TRABAJO FIN DE GRADO

Doble Grado en Matemáticas y Física

MÉTODOS ESTADÍSTICOS PARA BIG DATA

Pedro Martín Chávez

Dirigido por:
Dr. Rafael Pino Mejías

FACULTAD DE MATEMÁTICAS
Departamento de Estadística e Investigación Operativa
Sevilla, Diciembre 2019

Índice general

Resumen

La extracción de información y resultados de los grandes conjuntos de datos que se generan día a día en todo el mundo se ha convertido en uno de los objetivos principales a lo largo de la última década. El estudio y análisis de este Big Data mediante el ajuste de modelos estadísticos tradicionales no es posible. En este trabajo exponemos el BLB (Bag of Little Bootstraps) como un método viable para el tratamiento de estos datos y damos una descripción del mismo y de sus propiedades. Además, tratamos brevemente el Bootstrap y otros dos procedimientos relacionados y comparamos sus rendimientos con el del BLB. Finalmente, vemos una aplicación sencilla del BLB con ejemplo en el software estadístico libre R.

Abstract

The extraction of information and results from the large data sets that are generated day by day around the world has become one of the main objectives over the last decade. The study and analysis of this Big Data by adjusting traditional statistical models are not possible. In this paper, we expose the Bag of Little Bootstraps (BLB) as a viable method for the treatment of this data. We also give a description of the aforementioned method and its proprieties. In addition, we briefly discuss the Bootstrap and two other related methods and we compare them with BLB. Finally, we see a simple application of BLB with an example in the free statistical software R.

Capítulo 1

Introducción

Cuando hablamos de Big Data nos referimos a conjuntos de datos enormes, masivos en cuanto a tamaño y complejidad, y que no pueden ser tratados con las técnicas y herramientas habituales de análisis y gestión pues exceden la capacidad del software de las mismas. El origen del término Big Data parece encontrarse a mediados de los 90, en conversaciones de sobremesa en Silicon Graphics en la que participaba el reputado investigador en ciencias de la computación John Mashey [4]. Estos datos se generan de forma continua e ininterrumpida a partir de interacciones y transacciones entre personas y/o sistemas. Gigantes del comercio online como Amazon o multinacionales como Google producen una cantidad ingente de información, miles de millones de datos cada día, sin olvidar, por supuesto, las redes sociales (Facebook, Twitter, Instagram, etc.) que ponen en contacto a usuarios de todo el mundo. En el año 2013 se estimó un peso del orden del ZB para el volumen de datos producidos diariamente con un crecimiento del 40 % anual [8] .

Se suele hablar de una caracterización 3V del Big Data (Volumen, Velocidad y Variedad) que por lo general está bastante aceptada [17]. Volumen y Velocidad hacen referencia a la cantidad y rapidez con la que se generan los datos, así como al flujo de los mismos. Por su parte, Variedad abarca la gran diversidad de tipos de datos que se tratan, muchos de los cuales no están estructurados de forma tradicional o algunos incluso no están ni tan siquiera estructurados, pensemos, por ejemplo, en el estado de ánimo o los sentimientos que se extraen de una publicación en Facebook. Todo esto trae consigo la acumulación de "ruido" en el procesamiento de los datos y origina, a su vez, problemas de viabilidad computacional y estabilidad de los algoritmos. En los últimos años se han incluido 2 nuevas V's para caracterizar estos macrodatos [13]. Por un lado se habla de Veracidad para aludir a la calidad de los datos recogidos, pues un porcentaje de los mismos puede estar contaminado y, por otro, se habla de Valor para señalar el potencial o la utilidad de estos.

El desarrollo de procedimientos para el ajuste de modelos estadísticos sobre grandes volúmenes de datos es un campo en constante actualización, el enfoque que buscamos

en este trabajo es cómo aplicar estos modelos a Big Data. Los principales escollos a los que nos vamos a enfrentar son los tiempos de cálculo, que pueden ser excesivos, así como el tamaño de los conjuntos de datos.

Existen procedimientos estadísticos sólidos que son escalables computacionalmente a enormes conjuntos de datos. La escabilidad (del inglés *scalability*) se refiere a la habilidad de adaptación ante un crecimiento en magnitud con la menor pérdida posible de calidad. Una revisión sobre esto se puede encontrar en [12]. Dichos métodos se pueden agrupar en cuatro grandes categorías de acuerdo con [26]: métodos de submuestreo, los algoritmos basados en el divide y vencerás, los de memoria externa y los de actualización online.

Los procedimientos basados en el submuestreo utilizan aproximaciones que se fundamentan en la idea del uso de submuestras presentada por Politis y Romano, la cual se recoge en [21]. Entre las distintas técnicas propuestas en [26] destaca el método BLB (Bag of Little Bootstraps). Se profundizará sobre él a lo largo del documento y se encuentra a caballo entre esta categoría y la siguiente.

Los algoritmos de dividir y conquistar (o recombinar) se explican a través del dicho popular del divide y vencerás a la hora de afrontar un problema difícil y que viene a ser separarlo en partes más simples las veces que sea necesario hasta que su solución sea sencilla. Se trata de uno de los paradigmas más importantes en el diseño de algoritmos en Ciencias de la Computación y con él se combinan las soluciones de cada uno de los subproblemas para dar lugar a la solución del problema original. Este enfoque se aplica al análisis estadístico de Big Data, permitiéndonos someter los datos a un examen profundo y completo, minimizando el riesgo de perder información importante y ofreciéndonos una visualización detallada de los mismos. Este tipo de procedimientos constan en general de las siguientes etapas [9]. En primer lugar, los datos se dividen en bloques para, a continuación, ser procesados por separado. La aplicación de métodos analíticos a cada uno de estos subconjuntos se hace, en la medida de lo posible, en paralelo. Por último, se agregan las soluciones de dichos bloques para formar una solución final para el conjunto de datos completo.

En cuanto a los algoritmos de memoria externa, estos constan tanto de procedimientos para la administración de los datos como de procedimientos específicos para la realización de cálculos con los mismos. Esta gestión consiste en almacenar la información en el disco y procesarla iterativamente en subconjuntos de datos hasta que dicha información haya sido tratada en su totalidad. Esto se hace debido a que el disco ofrece mucha mayor capacidad que la memoria RAM pero, por el contrario, el acceso al mismo es considerablemente más lento que a la RAM. Las técnicas estadísticas deben ser implementadas de forma que se respete el procesamiento de los datos así gestionados. Destaca el paquete `bigmemory` del software libre R que aprovecha las características del sistema operativo con un lenguaje bajo nivel para proporcionar estructuras de datos capaces de soportar datos masivos. Estas estructuras son compatibles a su vez con las subrutinas de álgebra lineal de otros paquetes [14] .

Para todas aquellas aplicaciones que generan datos de forma continua, los cuales llegan a nuestra computadora secuencialmente, se pensó también en métodos que pudieran tratarlas. La idea de los algoritmos de actualización online es, como su propio nombre indica, ir actualizando repetidamente la estimación de los parámetros de interés a medida que se van recogiendo los sucesivos datos. En estas aplicaciones, los datos vienen fragmentados en secuencias por lo que interesa analizarlos progresivamente pero sin almacenarlos. Se desarrollan algoritmos de estimación iterativos y de inferencia estadística que se actualizan sucesivamente a medida que llegan los nuevos datos. Estos algoritmos han sido diseñados para ser computacionalmente eficientes y usar el mínimo almacenamiento posible ya que no suelen tener acceso a los datos históricos. Para profundizar en estos métodos consultar [23].

Una vez vista esta clasificación de los métodos estadísticos para Big Data, vamos a centrarnos y profundizar concretamente en el método BLB. Para ello, en el Capítulo 2, se dará una descripción del mismo y junto con sus propiedades. También se tratará más brevemente del método Bootstrap, anterior a este, junto con los algoritmos m out of n Bootstrap y Submuestreo, por la relación que guardan con ellos, y realizaremos una comparativa. En el Capítulo 3 se mostrará una aplicación del BLB con R, hablando previamente de la librería `datadr`, la cual será utilizada, y explicando resumidamente el modelo de regresión logísitica. Por último, finalizaremos con unas conclusiones de lo visto en este trabajo.

Capítulo 2

Bag of Little Bootstraps

En este capítulo se presenta una técnica inscrita en la familia de los métodos de dividir y recombinar que proporciona una vía para el ajuste y realización de inferencias para modelos estadísticos en Big Data. Tradicionalmente se considera al Bootstrap como un medio simple y poderoso para el desempeño de esta tarea, sin embargo, cuando entran en juego grandes conjuntos de datos, algo que sucede cada vez más frecuentemente, dicho procedimiento se vuelve mucho menos interesante pues las técnicas basadas en él son prohibitivamente exigentes computacionalmente hablando. Pese a que variantes como el Submuestreo o el m out of n Bootstrap pueden ser usadas para disminuir el coste de algunos de estos cálculos bootstraps, dichos métodos tienen, por lo general, escasa solidez en cuanto a la especificación de sus parámetros de ajuste y, además, necesitan normalmente conocer la tasa de convergencia del estimador, cosa que no sucede con el Bootstrap. La idea propuesta en [15] es el 'Bag of Little Bootstrap' (BLB), un nuevo algoritmo que mezcla aportaciones de los métodos anteriormente mencionados para proporcionar un medio robusto y computacionalmente eficiente.

2.1. Motivación y objetivo

El problema inferencial sobre el cálculo y la evaluación de la calidad de los estimadores siempre ha estado presente en el mundo de la estadística. Desde finales del siglo XX, con el desarrollo del Bootstrap y otros métodos relacionados, los progresos en este ámbito se han visto ligados al avance de la tecnología informática y es que el aumento de la velocidad y la capacidad de almacenamiento de los ordenadores brinda la posibilidad de que estos procedimientos se amplíen a conjuntos de datos más grandes. Sin embargo, el crecimiento de dichos conjuntos se está acelerando cada vez más, alcanzando muchos de ellos un volumen masivo. Con el fin de mitigar este inconveniente y poder trabajar con el Big Data, los recursos computacionales se están encaminando hacia arquitecturas paralelas y distribuidas que presentan nuevas capacidades para la manipulación de los datos, como la computación en nube y multinúcleo que da acceso a cientos o miles de

procesadores.

En un principio, el Bootstrap podría parecer idóneo para explotar esta tendencia hacia la computación paralela y distribuida pues cada unos de los diferentes remuestreos bootstraps se puede procesar de forma independiente por distintos nodos de cómputo. Además, este método ofrece otras características deseables en la configuración de datos masivos como son su naturaleza relativamente automática y su aplicabilidad a una amplia variedad de problemas inferenciales. A pesar de ello, la implementación habitual de esta rutina implica, como veremos en la Sección 2.3, un procedimiento computacional costoso. Alrededor del 63 % de los datos originales interviene en cada nuevo muestreo [7], lo cual puede traducirse en un agotamiento de los recursos computacionales cuando estos son procesados. De hecho, para grandes conjuntos de datos, incluso hallar una única estimación puntual en la totalidad del conjunto se transforma en un cálculo bastante exigente.

Una alternativa viable es trabajar con subconjuntos de datos. Ejemplos existentes dentro de esta inferencia basada en la simulación son el Submuestreo [21] y el m out of n Bootstrap [2]. En ambos casos, la idea es que un conjunto de datos de tamaño n se pueda procesar en múltiples conjuntos de tamaño $m < n$, calculando el estimador en cada uno de ellos. Por tanto, estos métodos parecen remediar inicialmente la deficiencia computacional clave del Bootstrap al requerir solo un cómputo repetido de las estimaciones de las muestras (o submuestras), la cuales se pueden escoger de forma que sean significativamente más pequeñas que el conjunto de datos original. Sin embargo, estos procedimientos también tienen inconvenientes como se verá en la Sección 2.4. Esto, junto con que son menos automáticos y fácilmente implementables que el Bootstrap, hace necesaria la búsqueda de una mejor opción.

La propuesta de los autores de *A scalable Bootstrap for massive data* [15] es un procedimiento automático y preciso que, como se recoge en el título del artículo, es escalable a grandes conjuntos de datos. Denominado BLB, se trata de un método que, combinando los aspectos positivos de los anteriormente vistos, cumple con el objetivo perseguido. Caracterizado por ser computacionalmente más favorable que el resto de procedimientos, su implementación es fácilmente paralelizable pudiendo aprovechar las capacidades de las plataformas modernas de procesamiento distribuido. También mantiene la aplicabilidad genérica del Bootstrap y ofrece propiedades estadísticas y de consistencia superiores a las otras alternativas mencionadas. Todo esto será tratado a lo largo del capítulo.

2.2. Notación

Consideremos una muestra $X_1, \ldots, X_n$ independiente e idénticamente distribuida (i.i.d.) a una distribución subyacente (desconocida) P y denotemos por

$$\mathbb{P}_n = n^{-1} \sum_{i=1}^{n} \delta_{X_i}$$

la correspondiente distribución empírica. Basándonos únicamente en los datos observados, calculamos una estimación $\hat{\theta}_n = \hat{\theta}\left(\mathbb{P}_n\right)$ de un parámetro poblacional (desconocido) $\theta = \theta(P)$. Para evaluar la calidad de $\hat{\theta}_n$ introducimos $\xi\left(Q_n(P), P\right)$ como medida, donde ξ toma valores sobre un espacio vectorial Ξ. Realmente se trata de un resumen de la distribución subyacente $Q_n(P)$ de una cantidad aleatoria $u(\mathbb{P}_n, P)$ con la que trabajaremos. En la práctica, no podemos calcular $\xi\left(Q_n(P), P\right)$ de forma directa debido a que no tenemos conocimiento de P y $Q_n(P)$, luego tenemos que estimar $\xi\left(Q_n(P), P\right)$ simplemente a través de los datos observados y la forma del estadístico u en cuestión. Notemos que, con el objetivo de darle un carácter más general, permitimos que ξ dependa de P, además de $Q_n(P)$, pero, por simplicidad, denotaremos $\xi\left(Q_n(P)\right)$ pues a menudo esta dependencia directa con P tiene una forma simple y solo involucra el parámetro θ.

Con el fin de esclarecer un poco la notación veamos a qué pueden referirse estas cantidades ofreciendo algunos ejemplos. Para el caso de $\hat{\theta}_n$, es posible estimar sencillamente una medida de correlación o los coeficientes de un modelo predictivo, y cuando hablamos de ξ y la elección de u, depende de nuestro objetivo inferencial: ξ puede tratarse de la varianza de $u(\mathbb{P}_n, P) = \hat{\theta}(\mathbb{P}_n)$, la esperanza de $u(\mathbb{P}_n, P) = \hat{\theta}(\mathbb{P}_n) - \theta(P)$ (es decir, el sesgo del estimador $\hat{\theta}_n$) o una región de confianza de $u(\mathbb{P}_n, P) = \sqrt{n}(\hat{\theta}(\mathbb{P}_n) - \theta(P))$.

2.3. Bootstrap

El Bootstrap es uno de los métodos estadísticos basados en el remuestreo, entre los que también se encuentran, por ejemplo, los Tests de Permutaciones o el Jackknife. Este grupo se fundamenta en la generación de nuevas muestras a partir de la muestra disponible. Hacen uso de la potencia computacional como alternativa a las técnicas tradicionales de la inferencia estadística en aquellos casos en los que no se dan las condiciones necesarias para emplearlas. Este tipo de métodos es especialmente útil en situaciones en las que el tamaño muestral es reducido, se utilizan modelos no paramétricos o cuando no se cumplen las habituales condiciones de independencia de observaciones, homocedasticidad, etc. En ciertas ocasiones, dicho proceso de remuestreo se aplica también a cada una de las muestras generadas a partir de la original.

Particularizando para el procedimiento Bootstrap, su origen se remonta al año 1979 cuando fue introducido por B. Efron en su artículo *Bootstrap Methods: Another Look at the Jackknife* [6]. Este trataba de explicar el jackknife en términos de un método

más primitivo y general que funcionara satisfactoriamente en una gran variedad de problemas de estimación. Efron da una descripción detallada del procedimiento que nosotros resumiremos a continuación. También expone una serie de ejemplos para la aplicación del mismo (varianza de la mediana, tasas de error en un análisis discriminante lineal, estimación de parámetros de regresión, etc.) y muestra como el Jackknife se puede considerar como un método de aproximación lineal para el Bootstrap.

Dada una muestra aleatoria $X_1, \ldots, X_n$ de una distribución desconocida P y una cantidad especificada $u(X_1, \ldots, X_n; P)$, el problema a resolver por el Bootstrap es el cálculo de la distribución de u sobre la base de los datos observados. Para ello se procede como sigue:

1. Construimos la distribución empírica de la muestra $\mathbb{P}_n$, como sabemos esto se hace asignando una probabilidad de $1/n$ a cada uno de los n casos de la muestra. La elección de $\mathbb{P}_n$ es porque constituye la estimación no paramétrica de máxima verosimilitud de la distribución poblacional P.

2. De la distribución empírica $\mathbb{P}_n$, extraemos una muestra aleatoria de tamaño n con reemplazamiento y la denotamos por $X_1^*, \ldots, X_n^*$. Es lo que se conoce como muestra bootstrap.

3. Aproximamos la distribución de muestreo del estadísitico $u(X_1, \ldots, X_n; P)$ por la distribución del estadístico bootstrap $u^* = u(X_1^*, \ldots, X_n^*; \mathbb{P}_n)$, es decir, la distribución de u^* está inducida por el mecanismo de remuestreo visto en 2.

Hemos visto que, en principio, el procedimiento Bootstrap es bastante simple y es que, en realidad, la complejidad del mismo radica en el cálculo de la distribución bootstrap. De acuerdo con [6], esto se puede hacer mediante uno de estos tres métodos:

- Método 1. Cálculo teórico directo. Sin embargo, en aplicaciones prácticas o bien no se puede conocer por medios analíticos la distribución del estadístico bootstrap, o bien es inviable generar todas las muestras bootstraps posibles.
- Método 2. Aproximación de Monte Carlo a la distribución bootstrap. Se fija un entero $r > 0$ y se generan este número de muestras bootstraps de tamaño n de $\mathbb{P}_n$. El histograma con los correspondientes valores de $u_1^*, \ldots, u_r^*$ se toma como una aproximación a la verdadera distribución bootstrap.
- Método 3. Este ve restringido su uso a cuando se quiere obtener la media y la varianza aproximada de la distribución bootstrap de u^*. Consiste en utilizar la expansión en serie de Taylor.

Debemos notar que con el Método 2 lo que obtenemos son aproximaciones de los estimadores bootstraps, por lo que además de la variabilidad atribuible a la muestra original se introduce un error o variabilidad debido a este remuestreo. No podemos olvidar además que el Bootstrap proporciona estimaciones en frecuencia aproximadas,

no estimaciones en verosimilitud aproximadas y, por lo tanto, los problemas de inferencia fundamental persisten, sin importar qué tan bien funcione el método. No obstante, existen diversos resultados teóricos basados en desarrollos asintóticos que garantizan que los resultados son generalmente consistentes en las situaciones inferenciales más comunes [1]. Sin embargo, hay situaciones donde el Bootstrap falla, por ejemplo, en la estimación de regiones de confianza para un extremo, en la de extremos para distribuciones no acotadas o en la de funciones no diferenciables de la distribución empírica, como se recoge en [2].

Veamos a continuación cómo aplicar lo visto a nuestro caso. Bajo la notación que estamos siguiendo, el Bootstrap simplemente nos proporciona, basándose en los datos, la siguiente aproximación $\xi\left(Q_n(P)\right) \approx \xi\left(Q_n\left(\mathbb{P}_n\right)\right)$. Sin embargo, como hemos dicho, $\xi\left(Q_n\left(\mathbb{P}_n\right)\right)$ no puede ser calculada siempre con exactitud, por este motivo recurrimos al Método 2, es decir, a la aproximación de Monte Carlo, para su estimación. Procedemos entonces remuestreando repetidamente n puntos i.i.d. a $\mathbb{P}_n$. Llamando $\mathbb{P}_n^*$ a las distribuciones empíricas de estas nuevas muestras, calculamos el estadístico $u(\mathbb{P}_n^*, \mathbb{P}_n)$ para cada muestra bootstrap. Finalmente, sea $\mathbb{Q}_n^*$ la distribución empírica de estos valores de u, aproximamos $\xi\left(Q_n(P)\right)$ por $\xi\left(\mathbb{Q}_n^*\right)$. Recogemos esto en el Algoritmo 1.

Input: Datos $X_1, \ldots, X_n$; r, número de iteraciones de Monte Carlo; $\hat{\theta}$, estimador de interés; u, estadístico de trabajo; ξ, medida de la calidad del estimador
Output: Una estimación de $\theta(P)$ y $\xi\left(Q_n(P)\right)$
// `Construir la distribución empírica de los datos`
$\mathbb{P}_n \leftarrow n^{-1} \sum_{i=1}^{n} \delta_{X_i}$
for $k \leftarrow 1$ **to** r **do**
  // `Remuestrear los datos`
  Escoger aleatoriamente y con reemplazamiento n índices $(k_1, \ldots, k_n)$ del conjunto $\{1, \ldots, n\}$
  $\mathbb{P}_{n,k}^* \leftarrow n^{-1} \sum_{i=1}^{n} \delta_{X_{k_i}}$
  $\hat{\theta}_{n,k}^* \leftarrow \hat{\theta}\left(\mathbb{P}_{n,k}^*\right)$
  $u_{n,k}^* \leftarrow u\left(\mathbb{P}_{n,k}^*, \mathbb{P}_n\right)$
end
// `Promediar los valores de` $\hat{\theta}\left(\mathbb{P}_{n,k}^*\right)$
$\hat{\theta}_n^* \leftarrow r^{-1} \sum_{k=1}^{r} \hat{\theta}_{n,k}^*$
// `Aproximar` $\xi\left(Q_n(\mathbb{P}_n)\right)$
$\mathbb{Q}_n^* \leftarrow r^{-1} \sum_{k=1}^{r} \delta_{u_{n,k}^*}$
$\xi_n^* \leftarrow \xi\left(\mathbb{Q}_n^*\right)$
return $\hat{\theta}_n^*$, ξ_n^*

Algoritmo 1: Bootstrap

Cabe señalar que la mayor parte del coste computacional del algoritmo se encuentra en la repetición del cálculo de los valores de u, que a su vez requiere un cálculo exigente

y repetido de las estimaciones $\hat{\theta}(\mathbb{P}_n^*)$ en las muestras.

2.4. Submuestreo y m out of n Bootstrap

Para intentar solventar las limitaciones del Bootstrap en situaciones como las enumeradas en la sección anterior y buscando una gama más amplia de resultados de consistencia, se comenzaron a explorar métodos como el Submuestreo y el m out of n Bootstrap. El desarrollo de los mismos también se vio favorecido por los problemas computacionales del Bootstrap descritos en el apartado 2.1 del capítulo.

En cuanto al m out of n Bootstrap, se trata de un esquema de remuestro, basado en el Bootstrap, pero con menos de n observaciones, en concreto, las nuevas muestras tienen un tamaño $m = o(n)$. En general, el método responde bien, tanto en las situaciones en las que el habitual Bootstrap no paramétrico funciona, como en las que no, es más, si el Bootstrap proporciona aproximaciones consistentes de primer orden, entonces este procedimiento que estamos tratando es capaz de facilitárnoslas hasta el segundo orden [2]. La elección de m en todo esto es una cuestión clave, aunque la selección de un tamaño de remuestreo óptimo requiere un cálculo significativamente mayor que puede eliminar cualquier ganancia computacional. El rendimiento depende de esta m, la intuición es que para valores pequeños cada estimación es ruidosa y para valores grandes hay muy pocas muestras, por lo que es difícil encontrar el valor adecuado para cada problema tratado (en [3] se discute sobre ello).

En el ámbito del submuestreo, la verdadera distribución muestral de un estadístico determinado se estima mediante una apropiada normalización de los valores de dicho estadístico, los cuales se calculan sobre submuestras de los datos. Para el contexto de muestras aleatorias i.i.d. con n observaciones, se escogen subconjuntos de tamaño $m < n$. Suponiendo que $m \to \infty$ y que $m/n \to 0$, el método es asintótocamente válido bajo condiciones mínimas. Esencialmente, todo lo que se requiere es que el estadístico, adecuadamente normalizado, posea una distribución limite bajo el modelo verdadero. Esto no sucede, por ejemplo, en el Bootstrap, donde es necesario que dicha distribución sea de alguna manera localmente uniforme en función del modelo desconocido [22]. Por lo tanto, como no se requiere de tal suavidad en esta teoría, el método es aplicable en entornos bastante complejos. Centrándonos un poco más en su funcionamiento, el Submuestreo se comporta bien, pese a tener suposiciones tan débiles, ya que cada subconjunto de tamaño m (tomado sin reemplazo de los datos originales) constituye en sí mismo una muestra de tamaño m del modelo verdadero. Intuitivamente, esto nos permite aproximar la distribución muestral del estadístico normalizado basándonos únicamente en m observaciones en lugar de n pues, si hay convergencia débil, las distribuciones correspondientes a ambos tamaños muestrales deben estar cercanas. Para profundizar en resultados asintóticos del Submuestreo ver [19] y consultar [21] para una revisión completa del procedimiento.

Veamos ahora cómo adaptar ambos procedimientos a nuestro problema. Para $m < n$, el m out of n Bootstrap (el Submuestreo) funciona remuestreando repetidamente m puntos i.i.d. a $\mathbb{P}_n$ (submuestreando sin reemplazamiento m puntos de $X_1, \ldots, X_n$). Llamando $\mathbb{P}^*_m$ a las distribuciones empíricas de estas nuevas muestras, calculamos el estadístico $u(\mathbb{P}^*_m, \mathbb{P}_n)$ para cada remuestro (submuestra). Finalmente, sea $\mathbb{Q}^*_m$ la distribución empírica de estos valores de u, calculamos $\xi\left(\mathbb{Q}^*_m\right) \approx \xi\left(Q_m(P)\right)$. La implementación del método se detalla en el Algoritmo 2 (Algoritmo 3).

Input: Datos $X_1, \ldots, X_n$; m, tamaño de remuestreo; r, número de iteraciones de Monte Carlo; $\hat{\theta}$, estimador de interés; u, estadístico de trabajo; ξ, medida de la calidad del estimador
Output: Una estimación de $\theta(P)$ y $\xi\left(Q_n(P)\right)$
// `Construir la distribución empírica de los datos`
$\mathbb{P}_n \leftarrow n^{-1} \sum_{i=1}^{n} \delta_{X_i}$
for $k \leftarrow 1$ **to** r **do**
  // `Remuestrear los datos`
  Escoger aleatoriamente y con reemplazamiento m índices $(k_1, \ldots, k_m)$ del conjunto $\{1, \ldots, n\}$
  $\mathbb{P}^*_{m,k} \leftarrow m^{-1} \sum_{i=1}^{m} \delta_{X_{k_i}}$
  $\hat{\theta}^*_{m,k} \leftarrow \hat{\theta}\left(\mathbb{P}^*_{m,k}\right)$
  $u^*_{m,k} \leftarrow u\left(\mathbb{P}^*_{m,k}, \mathbb{P}_n\right)$
end
// `Promediar los valores de` $\hat{\theta}\left(\mathbb{P}^*_{m,k}\right)$
$\hat{\theta}^*_m \leftarrow r^{-1} \sum_{k=1}^{r} \hat{\theta}^*_{m,k}$
// `Aproximar` $\xi\left(Q_m(\mathbb{P}_n)\right)$
$\mathbb{Q}^*_m \leftarrow r^{-1} \sum_{k=1}^{r} \delta_{u^*_{m,k}}$
$\xi^*_m \leftarrow \xi\left(\mathbb{Q}^*_m\right)$
return $\hat{\theta}^*_m$, ξ^*_m

Algoritmo 2: m out of n Bootstrap

Debemos notar que una vez finalizados los algoritmos hay que aplicar una corrección analítica (por ejemplo, $\sqrt{m/n}$ [12]) para, a su vez, aproximar $\xi\left(Q_n(P)\right)$. El motivo es que debido a que la variabilidad de un estimador en una submuestra difiere de la correspondiente en el conjunto completo de datos, se producen fluctuaciones en las estimaciones por lo que estos procedimientos deben realizar una reescala de su producción. Esta nueva escala requiere un conocimiento previo y un uso explícito de la tasa de convergencia del estimador en cuestión. Este es el principal inconveniente de dichos métodos, el cual hace que sean un poco menos "fáciles de usar" que el Bootstrap.

Input: Datos $X_1, \ldots, X_n$; m, tamaño de los subconjuntos; s, número de subconjuntos de la muestra; $\hat{\theta}$, estimador de interés; u, estadístico de trabajo; ξ, medida de la calidad del estimador
Output: Una estimación de $\theta(P)$ y $\xi\left(Q_n(P)\right)$
// `Construir la distribución empírica de los datos`
$\mathbb{P}_n \leftarrow n^{-1} \sum_{i=1}^{n} \delta_{X_i}$
for $j \leftarrow 1$ **to** s **do**
  // `Submuestrear los datos`
  Escoger aleatoriamente y sin reemplazamiento un subconjunto de m índices $\mathcal{I}_j \subset \{1, \ldots, n\}$
  $\mathbb{P}^*_{m,j} \leftarrow m^{-1} \sum_{i \in \mathcal{I}_j} \delta_{X_i}$
  $\hat{\theta}^*_{m,j} \leftarrow \hat{\theta}\left(\mathbb{P}^*_{m,j}\right)$
  $u^*_{m,j} \leftarrow u\left(\mathbb{P}^*_{m,j}, \mathbb{P}_n\right)$
end
// `Promediar los valores de` $\hat{\theta}\left(\mathbb{P}^*_{m,j}\right)$
$\hat{\theta}^*_m \leftarrow s^{-1} \sum_{j=1}^{s} \hat{\theta}^*_{m,j}$
// `Aproximar` $\xi\left(Q_m(\mathbb{P}_n)\right)$
$\mathbb{Q}^*_m \leftarrow s^{-1} \sum_{j=1}^{s} \delta_{u^*_{m,j}}$
$\xi^*_m \leftarrow \xi\left(\mathbb{Q}^*_m\right)$
return $\hat{\theta}^*_m$, ξ^*_m

Algoritmo 3: Submuestreo

2.5. BLB

De la conjunción de los métodos anteriores surge el BLB. La idea básica es escoger submuestras de menor tamaño (y sin reemplazamiento) del conjunto de datos original para, después, extraer (de estos subconjuntos) muestras bootstraps de tamaño igual al de los datos completos. Se consigue, sin embargo, una ponderación en estas muestras que permite calcular las estimaciones puntuales y las medidas de calidad del estimador a partir del remuestreo con un coste computacional asociado al tamaño de las submuestras y no al de los datos originales. La combinación de estos resultados procedentes de los distintos subconjuntos nos permite que el BLB no requiera una reescala analítica de su salida. Vemos entonces que el método se compone de dos procedimientos anidados, uno interno que aplica el Bootstrap a cada submuestra, y uno externo que combina estas múltiples estimaciones bootstraps. A continuación detallamos más rigurosamente el funcionamiento siguiendo el esquema de [15].

Dada la muestra aleatoria $X_1, \ldots, X_n$, se escogen de ella, de manera uniforme, aleatoria y sin reemplazamiento, s subconjuntos de tamaño $m < n$. Llamemos $\mathcal{I}_1, \ldots, \mathcal{I}_s \subset \{1, \ldots, n\}$ a los correspondientes conjuntos de índices (con $|\mathcal{I}_j| = m$, $\forall j$), a los que

también se les podría imponer la restricción de que fueran disjuntos, y sean

$$\mathbb{P}_{n,m}^{(j)} = m^{-1} \sum_{i \in \mathcal{I}_j} \delta_{X_i}$$

las distribuciones empíricas de cada uno de estos subconjuntos, $j = 1, \ldots, s$. Entonces el BLB combina, por ejemplo en promedio, los s estimadores puntuales calculados sobre los subconjuntos y ofrece también una estimación de $\xi(Q_n(P))$ que viene dada por

$$s^{-1} \sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right).$$

Lo fundamental es la utilización de Q_n pese a que la distribución $\mathbb{P}_{n,m}^{(j)}$ se construye solo a partir de m datos. Sin embargo, generalmente los términos $\xi(Q_n(\mathbb{P}_{n,m}^{(j)}))$ en la expresión anterior no se pueden computar de forma analítica, por lo que debemos calcularlos numéricamente utilizando la aproximación directa de Monte Carlo (al igual que en el Bootstrap de la Sección 2.3). Volvemos a muestrear n puntos i.i.d. a $\mathbb{P}_{n,m}^{(j)}$ y formamos la correspondiente $\mathbb{P}_{n,m}^{*}$. Seguidamente, calculamos $u(\mathbb{P}_{n,m}^{*}, \mathbb{P}_{n,m}^{(j)})$ en cada nueva muestra para poder formar la distribución empírica $\mathbb{Q}_{n,j}^{*}$ de estas estimaciones y así aproximar $\xi(Q_n(\mathbb{P}_{n,m}^{(j)})) \approx \xi(\mathbb{Q}_{n,j}^{*})$. El cálculo del estimador $\hat{\theta}$ en cada uno de los subconjuntos se hace mediante la media de las estimaciones puntuales determinadas previamente sobre las r muestras bootstraps de las correspondientes submuestras.

El hecho crucial que debe destacarse es el gran beneficio computacional que acarrea este método y es que, pese a tener un tamaño nominal n, cada nuevo remuestreo en el BLB alberga como mucho m datos (de los de partida) diferentes entre sí. Esto se traduce, por ejemplo, en que si $n = 1000000$ y tomamos $m = n^{\gamma}$ con $\gamma = 0.7$, cada muestra tendría a lo sumo 15849 datos distintos.

En nuestro caso, las remuestras van a ser generadas por un vector de frecuencias absolutas de una distribución multinomial uniforme de n ensayos sobre m objetos o categorías. Dicha distribución no es más que la generalización a m dimensiones de la binomial. Aunque la suma de las componentes del vector de cuentas sea igual a n, propiedad de la multinomial, la distribución empírica de estas muestras consiste en (a lo sumo) m datos distintos acompañados por las frecuencias previamente muestreadas. Como es posible representar cada una de dichas muestras simplemente manteniendo su distribución empírica, entonces el espacio de almacenamiento requerido para cada una de ellas solo crece en $O(m)$. Con esto lo que se consigue es una representación de datos ponderada, por lo que, si el estimador que utilizamos puede trabajar con ella, entonces el coste computacional de la estimación (tanto con respecto al tiempo como a la memoria) tampoco escala que n, sino en m. Pese a que esta propiedad no es válida para todos lo estimadores, sí que lo es para los estimadores más habituales como los de máxima verosimilitud o los de estimación por mínimos cuadrados. El Algoritmo 4 detalla los procedimientos descritos anteriormente para la construcción del método BLB.

Input: Datos $X_1, \ldots, X_n$; m, tamaño de los subconjuntos; s, número de subconjuntos de la muestra; r, número de iteraciones de Monte Carlo; $\hat{\theta}$, estimador de interés; u, estadístico de trabajo; ξ, medida de la calidad del estimador

Output: Una estimación de $\theta(P)$ y $\xi\left(Q_n(P)\right)$

for $j \leftarrow 1$ **to** s **do**

// `Submuestrear los datos`

Escoger aleatoriamente y sin reemplazamiento un subconjunto de m índices $\mathcal{I}_j = \{j_1, \ldots, j_m\} \subset \{1, \ldots, n\}$ (también es posible seleccionar los $\mathcal{I}_j$ de manera que sean subconjuntos disjuntos de tamaño m de una partición aleatoria de $\{1, \ldots, n\}$ previamente definida)

// `Construir la distribución empírica de la submuestra`

$\mathbb{P}_{n,m}^{(j)} \leftarrow m^{-1} \sum_{i \in \mathcal{I}_j} \delta_{X_i}$

for $k \leftarrow 1$ **to** r **do**

// `Remuestrear los datos`

Sample $(k_{j,1}, \cdots, k_{j,m}) \sim \text{Multinomial}(n, \mathbf{1}_m/m)$

$\mathbb{P}_{n,k_j}^{*} \leftarrow n^{-1} \sum_{l=1}^{m} k_{j,l}\, \delta_{X_{j_l}}$

$\hat{\theta}_{n,k_j}^{*} \leftarrow \hat{\theta}(\mathbb{P}_{n,k_j}^{*})$

$u_{n,k_j}^{*} \leftarrow u(\mathbb{P}_{n,k_j}^{*}, \mathbb{P}_{n,m}^{(j)})$

end

// `Promediar los valores de` $\hat{\theta}(\mathbb{P}_{n,k_j}^{*})$ `en las muestras bootstraps`

$\hat{\theta}_{n,j}^{*} \leftarrow r^{-1} \sum_{k=1}^{r} \hat{\theta}_{n,k_j}^{*}$

// `Aproximar` $\xi(Q_n(\mathbb{P}_{n,m}^{(j)}))$

$\mathbb{Q}_{n,j}^{*} \leftarrow r^{-1} \sum_{k=1}^{r} \delta_{u_{n,k_j}^{*}}$

$\xi_{n,j}^{*} \leftarrow \xi(\mathbb{Q}_{n,j}^{*})$

end

return $s^{-1} \sum_{j=1}^{s} \hat{\theta}_{n,j}^{*},\ s^{-1} \sum_{j=1}^{s} \xi_{n,j}^{*}$

Algoritmo 4: Bag of Little Bootstraps (BLB)

Pasamos ahora a ver las propiedades estadísticas propiamente dichas del BLB. Nos referimos a la consistencia asintótica (es decir, a la aproximación en probabilidad de $\xi(Q_n(P))$) y al grado de exactitud del método. Enunciamos aquí únicamente los resultados que se obtienen en [15] para el procedimiento, donde también se encuentran las pruebas de los mismos (ver su Apéndice). La manera de proceder es la habitual de los análisis bootstraps pero cabe destacar que no se tiene en cuenta explícitamente el error introducido por la aproximación de Monte Carlo.

Antes de enunciar el primer teorema, debemos introducir los conceptos de diferenciabilidad Hadamard y clase Donsker pues son necesarios para su comprensión. Ambas definiciones han sido extraidas de [25].

Definición 1. Consideremos dos espacios normados $\mathcal{P}$, $\mathcal{R}$. Sea $\mathcal{P}_\phi$ un subconjunto de $\mathcal{P}$ que contiene a P. Una aplicación $\phi : \mathcal{P}_\phi \mapsto \mathcal{R}$ se llama diferenciable Hadamard en P

si existe una aplicación lineal y continua $\phi'_P : \mathcal{P}_\phi \mapsto \mathcal{R}$ tal que

$$\left\| \frac{\phi\left(P + th_t\right) - \phi\left(P\right)}{t} - \phi'_P\left(h\right) \right\|_{\mathcal{R}} \to 0, \quad \text{cuando } t \to 0, \text{ para cualquier } h_t \to h.$$

La definición dada requiere que $\phi'_P : \mathcal{P}_\phi \mapsto \mathcal{R}$ exista como una aplicación sobre todo el espacio $\mathcal{R}$. Si este no es el caso, pero ϕ'_P existe en un subconjunto $\mathcal{R}_0$ y la sucesión $h_t \to h \in \mathcal{R}_0$, entonces ϕ se llama diferenciable Hadamard tangencialmente a este subconjunto.

Definición 2. Una clase $\mathcal{F}$ de funciones medibles $f : \mathcal{X} \mapsto \mathbb{R}$ se llama P-Donsker si la sucesión de procesos $\{\mathbb{G}_n f = \sqrt{n}(\mathbb{P}_n f - Pf) : f \in \mathcal{F}\}$, donde $\mathbb{P}_n f = n^{-1}\sum_{i=1}^n f(X_i)$ y $Pf = \int f dP$, converge en el espacio $\ell^\infty(\mathcal{F})$, el cual denota el conjunto de las funciones reales uniformemente acotadas en $\mathcal{F}$. El proceso límite es un proceso Gaussiano $\mathbb{G}_P$ con media cero y covarianza $\mathrm{E}(\mathbb{G}_P f \mathbb{G}_P g) = Pfg - PfPg$.

El resultado siguiente demuestra que la estimación proporcionada por el BLB para aproximar el valor poblacional $\xi(Q_n(P))$ es consistente. Para ello se exigen suposiciones estándar que se cumplen en la mayoría de los casos prácticos de interés y se tiene en cuenta que $m, n \to \infty$. No obstante, por lo general, se considera que m es una función de n que crece lentamente.

Teorema 1. *Supongamos que $\hat{\theta}_n = \phi(\mathbb{P}_n)$ y que $\theta = \phi(P)$, donde ϕ es diferenciable Hadamard en P tangencialmente a algún subespacio, con P, $\mathbb{P}_n$ y $\mathbb{P}_{n,m}^{(j)}$ vistas como aplicaciones de alguna clase Donsker $\mathcal{F}$ en $\mathbb{R}$ de manera que $\mathcal{F}_\delta$ sea medible para cada $\delta > 0$, donde $\mathcal{F}_\delta = \{f - g : f, g \in \mathcal{F}, \rho_P(f - g) < \delta\}$ y $\rho_P(f) = \sqrt{P(f - Pf)^2}$. Además, asumamos que $\xi(Q_n(P))$ es una función de la distribución de $u(\mathbb{P}_n, P) = \sqrt{n}(\phi(\mathbb{P}_n) - \phi(P))$ que es continua en el espacio de tales distribuciones con respecto a la métrica de la convergencia débil. Entonces,*

$$s^{-1}\sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) - \xi\left(Q_n(P)\right) \xrightarrow{P} 0$$

cuando $n \to \infty$, para cualquier sucesión $m \to \infty$ y para cualquier s fijo.

A continuación, vamos a caracterizar el orden de precisión del método BLB, es decir, la tasa de convergencia a $\xi(Q_n(P))$. Numerosos trabajos previos han demostrado que la estimación $\xi(Q_n(\mathbb{P}_n))$ del Bootstrap converge con una tasa que va como $O_P(1/n)$ o más rápido, suponiendo para ello que ξ se puede representar asintóticamente como una expansión en serie en potencias de $1/\sqrt{n}$ (ver, por ejemplo, [11]). Se va a ver que este grado de corrección coincide con el del BLB bajo estas suposiciones estándar siempre y cuando m y s sean elegidos con un valor suficientemente grande.

Teorema 2. *Supongamos que $\xi(Q_n(P))$ admite (asintóticamente) una expansión en serie como*

$$\xi\left(Q_n(P)\right) = z + \frac{p_1}{n^{1/2}} + \cdots + \frac{p_k}{n^{k/2}} + o\left(\frac{1}{n^{k/2}}\right),$$

donde z es una constante independiente de P y los p_k son polinomios en los momentos de P. Además, consideremos que la versión empírica de $\xi(Q_n(P))$ para cualquier j admite una expansión similar

$$\xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) = z + \frac{\hat{p}_1^{(j)}}{n^{1/2}} + \cdots + \frac{\hat{p}_k^{(j)}}{n^{k/2}} + o_P\left(\frac{1}{n^{k/2}}\right),$$

donde z es como se definió anteriormente y los $\hat{p}_k^{(j)}$ son polinomios en los momentos de $\mathbb{P}_{n,m}^{(j)}$ obtenidos al reemplazar los momentos de P en los p_k con los de $\mathbb{P}_{n,m}^{(j)}$. Entonces, suponiendo que $m \leq n$ y que $E(\hat{p}_k^{(1)})^2 < \infty$ para $k \in \{1, 2\}$, se tiene que

$$\left| s^{-1} \sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) - \xi\left(Q_n(P)\right) \right| = \sum_{k=1}^{2} O_P\left(\frac{\sqrt{\mathrm{Var}\left(\hat{p}_k^{(1)} - p_k | \mathbb{P}_n\right)}}{n^{k/2}\sqrt{s}} \right) + O_P\left(\frac{1}{n}\right) + O\left(\frac{1}{m\sqrt{n}}\right).$$

Por lo tanto, tomando $s = \Omega(\max\{n\,\mathrm{Var}(\hat{p}_1^{(1)} - p_1|\mathbb{P}_n), \mathrm{Var}(\hat{p}_2^{(1)} - p_2|\mathbb{P}_n)\})$ y $m = \Omega(\sqrt{n})$ obtenemos

$$\left| s^{-1} \sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) - \xi\left(Q_n(P)\right) \right| = O_P\left(\frac{1}{n}\right),$$

consiguiendo en este caso para el BLB el mismo nivel de corrección de orden superior que para el Bootstrap.

Comentamos brevemente que la simbología introducida para la elección de s y m corresponde a la notación Omega Grande, incluida dentro de la famalia de notaciones Landau y que viene a referirse a una cota inferior asintótica, es decir, a partir de un cierto punto tanto s como m crecen al menos como la expresión que se encuentra dentro de Ω. Sin embargo, valores considerablemente grandes de m pueden seguir siendo significatiamente menores que n con la condición $m/n \to 0$ cuando $n \to \infty$. Además, como $\mathrm{Var}(\hat{p}_k^{(1)} - p_k|\mathbb{P}_n)$ disminuye en probabilidad a medida que m y n aumentan, s también puede crecer más lento que n. Cabe mencionar también que, aunque la distribución empírica $\mathbb{P}_{n,m}^{(j)}$ esté construida a partir de m datos distintos, la expansión de $\xi(Q_n(\mathbb{P}_{n,m}^{(j)}))$ es en potencias de $1/\sqrt{n}$ y no de $1/\sqrt{m}$ pues $Q_n(\mathbb{P}_{n,m}^{(j)})$ es la distribución de los valores de u, los cuales se calculan sobre n puntos pese a que estos sean muestreados de $\mathbb{P}_{n,m}^{(j)}$. El hecho de que en realidad únicamente estemos "trabajando" con m datos se recoge por medio de los $\hat{p}_k^{(j)}$ ya que sí son polinomios en los momentos de esta $\mathbb{P}_{n,m}^{(j)}$.

Por último, damos dos resultados que nos ofrecen, respectivamente, un límite superior general a esta tasa de disminución de la varianza condicionada del párrafo anterior y un teorema equivalente al previo que se aplica sobre la variante alternativa del BLB que trabaja con subconjuntos disjuntos de los datos originales.

Observación 1. *Suponiendo que $E(\hat{p}_k^{(1)})^4 < \infty$, entonces* $\mathrm{Var}(\hat{p}_k^{(1)} - p_k|\mathbb{P}_n) = O_P(1/\sqrt{n}) + O(1/m)$.

Teorema 3. *Bajo las hipótesis del Teorema 2, y suponiendo que el BLB usa subconjuntos aleatorios disjuntos de los datos observados (en lugar de submuestras aleatorias simples), tenemos que*

$$\left| s^{-1} \sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) - \xi\left(Q_n(P)\right) \right| = O_P\left(\frac{1}{\sqrt{nms}}\right) + O\left(\frac{1}{m\sqrt{n}}\right).$$

Por lo tanto, si $s \sim n/m$ *y* $m = \Omega(\sqrt{n})$*, entonces*

$$\left| s^{-1} \sum_{j=1}^{s} \xi\left(Q_n\left(\mathbb{P}_{n,m}^{(j)}\right)\right) - \xi\left(Q_n(P)\right) \right| = O_P\left(\frac{1}{n}\right),$$

consiguiendo en este caso para el BLB el mismo nivel de corrección de orden superior que para el Bootstrap.

Para los resultados de esta sección hemos dicho que se establecen suposiciones estándar que se cumplen en la amplia mayoría de casos prácticos de interés. Sin ir más lejos, los estimadores de máxima verosimilitud o los de estimación por mínimos cuadrados son diferenciable Hadamard (ver [25]). Las hipótesis que se encuentran en los teoremas sobre ξ también se cumplen cuando se trata de un cuantil o un valor de una función de distribución (ver [11]). Por tanto, lo visto es válido para modelos como el de regresión lineal o logística que en concreto son con los que trabajaremos a continuación.

2.6. Comparación de modelos

En esta sección mostramos un resumen del trabajo realizado en los apartados 4, 5 y 6 del artículo *A scalable bootstrap for massive data* [15] donde sus autores comparan el rendimiento estadístico que proporcionan los distintos métodos vistos a lo largo del capítulo.

En primer lugar, el análisis empieza llevando a cabo un estudio de simulación sobre un único procesador. Se comienza viendo supuestos en los que los datos son simulados para así poder conocer las distribuciones P y $Q_n(P)$ y, en consecuencia, $\xi(Q_n(P))$. Esto es necesario ya que sino no se podría evaluar las corrección estadística de los distintos modelos.

Se considera un problema de regresión y uno de clasificación y, en ambos casos, los datos tienen la forma $X_i = (\tilde{X}_i, Y_i) \sim P$, i.i.d. para $i = 1, \ldots, n$, con $\tilde{X}_i \in \mathbb{R}^d$; $Y_i \in \mathbb{R}$ en la regresión e $Y_i \in \{0, 1\}$ en la clasificación. En cuanto a $\hat{\theta}_n$ y ξ, el primero estima un vector de parámetros en $\mathbb{R}^d$ relacionando $\tilde{X}_i$ con Y_i y el segundo se define como un procedimiento que calcula un conjunto de intervalos de confianza marginales al 95 % con los percentiles 2.5 y 97.5 de cada una de las componentes de la distribución $Q_n(P)$.

Se generan 2000 realizaciones muestrales del conjunto de datos, se estima $\hat{\theta}_n$ en cada una de ellas y, con su totalidad, se construye una aproximación fidedigna de $Q_n(P)$ con la que determinar el verdadero $\xi(Q_n(P))$. Luego, para otra realización muestral independiente, se calcula la estimación de $\xi(Q_n(P))$ de cada uno de los métodos, registrando sus valores en cada una de las iteraciones, así como el tiempo de procesamiento acumulado. Para valorar la evaluación de calidad del estimador (i.e., la ξ) de los distintos procedimientos se promedia en cinco de estas realizaciones muestrales (independientes entre sí, por supuesto) el promedio sobre todas las componentes del error relativo de la anchura de los intervalos de confianza estimados, el cual se calcula como $|c - c_0|/c_0$, donde c_0 y c hacen referencia, respectivamente, a la anchura verdadera y estimada del intervalo de confianza de una componente de $\xi(Q_n(P))$. Los tiempos de procesado se promedian siguiendo el mismo esquema de modo que se puede hacer una representación del error relativo frente al tiempo de procesamiento para los distintos procedimientos. Todo esto se implementa y ejecuta en MATLAB con un único procesador considerando, además, $r = 100$ y $m = n^\gamma$, $\gamma \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ para los diferentes experimentos.

A continuación, se describen los conjuntos de datos que se generan para los problemas de regresión y de clasificación. En el primero, la distribución subyacente P corresponde a un modelo lineal $Y_i = \tilde{X}_i^T \mathbf{1}_d + \epsilon_i$, con $n = 20000$ y $d = 100$, en el que $\tilde{X}_{i,j} \sim t$-Student(3) i.i.d. para $j = 1, \ldots, d$ y donde $\epsilon_i \sim$ Normal$(0, 10)$. En cuanto al segundo, se parte también de un modelo lineal $Y_i \sim$ Bernoulli$((1 + \exp(-\tilde{X}_i^T \mathbf{1}_d))^{-1})$ manteniendo n y la distribución de $\tilde{X}_i$ pero ahora con $d = 10$. El estimador $\hat{\theta}_n$ consiste, respectivamebte, en una regresión lineal por mínimos cuadrados y en una regresión logística ajustada por el método de Newton–Raphson para los dos problemas anteriores. A ambos tipos de regresión (la segunda la trataremos más en profundidad en el Capítulo 3) se les añade una pequeña penalización tipo L_2 en el vector de parámetros, con un peso de 10^{-5}, para mejorar la estabilidad numérica.

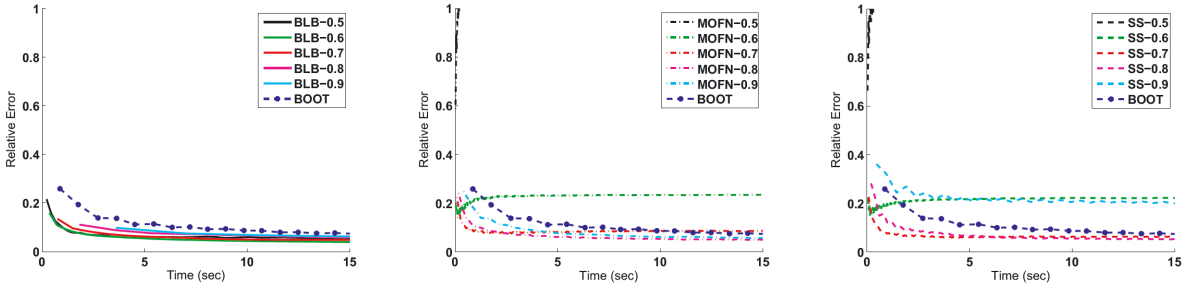

Figura 2.1: Error relativo vs. tiempo de procesamiento para el problema de regresión. La imagen izquierda compara el Bootstrap (BOOT) con el BLB, la central con el m out of n Bootstrap (MOFN), y la de la derecha con el Submuestreo (SS). Para BLB, MOFN y SS, se tiene $m = n^\gamma$ con el valor de γ para cada trayectoria dado en la leyenda. Los gráficos son los correspondiente a las letras (c), (f), (i) de la Fig. 1 de [15].

En la Figura 2.1 se muestran los resultados que se obtienen para el problema de regresión. El Bootstrap se utiliza como modelo de referencia para comparar el resto de métodos. Se observa que el BLB converge a un error relativo bajo más rápido que el

Bootstrap para todos los valores de $m = n^\gamma$ que se consideran, sin embargo, esto no sucede para el caso del m out of n Bootstrap, pues, como puede apreciarse en el gráfico central, su convergencia es deficiente para $m \leq n^{0.6}$. Por su parte, el rendimiento del Submuestreo es aún peor, convergiendo a un error relativo pobre incluso para $m = n^{0.9}$. Se menciona que valores bastante modestos de s son suficientes para conseguir estos resultados pues, como s indica el número de submuestras utilizadas, dichos valores guardan relación con el tiempo de procesamiento por lo que se encuentran implícitos en el eje de abscisas de las representaciones. En estos experimentos, el rango va desde 1-2 para $m = n^{0.9}$ hasta 10-14 para $m = n^{0.5}$, es decir, menores valores de s requieren mayores valores de m, de acuerdo con nuestra intuición y con lo que hemos visto en el Teorema 3.

La Figura 2.2 presenta ahora los resultados del problema de clasificación pero sin recoger los correspondientes al Submuestreo ya que, como antes, son estrictamente peores que los del m out of n Bootstrap. El primer gráfico muestra que el BLB continua convergiendo más rápido a un error relativo bajo, comparable o incluso mejor al del Bootstrap, excepto para el caso $m = n^{0.5}$, donde se mantiene siempre superior. Del mismo modo al caso del problema de regresión lineal, el rendimiento del m out of n Bootstrap sigue siendo pobre cuando $m \leq n^{0.6}$. Lo dicho en el párrafo previo sobre s permanece vigente pero el rango ahora va desde 1-2 para $m = n^{0.9}$ hasta 10-20 para $m \leq n^{0.6}$.

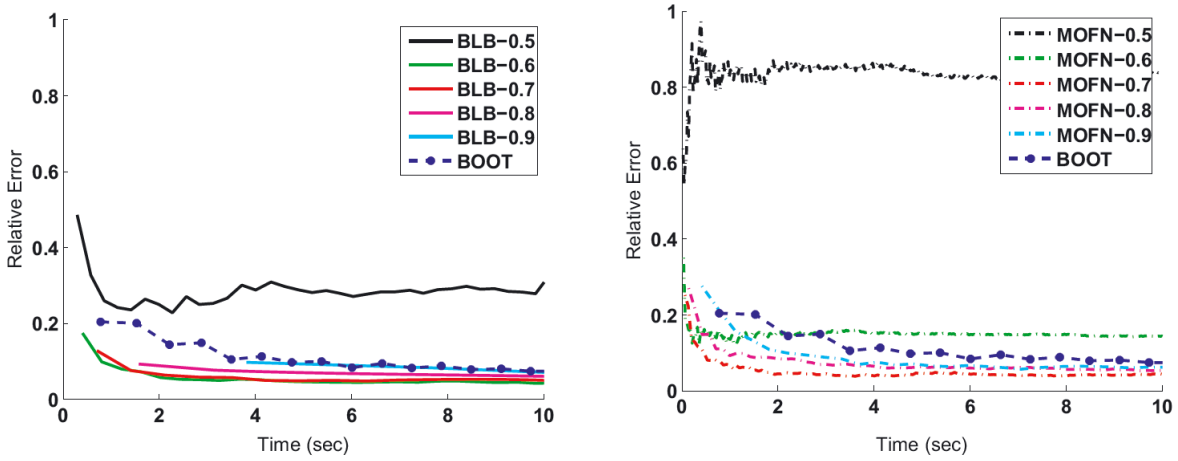

Figura 2.2: Error relativo vs. tiempo de procesamiento para el problema de clasificación. La imagen izquierda compara el Bootstrap (BOOT) con el BLB, y la de la derecha con el m out of n Bootstrap (MOFN). Para BLB y MOFN, se tiene $m = n^\gamma$ con el valor de γ para cada trayectoria dado en la leyenda. Los gráficos son los correspondiente a las letras (c), (f) de la Fig. 2 de [15].

Para profundizar un poco más en los casos en los que tanto el BLB como el m out of n Bootstrap convergen a errores relativos que no son comparables con el del Bootstrap para el problema de clasificación se expone un estudio del error relativo (tras la finalización del los algoritmos) en función de diversos valores de n cuando m es pequeño. La Figura 2.3 muestra que los errores relativos del BLB y del m out of n

Bootstrap son superiores al del Bootstrap para los valores más bajos de m pero que ambos decrecen hacia el de este último conforme n aumenta, manteniéndose inferior el del primero con respecto al del segundo. Esta disminución con el incremento de n se explica con los resultados teóricos de la sección anterior.

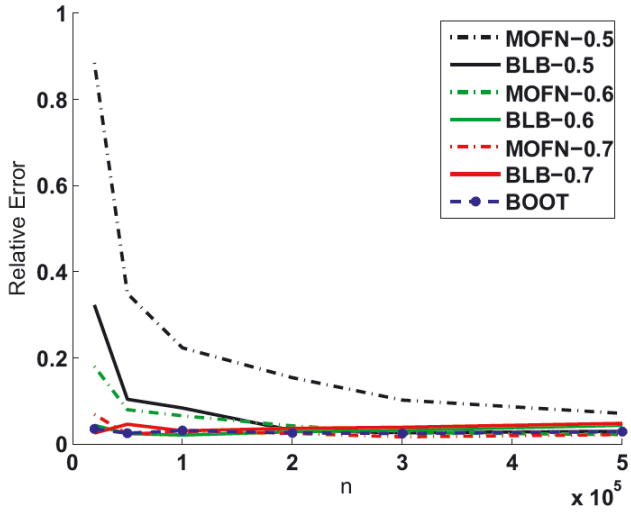

Figura 2.3: Error relativo (tras la convergencia) vs. n para Bootstrap (BOOT), BLB y m out of n Bootstrap (MOFN) en el problema de clasificación. Para BLB y MOFN, se tiene $m = n^\gamma$ con el valor de γ para cada trayectoria dado en la leyenda. El gráfico es el correspondiente a la letra (b) de la Fig. 3 de [15].

Una vez finalizado el estudio de simulación, se vislumbra que el BLB es el procedimiento con menor coste pues, por comparativa, es el que consigue un nivel de precisión óptimo en el menor tiempo, lo cual es señal de su capacidad de escalado a conjuntos de datos masivos (cuyos inconvenientes ya los tratamos con anterioridad en este documento).

Para confirman lo anterior, los autores se centran en el rendimiento computacional pero no sobre un solo procesador. En este caso, el análisis se hace mediante la implementación de los algoritmos en una plataforma de computación paralela y distribuida. Se comparan ahora el BLB y el Bootstrap para ver cual es el que tiene mejor capacidad de manejo de estos recursos informáticos de manera efectiva. El estudio de la escabilidad con el m out of n Bootstrap y con el Submuestreo no se contempla debido a los inconvenientes que presentan, ya vistos en esta sección, y a sus deficiencias con respecto al BLB.

La ejecución a escala del Bootstrap en este tipo de plataformas requiere implementar el estimador de un modo particular. En primer lugar, el conjunto de datos original se divide en un clúster de nodos de cómputo. A continuación, se paraleliza a lo largo de este clúster el cálculo, a la misma vez, de la estimación en cada remuestreo, y se calculan secuencialmente las siguientes muestras. Pese a que la viabilidad de este planteamiento parece factible, continúa teniendo algunos problemas como que, por ejemplo, para el cálculo del estimador, es necesaria una comunicación intranodo repetida con

datos intermedios lo que se traduce en un sobrecoste computacional. Además, hay que tener en cuenta que muchos sistemas de computación en clúster, almacenan datos solo en el disco, en lugar de en la memoria. Como dijimos en la introducción, las lecturas del disco son órdenes de magnitud más lentas que las lecturas de la memoria, luego el Bootstrap, al leer repetidamente un conjunto de datos muy grande en cada estimación, incurre también es un costo extremo.

En cuanto al BLB, por cómo es el método en sí mismo, su implementación en una arquitectura paralela es directa. Las distintas submuestras se distribuyen en diferentes nodos de cómputo y, a su vez, se utiliza el paralelismo intranodo para calcular los remuestreos que se generan a partir de cada uno de estos subconjuntos de datos. Como para generar y distribuir las submuestras únicamente se requiere una lectura de los datos al completo, el acceso al disco se produce una sola vez, y como el tamaño de las submuestras es relativamente pequeño, sus datos se almacenan en memoria para cuando sean requiridos, de forma que se alcanzan grandes ganancias computacionales permitiendo una escalabilidad efectiva.

Concretando para el experimento a gran escala, su ejecución la llevan a cabo en la plataforma de computación en nube Amazon EC2, implementándose en el lenguaje de programación Scala y utilizando el framework de computación en clúster de Spark. El estudio empírico se realizó únicamente para el problema de clasificación bajo las mismas especificaciones del inicio de esta sección pero con las siguientes modificaciones para adaptarse a esta escala mayor y al cálculo distribuido. Se generan 200 realizaciones independientes del conjunto de datos de tamaño n (en lugar de 2000) para calcular la verdadera distribución $Q_n(P)$, se selecciona $n = 6000000$ y $d = 3000$, de forma que los datos tienen un peso aproximado de 150 GB, y el último cambio es un factor de normalización en la distribución de $Y_i \sim \text{Bernoulli}((1+\exp(-\tilde{X}_i^T \mathbf{1}_d/\sqrt{d}))^{-1})$ previendo posibles dificultades a causa del alto valor de d.

Los resultados se muestran en la Figura 2.4. El gráfico de la izquierza corresponde a un almacenamiento del conjunto de datos al completo en el disco pues se dispone de un clúster de 10 nodos de trabajo cada uno con 6 GB de memoria y ocho núcleos de cómputo. Por su parte, en el de la derecha se muestran los resultados obtenidos con un clúster cuya memoria total es de 240 GB (20 nodos de trabajo cada uno con 4 núcleos), luego los datos se almacenan en caché en la memoria. En ambos casos, el BLB (en el que se ha usado $r = 50$, $s = 5$ y $m = n^{0.7}$) proporciona una salida de gran precisión mucho más rápido que el Bootstrap, que tarda un tiempo considerablemente mayor en alcanzar un error relativo comparable. Ambos métodos mejoran su rendimiento en el experimento de la derecha. Cabe mencionar que mientras el Bootstrap ofrece resultados conforme se van procesando los remuestreos, el BLB muestra un solo tiempo de procesamiento y error relativo debido a que está completamente paralelizado entre todas las submuestras.

Finaliza así el estudio del escalado computacional y de la comparativa general de la puede extraerse claramente el mejor desempeño del BLB con respecto al resto de métodos.

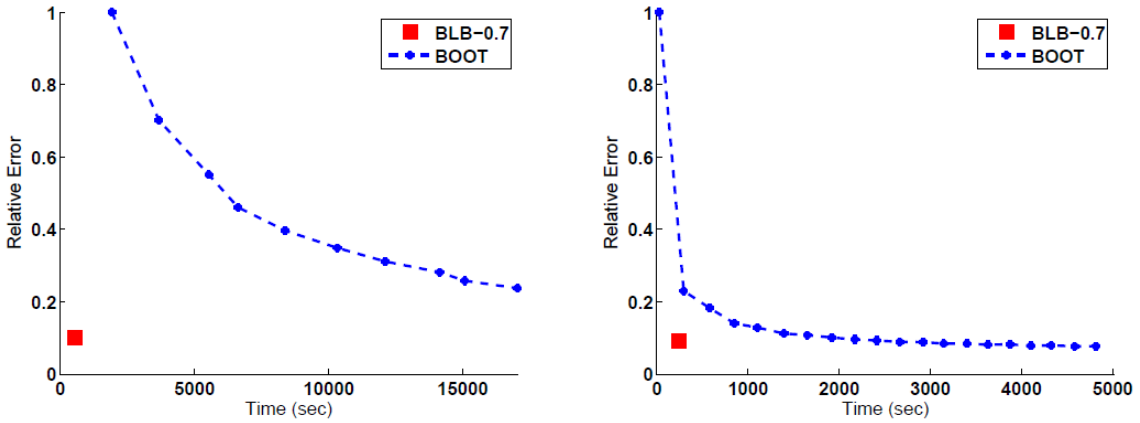

Figura 2.4: Error relativo vs. tiempo de procesamiento con datos de 150 GB en el problema de clasificación para Bootstrap (BOOT) y BLB (con $m = n^{0.7}$). La imagen izquierda muestra resultados con los datos guardados en el disco y la imagen derecha los muestra con los datos almacenados en caché en la memoria. Los gráficos corresponden con la Fig. 4 de [15].

Por último, la selección de los parámetros de ajuste que controlan el número de submuestras y remuestras así como el tamaño de las primeras es muy importante para alcanzar un buen rendimiento estadístico y computacional en el método BLB. Los autores de [15] sugieren $m = n^{0.7}$ como una opción razonable y efectiva en multitud de ocasiones y concluyen también que $r \geq 50$ y $s \geq 3$ son suficientes para conseguir un error relativo bajo. No obstante, proponen además un sistema adaptativo y automatizado para la determinación de r y s evitando así una elección conservadora e ineficientemente grande de sus valores. Esto se hace con el Algoritmo 5. Para seleccionar r de forma adaptativa en el bucle interno del BLB (ver Algoritmo 4) se procesan muestras continuadamente y se va actualizando $\xi^*_{n,j}$ hasta que deja de fluctuar significativamente. Para el caso de s, se procede igual, procesando submuestras pero ahora hasta que $s^{-1}\sum_{j=1}^{s}\xi^*_{n,j}$ se estabiliza.

Input: Sucesión de vectores $z^{(1)}, \ldots, z^{(t)} \in \mathbb{R}^d$; $w \in \mathbb{N}$, tamaño de la ventana $(< t)$; $\epsilon \in \mathbb{R}$, error relativo objetivo (> 0)
Output: `TRUE` si y solo si se considera que la sucesión de entrada ha dejado de fluctuar más allá del error relativo objetivo

if $\forall j \in [1, w], \frac{1}{d}\sum_{i=1}^{d}\frac{|z_i^{(t-j)} - z_i^{(t)}|}{|z_i^{(t)}|} \leq \epsilon$ **then**
  return *true*
else
  return *false*
end

Algoritmo 5: Selección de Parámetros

Capítulo 3

Aplicación del método BLB

En este capítulo mostramos un ejemplo de aplicación sencilla del método BLB llevado a cabo en R, un entorno de software libre para computación estadística y gráficos. Se dan unas nociones teóricas sencillas sobre regresión logística y una breve introducción al paquete `datadr` antes de acometer la aplicación propiamente dicha en la última sección del capítulo.

3.1. Regresión Logística

La regresión logística es un modelo que trata de analizar la capacidad de explicación de una serie de variables predictoras, que pueden ser cualitativas y/o cuantitativas (discretas y/o continuas), sobre una variable objetivo dicotómica (binaria). Se busca encontrar una relación entre la probabilidad de "éxito" de esta variable de respuesta y el conjunto de variables regresoras.

El modelo de regresión logísitica es un caso particular del modelo lineal generalizado (GLM, del inglés *Generalized Linear Model*), el cual se refiere a una amplia clase de modelos donde se generaliza la regresión lineal ordinaria de manera flexible permitiendo una variable de respuesta cuya distribución no tiene por qué ser una normal y permitiendo también que la relación entre esta variable y una combinación lineal de predictores sea a través de una función arbitraria y, por tanto, no necesariamente lineal. El GLM consigue unificar varios modelos estadísticos, incluidos los ya mencionados modelos de regresión lineal y logística pero también otros como la regresión de Poisson o modelos log-lineales. Para profundizar más sobre este tema consultar [20] donde Nelder y Wedderbur lo introducen o el libro del primer autor junto con McCullagh [18], aquí lo trataremos brevemente por no ser el objetivo del trabajo.

En primer lugar, por constituir un modelo de regresión, el GLM consta siempre de dos componentes:

- La componente aleatoria. Se refiere a una variable de respuesta o dependiente Y cuya distribución de probabilidad se encuentra dentro de la familia exponencial, o de modelos de dispersión exponencial, una gran clase distribuciones con muchas propiedades que las hacen extremadamente útiles para el análisis estadístico y que incluyen la normal, la binomial, la de Poisson o la gamma, entre otras. En concreto, nuestra función de densidad o de probabilidad, dependiendo de si la componente aleatoria es una variable continua o discreta, escrita en forma canónica es del tipo

$$f_Y(y;\theta,\phi) = a(y,\phi)\exp\left(\frac{y\theta - b(\theta)}{\phi}\right)$$

  donde θ es el parámetro natural, ϕ el parámetro de escala, $a(y,\phi)$ una función de normalización, que asegura que efectivamente tenemos una distribución, y $b(\theta)$ la función acumulativa, que es conocida. Tras unos breves cálculos es posible llegar a $E(Y) = \mu = b'(\theta)$ y $\text{var}(Y) = \phi\, b''(\theta)$. Habitualmente se denota $V(\mu) = b''(\theta)$ a la varianza cuando el factor de escala es la unidad y se denomina función de varianza.

- La componente sistemática. Corresponde a la cantidad que incorpora la información sobre las variables explicativas o independientes del modelo $X = (X_1, \ldots, X_p)^T$. Se trata de un predictor lineal con una serie de parámetros beta desconocidos que se denota por

$$\eta = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

Posteriormente, la relación entre ambas componentes se establece mediante una función link o vínculo, en particular, nos dice cómo es el enlace entre el predictor lineal y el valor esperado de la respuesta $\eta = g(\mu)$. Cuando esta función link es tal que $\eta = \theta = g(\mu)$ se denomina canónica. Si la relación se establece a la inversa, es decir, $\mu = h(\eta) = g^{-1}(\eta)$, lo cual siempre es posible por consideraciones que no trataremos de la función link, entonces $h(\cdot)$ se llama función respuesta.

Finalmente, para estimar los parámetros desconocidos del modelo, el vector $\beta \in \mathbb{R}^{p+1}$, se lleva a cabo una estimación de máxima verosimilitud, es decir, se escoge como estimador de β al argumento que maximiza la función verosimilitud. Recordemos que maximizar esta función es equivalente a maximizar el logaritmo de la misma. Dicha estimación es posible alcanzarla mediante el uso de algoritmos iterativos como el método de mínimos cuadrados reponderados iterativamente (IRLS, del inglés *Iteratively Reweighted Least Squares*).

Una vez conocidos los aspectos más importantes del GLM, pasamos a particularizar para el caso del modelo de regresión logística. Dado un vector aleatorio $X \in \mathbb{R}^p$, se desea predecir su relación con una variable dependiente dicotómica $Y \in \{0,1\}$. Considerando que X toma el valor x, se establece la hipótesis distribucional de que Y sigue una Bernoulli de parámetro su media μ y lo representamos por $Y|_{X=x} \sim \text{Bernoulli}(\mu(x))$. Con lo visto anteriormente, en el GLM la componente aleatoria corresponde a una distribución Bernoulli(μ) en la que el parámetro de escala es igual a la unidad ($\phi = 1$) y

la función cumulativa es $b(\theta) = \ln(1+e^\theta)$. De aquí obtenemos la media, que es más usual denotarla en este caso por $\pi = e^\theta/(1+e^\theta) = \mu$, y la varianza, $\text{var}(Y) = e^\theta/(1+e^\theta)^2 = \pi(1-\pi)$. Se escoge una transformación logit de π como función link canónica $g(\pi) = \ln(\pi/(1-\pi))$ de forma que la función respuesta también queda completamente definida $h(\eta) = e^\eta/(1+e^\eta)$. Esto completa la hipótesis estructural del modelo concluyendo finalmente que

$$\ln\left(\frac{\pi(x)}{1-\pi(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = \beta^T \underline{x}$$

donde hemos introducido la notación $\underline{x}^T = (1, x^T) \in \mathbb{R}^{1+p}$.

En cuanto a la estimación de parámetros, como hemos dicho usamos máxima verosimilitud, en este caso usando la probabilidad de Y condicionada a X. Supongamos que tenemos una muestra aleatoria $(y_i, x_i^T) \in \mathbb{R}^{1+p}$, $i = 1, \ldots, n$. Dado que $P[Y = y_i | X = x_i] = \pi(x_i)^{y_i} [1 - \pi(x_i)]^{1-y_i}$, entonces la función de verosimilitud y la log-verosimilitud vienen dadas, respectivamente, por

$$L(\beta; Y, X) = \prod_{i=1}^{n} \pi(x_i)^{y_i} [1 - \pi(x_i)]^{1-y_i}$$

$$\begin{aligned}\ell(\beta; Y, X) &= \sum_{i=1}^{n} [y_i \ln(\pi(x_i)) + (1 - y_i) \ln(1 - \pi(x_i))] \\ &= \sum_{i=1}^{n} \left[y_i \beta^T \underline{x}_i - \ln\left(1 + \exp\left(\beta^T \underline{x}_i\right)\right)\right]\end{aligned}$$

Para maximizar el logaritmo de la verosimilitud, se procede como es habitual, igualando el gradiente a cero. En este caso al gradiente se le suele llamar función score, s. Introducimos también la notación $\mathbf{X}$ para la matriz de datos (dimensión $n \times (1+p)$) que forman los $\underline{x}_i$, $\mathbf{y}$ para el vector con los datos y_i y $\boldsymbol{\pi}$ para el vector con las probabilidades $\pi(x_i)$. Así la ecuación se ecribe:

$$s(\beta) = \frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{n} \underline{x}_i \left(y_i - \frac{1 + \exp\left(\beta^T \underline{x}_i\right)}{\exp\left(\beta^T \underline{x}_i\right)}\right) = \mathbf{X}^T (\mathbf{y} - \boldsymbol{\pi}) = 0$$

Dado que no es posible su resolución analítica, deben utilizarse técnicas numéricas. Se puede probar que la función ℓ es convexa, luego la ecuación $s(\beta) = 0$ tiene un única solución. Aplicando el método de Newton–Raphson, se puede alcanzar la estimación iterativamente a partir de un estimador inicial $\hat{\beta}^{(0)}$ de acuerdo con

$$\hat{\beta}^{(k+1)} = \hat{\beta}^{(k)} + F^{-1}(\hat{\beta}^{(k)}) s(\hat{\beta}^{(k)})$$

La $F(\beta)$ es la matriz de Fisher que corresponde con la opuesta de la matriz Hessiana y cuya expresión es

$$F(\beta) = -\frac{\partial^2 \ell}{\partial \beta \partial \beta^T} = \sum_{i=1}^{n} \underline{x}_i \underline{x}_i^T \pi(x_i)(1 - \pi(x_i)) = \mathbf{X}^T \mathbf{W} \mathbf{X}$$

donde $\mathbf{W}$ es la matriz diagonal formada con los pesos $w_i = \pi(x_i)(1-\pi(x_i))$, $i = 1, \ldots, n$.

Las iteraciones del procedimiento de Newton-Raphson suelen reescribirse por conveniencia en la forma

$$\hat{\beta}^{(k+1)} = \left(\mathbf{X}^T\hat{\mathbf{W}}\mathbf{X}\right)^{-1}\mathbf{X}^T\hat{\mathbf{W}}\hat{\mathbf{z}}$$

consiguiendo así un paso de mínimos cuadrados ponderados sobre la variable $\hat{\mathbf{z}} = \mathbf{X}\hat{\beta}^{(k)} + \hat{\mathbf{W}}^{-1}(\mathbf{y} - \hat{\boldsymbol{\pi}})$. Se resuelve repetidamente, ya que, en cada iteración, $\hat{\boldsymbol{\pi}}$ cambia al actualizarse $\hat{\beta}$ y, por lo tanto, también lo hacen $\hat{\mathbf{W}}$ y $\hat{\mathbf{z}}$. Este método es el IRLS, por resolver el problema de los mínimos cuadrados ponderados:

$$\arg\min_{\beta}(\mathbf{z} - \mathbf{X}\beta)^T\mathbf{W}(\mathbf{z} - \mathbf{X}\beta)$$

Una vez obtenido el estimador $\hat{\beta}$, se calculan las probabilidades $\hat{\pi}(x_i) = \exp(\hat{\beta}^T\underline{x}_i)/(1+\exp(\hat{\beta}^T\underline{x}_i))$. Además, gracias a las propiedades asintóticas del estimador es posible conseguir intervalos de confianza para los coeficientes. Para cada $i \in \{0, 1, \ldots, p\}$, $\hat{\beta}_i \pm z_{1-\alpha/2} \times \widehat{SE}(\hat{\beta}_i)$ es un intervalo de confianza con un nivel de confianza asintótico $1-\alpha$ para β_i, siendo $\widehat{SE}(\hat{\beta}_i)$ la raíz cuadrada del elemento diagonal $i+1$ de $F^{-1}(\hat{\beta})$.

Pasamos ahora a dar una interpretación a los parámetros beta. Para ello es necesario definir el concepto *odds*. Dada una realización muestral x, la variable $Y|_{X=x}$ tiene una probabilidad $\pi(x)$ de ocurrir y una probabilidad $1-\pi(x)$ de no ocurrir. El cociente de estas probabilidades $\pi(x)/(1-\pi(x))$ se conoce como $\text{odds}(x) = \exp(\beta^T\underline{x})$ y representa la ventaja de la presencia (frente a la ausencia) de Y en individuos con $X = x$. A su vez, se define la *odds ratio* o razón de ventajas de una variable X_i como el cociente de las odds cuando dicha variable se incrementa una unidad $\text{or}(x_i) = \text{odds}(x_i+1)/\,\text{odds}(x_i)$ y representa cuanto más verosímil (o inverosímil) es la presencia (o ausencia) de Y en individuos con $X_i = x_i + 1$ que en individuos con $X_i = x_i$. De las expresiones

$$\text{odds}(0) = e^{\beta_0}$$

$$\text{or}(x_i) = \frac{\text{odds}(x_i+1)}{\text{odds}(x_i)} = e^{\beta_i}$$

se deduce que β_0 no es más que el log-odds cuando todas las variables explicativas son cero y β_i corresponde con el log-odds ratio de la variable X_i.

Por último, mencionamos la manera de proceder en función de la variable explicativa que tengamos. Sea X_i una de las variables explicativas X. Si es numérica (incluye las binarias), se trata del caso más sencillo pues se considera el valor de la variable en la observación muestral. Sin embargo, si no lo es y tenemos una variable cualitativa o categórica con $N \geq 2$ categorías, debemos construir $N-1$ variables binarias ficticias, denominadas *dummy*. La forma de hacerlo se ilustra con la siguiente matriz

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}_{(N-1)\times N}$$

donde las filas corresponden a las $N-1$ variables ficticias dummy y las columnas a las N categorías de la variable original X_i.

Completamos así esta revisión del modelo de regresión logística del que haremos uso en la Sección 3.3.

3.2. D&R con datadr

El paquete `datadr` [10] es una de las vías de implementación del dividir y recombinar (D&R, *Divide and Recombine* en inglés) en el entorno de programación estadística R. Se trata de una componente del proyecto de código abierto DeltaRho (`http://deltarho.org/`) cuyo objetivo es suministrar métodos y herramientas que permitan un análisis profundo de datos complejos de gran tamaño. D&R persigue proporcionar a los analistas de datos una mayor facilidad y flexibilidad en sus análisis estadísiticos dividiendo estos datos en subconjuntos de tamaño significativamente menor, aplicando métodos analíticos a los mismos y recombinando los resultados. La Figura 3.1 ilustra visualmente una representación de este proceso de D&R.

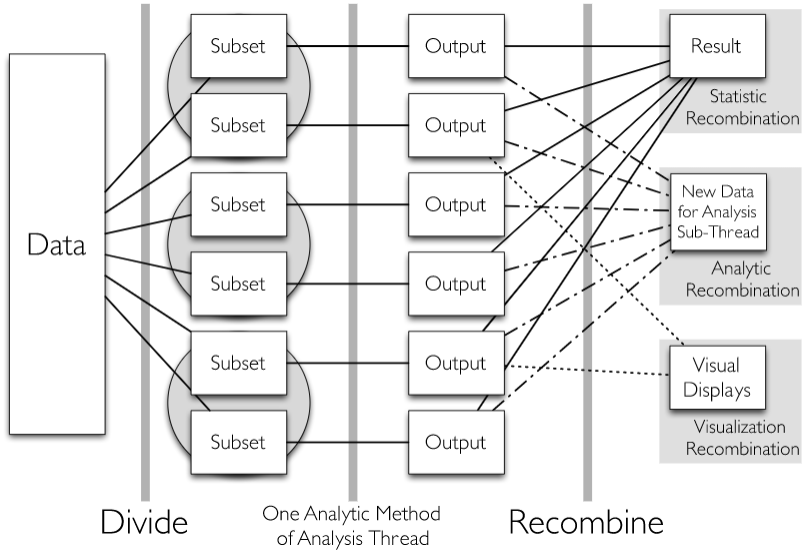

Figura 3.1: Diagrama esquemático del proceso D&R.

A continuación, presentamos un breve resumen del paquete comentando los tipos de datos más importantes con los que se trabaja, así como las principales funciones, entre las que se incluyen las que posteriormente utilizaremos en la siguiente sección. Para profundizar sobre esto se recomienda consultar el tutorial `http://deltarho.org/docs-datadr/` y la ayuda propia con la que cuenta R.

Comenzamos por los tipos de datos, los dos más importantes son el marco de datos distribuidos (ddf, del inglés *distributed data frame*) y el objeto de datos distribuidos

(ddo, del inglés *distributed data object*). El primero no es más que la división de un data frame en fragmentos, siendo cada fragmento un subconjunto de filas de este, que pueden localizarse en los nodos de un clúster. En cuanto al segundo, la noción es similar pero cada subconjunto, a su vez, puede ser una estructura arbitraria, luego los ddf se encontrarían dentro de los ddo. Se utiliza una estructura de pares clave-valor para almacenar ambos tipos de datos, es decir, un ddf/ddo es esencialmente una lista donde cada elemento de la misma es un par clave-valor, la clave sirve para etiquetar un subconjunto identificándolo unívocamente y el valor es este subconjunto propiamente dicho. El almacenamiento de los datos se realiza en la memoria, en un sistema de archivos estándar (por ejemplo, un disco duro) o en el HDFS (Hadoop Distributed File System). Con la primera forma no se requiere una conexión a un servidor (backend), sin embargo, conjuntos de datos que exceden la capacidad de la memoria sí necesitan una conexión backend para ser almacenados en disco o en HDFS. Independientemente del backend, la interfaz siempre permanece igual pero no se puede obviar el tamaño del conjunto de datos, como hemos dichos la memoria es ideal para datos pequeños, los de tamaño mediano son perfectos para el disco local y se recomienda el HDFS para BigData, cuando el volumen de datos ya es muy grande. Para este último caso es necesaria la utilización de RHIPE (R and Hadoop Integrated Programming Environment) que, como su propio nombre indica, es el motor que nos permite el uso de Hadoop totalmente desde R.

En lo referente a operaciones de datos, las dos principales funciones son `divide` y `recombine`. La primera fracciona un ddo/ddf ya sea por variables condicionantes o bien por subconjuntos elegidos al azar y la segunda toma los resultados de haber aplicado un cálculo a un ddo/ddf y los combina de diferentes maneras. Ambas integran el lenguaje de alto nivel que proporciona `datadr` para D&R y son suficientes para la mayoría de las operaciones que un usuario estándar lleva a cabo. En este paquete, a diferencia de otros marcos informáticos distrbuidos, no existe una noción predominante de evaluación diferida. La única excepción es la función `addTransform` que se encarga de aplicar transformaciones a cada uno de los subconjuntos de un ddo/ddf y que destaca por no aplicarlas inmediatamente, sino que las difiere hasta que se realiza alguna operación de datos.

El lenguaje de nivel inferior para realizar tares de D&R es MapReduce. El término en sí hace referencia a un modelo de programación para procesar y generar grandes conjuntos de datos con un algoritmo paralelizado y distribuido en un clúster, así como a su implementación asociada. Se compone de un procedimiento map, que realiza el filtrado y la clasificación, y un método de reducción (reduce), que realiza una operación de resumen. El sistema se organiza estableciendo el orden de los servidores distribuidos, ejecutando las diversas tareas en paralelo y administrando todas las comunicaciones y transferencias de datos entre las diversas partes de dicho sistema. Se trata de una especialización de la estrategia dividir-aplicar-combinar para el análisis de datos en la que las contribuciones clave son la escalabilidad y la tolerancia a fallos lograda para una gran variedad de aplicaciones mediante la optimización del motor de ejecución.

3.3. BLB con R

En primer lugar, el conjunto de datos con el que trabajaremos será el conocido como *Census Income* de *UCI Machine Learning Repository* [5]. La información que contiene fue extraída por Barry Becker de la base de datos de 1994 de la Oficina del Censo de los EE. UU. La recopilación de dicha información perseguía predecir si las ganancias de un individuo excedían los 50000 dólares anuales basándose en datos censales como la edad, la ocupación, la educación (formación / nivel de estudios), el estado civil, la raza, el sexo, las horas trabajadas por semana o el país de nacimiento entre otros. Dentro de R, este conjunto de datos se encuentra en el paquete `datadr` bajo el nombre `adult`.

El objetivo es enfrentarnos a un problema de clasificación comparando las estimaciones de un modelo de regresión logística global no distribuido con las correspondientes que resultan de aplicar un procedimiento BLB, comprobando el buen desempeño de este segundo procedimiento y pretendiendo verificar su extrapolación para conjuntos de volumen desorbitado, en los que el primer método carece de viabilidad.

El código completo se encuentra en el Apéndice. Nos disponemos aquí a realizar un comentario explicativo del mismo siguiendo el orden conforme al cual tendría que correrse por lo que es recomendable ojearlo conforme se avanza en la lectura. No obstante, presentamos aquí los resultados principales.

Tras instalar los paquetes necesarios y cargar los datos y su correspondiente librería procedemos a realizar una partición de estos. Dividimos los datos de manera aleatoria de forma que el 75 % de los mismos constituye el conjunto de entrenamiento, sobre el cual construiremos los modelos, y el 25 % restante conforma el conjunto test, que nos servirá, como su propio nombre indica, para testear la validez de nuestras predicciones. Debemos hacer notar que, como es habitual, hemos especificado (al azar) una semilla para la generación de números aleatorios en `R` de forma que produzcamos siempre el mismo fraccionamiento de datos para realizar nuestros cálculos y tengamos así idénticas condiciones de trabajo que alguien que estuviera interesado en reproducirlos y comprobarlos.

Cabe mencionar que en nuestro ejemplo los datos correspondientes a los ingresos están claramente desbalanceados. Tan solo un 24 % de los individuos gana al año más de 50k $, frente al 76 % restante, algo lógico si tenemos en cuenta que se trata de una renta anual considerable a la que el grueso de la población no suele llegar. Hablaremos sobre esto más adelante.

```
> # Proporción de 0s y 1s
> prop.table(table(adultEnt$income))

    <=50K      >50K
0.7607699 0.2392301
```

Procedemos a continuación a aplicar el modelo de regresión logística sobre la totalidad del conjunto de entrenamiento. La variable objetivo (binaria, por supuesto) es

`incomebin` que toma el valor 0 si los ingresos de una persona son iguales o inferiores a 50k $, y vale 1 en caso contrario. Queremos analizarla basándonos únicamente en los datos correspondientes a la formación de esa persona, al número de horas que trabaja por semana y a su sexo. De acuerdo con la clasificación del censo estadounidense, tenemos el nivel de estudios dividido en 16 categorías, desde preescolar (mínimo) hasta doctorado (máximo). La variable numérica `educationnum` será la que usemos por asignar un valor del 1 al 16 a dichas categorías en orden ascendente de dificultad. Las otras variables son `hoursperweek`, que también es numérica, y `sex`, que es dicotómica entre `Male` y `Female`. Para construir el modelo recurrimos a la función `glm` que ajusta modelos lineales generalizados. Especificando mediante una descripción simbólica la fórmula con la variable de respuesta y el predictor lineal, y seleccionando la distribución del error, en nuestro caso la familia binomial, se consigue la regresión logística.

```
> rglm

Call:  glm(formula = incomebin ~ educationnum + hoursperweek + sex,
    family = binomial(), data = adultEnt)

Coefficients:
 (Intercept)  educationnum  hoursperweek       sexMale
    -7.19077       0.35449       0.03271       1.20889

Degrees of Freedom: 24419 Total (i.e. Null);  24416 Residual
Null Deviance:      26870
Residual Deviance: 22100        AIC: 22110
> oddsratio
educationnum hoursperweek      sexMale
    1.425457     1.033253     3.349755
```

La interpretación de los parámetros estimados es interesante. Recordemos del final de la Sección 3.1 que $\hat{\beta}_0 = -7{,}191$ corresponde con el logaritmo neperiano de las posibilidades de tener ingresos superiores a 50k $ cuando las variables explicativas valen cero. En este caso las posibilidades son aproximadamente de 1 entre 1250 para una mujer desempleada sin nivel de estudios. El resto de parámetros equivalen a los log-odds ratio de las distintas variables. Estos nos dicen, por ejemplo, que un hombre tiene 3.35 veces más posibilidades que una mujer de tener unas ganancias anuales superiores a los 50000 dólares y que ascender un nivel en el criterio de clasificación educativo trae consigo un crecimiento del 43 % de las opciones de estar por encima de ese umbral de 50k $. En cuanto al odds ratio de las horas trabajadas por semana, el coeficiente no es significativo pues invertir una hora más de trabajo solo supone un aumento del 3 % de las posibilidades.

Seguidamente, comenzamos a sumergirnos dentro del paquete `datadr` propiamente dicho. Como hemos visto en la sección anterior, esta librería se maneja con unos tipos de datos concretos por lo que debemos convertir nuestro conjunto de entrenamiento. Haciendo uso de la comando `ddf` inicializamos un distributed data frame. Seleccionamos también la opción `update` para buscar información adicional, atributos que es posible que le falten al ddf y que es útil tenerlos para posibles cálculos posteriores. Por ejemplo, se detectan valores perdidos o se posibilita la obtención de medidas descriptivas sobre

el conjunto de datos completo. Es la manera habitual de proceder y trae consigo una ejecución automática de MapReduce para actualizar dichos atributos. Con la impresión de `adultDdf`, se observa que la primera línea de la salida nos indica que la estructura con los pares clave-valor se sustenta en la memoria, lo cual era de esperar por estar trabajando con un ejemplo sencillo en el que el volumen de datos es relativamente pequeño. No obstante, si se usaran los otros backends para almacenar y procesar conjuntos de datos más grandes, la interfaz siempre permanecería igual, lo cual es clave.

```
> adultDdf = ddf(adultEnt, update = TRUE)
* Running map/reduce to get missing attributes...
> adultDdf

Distributed data frame backed by 'kvMemory' connection

 attribute      | value
----------------+-------------------------------------------------------
 names          | age(int), workclass(fac), fnlwgt(int), and 13 more
 nrow           | 24420
 size (stored)  | 1.69 MB
 size (object)  | 1.69 MB
 # subsets      | 1

* Other attributes: getKeys(), splitSizeDistn(), splitRowDistn(), summary()
```

A continuación, procedemos a realizar la división de los datos con la función `divide`. El criterio que seguimos para dividirlos es un criterio alatorio, la elección de filas es arbitraria y, por tanto, también la de los subconjuntos. Para ello usamos la función `rrDiv` con la que así mismo podemos especificar el número aproximado de sus filas. Además, aplicamos una transformación a los datos posterior a esta división pero previa al escritura en memoria, eliminamos columnas de los subconjuntos para quedarnos únicamente con las que nos interesan. En la Figura 3.2 presentamos un gráfico del tamaño del subconjunto frente al índice como una comprobación visual de que efectivamente la dimensión de los 24 subconjuntos formados es aproximadamente la especificada.

```
> rrAdult = divide(adultDdf, by = rrDiv(1000), update = TRUE,
+                  postTransFn = function(x)
+                    x[,c("incomebin", "educationnum",
+                         "hoursperweek", "sex")])
* Verifying parameters...
* Applying division...
* Running map/reduce to get missing attributes...
> rrAdult

Distributed data frame backed by 'kvMemory' connection

 attribute      | value
----------------+-------------------------------------------------------
 names          | incomebin(num), educationnum(int), and 2 more
 nrow           | 24420
 size (stored)  | 605.48 KB
 size (object)  | 605.48 KB
 # subsets      | 24
```

```
* Other attributes: getKeys(), splitSizeDistn(), splitRowDistn(), summary()
* Approx. number of rows in each division: 1000
```

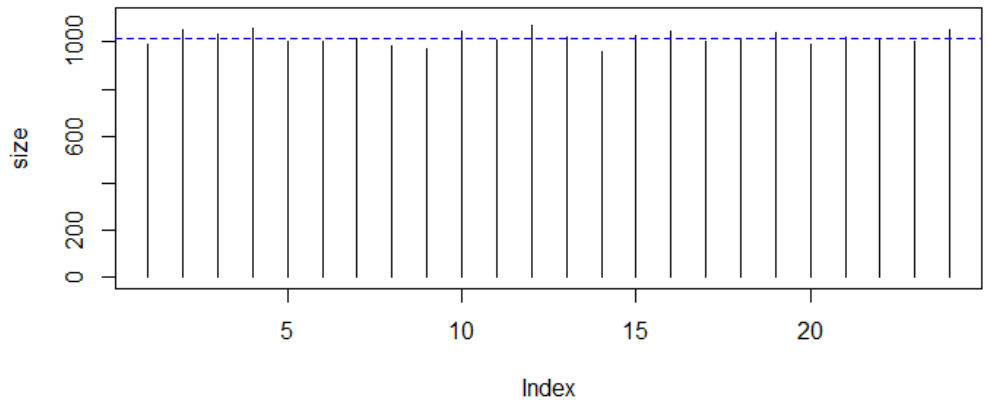

Figura 3.2: Representación del tamaño del subconjunto frente al índice. La línea azul indica el tamaño medio de los subconjuntos

El trabajo realizado hasta ahora con los datos del conjunto de entrenamiento (conversión a ddf y división del mismo) crea las condiciones necesarias para que el método BLB pueda ser empleado. Para ello, hacemos uso de la función `addTransform`, explicada en la sección anterior, para aplicar `drBLB` a cada uno de los subconjuntos aleatorios formados. `drBLB` es una función propia del paquete `datadr`, construida basándose en el Algoritmo 4 y que nos ayuda a implementar directamente el método. Hay que proporcionarle el estadístico ($\hat{\theta}$) que se calcula sobre las muestras bootstraps que debían tener un tamaño igual al número de observaciones totales. En este caso vamos a usar el estimador que ofrece `glm` pero recordemos que sea cual sea el elegido debe poder trabajar con representaciones ponderadas (aquí se especifican con el comando `weights`). Suministramos también a la función, además del número de muestras bootstraps (iteraciones de Monte Carlo, escogemos 50 por ser la recomendación de los autores de [15] que dijimos en la Sección 2.6), la métrica o medida (ξ) para la evalución de la calidad del estimador. Buscamos un intervalo de confianza al 95 % para los coeficientes de cada subconjunto por lo que utilizamos `quantile` (método percetil).

```
> adultBlb = addTransform(rrAdult, function(x)
+   {drBLB(x,
+          statistic = function(x, weights)
+            coef(glm(incomebin ~ educationnum + hoursperweek + sex,
+                     data = x, weights = weights, family = binomial())),
+          metric = function(T) c(mean(T),
+                                 quantile(T, c(0.025,0.975))),
+          R = 50,
+          n = nrow(rrAdult))})
*** finding global variables used in 'fn'...
  found: rrAdult
  package dependencies: datadr, stats
*** testing 'fn' on a subset... ok
```

```
> adultBlb

Transformed distributed data object backed by 'kvMemory' connection

 attribute      | value
----------------+------------------------------------------------------------
 size (stored)  | 605.48 KB (before transformation)
 size (object)  | 605.48 KB (before transformation)
 # subsets      | 24

* Other attributes: getKeys()
* Approx. number of rows in each division: 1000
```

Haciendo uso de `combMean` conseguimos una recombinación media tanto de los coeficientes como de los intervalos de confianza que nos proporciona los siguientes resultados:

```
> ICBLB
                CoefsBLB     ICBLB_L     ICBLB_U
(Intercept)  -7.25533333 -7.46607888 -7.05867092
educationnum  0.35730052  0.34373959  0.37154403
hoursperweek  0.03314598  0.03033059  0.03600495
sexMale       1.21978668  1.14106476  1.29939622
```

Para poner en contexto las estimaciones BLB debemos tener en cuenta los coeficientes correspondientes que hemos obtenido cuando hemos llevado a cabo una estimación global no distribuida con `glm`. Mostramos a continuación una tabla resumen con los resultados donde se añaden también los intervalos de confianza asintóticos para los coeficientes del modelo de regresión logística global cuya expresión habíamos mencionado en la Sección 3.1. Por comparación vemos que los resultados son razonablemente buenos, es más, inspirándonos en el criterio del error relativo visto en la Sección 2.6 podemos obtener, por un lado, que el error relativo medio al cotejar los coeficientes BLB con los coeficientes glm ($|\hat{\beta}_i^{glm} - \hat{\beta}_i^{BLB}|/|\hat{\beta}_i^{glm}|$) es inferior al 1 % y, por otro, que la anchura relativa media de los intervalos de confianza BLB estimados con respecto a sus coeficientes no llega al 11 %.

```
> round(Resumen,3)
             Coefsglm ICglm_L ICglm_U CoefsBLB ICBLB_L ICBLB_U
(Intercept)    -7.191  -7.406  -6.976   -7.255  -7.466  -7.059
educationnum    0.354   0.340   0.369    0.357   0.344   0.372
hoursperweek    0.033   0.030   0.036    0.033   0.030   0.036
sexMale         1.209   1.126   1.292    1.220   1.141   1.299
> rel = c(RelErrorCoefs = mean(abs((Coefsglm-CoefsBLB)/Coefsglm)),
+         RelWidthIC = mean(abs(widthBLB/CoefsBLB)))
> rel
RelErrorCoefs    RelWidthIC
  0.009796494   0.108741573
```

Tras confirmar, como cabía esperar, la gran similitud entre el modelo global y el modelo distribuido pasamos a testear las estimaciones. Lo primero que hacemos es definir una función, que dadas las variables explicativas y los coeficientes, nos calcule la probabilidad de tener unas ganancias anuales por encima de los 50000 dólares. Una

vez hecho esto, aplicamos esta función sobre el conjunto test tanto para la estimación global como para la estimación BLB y creamos las respectivas tablas de contingencia (Tablas 3.1 y 3.2) con los valores reales y los predichos suponiendo que se considera como 0 aquel individuo cuya probabilidad calculada de ingresar menos de 50k $ por año sea menor o igual que 0.5, y como 1 en otro caso.

	0 predic	1 predic
0 real	5760	382
1 real	1360	639

Tabla 3.1: Estimación glm

	0 predic	1 predic
0 real	5760	382
1 real	1361	638

Tabla 3.2: Estimación BLB

Obviamente, la diferencia entre ambas tablas de contingencia es mínima pues las estimaciones proporcionadas por ambos modelos son muy próximas. En los dos casos, el porcentaje de ceros acertados es del 93.78 % y, en el de unos, la estimación global tiene un acierto más (31.97 %) que la estimación BLB (31.92 %). En total, los aciertos son del 78.60 % y 78.59 %, respectivamente. Con estos datos se puede considerar que el testeo es idéntico. Recordemos que nuestro objetivo es comprobar que el método BLB es una buena alternativa al método de regresión logística global no distribuido de implementación inviable cuando se trabaja con enormes conjuntos de datos de complejidad elevada. Sin embargo, esto no significa que en circunstancias sencillas como las de esta sección, en la que ambas estimaciones son viables, el procedimiento BLB deba ofrecernos resultados considerablemente mejores que los de su oponente. No cabe esperar que esto suceda.

Por ejemplo, supongamos que queremos mejorar el porcentaje de unos que acierta la estimación BLB. Para hacerlo, la estimación glm global también tendría que conseguirlo. Si recordamos de cuando particionamos los datos, el conjunto de entrenamiento tenía tres veces menos de individuos con ingresos anuales superiores a 50k $ que con ingresos inferiores a 50k $ (24 % frente a 76 %). Una posible vía para conseguir esta mejora sería balancear el conjunto de entrenamiento, de forma que los modelos se construyeran teniendo la misma proporción de ceros que de unos para que así tuvieran una mejor capacidad de estimación sobre este segundo grupo. Para lograr lo anterior hacemos uso de la librería `caret` [16] que cuenta con las funciones `downSample` y `upSample`.

```
> library(caret)
> #adultEnt = downSample(adult[ient, -(p-1)], adult$income[ient],
> #                      yname = "income")
> adultEnt = upSample(adult[ient, -(p-1)], adult$income[ient],
+                     yname = "income")
```

Si n y N definen los totales de casos en las clases minoritaria y mayoritaria en el conjunto de entrenamiento de un problema de clasificación binaria, por un lado, el conocido como método Downsampling aplica una técnica de muestreo en la clase mayoritaria generando un conjunto de datos balanceado de tamaño $2n$ formado por los n casos de la clase minoritaria y una selección aleatoria también de n casos pero de entre los N de la mayoritaria. Por otro lado, el Upsampling aplica una técnica

de remuestreo en la clase minoritaria generando un conjunto de datos balanceado de tamaño $2N$ en que el tenemos por una parte los N casos de la clase mayoritaria y por otra una muestra bootstrap de tamaño N extraída de los n casos de la clase minoritaria. Ambos procedimientos se implementan directamente en R con las respectivas funciones `downSample` y `upSample` que acabamos de nombrar. Procediendo como hasta ahora con este nuevo conjunto de entrenamiento, es decir, corriendo el código redefiniendo `adultEnt` para que "escriba" sobre el ya cargado, se llega a que ambas técnicas suben la proporción de unos clasificados con acierto a costa de perder porcentaje de acierto total y de ceros correctamente clasificados. Los valores concretos los recogemos en la Tabla 3.3.

	0s acertados	1s acertados	total aciertos
downsampling (glm)	0.7115	0.6728	0.7020
downsampling (BLB)	0.7126	0.6723	0.7027
upsampling (glm)	0.7068	0.6778	0.6997
upsampling (BLB)	0.7069	0.6778	0.6998

Tabla 3.3: Proporción de estimaciones acertadas con los datos balanceados.

Por último, dejando a un lado este pequeño paréntesis sobre datos balanceados y volviendo con el testeo de nuestra estimación BLB, vamos a visualizar su comportamiento como clasificador binario. Para ello realizamos la representación de la curva ROC (del inglés *Receiver Operating Characteristic*) en la Figura 3.3. Para conseguirlo recurrimos a las funciones `prediction` y `performance` del paquete `ROCR` [24] que nos permiten implementarla de manera sencilla. El área bajo la curva (AUC = 0.7629, del inglés *Area Under Curve*), que coincide con el estadístico Wilcoxon-Mann-Whitney, nos da una medida de la excelencia del clasificador, cuanto más próxima sea a 1 mejor será su desempeño, lo cual indica que nuestra estimación tiene margen de mejora.

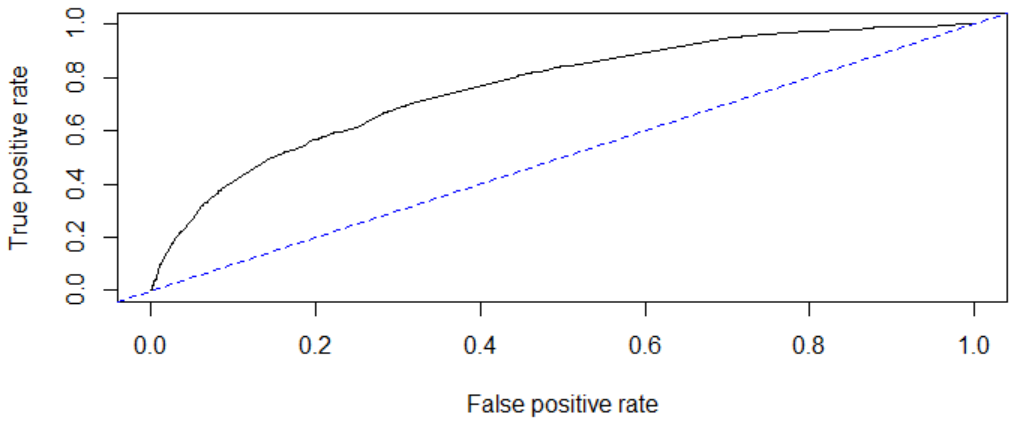

Figura 3.3: Razón de verdaderos positivos (tpr, true positive rate) frente a razón de falsos positivos (fpr, false positive rate). La línea negra corresponde a la curva ROC y línea azul indica tpr = fpr.

Conclusiones

Recapitulando lo visto a lo largo de este TFG podemos sacar en claro, de acuerdo con el Teorema 1 sobre consistencia del método y los Teoremas 2 y 3 sobre el grado de corrección y exactitud del mismo, que el procedimiento BLB comparte las fortalezas estadísticas del Bootstrap. Además, con la comparativa realizada en la Sección 2.6, extraemos un rendimiento superior tanto del BLB como del Bootstrap en detrimento del Submuestreo y el m out of n Bootstrap aun cuando se trabaja con un único procesador. Ahora bien, en el escalado a grandes conjuntos de datos se pone de manifiesto el fracaso del Bootstrap con respecto a la eficiencia computacional del BLB. Por último, rematamos en el Capítulo 3 estudiando la probabilidad de tener unos ingresos anuales por encima de 50000 dólares en función del sexo del individuo, su nivel de estudios y el número de horas que trabaja por semana. Comprobamos finalmente que los resultados estimados aplicando un procedimiento BLB coinciden con los obtenidos por un modelo de regresión logística global no distribuido.

Por tanto, se concluye en síntesis que el método BLB es un buen medio para analizar y obtener resultados de grandes conjuntos de datos y que cuenta con la capacidad de adaptarse fácilmente a las arquitecturas modernas de computación paralela y distribuida. No obstante, no debe verse tampoco como una solución definitiva que no presenta inconvenientes. Recordemos que sus requisitos de espacio de almacenamiento son de orden $O(m)$, por lo que para datos muy muy masivos, los subconjuntos con los que trabaja el BLB pueden ellos mismos ser demasiado grandes, de forma que la aplicación del método traiga consigo un coste computacional enorme. Dichas limitaciones hacen necesaria que la investigación en este ámbito continúe, buscando alternativas para estos problemas con conjuntos de datos muy masivos como podría ser un enfoque con 'Bag of Little m out of n Bootstraps'.

Apéndice

Recogemos aquí el código completo correspondiente a la Sección 3.3. Para trabajar con los datos balanceados simplemente hay que descomentar (en la parte de `PARTICIÓN ENTRENAMIENTO/TEST`) la carga de la librería `caret` y la técnica que queramos para implementar el balanceo de datos. Luego, correr el código de nuevo.

```
#####################
# PAQUETES NECESARIOS
# Paquete que implementa técnicas de división y recombinación
#install.packages("datadr")
# Paquete para el balanceo de datos
#install.packages("caret")
# Paquete para la curva COR
#install.packages("ROCR")


################
# CARGA DE DATOS
library(datadr)
data(adult)
help(adult)
summary(adult)


##############################
# PARTICIÓN ENTRENAMIENTO/TEST
set.seed(12345) # semilla
n = nrow(adult)
p = ncol(adult)
# Muestra del tamaño indicado del vector 1:n sin reemplazo
ient = sample(1:n, floor(0.75*n))
itest = setdiff(1:n, ient) # diferencia de conjuntos
adultEnt = adult[ient,] # conjunto de entrenamiento
#library(caret)
#adultEnt = downSample(adult[ient, -(p-1)], adult$income[ient],
#                      yname = "income") # Downsampling
#adultEnt = upSample(adult[ient, -(p-1)], adult$income[ient],
#                    yname = "income") #Upsampling
# Proporción de 0s y 1s
prop.table(table(adultEnt$income))
adultTest = adult[itest,] # conjunto test
```

```
#######################################
# MODELO DE REGRESIÓN LOGÍSTICA CON glm
rglm = glm(incomebin ~ educationnum + hoursperweek + sex,
           data = adultEnt, family = binomial())
rglm
summary(rglm)
Coefsglm = coef(rglm) # coeficientes del modelo
SEglm = summary(rglm)$coefficients[,2] # error estándar de los coefs
alfa = 0.05
# Coeficientes e intervalos de confianza asintóticos
ICglm = cbind(Coefsglm,
              ICglm_L = Coefsglm - qnorm(1-alfa/2)*SEglm,
              ICglm_U = Coefsglm + qnorm(1-alfa/2)*SEglm)
oddsratio = exp(Coefsglm[-1])
oddsratio

#############################################
# INICIALIZACIÓN DE UN DISTRIBUTED DATA FRAME
adultDdf = ddf(adultEnt, update = TRUE)
adultDdf
summary(adultDdf)

#####################################
# DIVISIÓN DEL DISTRIBUTED DATA FRAME
rrAdult = divide(adultDdf, by = rrDiv(1000), update = TRUE,
                 postTransFn = function(x)
                   x[,c("incomebin", "educationnum",
                        "hoursperweek", "sex")])
rrAdult
summary(rrAdult)
NSubconj = length(getKeys(rrAdult)) # número de subconjuntos
size = numeric(NSubconj)
# Bucle for para calcular el tamaño de cada subconjunto
for (i in 1:NSubconj)
  size[i] = nrow(rrAdult[[i]]$value)
# Representación del tamaño del subconjunto frente al índice
plot(size, type = "h", ylim = c(0,1100))
abline(h = mean(size), lty = 2, col = "blue")

##############################
# IMPLEMENTACIÓN DEL MÉTODO BLB
adultBlb = addTransform(rrAdult, function(x)
  {drBLB(x,
         statistic = function(x, weights)
           coef(glm(incomebin ~ educationnum + hoursperweek + sex,
                    data = x, weights = weights, family = binomial())),
         metric = function(T) c(mean(T),
                                quantile(T, c(0.025,0.975))),
         R = 50,
         n = nrow(rrAdult))})
adultBlb
# Media de los coeficientes y los intervalos de confianza
coefsBlb = recombine(adultBlb, combMean)
# Presentación de los resultados BLB
ICBLB = matrix(coefsBlb, ncol = 3, byrow = TRUE)
colnames(ICBLB) = c("CoefsBLB", "ICBLB_L", "ICBLB_U")
rownames(ICBLB) = c("(Intercept)", "educationnum", "hoursperweek", "sexMale")
```

```
ICBLB
widthBLB = ICBLB[,3] - ICBLB[,2] # anchura intervalos BLB
CoefsBLB = ICBLB[,1] # coeficientes BLB
# Resumen con las estimaciones glm y BLB
Resumen = cbind(ICglm, ICBLB)
round(Resumen,3)
# Error relativo de los coeficientes y anchura relativa de los IC BLB
rel = c(RelErrorCoefs = mean(abs((Coefsglm-CoefsBLB)/Coefsglm)),
        RelWidthIC = mean(abs(widthBLB/CoefsBLB)))
rel


############################
# TESTEO DE LAS ESTIMACIONES
# Función para calcular la probabilidad pi(x)
probrlog = function(x, coef)
{
  comblin = sum(coef*c(1,x))
  exp(comblin/(1+exp(comblin)))
}
# Datos a los que aplicar la función anterior
test = data.frame(adultTest[, c("educationnum", "hoursperweek")],
                  sex = adultTest$sex == "male")
head(test)
# Probabilidades estimadas con glm
probtestglm = apply(test, 1, probrlog, coef = Coefsglm)
# Probabilidades estimadas con BLB
probtestBLB = apply(test, 1, probrlog, coef = CoefsBLB)
# Tablas de contingencia
tablaglm = table(Real = adultTest$incomebin,
                 Predicglm = c(0,1)[(probtestglm>0.5)+1])
tablaglm
diag(prop.table(tablaglm, 1)) # proporción de 0s y 1s acertados con glm
sum(diag(prop.table(tablaglm))) # proporción total de aciertos con glm
tablaBLB = table(Real = adultTest$incomebin,
                 PredicBLB = c(0,1)[(probtestBLB>0.5)+1])
tablaBLB
diag(prop.table(tablaBLB, 1)) # proporción de 0s y 1s acertados con BLB
sum(diag(prop.table(tablaBLB))) # proporción total de aciertos con BLB
library(ROCR)
# Representación de la curva ROC
prediobj = prediction(probtestBLB, adultTest$incomebin)
plot(performance(prediobj, "tpr", "fpr"))
abline(a = 0, b = 1, col = "blue", lty = 2)
# Área bajo la curva
auc = as.numeric(performance(prediobj, "auc")@y.values)
cat("AUC test =", auc, "\n")
```

Referencias

[1] Bickel, P. J., and Freedman, D. A. Some Asymptotic Theory for the Bootstrap. *Ann. Statist. 9*, 6 (11 1981), 1196–1217.

[2] Bickel, P. J., Götze, F., and van Zwet, W. Resampling fewer than n observations: gains, losses, and remedies for losses. *Statist. Sinica 7* (1997), 1–32.

[3] Bickel, P. J., and Sakov, A. On the choice of m in the m out of n bootstrap and confidence bounds for extrema, 2008.

[4] Diebold, F. X. On the Origin(s) and Development of the Term 'Big Data'. *SSRN Electronic Journal* (09 2012).

[5] Dua, D., and Graff, C. UCI Machine Learning Repository. [http://archive.ics.uci.edu/ml], 2017. University of California, Irvine, School of Information and Computer Sciences.

[6] Efron, B. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics 7*, 1 (1979), 1–26.

[7] Efron, B., and Tibshirani, R. *An Introduction to the Bootstrap*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[8] Fan, W., and Bifet, A. Mining Big Data: Current Status, and Forecast to the Future. *SIGKDD Explor. Newsl. 14*, 2 (04 2013), 1–5.

[9] Guha, S., Hafen, R., Rounds, J., Xia, J., Li, J., Xi, B., and Cleveland, W. S. Large complex data: divide and recombine (D&R) with RHIPE. *Stat 1*, 1 (2012), 53–67.

[10] Hafen, R. datadr: Divide and recombine for large, complex data. [https://CRAN.R-project.org/package=datadr], 2018. R package version 0.8.6.1.

[11] Hall, P. *The Bootstrap and Edgeworth Expansion*. Springer Series in Statistics. Springer New York, 2013.

[12] Jordan, M. I. On statistics, computation and scalability. *Bernoulli 19*, 4 (09 2013), 1378–1390.

[13] Kalbandi, I., and Anuradha, J. A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. *Procedia Computer Science 48* (12 2015), 319–324.

[14] Kane, M., W. Emerson, J., and Weston, S. Scalable Strategies for Computing with Massive Data. *Journal of Statistical Software 55* (11 2013), 1–19.

[15] Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. I. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 76*, 4 (2014), 795–816.

[16] Kuhn, M. Contributions from Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., Candan, C., and Hunt, T. caret: Classification and Regression Training. [https://CRAN.R-project.org/package=caret], 2019. R package version 6.0-84.

[17] Laney, D. 3D Data Management: Controlling Data Volume, Velocity, and Variety. Tech. rep., META Group, February 2001.

[18] McCullagh, P., and Nelder, J. *Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1989.

[19] N. Politis, D., Romano, J., and Wolf, M. On the asymptotic theory of subsampling. *Statistica Sinica 11* (10 2001), 1105–1124.

[20] Nelder, J. A., and Wedderburn, R. W. M. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General) 135*, 3 (1972), 370–384.

[21] Politis, D., Romano, J., and Wolf., M. *Subsampling*. Springer, 1999.

[22] Politis, D. N., and Romano, J. P. The Stationary Bootstrap. *Journal of the American Statistical Association 89*, 428 (1994), 1303–1313.

[23] Schifano, E., Wu, J., Wang, C., Yan, J., and Chen, M.-H. Online Updating of Statistical Inference in the Big Data Setting. *Technometrics 58* (05 2015).

[24] Sing, T., Sander, O., Beerenwinkel, N., and Lengauer, T. ROCR: visualizing classifier performance in R. *Bioinformatics 21*, 20 (2005), 7881.

[25] van der Vaart, A. *Asymptotic Statistics*. Asymptotic Statistics. Cambridge University Press, 2000.

[26] Wang, C., Chen, M.-H., Schifano, E., Wu, J., and Yan, J. Statistical methods and computing for Big Data. *Statistics and Its Interface 9* (01 2016), 399–414.