trabajado con los modelos de regresión cuantil aditivos generalizados (QGAMs) y los métodos de bosques aleatorios y boosting, aplicados a problemas de regresión cuantil. En concreto, hemos analizado los resultados obtenidos de forma detallada y con una mayor extensión de lo que nos permite mostrar esta memoria. Así, cabe destacar que se ha estudiado el comportamiento de los tres modelos al separar nuestros problemas en función de la batería a la que pertenecen (DS-TS, MINLPLib-TS y QPLIB-TS). De esta forma, se ha reproducido el estudio llevado a cabo durante el tercer capítulo del trabajo para cada uno de estos conjuntos de problemas, comprobando que existe consistencia en los resultados, no solo al considerarlos de forma conjunta, sino también al diferenciarlos por batería.

Asimismo, esto nos ha permitido conocer en mayor medida a nuestros problemas y, así, determinar cómo se diferencian entre ellos y qué consideraciones adicionales debemos tener en cuenta a la hora de llevar a cabo nuestro aprendizaje. Por ejemplo, se comprueba que los problemas de la batería QPLIB-TS son muy diferentes al resto y, por lo tanto, aunque no tengamos un número elevado de los mismos, es fundamental incluirlos en el entrenamiento, si queremos obtener buenas predicciones en la muestra de test. Consideraciones similares han conducido a la construcción de las "familias" de problemas, consiguiendo así que nuestras muestras de entrenamiento recojan información variada de las tres baterías y, de esta forma, se obtengan predicciones buenas (y sobre todo consistentes) en el test.

En lo que respecta a los resultados recogidos en el tercer capítulo, para el conjunto total de datos, comprobamos también que eran consistentes, apoyándonos en las 10 submuestras construidas. Como ya se comentó, únicamente se mostraron diagramas de barras para la segunda de ellas, así como los *performance profiles* y los gráficos de importancia de los predictores para la primera. Sin embargo, se estudiaron en detalle los diagramas obtenidos para el total de las mismas, proporcionando a mayores tablas que recogían numéricamente todos los resultados obtenidos. En particular, pudimos concluir que la metodología de bosques aleatorios es la que consigue un mejor desempeño. Dicha técnica nos ha permitido, como vimos en este último capítulo, realizar un estudio exhaustivo de la precisión de las predicciones, a través de las muestras OOB, considerando así todos los problemas de nuestra base

de datos para llevar a cabo el entrenamiento, con el aumento de información proporcionada al modelo que eso conlleva. Claramente, estos resultados ya no son comparables con los obtenidos en el capítulo anterior, pero nos permiten aumentar el número de observaciones incluidas en la construcción del modelo, evitando la salvedad que podría ocasionar el no disponer de un conjunto de problemas lo suficientemente grande para la realización de este estudio. Es más, dicha posibilidad nos lleva a decantarnos definitivamente por la elección de esta metodología, frente a los otros modelos ajustados.

Siguiendo con esta línea de investigación, se podría intentar generalizar este proceso de aprendizaje sobre los criterios de ramificación del algoritmo de `RAPOSa`. Hasta ahora, únicamente hemos tratado de aprender sobre los mismos de manera que nuestro modelo consiga seleccionar el mejor de ellos para la resolución de un determinado problema. Por lo tanto, esta decisión nos lleva a resolver un problema concreto utilizando siempre el mismo criterio, en todos los nodos del avance del algoritmo. Sin embargo, esta técnica de aprendizaje podría intentar refinarse y, con ello, ser más precisa. Así, en lugar de fijar un solo criterio para cada problema, intentaríamos modificarlo a medida que crece el árbol. Con esto surge una nueva técnica, conocida como "aprendizaje online" y que, en particular, podría conseguir un mejor desempeño al adecuarse a la información adicional que se consigue en cada iteración del algoritmo de ramificación y acotación.

Por otra parte, existe un amplio abanico de posibilidades para la aplicación de estas técnicas de aprendizaje supervisado. Como ya comentamos, en esta memoria se están considerando seis configuraciones básicas de `RAPOSa`, cuya información se utiliza para llevar a cabo el proceso de aprendizaje. Así, nos estamos centrando en la selección del criterio de ramificación óptimo, manteniendo fijos el resto de elementos que constituyen el algoritmo de `RAPOSa`. Sin embargo, este no es el único enfoque posible, ya que, como ya comentábamos en el primer capítulo, existen numerosas mejoras del mismo (ver González-Rodríguez et al. (2020)) que podrían utilizarse o, incluso, combinarse, para así conseguir un mejor desempeño. Por lo tanto, tendremos que estas técnicas de aprendizaje son extrapolables, no solo a los criterios de ramificación, sino a otro tipo de decisiones que se toman a lo largo del algoritmo RLT.

Apéndice A

Ramificación y acotación: algoritmo RLT

En este apéndice describiremos en mayor detalle el algoritmo RLT, presentado en la Sección 1.2.2 de esta memoria. En concreto, tendremos que se trata de un algoritmo de ramificación y acotación, por lo que introduciremos de forma general lo que se entiende por este tipo de algoritmos.

La idea de los algoritmos de ramificación y acotación será dividir sucesivamente la región factible del problema de partida e ir resolviendo las versiones relajadas de los problemas resultantes. Así, se irá mejorando sucesivamente la cota inferior del valor óptimo del problema de partida. Por otra parte, cada vez que resolvamos una versión relajada y obtengamos una solución que cumpla las restricciones 1.3, dicha solución será factible en el problema original y, por tanto, nos proporcionará una cota superior del valor óptimo del problema de partida. En el momento en que la cota superior coincida con la inferior, habremos encontrado una solución óptima del problema de partida. En la literatura conocida, hay distintas técnicas de ramificación y acotación, que difieren en distintos aspectos, por ejemplo, en el criterio usado para dividir el conjunto factible o en el orden en el que se resuelven los problemas.

A continuación, incluiremos el algoritmo RLT, descrito en González-Rodríguez (2017):

Algoritmo 2 Algoritmo *Reformulation-Linearization Tecnique* (RLT)

- Inicialización:
  - Inicializamos la mejor solución como $x^* = \emptyset$ y el mejor valor objetivo como $v^* = \infty$. Tomamos $k = 1, T_1 = \{1\}$ y $\Omega^{1,1} = \Omega$.
  - Resolvemos $\text{LP}(\Omega^{1,1})$ y obtenemos una solución óptima $(x^{1,1}, X^{1,1})$ de $\text{LP}(\Omega^{1,1})$.
  - Determinamos la variable de ramificación x_p siguiendo la regla de ramificación presentada en 1.5. Si $\theta_p = 0$ paramos, pues $x^{1,1}$ resuelve el problema original $\text{PP}(\Omega)$. Así, actualizaríamos $x^* = x^{1,1}$ y $v^* = v[\text{LP}(\Omega^{1,1})]$. En caso contrario, tomamos $t = 1$ y procedemos con la etapa 1 del algoritmo.
- Etapa 1 (ramificación). Suponemos resuelto el problema $\text{LP}(\Omega^{k,t})$, en el cual se ha determinado la variable de ramificación x_p y $\theta_p > 0$. Puesto que $\theta_p > 0$, tendremos que $l_p^{k,t} < x_p^{k,t} < u_p^{k,t}$ donde $l_p^{k,t}$ y $u_p^{k,t}$ son las cotas inferior y superior, respectivamente, de x_p en el problema $\text{LP}(\Omega^{k,t})$. A continuacion tomamos la siguiente partición de $\Omega^{k,t}$:

Algoritmo 2 Algoritmo *Reformulation-Linearization Tecnique*

$$\Omega^{k,t_1} = \Omega^{k,t} \cap \{x \in \mathbb{R}^n : l_p^{k,t} \leq x_p \leq x_p^{k,t}\}, \text{ y}$$
$$\Omega^{k,t_2} = \Omega^{k,t} \cap \{x \in \mathbb{R}^n : x_p^{k,t} \leq x_p \leq u_p^{k,t},\}$$

siendo $t_1, t_2 \neq T_k$ y actualizando $T_k = (T_k - \{t\}) \cup \{t_1, t_2\}$.

- Etapa 2 (acotación). En esta etapa se resolverá el problema lineal $\text{LP}(\Omega^{k,t_1})$, obteniendo una solución óptima (x^{k,t_1}, X^{k,t_1}) de $\text{LP}(\Omega^{k,t_1})$ con valor óptimo $\text{LB}_{k,t_1} = v[\text{LP}(\Omega^{k,t_1})]$. Determinamos ahora la variable de ramificación x_p, siguiendo el criterio definido en 1.5, para Ω^{t,k_1}.

  - Si $\theta_p = 0$ entonces x^{k,t_1} resuelve $\text{PP}(\Omega^{k,t_1})$ y nos proporciona una cota superior de $\text{PP}(\Omega)$. En tal caso, si $V[\text{LP}(\Omega^{k,t_1})] < v^*$ (la cota superior obtenida es mejor que la actual), actualizaremos la mejor solución $x^* = x^{k,t_1}$ y el mejor objetivo $v^* = \text{LB}_{k,t_1}$.
  - Si $\theta_p > 0$ guardamos el índice p de la variable de ramificación, por si fuese necesario utlizarlo más adelante.

  Repetimos la etapa 2 con t_2 y procedemos con la etapa 3.

- Etapa 3 (poda). Construimos $T_{k+1} = T_k \backslash \{t \in T_k : \text{LB}_{k,t} \geq v\}$. De este modo, eliminamos de T_k todos los nodos que nos proporcionan una cota superior peor (más grande) que la que tenemos.

  - Si $T_{k+1} = \emptyset$ paramos.
  - En caso contrario, actualizamos $\Omega^{k+1,t_1} = \Omega^{k,t}, (x^{k+1,t}, X^{k+1,t}) = (x^{k,t}, X^{k,t})$, $\text{LB}_{k+1,t} = \text{LB}_{k,t}$ para todo $t \in T_{k+1}$ e incrementamos k en una unidad.

- Etapa 4 (selección del nodo). Seleccionamos un nodo activo (k, t), donde $t \in \arg\min_{t \in T_k}\{\text{LB}_{k,t}\}$ y volvemos a la etapa 1. Es decir, seleccionamos el nodo más prometedor, en función del valor objetivo de su nodo padre.

Asimismo, el teorema 3.6 enunciado en González-Rodríguez (2017) nos permite asegurar la convergencia del anterior algoritmo. En concreto, nos garantiza que dicho algoritmo terminará obteniendo un óptimo global para $\text{PP}(\Omega)$, ya sea en un número finito de iteraciones, como en un número infinito de ellas. Para más detalles acerca del enunciado y la demostración de dicho teorema, puede consultarse la referencia anterior. Además, en esa misma sección, podemos encontrar ilustrado el algoritmo sobre un ejemplo concreto, para facilitar el entendimiento del mismo.

Bibliografía

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Bussieck, M. R., Drud, A. S., and Meeraus, A. (2003). Minlplib-a collection of test models for mixed-integer nonlinear programming. *INFORMS J. on Computing*, 15:114–119.

Czyzyk, J., Mesnier, M. P., and Moré, J. J. (1998). The `NEOS Server`. *IEEE Journal on Computational Science and Engineering*, 5(3):68 – 75.

Dalkiran, E. and Sherali, H. (2013). Theoretical filtering of rlt bound-factor constraints for solving polynomial programming problems to global optimality. *Journal of Global Optimization*, 57(4):1147–1172.

Dalkiran, E. and Sherali, H. D. (2016). Rlt-pos: Reformulation-linearization technique-based optimization software for solving polynomial programming problems. *Mathematical Programming Computation*, 8:337–375.

de Boor, C. (1978). *A Practical Guide to Spline*, volume Volume 27.

Dolan, E. D. (2001). The `NEOS Server 4.0` administrative guide. Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213.

Fasiolo, M., Wood, S. N., Zaffran, M., Nedellec, R., and Goude, Y. (2021a). Fast calibrated additive quantile regression. *Journal of the American Statistical Association*, pages 1402–1412.

Fasiolo, M., Wood, S. N., Zaffran, M., Nedellec, R., and Goude, Y. (2021b). qgam: Bayesian nonparametric quantile regression modeling in r. *Journal of Statistical Software*, 100(9):1–31.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *Annals of Statistics*, 28:337–307.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, pages 1189–1232.

Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., Sahinidis, N., Vigerske, S., , and Wiegele, A. (2018). Qplib: a library of quadratic programming instances. *Mathematical Programming Computation*, 1:237–265.

Golub, G. and Loon, C. V. (1983). Matrix computations.

González-Rodríguez, Ossorio-Castillo, J., González-Díaz, J., Ángel, González-Rueda, M., Penas, D. R., and Rodríguez-Martínez, D. (2020). Computational advances in polynomial optimization: Raposa, a freely available global solver.

González-Rodríguez, B. (2017). Introducción a la programación no lineal. Master's thesis, Universidad de Santiago de Compostela.

Gropp, W. and Moré, J. J. (1997). Optimization environments and the `NEOS Server`. In Buhman, M. D. and Iserles, A., editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer, New York, NY, 2 edition.

Horst, R. (1990). Deterministic methods in constrained global optimization: some recent advances and new fields of application. *Naval Research Logistics (NRL)*, 37(4):433–471.

Horst, R. and Tuy, H. (1990). Global optimization: Deterministic approaches.

Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95.

Koenker, R. (2005). *Quantile Regression*. Econometric Society Monographs. Cambridge University Press.

Lasserre, J. B. (2015). *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press.

Meinshausen, N. (2006). Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999.

`R Core Team` (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Sherali, H. and Adams, W. (1989). A hierarchy of relaxations and covex hull characterizations for mixed-integer zero-one programming problems. *Department of Industrial Engineering and Operations Research, Virginia Polylechnic Institute and State University, Blacksburg, Virginia 24061*.

Sherali, H. D., Dalkiran, E., and Desai, J. (2012a). Enhancing rlt-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts. *Computational Optimization and Applications*, 52(2):483–506.

Sherali, H. D., Dalkiran, E., and Liberti, L. (2012b). Reduced rlt representations for nonconvex polynomial programming problems. *Journal of Global Optimization*, 52(3):447–469.

Sherali, H. D. and Tuncbilek, C. H. (1991). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1):101–112.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.