_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Análisis del algoritmo MINI para imputación de valores perdidos en conjuntos de datos pequeños y con variables continuas y categóricas

Marzo de 2017

Trabajo de fin de grado de: Gerard Cubas Rovira

Dirigido por: Fernando González Ladrón de Guevara
Marta Fernández Diego

Agradecimientos

La elaboración de este proyecto ha sido una tarea no exenta de esfuerzos, sobre todo en la fase experimental. Me gustaría agradecer su apoyo a las siguientes personas: Primero, y sobre todo a mis tutores Marta y Fernando, por la bibliografía que me han proporcionado, por su valiosos consejos, por todas las ideas que me han aportado, por el código de pre-procesado que me facilitaron y por sus ánimos durante todo el proceso. En segundo lugar, también quiero agradecer a Alfonso el hecho de compartir conmigo su trabajo y dejarme utilizar y modificar su código libremente. Por último, me gustaría agradecer a mis padres, Mamen y Gerardo; y a mi novia, Inma por su apoyo incondicional y por darme fuerzas en todo momento.

Resumen

La estimación del esfuerzo es un tema decisivo para la gestión de proyectos de software. Para alcanzar este objetivo se utilizan bases de datos históricas. Uno de los principales problemas a la hora de estimar dicho esfuerzo es que las bases de datos de las que se dispone, suelen estar incompletas y muchas técnicas de estimación del esfuerzo no toleran esta situación. Una posible solución al problema es eliminar los casos con valores perdidos, pero el tamaño de la base de datos con la que se trabaja se verá reducido y en la mayoría de las ocasiones afectará negativamente al resultado, sobre todo cuando se disponga de un número pequeño de casos. La otra opción es la imputación, es decir, asignar valores plausibles en los casos en que se produce la ausencia de un dato o se desconoce el valor de una o diversas variables.

El objetivo de este proyecto es replicar el algoritmo MINI, un método propuesto por Song y Shepperd en el año 2007 y en la publicación «A New Imputation Method for Small Software Project Data Sets». MINI es una combinación del algoritmo kNN (*k Nearest Neighbours*) y CMI (*Class Mean Imputation*) que consta de tres partes: la selección de las variables relevantes, la clasificación de los datos, y la imputación de los mismos. Para comprobar su efectividad se simulan datos perdidos según los mecanismos MAR y MCAR, diferentes patrones, y distintos porcentajes. Posteriormente se comparan los resultados de imputar dichos valores con los de los conjuntos de datos originales tomando como referencia los resultados obtenidos para la misma simulación mediante los algoritmos CMI y kNN. Como trabajo adicional, se introduce una mejora del algoritmo CMI que consiste en la utilización de estimadores robustos.

Índice general

Capítulo 1

Introducción

1.1 Descripción del problema

En estadística, los valores perdidos, o los datos ausentes, se dan cuando no hay un valor almacenado en la variable de una observación. Esto es una situación frecuente. Por ejemplo, en campos como la economía, la sociología o las ciencias políticas puede darse el caso de bases de datos incompletas porque los gobiernos prefieren no informar de malas estadísticas. Otro ejemplo es el abandono de proyectos de larga duración, o el caso de las encuestas, en las que muchas veces, la gente no responde por desconocer la respuesta a la pregunta. En la publicación [2] abordan el problema de los datos perdidos, conocidos como *missing data* en la literatura inglesa.

El artículo «A new imputation method for small software project data sets» [2] fue publicado en 2007 por Song y Shepperd en la revista *Journal of Systems and Software*. En él se propone un algoritmo al que llaman MINI (*Mean Imputation based k Nearest neighbours hot-deck Imputation*) que consta de tres partes: la extracción de las variables relevantes, a las que llaman *Key Features*, la clasificación de los datos y la imputación de los valores perdidos.

La extracción de las variables relevantes, selección de variables relevantes o *Feature Subset Selection* (FSS) es un problema con el que tiene que lidiar quien maneja grandes cantidades de datos. Surge cuando se quiere modelar la relación entre una variable dependiente y un conjunto de variables independientes. Su objetivo es reducir tanto como sea posible un conjunto de características que permitan representar una información. Los beneficios de este proceso son dos, por una parte, se reduce el coste computacional de los algoritmos que trabajan con la base de datos, y por otro lado se evita el sobre-ajuste que puede producir la excedencia de variables, o la calidad de ciertas características que por algún u otro motivo no presentan ningún tipo de relación con la variable dependiente. En la investigación [2] proponen un algoritmo (algoritmo 1) similar a ID3 y C4.5. En la sección 2.1 se describe en más detalle la extracción de las variables relevantes.

La clasificación de los datos consiste en agruparlos calculando la distancia entre el caso a imputar y cada clase, tal como se explica en la subsección 3.2.4.

La imputación de los datos consiste en asignar valores plausibles en los campos de la base de datos que contienen valores perdidos. Se realiza aplicando el algoritmo kNN a cada valor utilizando única y exclusivamente como vecinos proyectos de la clase más cercana.

1.2 Ámbito

La imputación de datos perdidos es una parte fundamental tanto de la minería de datos (*data mining*) como del aprendizaje automático (*machine learning*). No es raro ver como se usan indistintamente ambos términos. Son conceptos estrechamente relacionados cuya principal diferencia es el objetivo perseguido. La minería de datos trata de descubrir patrones anteriormente desconocidos. Mientras que el *machine learning* se usa para reproducir patrones conocidos y hacer predicciones basadas en esos patrones.

Los algoritmos de *machine learning* se dividen principalmente en dos áreas [3], el aprendizaje supervisado y el aprendizaje no supervisado. Por una parte, el supervisado trata de hacer predicciones buscando patrones en conjuntos de datos relacionando todas las variables con una variable especial, llamada variable objetivo. Por ejemplo, saber si un día será apto para jugar a tenis o no, en función de la previsión meteorológica (lluvia, viento, temperatura). Por otro lado, el aprendizaje no supervisado usa datos históricos que no están etiquetados. El fin es explorarlos para encontrar alguna estructura o forma de organizarlos. Ejemplos de aprendizaje no supervisado son los algoritmos de agrupamiento, las redes neuronales artificiales o los algoritmos de expectación-maximización. Todos los métodos empleados en este proyecto a excepción de k-means (C4.5, kNN, CMI, MINI) son de aprendizaje supervisado.

Este trabajo se encuentra enmarcado en una serie de estudios y publicaciones cuyo objetivo final es estimar el esfuerzo. Como muchas de las técnicas estadísticas para realizar dicha estimación no toleran bases de datos incompletas, si no queremos eliminar los casos que contienen *missing data*, acción poco conveniente en conjuntos donde se disponen pocas muestras, la única alternativa que nos queda es imputar esos datos.

1.3 Objetivos

El principal objetivo de este trabajo es el de replicar el algoritmo 2 de la subsección 3.2.4, propuesto en [2]. Del mismo modo, analizar su comportamiento para la imputación de datos perdidos en bases de datos de 100 y 150 casos, imputando variables tanto nominales como continuas que no sean el esfuerzo.

El propósito es desarrollar métodos que sirvan para imputar bases de datos incompletas que más adelante se utilicen para estimar el esfuerzo mediante procedimientos que no toleren la presencia de datos ausentes. Para replicar el algoritmo habrá que determinar un método de clasificación óptimo ya que tanto CMI como MINI precisan de una variable de clase.

Para comprobar la bondad del algoritmo propuesto se comparará su efectividad en términos de MMRE y del error de imputación nominal con la de los algoritmos de imputación kNN y CMI para diferentes porcentajes, mecanismos y patrones de datos perdidos.

Como objetivo secundario, se plantea, buscar aquellas variables que son relevantes para la imputación del esfuerzo. Para la selección de dichas variables se utilizará un árbol de decisión que se forme teniendo en cuenta principalmente la ganancia de información y que sirva para la

clasificación de los proyectos según el valor de la variable esfuerzo. Para poder generar el árbol de decisión es necesario discretizar previamente la variable mencionada. Y se tendrá que elegir el método más adecuado y el número de intervalos óptimo para realizar dicha discretización.

1.4 Trabajo relacionado

Jonsson y Wohlin, en [4], realizan un estudio sobre una base de datos en escala Likert, un tipo de datos ordinales típico en encuestas con opiniones subjetivas, y comprueban la efectividad de kNN para dicha base de datos, que consta de 54 elementos. En su estudio concluyen que el valor óptimo de k para esta situación es la raíz cuadrada del número de casos completos redondeada a un valor impar, y que el método se sigue comportando bien para grandes cantidades de datos perdidos.

Song, Shepperd y Cartwright, en [5], prueban los algoritmos CMI y kNN en la base de datos del ISBSG para dos muestras aleatorias de 50 y 100 casos. En su estudio, llegan a la conclusión de que CMI se comporta ligeramente mejor que kNN independientemente del tipo de mecanismo de datos perdidos. Además, concluyen que el mecanismo de pérdida puede asumirse MCAR cuando se trata este tipo de casos. En este artículo recomiendan utilizar CMI para conjuntos de datos pequeños. Song y Shepperd, en [2], el artículo donde presentan MINI, comparan CMI, kNN y MINI para *datasets* de los mismos tamaños que en [5]. En este estudio, sin embargo, concluyen que kNN se comporta mejor que CMI y que el mejor método de imputación es MINI en términos de MMRE, error de imputación nominal y varianza del *dataset* completo.

Song, Shepperd, Chen y col., en [6], prueban el impacto de aplicar el algoritmo C4.5 en un contexto de predicción del esfuerzo en la base de datos del ISBSG para conjuntos de datos incompletos, y los mismos conjuntos de datos, en los cuales se han imputado los datos ausentes utilizando para ello el algoritmo kNN. Para llevar a cabo sus pruebas inducen datos perdidos según diferentes porcentajes, para diferentes patrones de *missing data* y distintos mecanismos y extraen dos conclusiones principales: La primera es que la imputación mediante kNN puede mejorar la precisión de C4.5; la segunda es que tanto kNN como C4.5 se ven afectados por el mecanismo, el patrón de la pérdida y por el porcentaje de datos perdidos.

González Ladrón de Guevara y Fernández Diego, en [7] y [8], realizan dos estudios sistemáticos de mapeo sobre la base de datos del ISBSG en uno evalúan sus potenciales y limitaciones, y en el otro las variables más utilizadas en investigaciones relacionadas con la estimación del esfuerzo. Para ello recurren a artículos y actas de congresos de IEEEXplore, ACM DIgital Library, ScienceDirect y WebofNoledge. En la subsección 3.1.1 hay una descripción más detallada de sus estudios y de las conclusiones a las que llegan.

Pan, en [9], compara el algoritmo kNN del paquete VIM [10] con métodos de imputación por media y por mediana para imputar valores que sirvan para posteriormente generar modelos regresivos en los que la variable dependiente sea el esfuerzo. Para llevar a cabo dicha tarea utiliza 3 subconjuntos de datos de la base de datos del ISBSG, de 30, 100 y 300 casos respectivamente. En estos subconjuntos genera datos perdidos según tres porcentajes diferentes y el mecanismo MCAR, los datos ausentes se distribuyen entre todas las variables. Posteriormente realiza la imputación de los 9 *datasets* generados mediante los métodos mencionados. Finalmente genera modelos de regresión lineal para cada conjunto de datos completo y para el *dataset* que contiene valores perdidos. En los modelos regresivos la variable dependiente es el esfuerzo y las variables independientes, las otras cuatro imputadas. A partir de los resultados, la autora concluye que el mejor modelo para grandes *datasets* es kNN, mientras que para los pequeños en ocasiones será

mejor no realizar imputación o llevarla a cabo utilizando el método de la mediana dependiendo del tamaño y del porcentaje de datos perdidos.

Serrano, en [11], programa un algoritmo kNN nuevo con tres mejoras adicionales sobre el kNN del paquete de R VIM [10]. La función kNN utilizada en este trabajo es la suya. Y la función de imputación según el algoritmo MINI (algoritmo 2), es una modificación de la misma. La primera mejora es el diseño de una función moda para imputar las variables nominales y donde se resuelven los empates en función de la distancia al caso a imputar, a diferencia de como se hace en la mayoría de paquetes de R, que es aleatoriamente. La segunda mejora es la capacidad de calcular la distancia entre variables nominales que contienen más de un valor, a las que llama variables multi-factor. Y la tercera es la posibilidad de ponderar el valor de la variable de los vecinos para la imputación en proporción a su distancia inversa. En este trabajo, la selección de variables relevantes se lleva a cabo utilizando un método basado en mRMR (*minimal Redundancy - Maximal Relevance*). Un algoritmo iterativo de aprendizaje supervisado en el que en cada paso se selecciona la variable que presenta la máxima ganancia de información con respecto a la variable a imputar, a la cual se le resta la media de la ganancia de información calculada con respecto a las variables ya seleccionadas. De esta forma se ordenan los atributos según un ranking y se van añadiendo variables a las ya seleccionadas como *Key Features*. Para ello calcula el MRE (medida de error descrita en la subsección 3.2.7) de ir generando uno a uno valores perdidos en la variable a imputar para todos los casos del *dataset* y calcula posteriormente la media de este valor. De esta forma, se van añadiendo variables a la lista de *Key Features* mientras el MMRE desciende, y se dejan de seleccionar características cuando el MMRE empieza a aumentar.

1.5 Descripción de las partes del documento

El capítulo 2 es una introducción teórica al problema abordado en este proyecto, este capítulo se divide a su vez en tres secciones. La sección 2.1 trata del proceso de selección de variables relevantes, para ello se introducen algunos conceptos de teoría de la información y de los árboles de decisión, poniendo ambos conceptos en relación con la selección de características clave. La sección 2.2 consiste en una introducción a la teoría de valores perdidos en la que se habla de los diferentes mecanismos y patrones de datos ausentes, y las distintas técnicas que se utilizan para afrontar esta situación. Finalmente se describe con más detalle los algoritmos kNN y CMI. En la sección 2.3 se expone el problema de los *outliers* o valores atípicos, su detección y de algunas posibles soluciones a este recurrente problema en el tratamiento de datos. En el capítulo 3 se plantea la metodología empleada. En la sección 3.1 se describe la base de datos utilizada en el trabajo, se exponen los motivos por los que se escoge R como lenguaje de programación para llevar a cabo todo el proyecto, y se hace una breve presentación del lenguaje. La sección 3.2 es una descripción de los procedimientos realizados. Y finalmente en los capítulos 4 y 5 se exponen los resultados obtenidos en el experimento, se extraen conclusiones a partir de los mismos, se habla de las limitaciones de los procedimientos utilizados y del algoritmo MINI y se proponen algunas ideas sobre las que trabajar en el futuro.

Capítulo 2

Estado del arte

2.1 Selección de variables relevantes

Los algoritmos de *machine learning* obtienen cierto conocimiento a partir de información inteligible para los ordenadores, este conocimiento puede utilizarse para imputar datos como es nuestro caso. Desafortunadamente, su éxito normalmente depende de la calidad de los datos sobre los que se opera. Si el *dataset* con el que se trabaja contiene información irrelevante, los algoritmos de *machine learning* pueden producir resultados menos precisos y comprensibles, o incluso hacer que las predicciones sean completamente erróneas [12].

El proceso de selección de variables relevantes, también conocido como FSS, de sus siglas en inglés, (*Feature Subset Selection*) es el proceso de identificar y eliminar tanta información irrelevante y redundante como sea posible. FSS es muy utilizado en disciplinas donde se manejan grandes cantidades de datos (*Big Data*) como la minería de datos, reconocimiento de patrones, predicciones financieras o inteligencia artificial. Otro de sus beneficios, es que al disminuir las variables con las que trabajamos, estamos reduciendo el tamaño de nuestro *dataset* y esto se traducirá en un menor coste computacional a la hora de implementar los algoritmos.

Si inicialmente hay n posibles características, entonces existen n^2 combinaciones de subconjuntos posibles. La única forma de encontrar el más adecuado sería probarlos todos. No obstante, esto no es tan buena idea como puede parecer inicialmente debido a que posteriormente habría que imputar los valores perdidos para todos los casos y habría que calcular los errores también para todos los subconjuntos utilizados. No es que sea inviable realizar esta operación, pero sí muy costosa computacionalmente.

2.1.1 FSS basada en teoría de la información

Entropía y ganancia de información

La teoría de la información, también conocida como teoría matemática de la información, es una propuesta planteada por Claude E. Shannon y Warren Weave a finales de la década de 1940. En sus orígenes fue aplicada con objetivos de codificación de canal y criptografía. Pronto se extendió su ámbito de aplicación a otros campos como el procesamiento de señales, el tratamiento digital de imágenes, la minería de datos, el aprendizaje automático o la inteligencia artificial.

Shannon introdujo en 1948 el concepto de entropía [13]. La entropía en teoría de la información tiene mucho que ver con la incertidumbre que existe en cualquier experimento o señal aleatoria. Es también la cantidad de «ruido» o «desorden» que contiene o libera un sistema. Shannon ofrece una definición de entropía que satisface las siguientes afirmaciones:

- La medida de información debe ser proporcional (una función lineal y continua). Es decir, el cambio pequeño en una de las probabilidades de aparición de uno de los elementos de la señal debe alterar poco el valor de la entropía.

- Si todos los elementos de la señal son equiprobables a la hora de aparecer, entonces la entropía será máxima.

Una descripción formal de la entropía sería:

Si consideramos X una variable aleatoria con k posibles valores podríamos definir la entropía para variables discretas como:

$$E(X) = -\sum_{i=1}^{k} p(x_i)\ log_2\ p(x_i) \tag{2.1}$$

y para variables continuas de la siguiente forma:

$$E(X|Y) = -\int p(x_i)\ log_2\ p(x_i)dx \tag{2.2}$$

Es importante mencionar que se usan logaritmos en base 2 porque la información está codificada en bits.

Dadas dos Variables X e Y con k y l posibles valores respectivamente, la entropía de X condicionada a Y sería:

$$E(X|Y) = -\sum_{i=1}^{k}\sum_{j=1}^{l} p(x_i, y_j)\ log_2\ p(x_i|y_j) \tag{2.3}$$

Siendo $p(x_i, y_j)$ la probabilidad conjunta de las variables aleatorias X e Y y $p(x_i|y_j)$ la probabilidad condicionada de X sobre Y.

De igual forma, para el caso de variables continuas:

$$E(X|Y) = -\int\int p(x_i, y_j)\ log_2\ p(x_i|y_j)dxdy \tag{2.4}$$

En teoría de la información, la información mutua, transinformación o ganancia de información de dos variables se describe como una cantidad que mide la dependencia mutua de las dos variables, es decir, es una medida de la reducción de la entropía de una variable aleatoria, X, conociéndose el valor de otra variable, Y.

$$I(X;Y) = E(X) - E(X|Y) = \sum_{i=1}^{k}\sum_{j=1}^{l} p(x_i, y_j)\ log_2\ \frac{p(x_i, y_j)}{p(x_i)p(y_j)} \tag{2.5}$$

Teoría de la información aplicada a la selección de variables relevantes

En este trabajo, la teoría de la información y sobre todo el concepto de ganancia de información tienen especial relevancia ya que la base de datos de la que se parte cuenta con 118 variables por proyecto, y aunque se realice un filtrado inicial (veáse la subsección 3.2.2), para la clasificación de los datos según el algoritmo propuesto en la subsección 3.2.3 no son necesarias todas las variables [12]. Y no solo es que sean prescindibles, sino que además de aumentar el tiempo de procesado de los datos innecesariamente, la utilización de más variables de las necesarias nos puede llevar a obtener peores resultados debido a un sobre-ajuste (*over-fitting*). Esta situación se da al crear modelos que se ajustan demasiado a la evidencia, y como consecuencia, se comportan mal frente a nuevos ejemplos.

A la hora de imputar los datos que presentan valores perdidos en un conjunto de datos es importante saber seleccionar aquellas variables que presenten un alto grado de correlación con la variable a imputar.

En la publicación [2] se refieren a el proceso de selección de variables relevantes como *Key Feature Selection* y utilizan un algoritmo (algoritmo 1) muy similar al de los árboles ID3 y C4.5, basado en la teoría de la información y concretamente en el concepto de ganancia de información. En este algoritmo buscan clasificar los datos utilizando como variable de clase *Normalised Work Effort Level 1* y van seleccionando mediante un proceso iterativo las variables que presentan mayor ganancia de información con respecto a esta, hasta haber completado la clasificación de los datos.

Un Conjunto de datos, es un número de observaciones $n \in N$ proyectos de software, y se denota como $\mathbf{X} = X_1, X_2, \cdots, X_n$. Asumimos que el *dataset* $\mathbf{X}$ puede dividirse en dos subconjuntos, uno de datos observables $\mathbf{X}_{obs}$, y otro de datos perdidos $\mathbf{X}_{per}$, así $\mathbf{X} = \{\mathbf{X}_{obs}, \mathbf{X}_{per}\}$. Si el componente de datos perdidos es nulo, entonces podemos decir que el *dataset* está completo.

Cada elemento o proyecto $X_i \subset \mathbf{X}$ donde $i = 1, 2, \cdots, n$ es un caso, y si se define p cómo la dimensión del espacio de variables o atributos, el caso X_i puede denotarse como $(x_{i1}, x_{i2}, \cdots, x_{ip})$, $p < 1$. Siendo $1 \leq i \leq n$ y $1 \leq j \leq p$, entonces x_{ij} es el valor del caso i y la variable j del conjunto de datos $\mathbf{X}$. Las variables, también conocidos como características o *features* son un conjunto de atributos utilizados para diferenciar los distintos proyectos. Denotamos como F_j a la j-ésima variable del *dataset* donde $j = 1, 2, \cdots, p$. Dentro del conjunto de datos $\mathbf{X}$ asumimos que hay una característica especial que actúa como variable de clase, entonces tenemos N_c clases diferentes $c = 1, 2, \cdots, N_c$. Además, para cada caso, $X_i (1 \leq i \leq n)$, denotamos los valores de sus características perdidas cómo $X_i^{per} = \{x_{ij}, ?\}^f$

Supóngase que $s_i(i = 12, \cdots, Nc)$ es el número de casos en $\mathbf{X}$ en la clase o grupo C_i $(i = 1, 2, \cdots, Nc)$. La información necesaria para clasificar la base de datos $\mathbf{X}$ puede ser descrita como:

$$I(s_1, s_2, , \cdots, s_{Nc}) = -\sum_{i=1}^{Nc} \frac{s_i}{n} log_2 \frac{s_i}{n} \tag{2.6}$$

Si la característica F presenta v valores distintos, $\{a_1, a_2, \cdots a_v, \}$. Dicha *feature* puede utilizarse para dividir el conjunto de datos $\mathbf{X}$ en v subconjuntos $\{X_1, X_2, \cdots, X_v\}$, así $X_j(j = 1, 2, \cdots, v)$ contiene esos casos en $\mathbf{X}$ donde F toma cómo valor $a_j(j = 1, 2, \cdots, v)$. Si X_j contiene s_{ij} casos de la clase C_i, la información necesaria para clasificar todos casos en los diferentes subconjuntos X_j mediante la característica F, también conocida como entropía viene dada por la siguiente expresión según [2]:

$$E(F) = \sum_{j=1}^{v} \frac{s_{1j} + s_{2j} + \cdots s_{Ncj}}{n} I(s_1, s_2, , \cdots, s_{Nc}) \tag{2.7}$$

La información necesaria para clasificar un subconjunto de datos X_j es:

$$I(s_1, s_2, , \cdots, s_{Nc}) = -\sum_{i=1}^{Nc} \frac{s_{ij}}{n} log_2 \frac{s_{ij}}{n} \tag{2.8}$$

La ganancia de información de la característica F viene dada por :

$$Ganancia(F) = I(s_1, s_2, \cdots, s_{Nc}) - E(F) \tag{2.9}$$

Y el ratio de ganancia según [14] es:

$$Ratio\ de\ Ganancia = \frac{I(s_1, s_2, \cdots, s_{Nc}) - E(F)}{E(F)} \tag{2.10}$$

El algoritmo de FSS descrito en [2] como *Key Feature Extraction* está basado en un modelo heurístico y podría categorizarse dentro de los diferentes tipos de FSS como CFS(*Correlation-based Feature Selection*). En este artículo proponen el algoritmo 1.

2.1.2 Árboles de decisión aplicados a FSS

Un árbol de decisión es un modelo de predicción. Partiendo de un conjunto de datos se obtiene un diagrama de construcciones lógicas muy similar a los sistemas de predicción basados en reglas [15]. Sus aplicaciones son muy diversas, desde la economía, hasta campos como el de la inteligencia artificial o la minería de datos en tareas como la clasificación, la regresión, el agrupamiento o la selección de variables relevantes. Estos modelos predictivos se han hecho muy populares debido a su robustez y a su velocidad de ejecución.

Algoritmo 1 Extracción Características Clave. `Fuente:` A New Imputation Method for Small Software Project Data Sets [2]

Entrada: lista-de-características, casos
Salida: CaracterísticasClave.

si los casos son todos de la misma clase **entonces**
  devolver CaracterísticasClave;
fin si
si lista-de-características está vacía **entonces**
  devolver CaracterísticasClave;
fin si
para cada característica Fi de lista-de-características **hacer**
  Calcular la ganancia de información (Fi) según la fórmula ...
fin para
Busca la característica F con mayor ganancia de información;
lista-de-características -= F;
CaracterísticasClave $\cup = F$;
para cada característica de valor conocido a_i de la lista-de-características **hacer**
  Hacer que X_i sea el conjunto de casos para los cuáles $F = a_i$;
  si X_i no está vació **entonces**
    Llamar al algoritmo ExtracciónCaracterísticasClave(lista-de-características, casos);
  fin si
fin para

Un árbol de decisión puede entenderse como una colección de condiciones organizadas jerárquicamente, desde un punto de vista gráfico podría verse como un diagrama de flujo. En él, cada **nodo** es un test para un atributo, y representa el momento en el que se ha de tomar una decisión entre varias posibles. Las **ramas** se corresponden a las posibles decisiones tomadas en los nodos. Las **hojas** proporcionan la clasificación de la instancia. Para clasificar un caso, se comienza en el nodo raíz, se aplica el test al atributo del nodo en el que nos encontramos y se sigue por la rama que corresponde al valor de dicho atributo en el caso que se está clasificando. El proceso se repite hasta alcanzar una hoja, donde el caso se encuentra clasificado.

El algoritmo ID3

ID3 (*Iterative Dichotomiser 3*) [16] , fue introducido por Quinlan en 1987, es un algoritmo utilizado para hacer un árbol de decisión a partir de un conjunto de datos. El algoritmo se basa en maximizar la ganancia de información en cada escisión, nivel de división o nodo del árbol teniendo en cuenta todos los atributos disponibles. Es un precursor del algoritmo C4.5 y solo funciona con variables nominales[1], además es necesario que la variable a clasificar sea de tipo binario.

El algoritmo ID3, igual que la mayoría de algoritmos de inducción de árboles de decisión, construye el árbol de forma top-down, desde la raíz hasta las hojas, utilizando una estrategia *"divide y vencerás"*, también conocida como *"basada en el criterio de partición"* (en inglés, *splitting*).

Un ejemplo típico para explicar los árboles de decisión es el de los días que se podrá jugar al tenis en función de la previsión de cómo está el cielo, el viento y la temperatura [16].

[1] Una clasificación genérica de las variables es dividirlas en continuas o cuantitativas y categóricas o nominales, para una descripción más detallada de los tipos de variables véase la subsección 3.1.1.

Día	Cielo	Temperatura	Humedad	Viento	Jugar Tenis
D1	Sol	Calor	Alta	Débil	No
D2	Sol	Calor	Alta	Fuerte	No
D3	Nubes	Calor	Alta	Débil	Si
D4	Lluvia	Frío	Alta	Débil	Si
D5	Lluvia	Baja	Normal	Débil	Si
D6	Lluvia	Baja	Normal	Fuerte	No
D7	Nubes	Baja	Normal	Fuerte	Si
D8	Sol	Frío	Alta	Débil	No
D9	Sol	Baja	Normal	Débil	Si
D10	Lluvia	Frío	Normal	Débil	Si
D11	Sol	Frío	Normal	Fuerte	Si
D12	Nubes	Frío	Alta	Fuerte	Si
D13	Nubes	Calor	Normal	Débil	Si
D14	Lluvia	Frío	Alta	Fuerte	No

Tabla 2.1: Condiciones meteorológicas

Dada la tabla 2.1. El primer paso para generar un árbol de decisión ID3 es calcular la ganancia de información de cada atributo con respecto a la variable de clase para seleccionar el nodo raíz, la variable de clase es aquella que indica si se podrá o no jugar al tenis. Para ilustrar el ejemplo se utiliza el paquete de R[2] *FSelector* [14].

```
> information.gain(Jugar.Tenis~.,condiciones.meteorologicas)
attr_importance
Cielo              0.17103394
Temperatura       0.02025554
Humedad            0.10524435
Viento             0.03335912
```

Como se puede observar, el nodo raíz es la variable "Cielo" ya que es aquella cuya ganancia de información con respecto a la variable de clase (Jugar.Tenis). Así que nuestro árbol ID3 toma la siguiente forma:

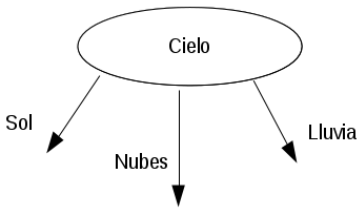


Figura 2.1: primera parte del árbol

Posteriormente seleccionamos los casos para los que la variable "Cielo" toma el valor "Sol".

```
sol<-subset(condiciones.meteorologicas, Cielo=="Sol")
```

[2]En la subsección 3.1.3 se introduce el lenguaje de programación R, véase para más información.

	Cielo	Temperatura	Humedad	Viento	Jugar.Tenis
1	Sol	Calor	Alta	Débil	No
2	Sol	Calor	Alta	Fuerte	No
8	Sol	Frío	Alta	Débil	No
9	Sol	Baja	Normal	Débil	Si
11	Sol	Frío	Normal	Fuerte	Si

Tabla 2.2: Cielo=Sol

y calculamos la ganancia de información para todas esas variables.

```
> information.gain(Jugar.Tenis~.,sol)
attr_importance
Cielo              0.00000000
Temperatura       0.39575279
Humedad            0.67301167
Viento             0.01384429
```

Aquí se aprecia que el nodo que sigue a la rama “Sol” será la variable “Humedad”.

En todos los casos en los que la variable “Cielo” toma el valor “Nubes” se puede jugar a tenis según se puede ver en la tabla 2.3. Por lo tanto todos estos casos ya se encuentran clasificados y nos encontramos ante una hoja del árbol.

	Cielo	Temperatura	Humedad	Viento	Jugar.Tenis
3	Nubes	Calor	Alta	Débil	Si
7	Nubes	Baja	Normal	Fuerte	Si
12	Nubes	Frío	Alta	Fuerte	Si
13	Nubes	Calor	Normal	Débil	Si

Tabla 2.3: Cielo=Nubes

A continuación, seleccionamos el subconjunto en el que la “Cielo” es igual a “Lluvia”’ y calculamos la ganancia de información.

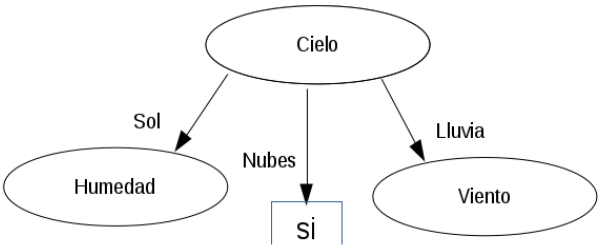


Figura 2.2: segunda parte del árbol

```
> lluvia<-subset(condiciones.meteorologicas, Cielo=="Lluvia")
> information.gain(Jugar.Tenis~.,lluvia)
attr_importance
Cielo              0.00000000
```

```
Temperatura       0.01384429
Humedad            0.01384429
Viento             0.67301167
```

En este punto nos encontramos con un árbol como el de la figura 2.2. A partir de aquí solo quedaría acabar de construir el árbol filtrando los subconjuntos sol y lluvia por los diferentes tipos de "Humedad" y "Viento" respectivamente, hasta ver que en cada *subset* final solo hay casos en los que se puede jugar a tenis o no se puede hacer. Solo entonces habremos hecho una correcta clasificación del árbol. El resultado final se puede ver en la figura 2.3.

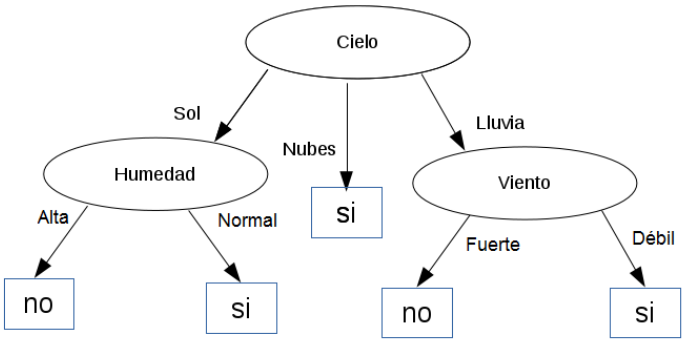


Figura 2.3: árbol final

A partir de la figura 2.3 si utilizamos el árbol con un objetivo predictivo y teniendo las condiciones meteorológicas para un nuevo día podríamos predecir si el día es apto o no lo es para jugar a tenis. Por otra parte, si nuestro objetivo es la selección de variables relevantes, vemos que la variable temperatura, que si aparece en la tabla 2.1, no se utiliza en el árbol generado. Por tanto, si nuestro objetivo es saber si podremos Jugar a tenis o no en función de las variables "Cielo", "Temperatura" y "Humedad" con algún tipo de mecanismo de predicción o imputación que no ha de ser necesariamente un árbol de decisión, podemos concluir que la variable "Temperatura" es la que menos influye en si se puede o no jugar a tenis y podría ser descartada.

El algoritmo C4.5

Se hizo bastante popular después de ser clasificado en el primer puesto del Top 10 de algoritmos de minería de datos en 2008 según la serie de libros Springer LNCS *Lecture Notes in Computer Science*

Mientras que kNN es un algoritmo utilizado tanto para imputación cómo para clasificación basado en el aprendizaje supervisado. C4.5 [17] se basa en el aprendizaje inductivo y aunque puede trabajar con *datasets* incompletos, se utiliza para la clasificación.

El funcionamiento de C4.5 es muy similar al de ID3, pero en vez de trabajar con ganancia de información, utiliza el ratio de ganancia.

Puede entenderse como una modificación y una mejora del algoritmo ID3 por su autor original en 1993, aplicándole el proceso de poda, que permite obtener árboles de menor tamaño y mayor precisión, además también permite trabajar con atributos cuantitativos, tolera valores perdidos, siempre y cuando no se den en la variable de clase y permite que el atributo de clase no sea binario. El algoritmo con poda consta de tres etapas [18].

En la primera, las reglas se extraen del árbol de decisión. Cada camino desde la raíz del árbol hasta sus hojas puede explicarse como un producto de reglas de la forma:

if $Cond_1 \hat{} Cond_2 \hat{} \cdots \hat{} Cond_n$ then $Class\ C$

Donde las $Cond_i$ son condiciones del camino y C es la clase etiquetada por la hoja, a la que llamaremos clase predicha por la regla.

En la segunda etapa, todas las reglas se agrupan de acuerdo a sus clases predichas. Para cada clase, el conjunto de condiciones aplicado a dicha clase es simplificando eliminando aquellas reglas que al ser removidas no disminuyen la predicción del algoritmo.

Finalmente, se comprueba cada regla para ver si eliminándola se puede reducir el error del conjunto de condiciones de entrenamiento.

El algoritmo M5

Los árboles LMT (*Logistic Model Tree*) [1], como los árboles de regresión, predicen un valor numérico para una instancia que es definida sobre un conjunto fijo de atributos numéricos o nominales. A diferencia de los árboles de regresión ordinarios, los LMT construyen una aproximación lineal por partes. El modelo de árbol final consta de un árbol con funciones de regresión lineal en las hojas, y la predicción para una instancia se obtiene llevando a cabo una clasificación similar a la de los árboles de decisión hasta una hoja y utilizando el modelo lineal asociado con esa hoja para predecir variables continuas.

El algoritmo M5 [19] se utiliza para construir árboles de la siguiente manera. Primero, se remplazan todos los atributos nominales por los binarios, a continuación se desarrolla un árbol de regresión sin poda, para ello se utiliza la reducción de la varianza como criterio de división. A continuación, se colocan modelos de regresión lineal en cada nodo del árbol. En este punto los atributos empleados en la regresión están restringidos a aquellos que se producen en el sub-árbol que conecta el nodo raíz con la hoja. La selección adicional de variables en los modelos se lleva a cabo añadiendo términos para minimizar una estimación de error que introduce una penalización por cada parámetro utilizado en el modelo. Después de haber construido el árbol y los modelos regresivos se lleva a cabo un proceso de poda en el que se eliminan variables del árbol teniendo en cuenta el error final de estimación de cada modelo.

2.2 Teoría de valores perdidos y aplicación

2.2.1 Introducción

Es frecuente que en estudios donde se maneja gran cantidad de información no se disponga de toda. El origen de los datos ausentes, puede deberse a varios factores, como el borrado accidental o el no registro de algún valor a la hora de recopilar los datos. Por ejemplo, es típico

en encuestas que los participantes no respondan a alguna pregunta por el desconocimiento de la respuesta, por cansancio, desinterés o por rechazo a proporcionar cierta información. El conjunto de métodos que tratan de asignar valores posibles a los datos incompletos son conocidos como técnicas de imputación. Estos métodos tratan de deducir el valor de los datos ausentes a partir de un conjunto de datos conocido.

2.2.2 Mecanismos de datos perdidos

Little y Rubin [20] establecieron en 1976 un marco teórico con el que trabajar con datos perdidos, que se sigue utilizando hoy en día. Este marco divide los mecanismos de *missing data* según la relación entre los valores perdidos y el resto de variables. Según esta clasificación existen tres tipos: MCAR (Missing Completly At Random), MAR (*Missing At Random*) y MNAR (*Missing Not At Random*).

MCAR (Missing Completly At Random)

El mecanismo MCAR se da cuando la perdida de los datos es completamente aleatoria y los valores perdidos no están relacionados con ninguna variable.

Si se supone Z Como la matriz que incluye tanto los valores perdidos como los que no lo son y llamamos Z_{obs} a los datos completos y Z_{per} a los datos perdidos. El subconjunto de valores perdidos puede definirse como la matriz R, siendo i el i-ésimo caso y j la j-ésima características [5].

$$R = \begin{cases} 1 & \text{si } Z_{ij} \text{ es un valor perdido} \\ 0 & \text{si } Z_{ij} \text{ no es un valor perdido} \end{cases} \tag{2.11}$$

Una descripción formal del mecanismo MCAR es:

$$p(R|Z) = p(R); \forall Z \tag{2.12}$$

MAR (Missing At Random)

Este mecanismo se da cuando los datos perdidos presentan algún tipo de relación con los datos observados [2]. En este mecanismo el patrón de la pérdida se puede predecir a partir de otras variables:

$$p(R|Z) = p(R|Z_{obs}); \forall Z_{mis} \tag{2.13}$$

MNAR (Missing Not At Random)

También conocido como NI (*Not Ignorables*) se da cuando el mecanismo de la pérdida no es aleatorio, los valores perdidos están relacionado con los mismos valores perdidos y el patrón de datos no puede obtenerse a partir de los datos observados. En lenguaje matemático:

$$p(R|Z) \neq p(R); \forall Z \tag{2.14}$$

$$p(R|Z) = p(R|Z_{mis}) \forall Z \tag{2.15}$$

Esta es la peor de las tres situaciones ya que la información de la que se dispone no está relacionada con la pérdida. Y poco se puede hacer a no ser que se conozca algo sobre los motivos por los que no están disponibles los datos [21].

2.2.3 Patrones de datos perdidos

Los patrones de datos perdidos describen la distribución de los datos perdidos en el *dataset*. La matriz indicadora de datos perdidos R nos muestra el patrón. Esta matriz se forma mediante la Ecuación 2.11.

Existen muchos tipos diferentes de patrones de datos perdidos, Enders los clasifica en 6 tipos en [21]. El univariante (*univariate pattern*), el unidad sin respuesta (*unit nonresponse pattern*), el monótono (*monottone pattern*), el general (*general pattern*), el patrón planificado (*planed missing pattern*) y el variable latente (*latent variable pattern*).

En el patrón univariante solo una variable presenta *missing data* y aunque es muy raro que se dé en casos prácticos, se suele usar en estudios experimentales.

El (*unit nonresponse pattern*) es típico en encuestas en las que algunas preguntas son más comprometidas que otras.

En el patrón monótono primero las variables se pueden organizar de tal forma que para un conjunto de variables $x_1, x_2, \cdots, x_n$ si x_i es un valor perdido también lo serán $x_{i+1}, x_{i+2}, \cdots, x_{i+n}$. Es muy frecuente en estudios largos en el tiempo donde los participantes abandonan.

En el patrón general o patrón arbitrario los datos se distribuyen de una forma aparentemente aleatoria entre todas las variables.

El patrón planificado se corresponde al caso de una encuesta en la que se les ha pasado diferentes formularios a los participantes.

Por último, el patrón de variable latente se da únicamente en el análisis de variables latentes como en los modelos de ecuaciones estructurales. Podría considerarse un caso de patrón univariante para el cual se han perdidos todas las muestras de la variable.

Es importante recalcar que el patrón únicamente hace referencia a la distribución de los datos y no tiene especial relevancia en su imputación [20]. Aunque sí es importante en determinadas ocasiones, ya que hay algunos tipos de métodos que no funcionan adecuadamente con determinados patrones o mecanismos de datos perdidos.

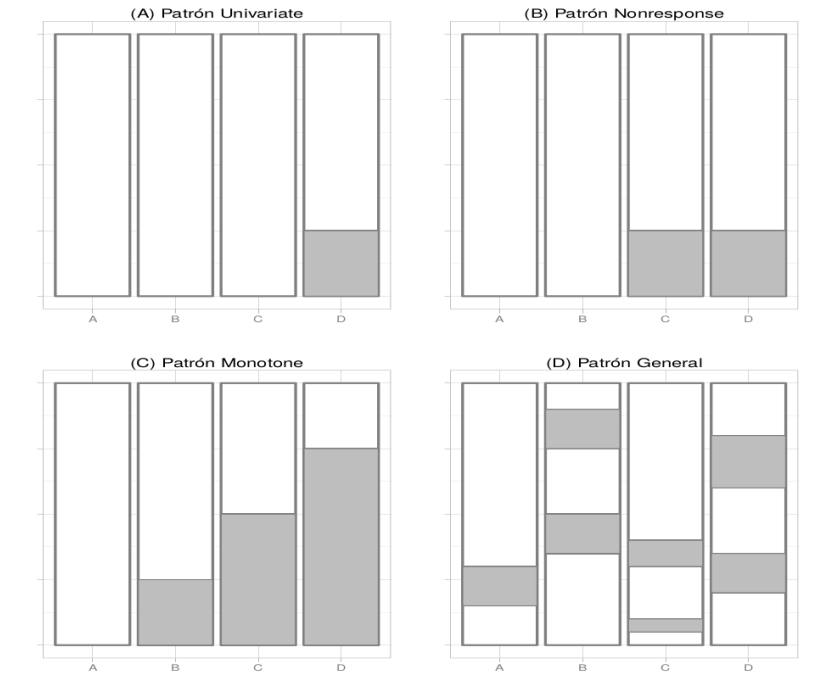


Figura 2.4: Patrones de missing data. `Fuente:` Applied Missing Data Analysis [21]

2.2.4 Técnicas para el tratamiento de valores perdidos

El problema de los datos perdidos, ha sido estudiado durante 40 años. Las técnicas empleadas para el tratamiento de datos perdidos pueden clasificarse en tres tipos. Técnicas de borrado, técnicas tolerantes y técnicas de imputación.

Las técnicas de borrado eliminan los valores perdidos y se dividen en dos tipos: LD (*Listwise Deletion*)y PD (*Pairwise Deletion*). LD elimina completamente los casos con valores perdidos mientras que PD considera cada característica de forma independiente, de manera que, a la hora de operar con cada característica o variable de la base de datos, se tienen en cuenta únicamente los casos con valores completos. Esto implica que diferentes cálculos, utilizan casos diferentes y presentaran distinto número de muestras.

Las técnicas tolerantes son aquellas que permiten trabajar con una base de datos incompleta. Un ejemplo es el algoritmo C4.5 [17] (véase la subsección 2.1.2), empleado en este trabajo para la selección de variables relevantes. Este método tolera los datos perdidos para todas las variables, menos la variable de clase.

Las técnicas de imputación estiman los valores perdidos y posteriormente los insertan en la base de datos para generar un conjunto de datos completo. Este trabajo se centra en las técnicas de imputación. Las técnicas más utilizadas para la imputación son las siguientes:

- *Mean Imputation* (MI) es una de las técnicas más sencillas de imputación. En ella los valores ausentes se sustituyen por la media aritmética de los observados. Presenta el problema de que subestima la varianza.

- *Regression Imputation* (RI) consiste en extraer un modelo regresivo i.e. un conjunto de ecuaciones de regresión, cuyos coeficientes se construyen a partir de los datos existentes. Estas

ecuaciones modelarán el comportamiento de la variable o las variables dependientes y servirán para calcular los valores de los datos perdidos a partir de las variables independientes, que serán conocidas *a priori*.

- *Hot-deck imputation* (HDI) son las técnicas que utilizan casos similares (vecinos). Dichos vecinos son escogidos teniendo en cuenta su similitud con el caso que se pretende imputar. Estos métodos siguen subestimando la varianza, pero no tanto como los anteriores. Algunos ejemplos son:

  - *Random Value* Es el más sencillo y consiste en escoger a los vecinos aleatoriamente.

  - *Class Mean Imputation* (CMI) divide la base de datos en Nc subconjuntos o clases diferentes, posteriormente aplica MI independientemente para cada clase [2], [5].

  - *Similar Response Pattern Imputation* (SRPI) identifica el caso más similar al caso que se desea imputar utilizando la distancia por mínimos cuadrados y se sustituye su valor por el de este caso [21].

  - *k Nearest Neighbours* (kNN) selecciona los k casos más cercanos al caso a imputar. Posteriormente se utilizan estos k casos para sustituir el valor perdido por uno imputado mediante algún tipo de método que normalmente es la media aritmética o la moda, dependiendo de si tratamos con variables continuas o categóricas [2], [4]-[6], [9], [11], [18].

- *Cold-Deck Imputation* (CDI). Funciona igual que la imputación Hot-deck, pero empleando datos externos a los de la base de datos que se pretende imputar. Estos datos externos suelen ser una versión previa de la misma base de datos [9].

- *Maximum Likelihood* (ML) consiste en hacer una suposición sobre la función distribución de los datos con valores perdidos que suele ser la distribución normal, aunque también se puede aplicar con otros tipos de distribuciones [20].

- *Imputación Múltiple* consiste en imputar un número $m > 1$ de veces la variable aleatoria a partir de la función distribución de la variable a imputar. Obteniendo así m *datasets* completos con el objetivo de posteriormente analizar el comportamiento de estos *datasets* con técnicas que no toleren los datos perdidos y así reducir el problema de la varianza. Presenta el inconveniente de requerir un elevado coste computacional.

2.2.5 El algoritmo kNN

Dado un conjunto de datos, y un caso particular de ese conjunto, el algoritmo *k Nearest Neighbours* (kNN) trata de identificar los k casos más similares al caso particular. Para calcular la similitud entre dos casos se considera a cada caso como un vector y se utiliza una medida de distancia (la Euclídea por ejemplo), para medir la diferencia entre los valores de una misma posición del vector, es decir, la misma variable. Una vez hemos obtenido la distancia entre todas las variables, se calcula la distancia entre casos como la suma de las distancias entre sus variables. Cuanto menor es la distancia entre dos casos, más similares se consideran [2], [4]-[6], [9], [11], [18].

Las aplicaciones de kNN son muy diversas, se puede utilizar como método de clasificación para el reconocimiento de patrones, como método de agrupación (clustering) o como algoritmo

de imputación. Si se utiliza para imputar datos perdidos, el procedimiento consiste en buscar los k valores más cercanos al caso sobre el cual se va a realizar la imputación y posteriormente se utilizan los valores de la variable a imputar de los vecinos para calcular y asignar un valor a la misma variable del caso a imputar.

La similitud entre dos casos es un concepto muy subjetivo. Y aunque la distancia se define mediante una expresión matemática, existen muchas formas diferentes de hacerlo. Algunas de las distancias más conocidas son la Minkowski, la Eculídea, la Manhattan, o la Chebysev. En [2] utilizan un tipo de distancia similar a la Manhattan definida por ellos mismos. Por otra parte, en [10] y [11] utilizan la distancia Gower para su implementación de la función kNN. En [4] Utilizan la Euclídea para su experimento.

Otro parámetro muy importante del algoritmo kNN es el número de k. La elección de este parámetro depende mayoritariamente de los datos con los que se trabaja. Para valores grandes de k, el algoritmo tomará como vecinos casos que se encuentren alejados del caso a imputar, mientras que valores pequeños de k, donde el caso extremo es $k = 1$, puede producir resultados erróneos debido a la existencia de *outliers* (puntos inconsistentes con el resto del dataset). En [2] utilizan un valor de $k = 2$, debido a la sugerencia de una publicación de Gada Kadoda, Michelle Cartwright y Martin Shepperd de 2001 (*Issues of the effective use of CBR technology for software project prediction*).

2.2.6 Los algoritmos CMI y MINI

Class Mean Imputation [2], [5] remplaza cada valor perdido por la media de la clase o del *cluster* a la hora de imputar variables continuas. Para el caso del tratamiento de atributos nominales se emplea la moda en vez de la media. En este método la varianza es subestimada, no obstante, no tanto como en el caso de *Mean Imputation*. Sin embargo, la precisión de la imputación de este método suele ser inferior a la de kNN. El algoritmo MINI [2] trata de mejorar los problemas con la varianza y además aumentar el grado de precisión en la imputación de los datos ausentes.

2.3 Análisis de outliers

2.3.1 Casos atípicos

Se denominan casos atípicos u *outliers* a aquellas observaciones con características diferentes de las demás. Este tipo de casos no pueden ser caracterizados como benéficos o problemáticos, sino que deben ser contemplados en el contexto del análisis y debe evaluarse el tipo de información que pueden proporcionar.

Su principal problema radica en que estos casos pueden no ser representativos de la población llegando a distorsionar el comportamiento de los contrastes estadísticos. Por otra parte, aunque diferentes a la mayor parte de la muestra, pueden ser indicativos de las características de un segmento válido de la población y, por consiguiente, una señal de la falta de representatividad de la muestra.

Los *outliers* pueden clasificarse en 4 tipos [22]:

1. Los que surgen de un error de procedimiento, como puede ser la entrada de datos o la codificación.

2. Observaciones que ocurren como consecuencia de un suceso extraordinario.

3. Observaciones cuyos valores caen dentro del rango de las variables observadas. Pero en las que la combinación de dichas variables es única.

4. Datos extraordinarios para los cuales no se puede encontrar una explicación subyacente.

Para detectar la presencia de *outliers* se pueden utilizar métodos univariantes o multivariantes. Ante los primeros la mejor alternativa es recurrir a tests, como el test de Dixon o el test de Grubb. Este procedimiento no es sencillo. Para empezar, versiones simples de los tests pueden fallar si hay más de un *outlier* presente. Además, debemos decidir si una vez detectados los casos atípicos los descartamos o no.

El caso mutlivariante es todavía más preocupante y complejo ya que los *outliers* multivariantes son observaciones que se consideran extrañas no por el valor que toman en una determinada variable, sino en el conjunto de ellas. Son más difíciles de identificar que los *outliers* unidimensionales, dado que no pueden considerarse "valores extremos", como sucede cuando se tiene una única variable bajo estudio. Su presencia tiene efectos todavía más perjudiciales que en el caso unidimensional, porque distorsionan no sólo los valores de la medida de posición (media) o de dispersión (varianza), sino muy especialmente, las correlaciones entre las variables [23].

2.3.2 Estadística robusta

La estadística robusta es una forma de analizar los resultados cuando se supone que presentan una pequeña proporción de *outliers*. La mayoría de estimaciones de tendencia central (ej. media aritmética) y de dispersión (ej. desviación típica) dependen de la interpretación de que los datos utilizados sean una muestra aleatoria de un conjunto de datos que sigue una distribución normal. No obstante, en la mayoría de casos, los datos analizados se alejan de ese modelo.

Supongamos por ejemplo el siguiente conjunto de datos del ejemplo de [24]:

4.5 4.9 5.6 4.2 6.2 5.2 9.9

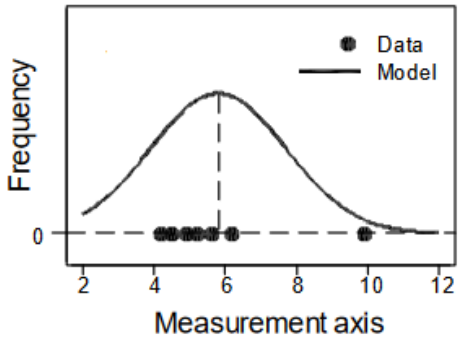


Figura 2.5: Distribución del conjunto de datos con outliers. `Fuente:` Robust statistics: a method of coping with outliers [24].

Incluso en una muestra tan pequeña, el valor 9.9 parece no ser representativo de ella. Si incluimos este valor en los cálculos obtenemos $\overline{x} = 5.8$ y $s = 1.9$. Estos estadísticos empleados para definir un modelo basado en una distribución normal describen los datos, pero no lo hacen bien.

Una interpretación más adecuada de estos datos es que presentan una media que aproximadamente vale 5. Y una de desviación estándar de en torno a 1, con un *outlier* de 9.9. Si no tenemos en cuenta el caso atípico para los cálculos, nos encontramos con que $\overline{x} = 5.1$ y $s = 0.7$. Estas estadísticas nos proporcionan un modelo posible para la mayoría de datos (figura 2.6) a pesar de no advertirnos sobre la posible presencia de *outliers*. Esta aproximación es preferible en las aplicaciones de análisis estadístico.

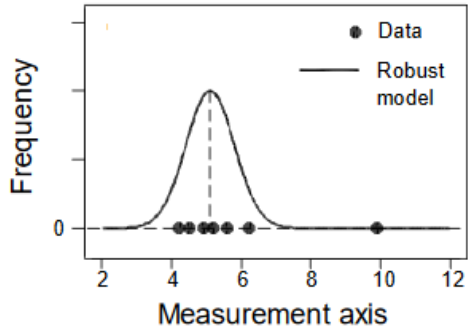


Figura 2.6: Distribución del conjunto de datos sin outliers. `Fuente:` Robust statistics: a method of coping with outliers [24].

Estimadores robustos

Para reducir el problema de los *outliers* univariados los estadísticos robustos proporcionan un procedimiento alternativo, este procedimiento nos ofrece un modelo en el que se describe la parte "buena" de los datos, sin la necesidad de identificar observaciones específicas y excluirlas. Primero estudiaremos el estimador de la mediana y posteriormente el estimador M de Huber.

Estimador mediana/ método MAD

En este método extraemos el valor central de los datos ordenados (mediana) como la estimación de la media.

4.2 4.5 4.9 **5.2** 5.6 6.2 9.9

Se puede apreciar que la media no aumenta por mucho que lo haga el valor del *outlier*. La mediana es un estimador robusto de la media. La denotamos por $\hat{\mu} = 5.2$ para distinguirla de la media aritmética ordinaria ($\overline{x}$).

Para estimar la desviación típica calculamos las diferencias entre los valores del conjunto de datos y la mediana:

-1.0 -0.7 -0.3 0.0 0.4 1.0 4.7

Posteriormente ordenamos de forma creciente el valor absoluto de estas diferencias y seleccionamos la mediana de estos valores (MAD, por sus siglas en inglés *median absolute difference*).

0.0 0.3 0.4 **0.7** 1.0 1.0 4.7

Por lo tanto, obtenemos un valor de MAD=0.7. De nuevo un aumento en el valor atípico no repercutirá en el cálculo de MAD. Para la estimación de la desviación estándar, a la que denotaremos por $\hat{\sigma}$, multiplicaremos el valor de MAD por una constante k que presenta un valor cercano a 1.5 para datos que siguen una distribución normal. Proporcionándonos un valor de $\hat{\sigma} = 1.05$

Estimador M de Huber

El método de Huber utiliza más información proporcionada por los datos. En este método, transformamos progresivamente los datos a través de un proceso denominado winsorización. Asumiendo que tenemos estimaciones iniciales llamadas $\hat{\mu}_0$ y $\hat{\sigma}_0$ (podrían ser estimaciones mediana-MAD, o simplemente $\overline{x}$ y s). Si un valor x_i cae por encima $\hat{\mu}_0 + 1.5\sigma_0$ lo remplazamos por $\tilde{x}_i = \hat{\mu}_0 + 1.5\sigma_0$. De la misma forma, si el valor cae por debajo de $\hat{\mu}_0 - 1.5\sigma_0$ lo reemplazaremos por $\tilde{x}_i = \hat{\mu}_0 - 1.5\sigma_0$. En caso de que no se cumpla ninguna de las condiciones anteriores dejamos su valor original $\tilde{x}_i = x_i$. A partir de aquí calculamos una estimación mejorada de la media como $\hat{\mu}_1 = mean(\tilde{x}_i)$ y de la desviación típica como $\hat{\sigma}_1 = 1.134 \times stdev(\tilde{x}_i)$. (El factor 1.134 es utilizado para la distribución normal, dando un valor de 1.5 para el multiplicador más usado en el proceso de winsorización.)

El *dataset* de 7 muestras es demasiado pequeño para aplicarle un proceso de winsorización, pero sirve como ilustración del método. Utilizando $\hat{\mu}_0 = 5.2$ y $\hat{\sigma}_0 = 1.05$, la winsorización transforma el conjunto de datos inicial en:

4.5 4.9 5.6 4.2 6.2 5.2 **6.675**

Las estimaciones mejoradas de este nuevo conjunto son $\hat{\mu}_1 = 5.34$ y $\hat{\sigma}_1 = 1.04$. Este procedimiento se repite iterativamente utilizando en cada ciclo las estimaciones mejoradas del anterior. Finalmente, el proceso converge, y los resultados son las estimaciones robustas. Para el conjunto de datos del ejemplo:

$\hat{\mu}_{Hub} = 5.36$ y $\hat{\sigma}_{Hub} = 1.15$

Capítulo 3

Metodología

El siguiente capítulo consiste en una descripción de los procedimientos utilizados para alcanzar los objetivos y se encuentra dividido en dos secciones: Herramientas de trabajo y diseño del experimento.

En la sección de herramientas de trabajo se presenta la base de datos empleada en el experimento junto a una descripción más detallada de la misma y los diferentes de datos que la componen. A continuación se describen los motivos por los que se elige R para implementar todas las funciones y todos los scripts de este proyecto, y se hace una breve introducción al lenguaje de programación.

Por otro lado, la sección de diseño e implementación del experimento trata de la planificación del proyecto, en ella se describen los algoritmos empleados y la estructuración del código fuente, así como una descripción detallada del mismo y una explicación individual del comportamiento de cada función.

3.1 Herramientas de trabajo

3.1.1 Fuente de los datos

La base de datos del ISBSG

El *benchmarking* consiste en tomar "comparadores" o *benchmarks* a aquellos productos, servicios y procesos de trabajo que pertenezcan a organizaciones que evidencien las mejores prácticas sobre el área de interés, con el propósito de transferir el conocimiento de las mejores prácticas y su aplicación.

El ISBSG (*International Software Benchmarking Standards Group*) es una organización sin ánimo de lucro fundada en 1997 con el objetivo de recopilar una base de datos con el histórico de proyectos relacionados con la IT (*Information Technology*) que pueda servir para mejorar los procesos de software y los productos relacionados. El objetivo del ISBSG es ayudar a cualquier

tipo de organizaciones a realizar sus propios análisis sobre la base de datos, tanto para profesionales del sector que pretenden llevar a cabo una planificación realista como para investigadores que buscan obtener nuevos modelos matemáticos que sirvan para mejorar esta industria.

El ISBSG cuenta con una base de datos que actualmente se encuentra en su decimotercera versión y consta de 6760 casos. La base de datos del ISBSG está organizada por versiones y este trabajo está realizado con la versión 12, que contiene 6006 proyectos comprendidos entre los años 1989 y 2013 de sectores cómo aseguradoras, banca, servicios, comunicaciones o gobierno de hasta 25 países. La figura 3.1 muestra el número de proyectos en función de la versión de la base de datos del ISBSG. Es importante señalar que el trabajo realizado en la publicación [2] parte de la versión 7 de esta misma base de datos, que contaba con 1238 proyectos. Y esto, entre otras causas, influirá en los resultados obtenidos. Los principales inconvenientes de este repositorio son dos: el primero es que los datos no son homogéneos, es decir, provienen de diferentes fuentes y además cuentan con elevado porcentaje de *outliers* (véase sección 2.3). El segundo es que la mayoría de variables presenta un número excesivamente elevado de datos perdidos. No obstante, la base de datos cuenta con dos campos que sirven para evaluar la calidad de las muestras y que pueden ser utilizados para llevar a cabo procesos de filtrado.

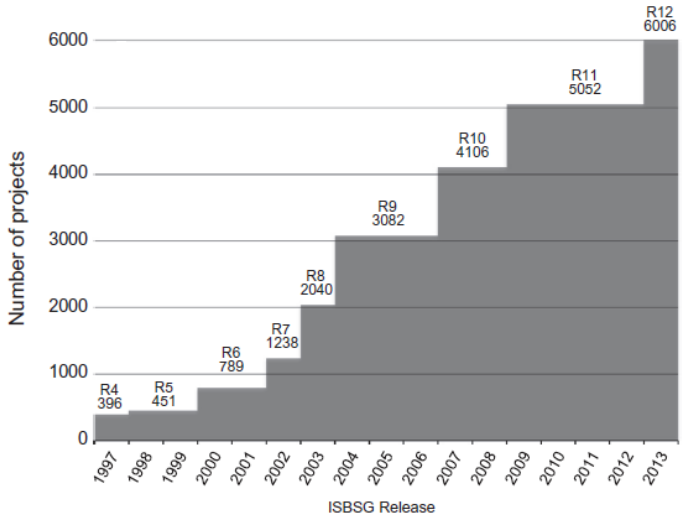


Figura 3.1: Número de proyectos según la versión del ISBSG. `Fuente:`Potential and limitations of the ISBSG dataset in enhancing software engineering research: A mapping review [7].

Cada proyecto de la versión 12 del ISBSG está compuesto por 125 variables y el problema con la selección de variables relevantes de la base de datos para la predicción del esfuerzo es que no existe consenso acerca de qué variables deben ser seleccionadas, o qué métodos utilizar para llevar a cabo dicha selección.

En [7] Fernando González Ladrón de Guevara y Marta Fernández Diego realizan un estudio sistemático de mapeo para el cual parten de 306 artículos y actas de congresos de IEEEXplore, ACM DIgital Library, ScienceDirect y WebofNoledge hasta finales de junio de 2012. Y tras un proceso de filtrado en el que eliminan duplicados y seleccionan aquellos en los que la base de datos del ISBSG presenta especial relevancia se quedan con 129 proyectos. A partir de estas publicaciones extraen diversas conclusiones. Una de ellas es que casi tres cuartas partes de los estudios tratan métodos de estimación. De las 129 publicaciones iniciales, 91 (70.5 %) se centran en la variable esfuerzo. El 36 % está relacionado con propiedades de los conjuntos de datos y el 25 % con modelos de calibración. Por otra parte, los estudios que se centran en métodos de estimación la mayoría emplea modelos regresivos (61.2 %), seguidos por métodos de aprendizaje

automático (35.7 %) y en tercer lugar se encuentran los métodos de estimación por analogía (22.5 %).

En [8] Fernando González Ladrón de Guevara, Marta Fernández Diego y Chris Lokan realizan otro estudio sistemático de mapeo que se centra en el uso de las variables de la base de datos del ISBSG. Este estudio se lleva a cabo a través de 107 artículos y actas de congresos utilizando los mismos orígenes de datos que en [7], en estas publicaciones la variable dependiente es el esfuerzo o la productividad y las independientes varían según el artículo. Entre los criterios para la selección de variables independientes se encuentran la creencia en la relación de dichas variables con el esfuerzo, la utilización de las mismas que otros estudios previos o la utilización de métodos estadísticos para determinar relaciones entre las variables. Hay una descripción más detallada de las variables independientes más utilizadas en los artículos del estudio en el Apéndice A. En las publicaciones las variables utilizadas para llevar a cabo el proceso de filtrado son principalmente *Data Quality Rating* y *Count Approach*, en algunos artículos también se utilizan las variables *Resource Level*, *Unadjusted Function Points* y *Recording Method*. En un 36 % de los artículos se eliminan casos completos en aquellos proyectos que se detecta la presencia de *outliers* (véase la sección 2.3), utilizando para ello variables de los grupos *size*, *effort* y *PDR*. Por otro lado, la mayoría de métodos de estimación están basados en modelos regresivos, a veces solos, pero en la mayoría en combinación con otros métodos.

Tipos de datos

Una base de datos o *dataset* está compuesta por un conjunto de casos u observaciones cada uno de las cuales presenta una serie de características o variables a las que llamamos atributos. De tal forma que se puede entender como una matriz en la que las filas son los casos y las columnas los atributos en [3] de la siguiente manera:

- *Los atributos cuantitativos* se dividen en:
  - *Atributos con valores continuos* , por ejemplo, el peso de una persona, o la variable de la base de datos del ISBSG *Project Elapsed Time*.
  - *Atributos con valores discretos,* por ejemplo, el esfuerzo normalizado *Normalised Work Effort Level.1* o *Unadjusted Function Points*. Un caso particular de estos atributos se da cuando sólo toman valores binarios, i.e. 0 o 1 y que representa la presencia o ausencia de una determinada característica. Este tipo de atributos es muy frecuente cuando se discretizan variables continuas, como en el caso de este trabajo (véase subsección 3.2.3). En [9], por ejemplo, se discretiza la variable *Language Type* y se convierte en 7 variables binarias sobre las que posteriormente se aplica regresión lineal.
  - *Atributos con valores definidos sobre intervalos,* por ejemplo, la duración de un suceso discretizada o el tamaño de un proyecto. En la base de datos con la que trabajamos, lo podemos encontrar por ejemplo en *Relative Size*.
- *Los atributos cualitativos o Categóricos* se dividen en:
  - *Atributos con valores nominales o no ordenados,* por ejemplo, *Industry Sector* o *Application Group*.

- *Atributos con valores ordinales,* por ejemplo, el rango de un militar o *Language Type* en la base de datos del ISBSG, que hace referencia al nivel de abstracción del lenguaje utilizado.

3.1.2 El lenguaje de programación utilizado

S es un lenguaje estadístico desarrollado inicialmente por AT&T hacia finales de los 70 y enfocado al análisis de datos. En 1980 fue distribuido por primera vez fuera de AT&T posteriormente fue implementado comercialmente junto a una GUI bajo el nombre de S-PLUS por TIBCO Software Inc.

R es un lenguaje de programación interpretado y orientado a objetos basado en lenguaje S. Fue desarrollado inicialmente por Robert Gentleman y Ross Ihaka del departamento de Estadística de la Universidad de Auckland en 1993. Actualmente su desarrollo es responsabilidad del R Development Core Team y los trabajos de los contribuidores son distribuidos bajo CRAN [25] (*Comprhensive R Archive Network*). R está distribuido bajo una GNU GPL (GNU Global Public License) [26] de forma que cualquier interesado puede acceder al código, modificarlo y contribuir al proyecto con su trabajo.

La interfaz gráfica de R es RStudio. RStudio es un entorno de desarrollo integrado (IDE) para R. Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

Entre las ventajas de R destacan que es multi-plataforma, su estructura de programación funcional orientada a objetos, que es software libre, y cumple con las cuatro libertades según la definición de software libre de Richard Stallman [27], hecho que permite que el proyecto cuente con una comunidad de colaboradores muy activa entre la que se encuentran investigadores académicos, especialistas en marketing, programadores profesionales o analistas de datos. Actualmente hay más de 9400 paquetes disponibles. Además, a diferencia de otros programas de software estadístico que no son un lenguaje de programación, es decir, que basan la interacción con el usuario en una interfaz gráfica cómo SPSS, R permite automatizar tareas y análisis y crear funciones. [28]

Por estos motivos todo el análisis estadístico y todo el trabajo de programación de este proyecto ha sido realizado en R.

3.1.3 Introducción a R

Operaciones básicas

El primer paso para utilizar R es o bien iniciarlo desde el terminal con el comando R o bien arrancar el entorno de desarrollo RStudio. Al realizar alguna de estas acciones lo primero que vemos en la consola es el símbolo de sistema (>). Los comandos más básicos son la asignación de variables mediante el operador <- y operaciones aritméticas (+ - / *).

```
> a<-1 #Esto es un comentario
> b<-2
> b-a
[1] 1
```

Los comentarios en R se realizan mediante el símbolo #. A partir de la almohadilla el intérprete ignorará el resto de la línea.

Un vector es un conjunto de variables del mismo tipo. La creación de vectores se realiza mediante la función c( ) de concatenación. En el siguiente ejemplo se crea un vector concatenando las variables que ya se encontraban asignadas en el entorno de trabajo global. posteriormente se crea otro vector y se suman ambos.

```
> A<-c(a,b)
> B<-(0,1)
> A+B
[1] 1 3
```

Las operaciones aritméticas entre vectores se realizan con los mismos operandos.

Estructuras de datos en R

Tipos de datos básicos

Existen varios tipos de datos en R. En la siguiente tabla se pueden ver los más importantes.

logical	Números booleanos que permiten dos estados, TRUE o FALSE.
numeric	Números reales.
integer	Números complejos
complex	Datos que pueden contener números complejos
character	Datos encerrados entre comillas que pueden contener cualquier tipo de carácter y números.

Tabla 3.1: Tipos de datos en R

Objetos en R

Vectores

El tipo de datos básico en R es el vector [28]. Es decir, todos los elementos que no sean definidos de otra forma son vectores. La variable "a" del ejemplo anterior es un vector numérico de una única posición.

Para conocer el tipo de datos de una variable se puede hacer mediante la función class(). En el siguiente ejemplo vemos que el vector "A" que habíamos creado anteriormente es de la clase *numeric*. Posteriormente creamos un vector lógico y comprobamos su tipo de datos.

```
> class(A)
[1] "numeric"
> class(c(TRUE,FALSE))
[1] "logical"
> class(class(A))
[1] "character"
```

Para generar vectores de forma automática podemos hacerlo mediante el operador ":" o con la función seq( ). El funcionamiento del operador ":" es el mismo que en MATLAB, sirve para

crear vectores compuestos por secuencias regulares entre dos números por intervalos de una unidad. La función seq( ) realiza la misma operación, es decir, genera secuencias regulares, pero el incremento o decremento no tiene por qué ser una unidad. En el siguiente ejemplo se ve una llamada a la función en la que el numero inicial es 2, el final 22, y los incrementos se producen por intervalos de 4.

```
seq(2,22,4)
[1]  2  6 10 14 18 22
```

Matrices

Una matriz, igual que en la mayoría de lenguajes de programación es un vector de dos dimensiones. Las matrices se inicializan utilizando la función matrix( ). En el siguiente ejemplo se le pasa un vector a la función matrix( ) y se le indica que el número de filas deseadas es 2.

```
> X<-matrix(seq(5,20,5),nrow=2)
> X
[,1] [,2]
[1,]    5   15
[2,]   10   20
```

Para acceder a la posición de una matriz se indica entre corchetes el número de la fila y la columna. Si queremos acceder a una fila o una columna completa dejamos en blanco el número de la fila o de la columna.

```
> X[1,2]
[1] 15
> X[1,]
[1]  5 15
> X[,2]
[1] 15 20
```

Factores

Este tipo de objeto es fundamental para el análisis estadístico pues es la forma como se tratan las variables categóricas. Para definir una variable categórica en R se utiliza la función factor.

```
> cielo<-factor(c("Sol","Sol","Nubes","Lluvia","Lluvia","Lluvia","Nubes","Sol",
    "Sol"))
> cielo
[1] Sol    Sol    Nubes  Lluvia Lluvia Lluvia Nubes  Sol    Sol
Levels: Lluvia Nubes Sol
```

Listas

Son vectores cuyos elementos pueden ser de distinto tipo. Son similares a las estructuras en C [28], o los mapas en MATLAB y se utilizan mucho en R. El lenguaje de programación solo permite que las funciones devuelvan una única variable u objeto, por lo que es muy típico utilizar listas que contengan diversas variables u objetos como valores de retorno.

```
> dia<-list(cielo="Soleado",temperatura=26,humedad="Normal",viento="Débil",
    Jugar.Tenis=TRUE)
```

```
> dia
$cielo
[1] "Soleado"

$temperatura
[1] 26

$humedad
[1] "Normal"

$viento
[1] "Débil"

$Jugar.Tenis
[1] TRUE
```

Para acceder a cada elemento de la lista se puede hacer de varias formas:

```
>dia[2]
$temperatura
[1] 26

> dia["temperatura"]
$temperatura
[1] 26
```

Para acceder únicamente al valor también existen diferentes maneras de hacerlo:

```
> dia[["temperatura"]]
[1] 26
> dia$temperatura
[1] 26
```

Data Frames

El *data frame* presenta una estructura similar a la de las matrices, organizado por filas y columnas. Pero se diferencia de estas en que cada columna puede ser un tipo de datos distinto. De forma intuitiva podría decirse que un *data frame* es a una matriz, lo que una lista a un vector.

```
> dataset<-data.frame(cielo,temperatura=c(24,23,21,17,18,19,22,25,23))
> dataset
cielo temperatura
1    Sol          24
2    Sol          23
3  Nubes          21
4 Lluvia          17
5 Lluvia          18
6 Lluvia          19
7  Nubes          22
8    Sol          25
9    Sol          23
```

Para seleccionar una posición de un *data frame* o seleccionar una fila el procedimiento es el mismo que con las matrices. No obstante, si queremos seleccionar una columna existen diferentes formas de hacerlo:

```
> dataset$cielo
```

```
[1] Sol    Sol    Nubes  Lluvia Lluvia Lluvia Nubes  Sol    Sol
Levels: Lluvia Nubes Sol
> dataset[,"cielo"]
[1] Sol    Sol    Nubes  Lluvia Lluvia Lluvia Nubes  Sol    Sol
Levels: Lluvia Nubes Sol
```

Estas operaciones extraen las columnas en forma de vector. También se pueden usar otras instrucciones que filtren el *data frame* como por ejemplo [ ](corchetes sin coma) o la función subset( ).

```
> subset(dataset,cielo=="Sol")
cielo temperatura
1   Sol          24
2   Sol          23
8   Sol          25
9   Sol          23
```

Valores perdidos

En estadística es frecuente la situación de no disponer de todos los datos, bien por no poder recolectarlos o bien porque se queden algunas preguntas de una encuesta sin responder por desconocimiento de las respuestas. Otra característica de R es su capacidad para tratar este tipo de valores. En R los valores ausentes se simbolizan mediante **NA** (*Not Available*) [11], [28]. Es importante no confundir estos valores ni con los NULL, que representan variables no inicializadas, ni con los NaN (*Not a Number*). NA representa un valor existente pero que no conocemos.

La función is.NA( ) nos permite conocer si los valores de una variable son NA. Además, muchas funciones cuyas operaciones matemáticas no toleran la existencia de NAs, admiten como parámetro de entrada un booleano llamado na.rm que indica si se eliminan los NAs.

```
> vector<-c(2,3,NA,4)
> is.na(vector)
[1] FALSE FALSE  TRUE FALSE
> mean(vector)
[1] NA
> mean(vector,na.rm=TRUE)
[1] 3
```

Funciones

Para crear una función se utiliza la sentencia function y la estructura es la siguiente:

```
nombreFuncion<-function(argumento1=numero.argumento1,....,argumenton=numero.
    argumenton){
#cuerpo de la función
return(variable.a.devolver )
}
```

Se utiliza el operador = en los argumentos de entrada para asignarles valores por defecto en caso de que no se indique en la llamada a la función. Como se ha mencionado anteriormente en el Subpárrafo 3.1.3, si se quiere devolver más de un objeto, la forma es uniéndolos en una lista y devolviendo esa lista.

En el siguiente ejemplo se define una función que sirve para calcular las operaciones de suma, resta, multiplicación y división sobre un número y almacenarlas en un objeto de tipo lista.

```
operaciones<-function(numero1,numero2=1){
sum<-numero1+numero2
res<-numero1-numero2
mul<-numero1*numero2
div<-numero1/numero2
resultado<-list(suma=sum,resta=res,multiplicacion=mul,division=div)
return(resultado)
}
```

Si realizamos una llamada a la función se pueden ver los resultados. En caso de que el segundo número fuera 1, no necesitaríamos pasar este parámetro a la función, ya que es su valor por defecto.

```
> operaciones(4,2)
$suma
[1] 6

$resta
[1] 2

$multiplicacion
[1] 8

$division
[1] 2
```

Si una función no tiene sentencia return( ) devolverá la última operación realizada.

La siguiente tabla es una recopilación de las funciones de R más utilizadas, estas funciones pertenecen al paquete base [29]. El paquete base es el que viene instalado con R por defecto. Adicionalmente se pueden instalar tantos paquetes como se desee mediante el comando install.packages("CRAN"), la tabla 3.3 muestra una descripción de los paquetes que se utilizan en este proyecto.

rbind( )	Une dos data frames por filas.
cbind( )	Une dos data frames por columnas.
ncol( )	Devuelve el número de columnas de un data frame.
nrow( )	Devuelve el número de filas de un data frame.
read.csv( )	Leer un archivo de tipo .csv.
is.na( )	Devuelve un objeto del mismo tipo que el que se le pasa como parámetro (vector, matriz, data frame, lista) que contiene valores lógicos TRUE en las posiciones donde hay NAs y FALSE donde no los hay.
names( )	Devuelve los nombres de una lista o de las columnas de un data frame
rm ( )	Elimina un objeto.

Tabla 3.2: Funciones más utilizadas del paquete base

RWeka [30]	Es una interfaz en R para Weka. Weka es una colección de algoritmos de aprendizaje automático para tareas de minería de datos escrita en Java, contiene herramientas para el pre-procesado de datos, clasificación, regresión, agrupamiento y reglas de asociación y visualización.
FSelector [14]	Paquete que incluye funciones para la selección de variables de un conjunto de datos determinado.
arules [31]	Proporciona la infraestructura para la representación, manipulación y análisis de datos y patrones. En este trabajo se usa para discretizar variables continuas.
ggplot2 [32]	Una implementación de la gramática de gráficos en R. Combina las ventajas de los gráficos de base y de latice. Es una herramienta muy potente que permite representar todo tipo de gráficos y adecuarlos a las necesidades del usuario mediante un sistema de condiciones.
VIM [10]	Paquete para inspección visual de *data frames* con datos perdidos. Adicionalmente dispone de varios métodos de imputación hotdeck entre los que se encuentra kNN.
foreach [33]	Paquete que da soporte a la construcción de bucles de tipo foreach. Los bucles foreach permiten realizar iteraciones sobre un conjunto de elementos sin la necesidad de emplear para ello un contador explícito. En este sentido, su ejecución es similar al de la función lapply(), pero sin requerir una función de evaluación. Su uso facilita la computación en paralelo.
doParallel [34]	Proporciona la infraestructura necesaria para la función %dopar% utilizando para ello el paquete *parallel*. Esta función se emplea en equipos con procesadores de varios núcleos para llevar a cabo la computación en paralelo.
robustbase [35]	Conjunto de herramientas básicas para el análisis estadístico robusto. Incluye metodología de regresión, selección de modelos y estadísticos multivariable.
stargazer [36]	Produce código LaTeX, así como HTML y CSS para formar tablas, o para mostrar resultados de análisis regresivo.

Tabla 3.3: Paquetes de R utilizados.

3.2 Diseño e implementación del experimento

3.2.1 Planteamiento

El diseño de la fase experimental del proyecto se plantea de la siguiente forma. En la primera etapa, a la que llamaremos fase de pre-procesado, se eliminan aquellos proyectos de peor calidad. También se descartan manualmente las variables utilizadas para el filtrado y aquellas que se consideran poco relacionadas con la variable esfuerzo o que presentan más de un 40 % de datos perdidos. A continuación, se borran los proyectos que contienen datos ausentes, quedándonos así con un *dataset* completo de 621 muestras. La segunda fase, es la de la selección de las características clave, en esta fase, además de descartar otras variables, también se elegirá el método de discretización más adecuado para la variable de clase (*Normalised.Work.Effort.Level.1*) y el número de clases escogido en el proceso de discretización. Posteriormente se seleccionan dos muestras aleatorias de 100 y 150 casos, donde se generan datos perdidos artificialmente según 2 mecanismos, 5 patrones, 7 porcentajes de datos ausentes y se repite el proceso 50 veces. De esta forma se generan un total de 7000 *datasets*. Finalmente se imputan los datos ausentes de los *data frames* incompletos mediante CMI, MINI y kNN[1]. Como los datos ausentes se han generado de forma artificial, se utilizarán los conjuntos de datos originales, junto a los *datasets* donde se ha

[1]En el experimento se utiliza un número de vecino k=2, el mismo número que en la publicación [2], tanto para kNN, como para MINI.

llevado a cabo la imputación para el cálculo del MMRE y del error de imputación nominal en el análisis de los resultados. Hay una descripción general del planteamiento del experimento en el diagrama de la figura 3.2.

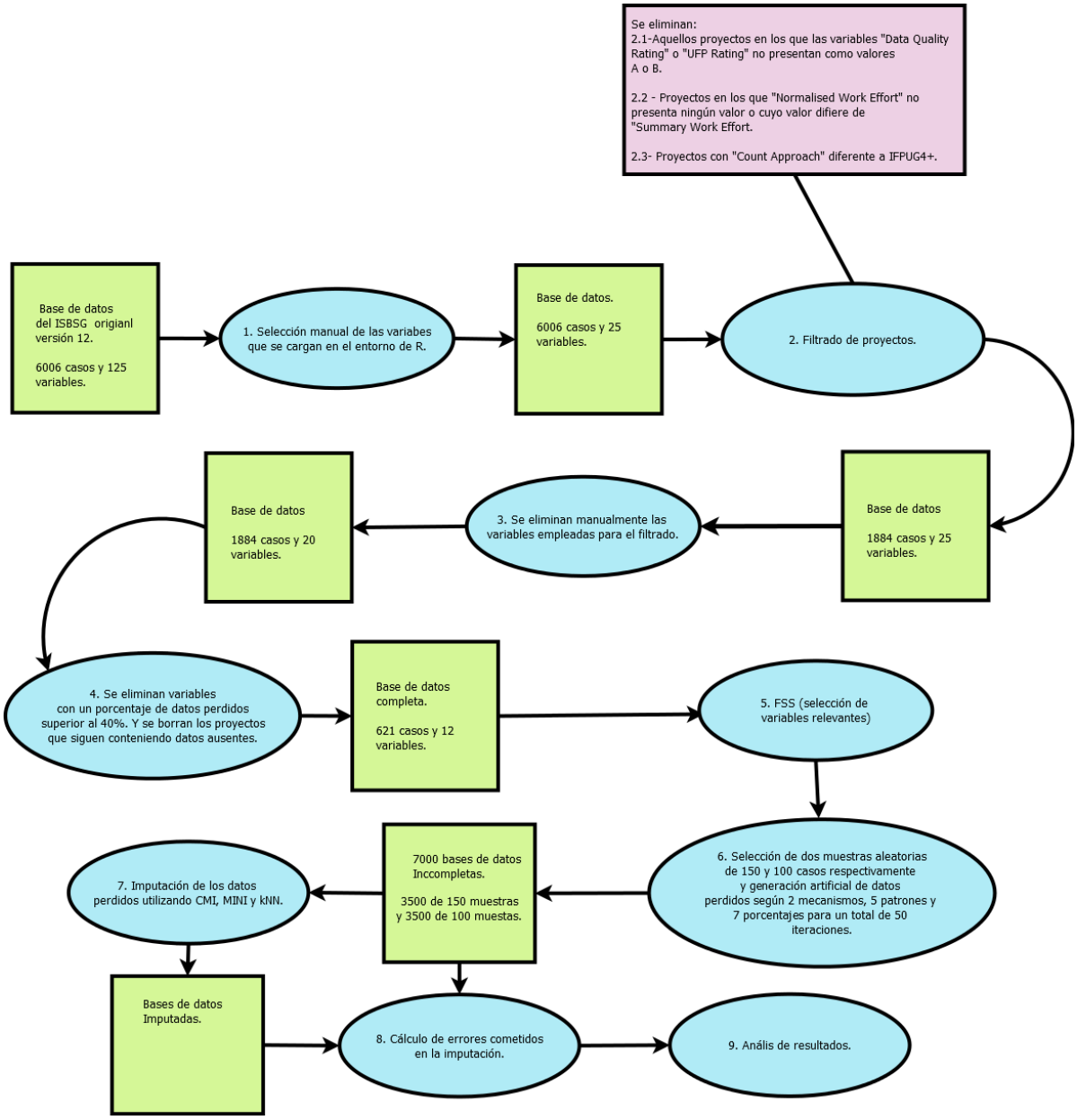


Figura 3.2: Esquema del proceso

A la hora de generar los conjuntos de datos incompletos se tienen en cuenta las siguientes consideraciones:

- Mecanismos de datos perdidos (se utilizan los mismos mecanismos de datos ausentes que en [5] y en [2])

  MCAR: Se inducen datos perdidos aleatoriamente en la variable o las variables deseadas.

  MAR: La implementación del mecanismo MAR está basada en el tamaño del proyecto, cuánto mayor es el proyecto, mayor es la probabilidad de que contenga datos perdidos. Primero se ordenan los casos de acuerdo a la variable *Functional.Size* y se divide el *dataset* en cuatro partes, sobre las que se generarán diferentes porcentajes de *missing data*. El porcentaje vendrá dado en proporción a la media del tamaño de los proyectos de cada parte. Para la parte i-ésima el porcentaje de datos perdidos será de $\frac{M_i}{\sum_{i=1}^{4} M_i} \times p\,\% \times n$ donde M_i es la media del tamaño del proyecto de la í-esima parte, $p\,\%$ es el porcentaje total de datos perdidos, y n es el número de casos en el *dataset*.

- Patrones de las pérdidas

  Se generan en total 5 patrones: univariante, en una variable continua y una nominal; y bivariante, en dos variables continuas, dos nominales, y una continua y otra nominal. Al patrón en el que se generan datos ausentes en una variable nominal y una continua lo denominaremos patrón de pérdida mixto.

- Porcentajes de *missing data*

  Para cada patrón y para cada mecanismo de datos perdidos se generan siete porcentajes diferentes de datos perdidos: 10 %, 15 %, 20 %, 25 %, 30 %, 35 % y 40 %.

3.2.2 Pre-procesado de los datos

La base de datos de la que partimos es la versión 12 del repositorio de proyectos ISBSG. Esta base de datos consta de 6006 casos y 125 variables por proyecto. Ya que dentro de la base de datos, la información de la que se dispone no presenta el mismo grado de fiabilidad, y con el objetivo de partir de un *dataset*, que aunque más reducido, esté completo, y compuesto únicamente por casos cuya información presente una calidad mínima, se utilizaran unos scripts proporcionados por los tutores de este TFG (González Ladrón de Guevara y Fernández Diego [7], [8]), para el pre-procesado de los datos. Todos estos scripts se encuentran coloreados de verde botella en el diagrama de la figura 3.3.

El script principal es esem2016_pre.R. Este script llama a import_ISBSG.R y a filter_projects.R. En import_ISBSG.R se carga la base de datos con los 6006 casos y se eliminan manualmente las variables que se consideran irrelevantes hasta quedarnos con un total de 25. Posteriormente, en filter_projects.R se eliminan los proyectos cuyas variable *Data.Quality.Rating* y *UFP.rating* no presentan cómo valores o "A" o "B"[2]. También se eliminan los proyectos que no tienen ningún valor en la variable *Normalised.Work.Effort.Level.1*[3] y aquellos en los que el valor de esta variable difiere del de *Summary.Work.Effort*. También en los que la variable *Count.Aprroach* no toma como valor IFPUG 4+. En este punto nos encontramos con 1884 casos y 25 variables. Se descartan las variables que han sido necesarias para el filtrado de los datos,

[2] La calidad de los datos se encuentra clasificada en un ranking que puede tomar cuatro valores, de "A" a "D".
[3] Véase la subsección 3.1.1 y el Apéndice A para una información detallada de las variables.

quedándonos con 20. Finalmente se borran aquellas características que tienen un porcentaje de NAs superior al 40 % mediante el método elimina.var.con.demasiados.NAs( ). Así nos quedamos con las 12 variables que se pueden ver en tabla 3.4.

Industry.Sector
Application.Group
Development.Type
Development.Platform
Language.Type
Primary.Programing.Language
Functional.Size
Adjusted.Functio.Points
Normalised.Work.Effort.Level.1
Project.Elapsed.Time
X1st.Data.Base.System
Used Methodology

Tabla 3.4: Variables después del proceso de filtrado inicial

Estas serán las variables de las que partamos para implementar la clasificación de los datos, y de las cuales descartemos aquellas consideradas como menos relevantes. Finalmente se eliminan los casos que contienen valores perdidos para así poder trabajar con una base de datos completa en la cual se pueda medir la efectividad de los métodos de imputación. En este punto nos encontramos con un *data frame* de 621 casos.

3.2.3 Selección de las características clave o Key Feature Extraction

El algoritmo 1 [2], propuesto para la selección de las variables relevantes es muy parecido al algoritmo ID3 [16], no obstante, como nuestra base de datos presenta tanto variables continuas como categóricas e ID3 únicamente trabaja con variables nominales, se ha optado por utilizar C4.5 [17], que es muy similar, pero que en vez de trabajar con ganancia de información utiliza el ratio de ganancia (para más detalles ver la subsección 2.1.2 y la subsección 2.1.2). Los nodos en un árbol de decisión corresponden a las características seleccionadas y las hojas a sus valores asociados. La función utilizada para implementar C4.5 es J48 del paquete *RWeka* [30]. *RWeka* es una interfaz para R de la famosa plataforma de software para el aprendizaje automático y la minería de datos *Weka*, escrita en Java y desarrollada en la Universidad de Waikato. J48 tiene como entradas: una fórmula que modeliza la clasificación de los datos, los datos, la acción a realizar con los datos perdidos y una variable de control. Mediante el parámetro de control se pueden tomar diversas acciones, cómo impedir la poda, impedir el crecimiento de subárboles o establecer un número mínimo de instancias por hoja. La opción para establecer el número mínimo de instancias es la única que utilizaremos, y lo haremos con el objetivo de limitar el tamaño del árbol. Este parámetro se establece en la llamada a la función, dentro del parámetro Weka_Control, con el comando -M seguido del número mínimo de instancias por hoja que queremos establecer.

Al aplicar el algoritmo C4.5 a la variable *Normalised.Work.Effort.Level.1* nos encontramos ante el problema de que es una variable continua. Una opción sería utilizar el algoritmo M5[4] [16]. No obstante, como nuestro objetivo es la selección de variables relevantes y no la clasificación se opta por discretizar la variable de clase. La elección del método de discretización, el valor de M y del número de intervalos o clases influirá notablemente en la selección de las variables relevan-

[4]Esta opción es descartada ya que se generan árboles muy reducidos, en los que casi todo el proceso de clasificación se lleva a cabo mediante los modelos regresivos de las hojas (véase la subsección 2.1.2 para más información).

tes [37] (véase la tabla 4.1). Para ello utilizaremos dos métodos de clasificación no supervisada (discretización por mismo intervalo y discretización por misa frecuencia) y un método de clasificación supervisada (discretización por K-means). K-means es un algoritmo de agrupamiento (*clustering*), que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. El proceso de discretización se lleva a cabo utilizando el paquete de R *arules* [31].

A la hora de seleccionar el número óptimo de clases, el método de discretización y el valor del parámetro M buscaremos un compromiso entre complejidad del árbol de decisión y bondad de la clasificación, es decir, intentaremos crear un árbol lo menos complejo posible, y que clasifique correctamente el mayor número de casos. Para eso atenderemos a tres parámetros de la función evaluate_Weka_classifier( ) del paquete *RWeka* [30], El coeficiente kappa y el error absoluto relativo y el error absoluto medio.

El Coeficiente kappa de Cohen es una medida estadística que ajusta el efecto del azar en la proporción de la concordancia observada para elementos cualitativos

la ecuación para kappa es:

$$kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \tag{3.1}$$

Donde $Pr(a)$ es el acierto relativo en la clasificación y $Pr(e)$ es la probabilidad hipotética de acierto por azar. El valor óptimo de kappa es 1 y se da cuando $Pr(a) = 1$ y $Pr(e) = 0$. Y el peor valor que puede presentar esta medida es 0 y se da cuando el acierto en la clasificación a partir de las reglas del árbol de decisión es el mismo que si clasificaremos los casos aleatoriamente.

El error absoluto relativo o RAE se define como:

$$RAE = \frac{\sum_{i=1}^{N} \left|\widehat{\theta}_i - \theta_i\right|}{\sum_{i=1}^{N} \left|\overline{\theta} - \theta_i\right|} \tag{3.2}$$

Donde θ_i es la clase a la que pertenece el caso i, $\widehat{\theta}_i$ el resultado de la clasificación y $\overline{\theta}$ el valor medio de θ.

Y el error absoluto medio o MAE es:

$$MAE = \sum_{i=1}^{N} \left|\widehat{\theta}_i - \theta_i\right| \tag{3.3}$$

El script FSS.R (color naranja en el diagrama de la figura 3.3) genera diversos árboles de de decisión, para diferentes números de clases y distintos valores de M, utilizando para ello la función Feature.Subset.Selection( ). Esta función nos devuelve un *data frame* que contiene los valores de kappa (coeficiente kappa de Cohen), RAE (*Relative Absolute Error*), MAE (*Mean Absolute Error*), RMSE (*Root Mean Squared Error*) y RRAE (*Root Relative Absolute Error*).

3.2.4 Imputación de los datos

Medida de las Distancias

La forma de medir las distancias es un tema de vital importancia tanto a la hora de clasificar los datos, como a la hora de buscar los vecinos.

[11] Utiliza la distancia Gower para la función kNN que implementa. En [2] utilizan un tipo de distancia similar a la Manhattan que definen ellos mismos. En este trabajo, con el objetivo de ser rigurosos en la comparación entre MINI y kNN y por ser una distancia más extendida y reconocida se opta por utilizar la Gower para ambos algoritmos. Dicha distancia se define de la siguiente forma:

Distancia entre valores continuos: $D(x_{ij}, x_{i'j})$ es la distancia entre x_{ij} y $x_{i'j}$ $(1 \leq i, i' \leq n)$. Donde se supone continua a la j-ésima característica $F_j(j = 1, 2, \cdots, p)$ del conjunto de datos $\mathbf{X}$

$$D(x_{ij}, x_{i'j}) = \frac{1}{max\{x_{rj}\} - min\{x_{rj}\}} |x_{ij} - x_{i'j}| \quad (r = 1, 2, \cdots, n) \tag{3.4}$$

Media de valores continuos de una clase

$$M_j^c = \frac{1}{\|C_c\|} \sum_{r=1}^{n} x_{r,j}^c \tag{3.5}$$

Donde $\|C_c\|$ es el número de casos en la clase C_c; $c \in \{1, 2, \cdots, Nc\}$; $x_{r,j}^c$ indica que el valor de la variable $x_{r,j}$ pertenece a la clase C_c.

Distancia entre valores nominales: $d(x_{ij}, x_{i'j})$ es la distancia entre x_{ij} y $x_{i'j}$ $(1 \leq i, i' \leq n)$. Donde se supone categórica a la j-ésima característica $F_j(j = 1, 2, \cdots, p)$ del conjunto de datos $\mathbf{X}$

$$d(x_{ij}, x_{i'j}) = \begin{cases} 1 & \text{si } x_{ij} \neq x_{i'j} \\ 0 & \text{si } x_{ij} \equiv x_{i'j} \end{cases} \tag{3.6}$$

Moda de valores nominales

$$M_j^c = \frac{1}{\|C_c\|} \sum_{r=1}^{n} x_{r,j}^c \tag{3.7}$$

donde

$$M_j^c = \text{máx}_{1 \leq i \leq n}\{P(x_{ij}^c)\}$$

Distancia generalizada entre dos casos. Para dos casos cualquiera $X_i, X_j \subset \mathbf{X}(ij = 1, 2, \cdots, \hat{} \, i \neq j)$ la distancia entre ellos $D(X_i, X_j)$ es la siguiente:

$$D(X_i, X_j) = \frac{\sum_{k=1}^{p} w_k \cdot \delta_k(i,j) \cdot |x_{i,k} - x_{j,k}|}{\sum_{k=1}^{p} w_k \cdot \delta_k(i,j)} \tag{3.8}$$

donde

$$|x_{i,k} - x_{j,k}| = \begin{cases} D(x_{ij}, x_{i'j}) & \text{si la característica } k \text{ es continua} \\ d(x_{ij}, x_{i'j}) & \text{si la característica } k \text{ es nominal} \end{cases}$$

El término w es un vector de pesos que permite dar más importancia a unas variables sobre otras, en este trabajo no se utiliza. Y el componente $\delta_k(i,j)$ es 0 cuando se desconoce el valor de la variable $x_{i,k}$, el de $x_{j,k}$ o el de ambas.

Distancia generalizada entre un caso y una clase, Siendo $D(i,c)$ la distancia entre el caso con valores perdidos X_i y la clase $C_c(c = 1, 2, \cdots, Nc)$. Se define como:

$$D(i,c) = \frac{1}{NoCc} \sum_{j \in NoCc} D(X_i, X_j) \tag{3.9}$$

Dónde $c \in \{1, 2, \cdots, Nc\}$, X_j es un caso que pertenece a la clase C_c y $NoCc$ es el número de casos de la clase C_c.

Algoritmo de imputación

El algoritmo MINI se describe en el algoritmo 2.

3.2.5 Resolución de conflictos asociados a la computación en paralelo

En la función min.dis.caso.clase( ) del Apéndice B se ha probado primero a realizar la computación en paralelo en el cálculo de las distancias entre el caso a clasificar y todos los casos de una clase y repetir este proceso de forma secuencial para todas las clases. Posteriormente se ha probado a calcular las distancias entre el caso a clasificar y cada clase en paralelo, y calcular las distancias entre casos de forma secuencial, reduciéndose el tiempo de ejecución de la función de imputación en más de la mitad. En general en R, cuando trabajamos con bucles anidados es preferible paralelizar el bucle externo [38], aunque no siempre es así, en determinadas ocasiones podría ser conveniente paralelizar el interno debido a inconsistencias en el balanceo entre el número de procesos y el coste computacional de los mismos.

3.2.6 Descripción del código fuente empleado

El código fuente se encuentra estructurado en 9 scripts. El principal es Experimento.R, desde el cual se llama a esem2016_pre.R para cargar la base de datos y eliminar los casos de peor calidad y variables no relevantes mediante métodos de filtrado. Posteriormente se seleccionan aquellas variables de entre las restantes en el script FSS.R utilizando como método de envoltura para el proceso de *Feature Subset Selection* el algoritmo C4.5, en este script también se elegirá el método más adecuado para discretizar la variable de clase y se escogerá el número de clases más adecuado. Funciones.R y Funciones_preliminares.R contienen funciones utilizadas por el

Algoritmo 2 Algoritmo MINI. `Fuente:` A New Imputation Method for Small Software Project Data Sets [2]

Entrada: Conjunto de datos $\widehat{\mathbf{X}}$ con la variable de clase, Características Clave f_k
Salida: valor estimado de los valores perdidos x_{ij}.

1: Busca en el *dataset* $\widehat{\mathbf{X}}$ los índices de los valores perdidos $S_m = \{(ij)\}^{N_m}$ donde i y j son respectivamente el número de caso y el número de la característica del dato perdido x_{ij} respectivamente, y N_m es el número de valores perdidos.
2: **para** cada $ij \in Sm$ **hacer**
3: **para** cada clase $c \in C_k$ **hacer**
4: Utilizar la fórmula 3.9 para calcular la distancia $D(i, c)$ entre el caso X_i y la clase c utilizando para dicho cálculo las características clave
5: **fin para**
6: $d(i, c) = \text{mín}_{1 \leq c \leq Nc}\{D(i, c)\}$
7: Busca el conjunto de vecino DoN en la clase c mediante el método de kNN
8: **si** F_j es una variable continua **entonces**
9: Utilizar la fórmula 3.5 para calcular la media M_j^c de la característica j del conjunto de vecinos.
10: **si no**
11: F_j es una variable categórica
12: Utilizar la fórmula 3.7 para calcular la moda M_j^c de la característica j del conjunto de vecinos.
13: **fin si**
14: $\widehat{x}_{ij} = M_j^c$
15: $X_m \cup \widehat{x}_{ij}$
16: **fin para**
17: **devolver** X_m

script Experimento.R y por FSS.R. Por último, Resultados.R carga un *data frame* de salida de Experimento.R. y se utiliza para generar tablas y gráficos a partir de los resultados obtenidos.

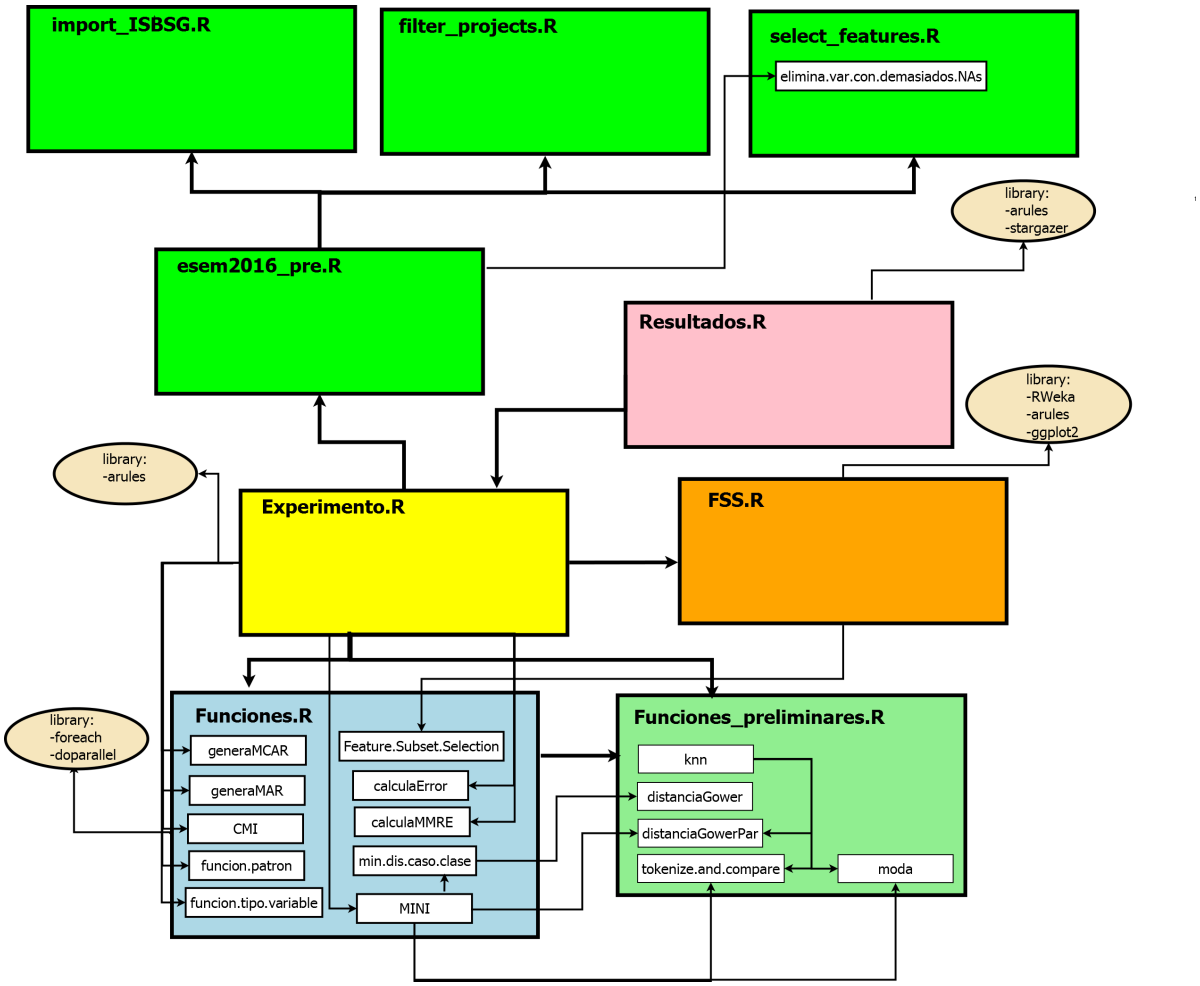


Figura 3.3: Diagrama del código utilizado

En el diagrama de la figura 3.3 se puede ver una representación de todos los scripts utilizados en el proyecto. En él, cada rectángulo coloreado representa un script. Las funciones se representan utilizando rectángulos en blanco dentro de cada script. Los paquetes de R utilizados por cada script se simbolizan mediante elipses de color carne. Las dependencias entre los scripts se han representado utilizando flechas gruesas. Y las llamadas a las funciones con flechas finas.

El script Funciones.R (sección B.1) es llamado por Experimento.R. Las funciones generaMCAR( ) y generaMAR( ) son utilizadas para simular datos perdidos según los mecanismos MCAR y MAR (véase la subsección 2.2.2) respectivamente en la variable o las variables deseadas. Estas dos funciones, junto a MINI, se han realizado modificando funciones previas creadas por Alfonso Polo [11]. Para poder distinguir las líneas de código que se han aportado, estas se encuentran encerradas entre dos comentarios ### G. MINI utiliza la función distanciaGowerPar( ) (distancia Gower en paralelo) para calcular la distancia entre casos, también se sirve de la función min.dis.caso.clase( ), para calcular la clase más próxima según la Ecuación 3.9 . A su vez min.dis.caso.clase( ) utiliza la función distanciaGower( ). CMI es una función que realiza la imputación según el método *Class Mean Imputation*. funcion.tipo.variables( ) sirve para identificar el tipo de variables (Continua, Nominal o Mixta) en una lista que contiene elementos del tipo *character* con variables de la base de datos del ISBSG. funcion.patron( ) nos dice si el patrón es univariante, bivariable o multivariante (más de dos variables). CalculaMMRE( ),

CalculaMdMRE( ) y calculaError( ) son las funciones empleadas para calcular las medidas de error para variables continuas y categóricas respectivamente.

Funciones.R carga el script Funciones_preliminares.R. En este script se encuentran aquellas funciones utilizadas en Experimento.R creadas por Alfonso Polo [11] y que no se han modificado. knn( ) realiza la imputación mediante el algoritmo kNN. Tokenize.and.Compare( ) sirve para calcular la distancia entre variables que presentan varios valores como pueden ser *Organisation Type* o *Application Type*.

Adicionalmente, se crea un paquete de R llamado MINI en el que se añaden las funciones de los scripts Funciones.R y Funciones_preliminares.R (Apéndice B). El objetivo de crear dicho paquete es facilitar la reestructuración del código y su reutilización en el futuro. En la figura 3.4 puede verse una captura de pantalla de la creación de este paquete mediante el entorno de desarrollo RStudio. En esta captura de pantalla también se puede observar el archivo de descripción del paquete, la consola desde la que se carga el paquete mediante el comando library( ), el panel de compilación y el directorio de trabajo.

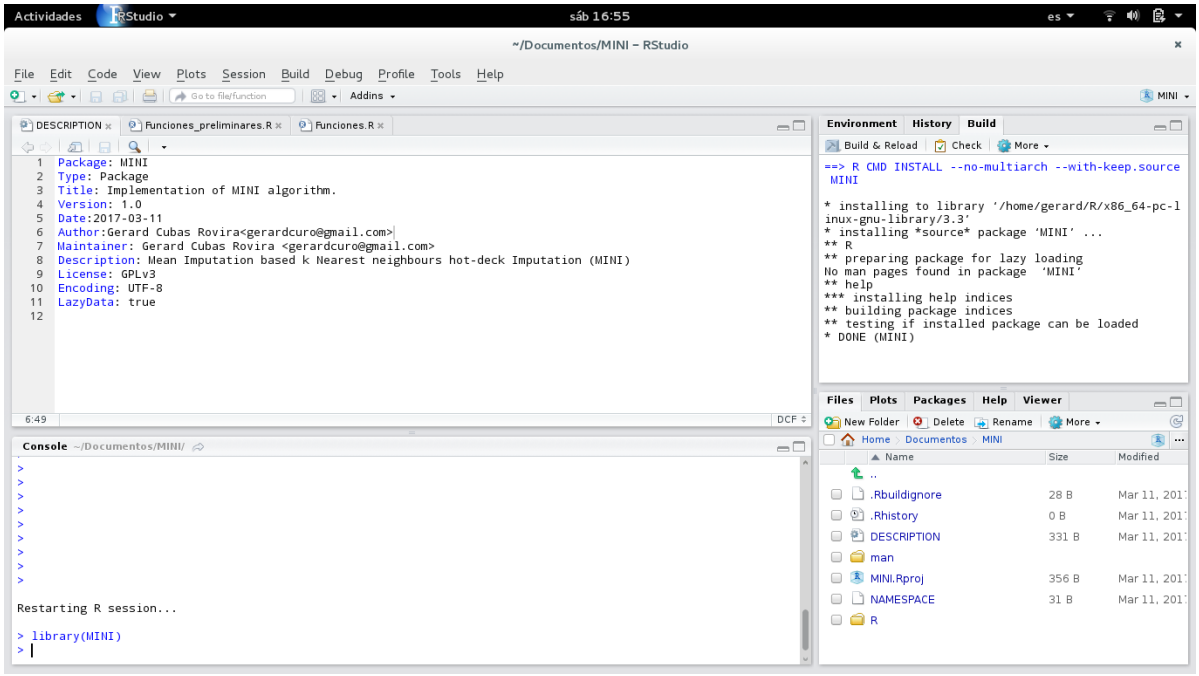


Figura 3.4: Captura de pantalla de la creación del paquete MINI

3.2.7 Medida del error

Para la medida del error en variables continuas se propone en [2] el MMRE (*Mean Magnitude of Relative Error*). El MRE o magnitud de error relativo para una variable imputada en un caso concreto se define como:

$$MRE = \frac{|x_i - \widehat{x_i}|}{x_i} \tag{3.10}$$

siendo $\widehat{x}$ el valor predicho para x

El MMRE se calcula aplicando la media del MRE en los valores imputados. Un problema muy típico del MMRE es que es muy sensible a los *outliers*. Por este motivo también se empleará como medida de error para variables continuas el MdMRE es decir, la mediana del MRE.

Para las variables nominales se calculará el error con la siguiente fórmula:

$$\epsilon = 1 - \frac{N_{cor}}{N_{per}} \tag{3.11}$$

Donde N_{cor} es el número de valores imputados correctamente y N_{per} es el conjunto de valores perdidos.

Capítulo 4

Resultados experimentales

4.1 Selección de variables, número de clases y método de discretización

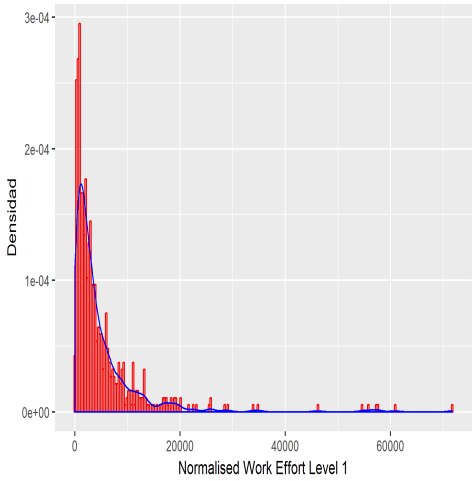
El proceso de selección de variables relevantes, tras el filtrado inicial, se lleva a cabo mediante el algoritmo C4.5 [17] de RWeka [30]. Como ya se ha mencionado previamente en la subsección 3.2.3, para poder lidiar con una variable de clase numérica es necesario aplicar un proceso de discretizado previo. Antes de discretizar la variable se lleva a cabo una transformación logarítmica del esfuerzo normalizado, el objetivo de esta transformación es conseguir una distribución similar a una gaussiana. En la figura 4.1 pueden verse las distribuciones de la variable de clase (*Normalised Work Effort Level 1*) aplicando y sin aplicar la transformación logarítmica.

Posteriormente se realiza una llamada a la función Feature.Subset.Selection( ) con la finalidad de elegir el mejor método de discretización y se presta atención a los valores del coeficiente kappa y de RAE (*Relative Absolute Error*).

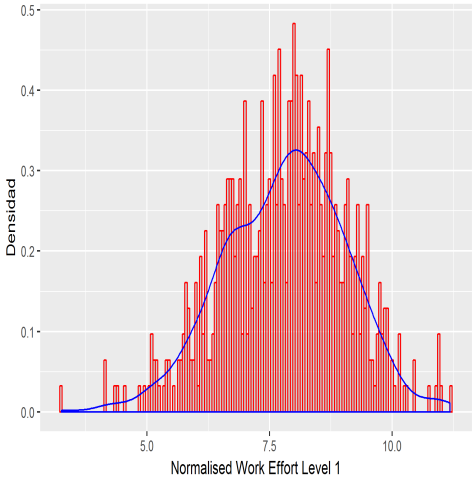
```
ISBSG$Normalised.Work.Effort.Level.1<-log(ISBSG$Normalised.Work.Effort.Level.1)
Output.FSS<-Feature.Subset.Selection(data=ISBSG,metodos=c("frequency","cluster"
    ,"interval"),Ms=1:30,Nc=2:10)
```

El siguiente código, se utiliza para obtener el valor de M (mínimo número de instancias por hoja) que maximiza kappa para el método de discretización de k-means según el número de intervalos de la discretización. Este código se repite para cada uno de los tres métodos de discretización propuestos con el objetivo de obtener los valores de M que maximizan kappa y que minimiza RAE en función del número de clases.

```
Ms<-vector(mode="numeric",length=9)
for(i in 2:10){
subset<-subset(Output.FSS,Numero.Clases==i & Metodo.Discretizacion=="cluster")
Ms[i-1]<-subset$M[ which.max(subset$kappa)]
}
subset<-subset(Output.FSS,Numero.Clases==2:10 & M==Ms & Metodo.Discretizacion==
    "cluster")
```



(a) Histograma de *Normalised Work Effort Level 1* sin transformación logarítmica



(b) Histograma de *Normalised Work Effort Level 1* al cual se le ha aplicado una transformación logarítmica

Figura 4.1: Histogramas de *Normalised Work Effort Level 1*.

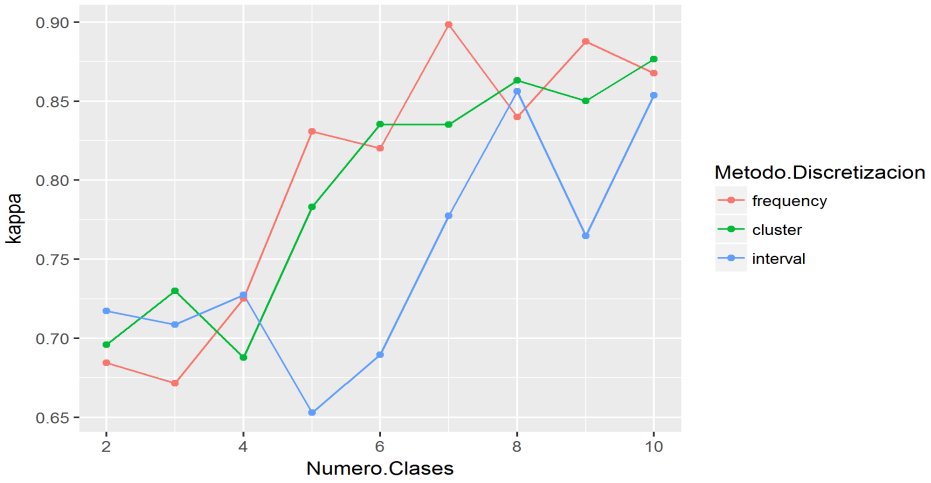


Figura 4.2: Kappa en función del número de clases para los tres métodos de discretización

La figura 4.2 muestra una tendencia creciente del coeficiente kappa de Cohen. Esa tendencia es debida a que al aumentar el número de clases, la probabilidad de clasificar correctamente por azar cada caso disminuye, además, un aumento en el número de intervalos en el proceso de discretización, repercute en una clasificación más precisa.

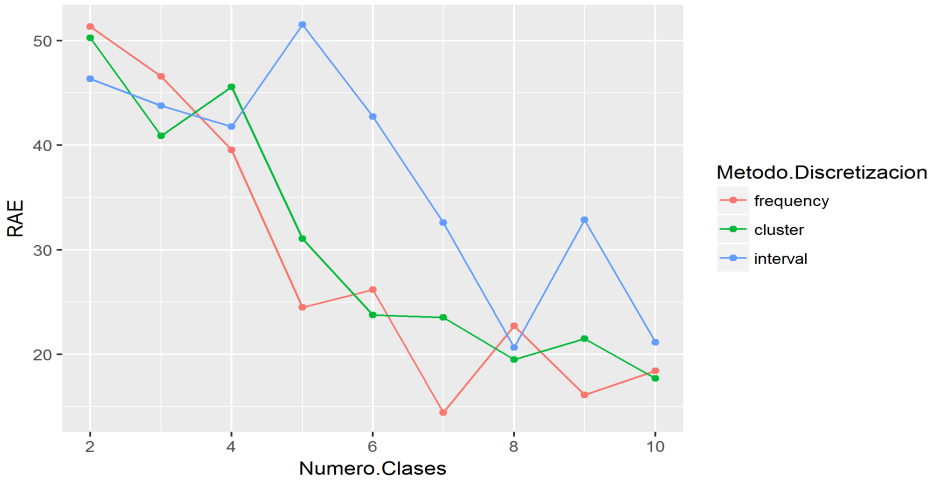


Figura 4.3: RAE en función del número de clases para los tres métodos de discretización

Por otra parte, en la figura 4.3 se puede observar como RAE disminuye con el número de clases. Estos resultados son consecuencia de que al aumentar el número de clases, se producen menos errores en la clasificación, llegando al caso extremo en el que si tuviéramos el mismo número de clases que de casos, existiría una hoja para cada caso, y todos serían clasificados correctamente.

A partir de las figuras 4.2 y 4.3 se puede concluir que el método de discretización que peor se comporta de los tres planteados es el de discretización por intervalo, salvo cuando se utilizan un número de clases $Nc = 2$. También se puede deducir que el método más adecuado será frecuencia, a excepción de cuando se utilicen números de clases $Nc = 6$, $Nc = 8$ o $Nc = 10$, en estos casos el método de discretización que maximiza kappa y minimiza RAE será k-means. Partiendo de estas gráficas, se concluye que el valor óptimo del número de clases es $Nc = 7$ y que el método de discretizado más adecuado es frecuencia ya que son los valores que maximizan kappa y minimizan RAE.

El siguiente código sirve para seleccionar las variables relevantes o *Key Features*, es una llamada al algoritmo C4.5 [17] (función J48 de RWeka [14]) habiendo discretizado previamente el esfuerzo normalizado en siete clases por frecuencia, utilizando para ello la función discretize( ) del paquete *arules* [31]. Posteriormente se hace una llamada a java utilizando el código rJava::.jcall para extraer las variables consideradas como nodos del árbol de decisión, serán estas variables con las que nos quedemos.

```
ISBSGd<-ISBSG
ISBSGd$Normalised.Work.Effort.Level.1<-discretize(ISBSG$Normalised.Work.Effort.
    Level.1, method="frequency", categories=7)
m<- J48(Normalised.Work.Effort.Level.1~., data = ISBSGd, control = Weka_control
    (M=5))
x <- rJava::.jcall(m$classifier, "S", "graph")
y <- parse_Weka_digraph(x, plainleaf = TRUE)
Features<-unique(y$nodes[,2][y$nodes[,2]!=""])
Features<-c("Normalised.Work.Effort.Level.1",Features)
print(Features)
```

M	IS	AG	DT	DP	LT	PPL	FS	AFP	PET	1DBS	UM	Hojas	MAE
1	x	x	x	x	x	x	x	x	x	x	x	1070	0.035
5	x		x	x	x	x	x	x	x	x	x	210	0.155
10	x			x	x	x	x	x	x			89	0.193
15					x	x	x	x	x			105	0.180
20						x	x	x	x			66	0.198
25						x	x	x				65	0.200
30						x	x	x	x			67	0.194

Tabla 4.1: Variables independientes seleccionadas para el método de discretización frecuencia, para 7 clases y para diferentes valores de M y número de hojas y MAE (*Mean Absolute Error*) del árbol generado.

En la tabla 4.1, se pueden ver las variables independientes seleccionadas fijando diferentes valores de M para el método de discretización por frecuencia. Aunque podrían ser descartadas hasta 8 variables, si atendemos al criterio de minimizar MAE (*Mean Absolute Error*), seleccionaremos un valor de $M = 5$, descartartando así la variable independiente *Application Group*, y nos quedaremos con las variables de la tabla 4.2.

Industry.Sector
Development.Type
Development.Platform
Language.Type
Primary.Programing.Language
Functional.Size
Adjusted.Functio.Points
Normalised.Work.Effort.Level.1
Project.Elapsed.Time
X1st.Data.Base.System
Used Methodology

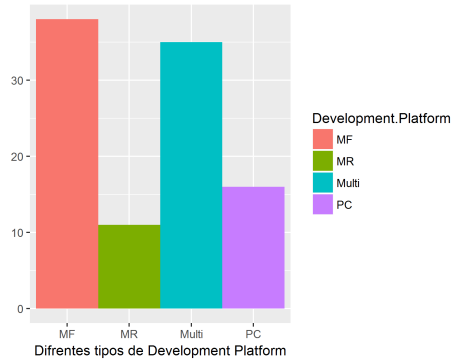
Tabla 4.2: Variables después del proceso de FSS

4.2 Análisis descriptivo de variables para los datasets ISBSG100 e ISBSG150

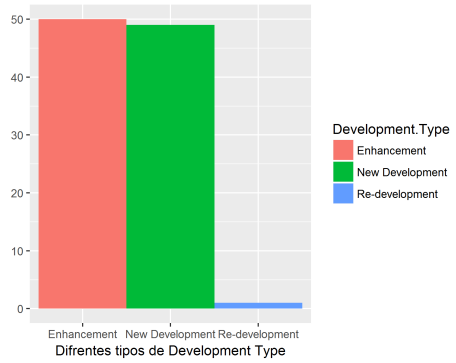
4.2.1 Variables sobre las cuales se generan datos perdidos

Las variables nominales escogidas para llevar a cabo el análisis son *Development Type* y *Development Platform*. En las figuras 4.4b y 4.4a puede apreciarse la distribución de dichas variables en el *dataset* de 100 casos, y en las figuras 4.5b y 4.5a en el *dataset* de 150. El hecho de que no estén distribuidas uniformemente repercutirá posteriormente en los resultados obtenidos.

Las variables continuas sobre las que se generan datos perdidos son *Project Elapsed Time* y *Adjusted Function Points*. La información general de estas dos variables en los conjuntos de datos ISBSG100 e ISBSG150 se detalla en la tabla 4.3 donde la columna N representa el tamaño del *dataset*.

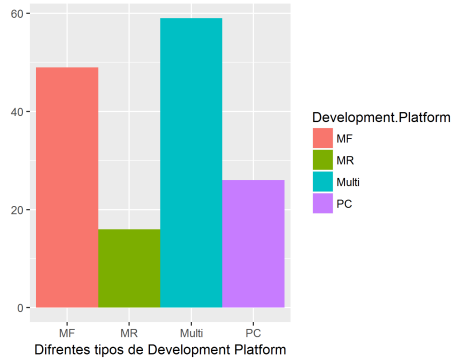


(a) Histograma de la distribución de la variable "Development Platform".

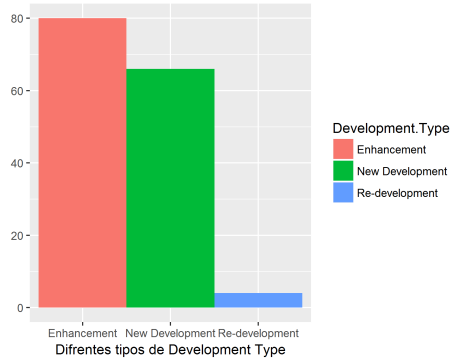


(b) Histograma de la distribución de la variable "Development Type".

Figura 4.4: Distribuciones de DP y DT en ISBSG100.



(a) Histograma de la distribución de la variables "Development Platform".



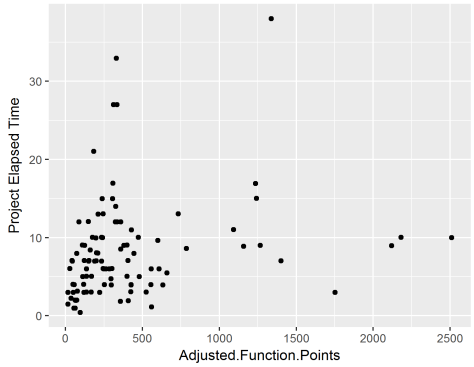
(b) Histograma de la distribución de la variables "Development Type".

Figura 4.5: Distribuciones de DP y DT en ISBSG150.

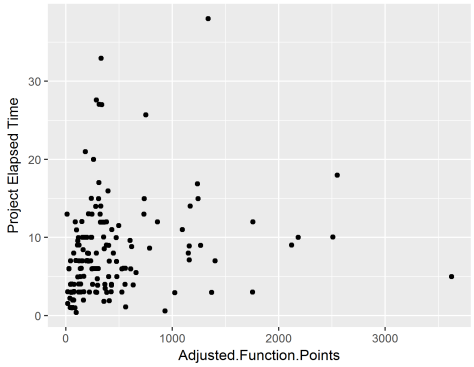
Variable	N	Media	Desv. Típica	Min	Max
Adjusted.Function.Points	100	406.560	471.319	16	2,509
Adjusted.Function.Points	150	444.967	545.650	10	3,622
Project.Elapsed.Time	100	8.088	6.306	0.400	38.000
Project.Elapsed.Time	150	8.180	6.212	0.400	38.000

Tabla 4.3: Variables continuas imputadas

Las figuras 4.6a y 4.6b representan diagramas de dispersión de las variables continuas imputadas. En ellas se puede advertir la presencia de *outliers* [23], [24] del tipo multivariable. Estos valores atípicos son bastante problemáticos para llevar a cabo la imputación, sobre todo para los algoritmos kNN y MINI debido a que su presencia no afecta únicamente a la hora de llevar a cabo la estimación de la media, sino que también afecta a las correlaciones entre las variables y supone un problema a la hora de escoger los vecinos para llevar a cabo la imputación. Además, estos *outliers* representan proyectos reales, y su eliminación nos proporcionaría resultados sesgados, por lo que no es aconsejable borrarlos.



(a) Gráfico de dispersión del *dataset* de 100 casos.

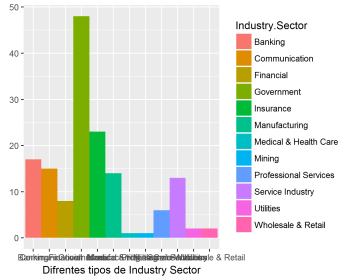


(b) Gráfico de dispersión del *dataset* de 150 casos.

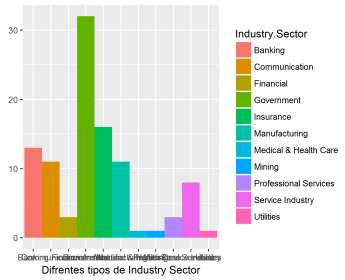
Figura 4.6: Gráficos de dispersión de las variables PET y AFP para ISBSG100 e ISBSG150.

4.2.2 Variables independientes

Las variables nominales sobre las que no se generan datos perdidos, pero que si se utilizan en el cálculo de las distancias para escoger los vecinos en el algoritmo de kNN, y para clasificar los casos y escoger los vecinos en el algoritmo MINI son: *Industry Sector*(figura 4.7), *Language Type*(figura 4.8), *Primary Programming Language*(figura 4.9), *X1st Data Base System*(figura 4.10) y *Used Methodology*(figura 4.11).

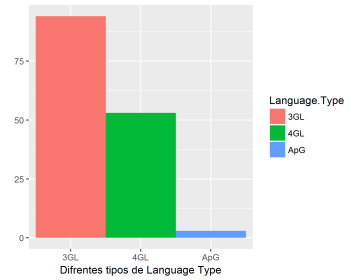


(a) Histograma de la distribución de la variable "Industry Sector" en ISBSG150.

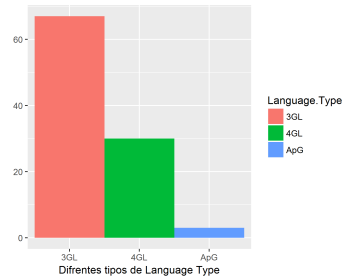


(b) Histograma de la distribución de la variable "Industry Sector" en ISBSG100.

Figura 4.7: Distribuciones de "Industry Sector" en conjuntos de datos de 150 y 100 casos.

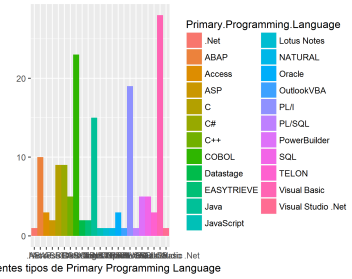


(a) Histograma de la distribución de la variable "Language Type" en ISBSG150.

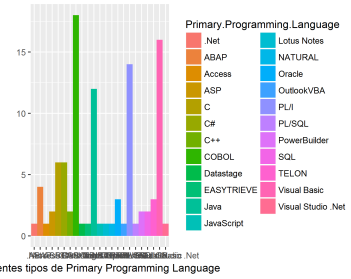


(b) Histograma de la distribución de la variable "Language Type" en ISBSG100.

Figura 4.8: Distribuciones de "Language Type" en conjuntos de datos de 150 y 100 casos.

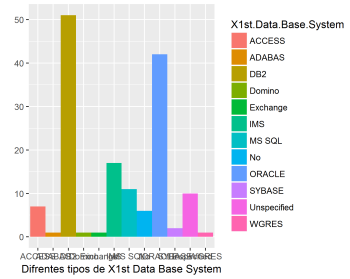


(a) Histograma de la distribución de la variable "Primary Programming Language" en ISBSG150.

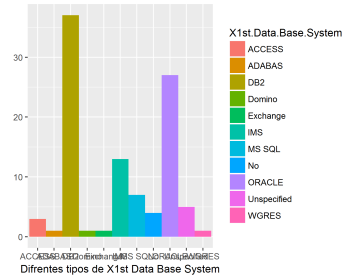


(b) Histograma de la distribución de la variable "Primary Programming Language" en ISBSG100.

Figura 4.9: Distribuciones de "Primary Programming Language" en conjuntos de datos de 150 y 100 casos.

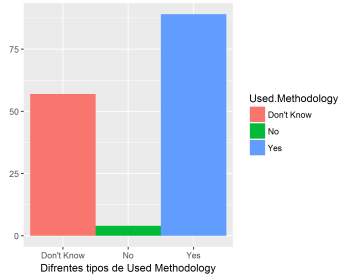


(a) Histograma de la distribución de la variable "X1st Data Base System" en ISBSG150.

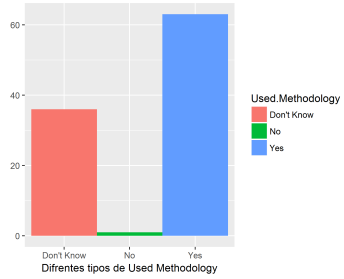


(b) Histograma de la distribución de la variable "X1st Data Base System" en ISBSG100.

Figura 4.10: Distribuciones de "X1st Data Base System" en conjuntos de datos de 150 y 100 casos.



(a) Histograma de la distribución de la variable "Used Methodology" en ISBSG150.



(b) Histograma de la distribución de la variables "Used Methodology" en ISBSG100.

Figura 4.11: Distribuciones de "Used Methodology" en conjuntos de datos de 150 y 100 casos.

Las variables continuas sobre las que no se generan datos perdidos, pero que si se utilizan en el cálculo de las distancias para los algoritmos kNN y MINI son: *Functional.Size* y *Normalised.Work.Effort*. En la tabla 4.4 se puede ver un resumen de las características de estas variables (media, desviación típica, mínimo y máximo) para los conjuntos de datos de 100 y 150 casos.

Variable	N	Media	Desv. Típica	Min	Max
Functional.Size	100	401.370	469.887	16	2,509
Functional.Size	150	438.947	541.210	10	3,622
Normalised.Work.Effort.Level.1	100	4,936.660	7,792.218	144	71,729
Normalised.Work.Effort.Level.1	150	5,158.653	7,450.430	62	71,729

Tabla 4.4: Variables continuas sobre las que no se generan datos perdidos.

4.3 Comparación de los diferentes métodos de imputación

4.3.1 Patrones de pérdida univariante y bivariante

Los resultados de las siguientes tablas se obtienen aplicando la media aritmética a los diferentes MMREs y errores de imputación nominal para el conjunto de casos indicado en cada apartado. Por ejemplo, el valor de la primera fila y la tercera columna de la tabla 4.5 es el resultado de aplicar la media del error de imputación nominal de todos los conjuntos de datos donde se ha generado un 10 % de datos ausentes y se ha imputado mediante CMI, es decir, tanto generando datos perdidos según mecanismos MCAR y MAR, y patrones de datos ausentes univariante y bivariante, y para cada una de las 50 iteraciones del experimento.

La tabla 4.5 muestra que al aumentar el porcentaje de datos perdidos, no aumenta siempre el MMRE para las variables continuas, y se debe a que a que el MRE está directamente relacionado con las posiciones de las celdas donde se han generado los datos ausentes, es decir, penaliza la imputación de valores originales bajos sobre valores originales altos. Por otro lado, para las variables nominales, al aumentar el porcentaje de datos perdidos, sí se aprecia una tendencia creciente en el error de imputación nominal. Esta predisposición puede verse en los tres algoritmos propuestos, aunque se advierte con mayor claridad en MINI y kNN.

Tipo de Variable	Porcentaje de datos ausentes	Método de imputación		
		CMI	MINI	kNN
nominal	10(%)	0.508	0.447	0.370
	15 (%)	0.504	0.470	0.378
	20 (%)	0.512	0.490	0.392
	25 (%)	0.514	0.505	0.408
	30 (%)	0.529	0.532	0.421
	35 (%)	0.543	0.547	0.436
	40 (%)	0.561	0.557	0.451
continua	10 (%)	0.842	0.965	0.765
	15 (%)	0.856	1.016	0.808
	20 (%)	0.854	1.033	0.819
	25 (%)	0.853	1.039	0.899
	30 (%)	0.882	1.056	0.866
	35 (%)	0.868	1.053	0.864
	40 (%)	0.874	1.039	0.833

Tabla 4.5: MMREs y errores de imputación nominal de los tres métodos para diferentes porcentajes de datos ausentes.

A partir de los resultados obtenidos se concluye que el algoritmo de imputación que mejores resultados presenta en términos del MMRE y del error de imputación nominal es kNN, seguido por MINI para variables nominales y por CMI para continuas.

Tipo de variable	Tamaño del dataset	CMI	MINI	kNN
nominal	100	0.474	0.505	0.420
	150	0.575	0.509	0.396
continua	100	0.802	0.959	0.859
	150	0.921	1.098	0.813

Tabla 4.6: MMREs y errores de imputación nominal de los tres métodos para *datasets* de distintos tamaños.

En la tabla 4.6 se puede observar, al contrario de lo esperado, que a partir de conjuntos de datos con mayor número de casos, la precisión de la imputación disminuye para los algoritmos CMI y MINI. De nuevo se puede apreciar que el error más bajo corresponde a kNN, seguido por CMI, excepto para el conjunto de 150 casos y para variables nominales, en cuyo caso, el segundo algoritmo que mejor se comporta es MINI. Los resultados se deben a que ISBSG150 contiene una proporción más elevada de *outliers* que ISBSG100.

Tipo de variable	Número de variables con datos ausentes	CMI	MINI	kNN
nominal	1	0.561	0.533	0.436
	2	0.488	0.480	0.380
continua	1	0.888	1.050	0.876
	2	0.835	1.007	0.796

Tabla 4.7: MMREs y errores de imputación nominal para diferentes números de variables con datos perdidos.

En la tabla 4.7 se ve que al distribuir el número de datos ausentes entre dos variables, existen más casos con la variable a imputar conocida, y que por tanto pueden actuar como donantes,

mostrando así una mejoría en la precisión de la imputación al distribuir entre varias variables el número de datos ausentes.

Tipo de Variable	Mecanismo	CMI	MINI	kNN
nominal	MCAR	0.526	0.465	0.386
	MAR	0.522	0.549	0.430
continua	MCAR	1.082	1.280	1.094
	MAR	0.641	0.777	0.579

Tabla 4.8: MMREs y errores de imputación nominal para tabla con diferentes mecanismos datos ausentes

A partir de la tabla 4.8 se puede deducir que el error de imputación nominal es ligeramente inferior para el mecanismo MCAR. Mientras que para las variables continuas, el MMRE es considerablemente más bajo para el mecanismo MAR. La explicación de estos resultados es que el mecanismo MAR genera un mayor porcentaje de datos perdidos en los proyectos donde la variable *Functional Size* toma valores elevados, y las variables continuas que se imputan, es decir, *Adjusted Function Points* y *Project Elapsed Time*, presentan un alto grado de correlación con dicha variable. Por lo tanto, al generar un mayor número de datos ausentes en los casos en los que la variable a imputar toma valores más altos, el MRE de dichas imputaciones tenderá a ser más bajo que el MRE resultado de generar datos perdidos aleatoriamente a lo largo del *dataset*, recordemos que el MRE penaliza la imputación de valores bajos de la variable con respecto a los altos.

Otra medida de error es la tasa de crecimiento del error de imputación nominal γ , que refleja el aumento del error conforme el número de casos disminuye. γ se define de la siguiente forma:

$$\gamma = \frac{V_c - V_r}{V_r} \tag{4.1}$$

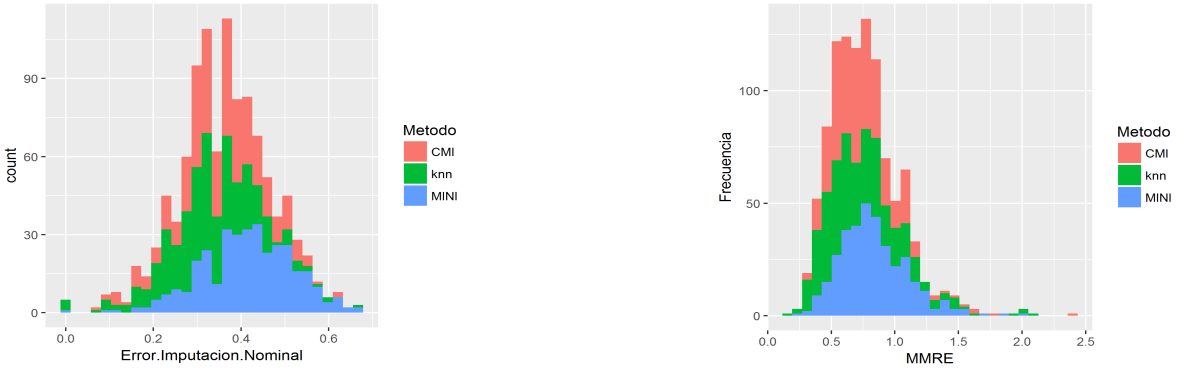
Donde V_c es el valor del error para variables nominales del método que se pretende comparar para el conjunto de datos de la comparación y V_r es el error para variables nominales del mismo método para el *dataset* de referencia. El objetivo es investigar el cambio del error de imputación nominal que se produce cuando el número de casos del *dataset* disminuye. Para eso se toma ISBSG150 como *dataset* de referencia y se compara con ISBSG100.

La tabla 4.9 es una representación de la tasa de crecimiento del error de imputación nominal para los tres algoritmos de imputación, para los mecanismos de datos perdidos MCAR y MAR, y para los patrones de datos perdidos univariante y bivariante. En ella se puede observar como aumenta el error en kNN y MINI cuando el mecanismo de datos perdidos es MCAR. En el caso de CMI disminuye para este mismo mecanismo. Cuando se generan datos perdidos según el mecanismo MAR, el error disminuye para todos los casos menos para el del patrón de datos perdidos bivariante en combinación con kNN.

Mecanismo	Patrón	Método	p10	p15	p20	p25	p30	p35	p40
MCAR	Univariate	CMI	-15.8	-16.8	-10.9	-13.6	-11.3	-11.0	-11.2
		MINI	16.6	5.0	14.5	10.0	7.3	6.3	0.1
		knn	19.9	4.7	8.9	18.5	14.7	15.0	12.7
	Bivariate	CMI	-22.3	-25.7	-23.7	-16.6	-19.7	-19.2	-15.7
		MINI	11.5	-4.9	0.5	11.8	9.6	2.5	3.3
		knn	4.4	3.2	10.7	15.9	11.1	9.6	10.7
MAR	Univariate	CMI	-18.0	-22.5	-24.5	-20.8	-13.9	-7.9	3.9
		MINI	-8.5	-5.9	-18.6	-12.6	-6.3	-7.8	-4.1
		knn	0.1	-1.8	-3.8	-2.3	-5.0	0.2	7.0
	Bivariate	CMI	-33.3	-29.4	-27.3	-27.7	-22.3	-15.3	-7.1
		MINI	-4.2	0.5	-9.6	-2.8	-3.8	-6.2	1.4
		knn	4.7	4.0	-2.0	6.7	1.7	5.7	5.2

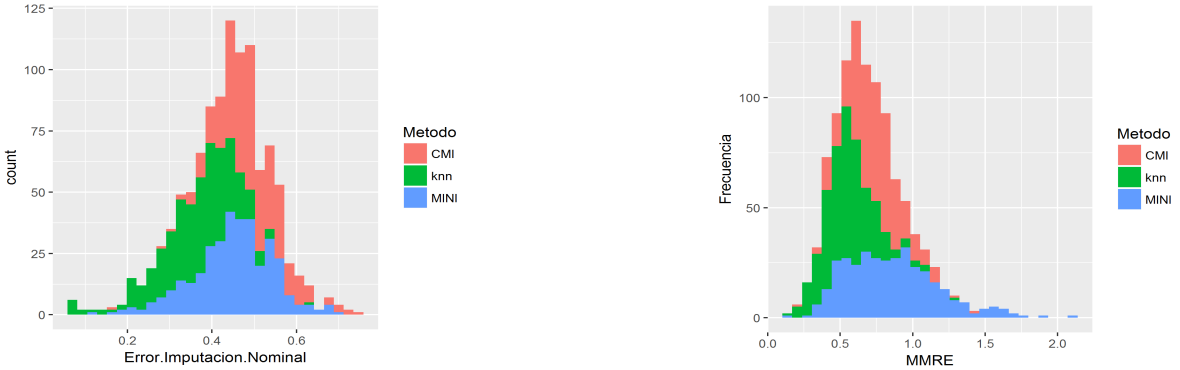
Tabla 4.9: Tasa de crecimiento del error de imputación nominal para una y dos variables

4.3.2 Patrón de pérdida mixto



(a) Histograma del error de imputación nominal. **(b)** Histograma del MMRE.

Figura 4.12: Histogramas del error nominal para *datasets* de 100 muestras según el mecanismo MCAR.

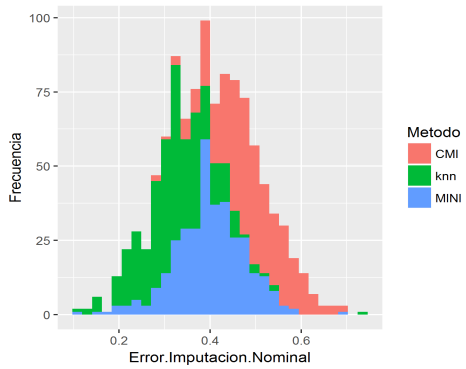


(a) Histograma del error de imputación nominal. **(b)** Histograma del MMRE.

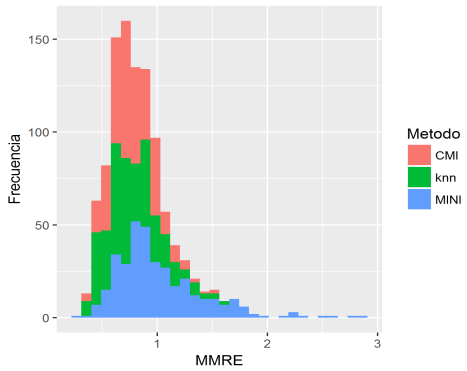
Figura 4.13: Histogramas del error nominal para *datasets* de 100 muestras según el mecanismo MAR.

Las figuras 4.12, 4.13, 4.14 y 4.15 representan histogramas del MMRE y del error de imputación nominal del patrón de pérdida mixto (variables *Project Elapsed Time* y *Development*

Type), para los dos conjuntos de datos de diferentes tamaños y para los mecanismos de *missing data* MAR y MCAR.

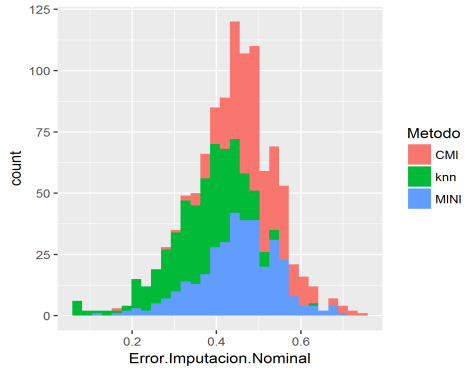


(a) Histograma del error de imputación nominal.

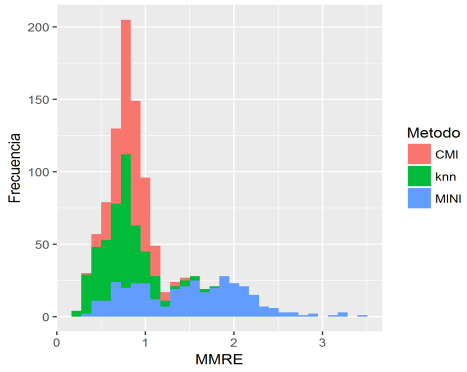


(b) Histograma del MMRE.

Figura 4.14: Histogramas del error nominal para *datasets* de 150 muestras según el mecanismo MCAR.



(a) Histograma del error de imputación nominal.



(b) Histograma del MMRE.

Figura 4.15: Histogramas del error nominal para *datasets* de 150 muestras según el mecanismo MAR.

En la figura 4.14 se analiza el resultado de generar datos perdidos en el *dataset* de 150 casos según un mecanismo MCAR y posteriormente imputar los datos faltantes mediante los tres algoritmos planteados. En este histograma se observa que el error de imputación nominal más bajo en la mayoría de los casos es para kNN, y el más alto es para MINI, mientras que el MMRE es similar para los tres algoritmos propuestos. El histograma de la figura 4.15 representa el mismo conjunto de datos anterior para el mismo patrón, pero según un mecanismo de *missing data* MAR. En estas figuras la distribución del MMRE para el algoritmo MINI oscila entre 0.3 y 3.5, para kNN lo hace entre 0.2 y 1.4 y para CMI entre 0.4 y 1.4. La mayoría de los MREs tanto de CMI como de kNN son de 0.5 y 0.4 y siguen una distribución que podría categorizarse como normal. Los errores de imputación nominal más bajos son para kNN, que destaca por presentar los mejores resultados de imputación, para este tipo de variables MINI se comporta ligeramente mejor que CMI.

4.4 Revisión y perfeccionamiento del algoritmo CMI

El algoritmo CMI (*Class Mean imputation*) [5] se basa en clasificar los casos de la población de estudio, y posteriormente imputar los datos perdidos utilizando para ello la media para variables continuas y la moda en el caso de las nominales como se ha descrito en la subsección 2.2.6. No obstante, como estamos trabajando con conjuntos de datos heterogéneos con una gran presencia de *outliers*, se realiza una segunda prueba experimental en la que se modifica este método sustituyendo la imputación a través de la media por otros estimadores más robustos y menos sensibles a la presencia de valores atípicos [24] (véase la subsección 2.3.2). Estos estimadores robustos consisten en la mediana y la M de Huber.

Existen dos motivos principales para la aplicación de estos estimadores robustos al algoritmo CMI y no hacerlo para kNN ni para MINI: El primero es que los estimadores únicamente mejoran los resultados en el caso de la presencia de *outliers* univariantes, y CMI únicamente precisa de la variable a imputar y la variable de clase para llevar a cabo la imputación. Mientras que tanto kNN como MINI necesitan de un conjunto de variables más amplio, que se utilicen para calcular la clase más cercana en el caso de MINI, y para seleccionar los vecinos en ambos. El segundo motivo y el principal para realizar las pruebas con estimadores robustos solo en CMI es que al utilizar un número de vecinos $k = 2$, la aplicación de la media, la mediana, y la M de Huber para dos valores nos proporcionará los mismos resultados.

Tanto la mediana cómo la M de Huber, se utilizan para remplazar al estimador de la media y como consecuencia únicamente se podrán utilizar para imputar variables continuas. El cálculo de la M de Huber se realiza mediante el paquete de R *robustbase* [35] y la función huberM( ), para el cálculo de la mediana se utiliza la función median( ) del paquete *base* [29].

En este experimento adicional, el procedimiento es el mismo que en el experimento principal, se realiza la prueba de aplicar CMI, imputando con la media, moda y M de Huber para un total de 200 iteraciones. En esta prueba se realiza el cuádruple de iteraciones debido a que el tiempo de ejecución de CMI es considerablemente inferior al de MINI o kNN. Los porcentajes de datos perdidos, patrones de *missing data* y mecanismos de datos ausentes no cambian.

Tipo de Variable	Porcentaje de datos ausentes	Método de imputación		
		Media	Mediana	M de Huber
continua	10 (%)	0.870	0.676	0.720
	15 (%)	0.849	0.674	0.715
	20 (%)	0.862	0.679	0.723
	25 (%)	0.861	0.677	0.722
	30 (%)	0.860	0.680	0.725
	35 (%)	0.875	0.701	0.747
	40 (%)	0.895	0.718	0.766

Tabla 4.10: MMREs de CMI utilizando los estimadores media, mediana y M de Huber para diferentes porcentajes de datos ausentes.

Los resultados de la tabla 4.10 y de la tabla 4.11 muestran que el mejor estimador de los tres para el algoritmo CMI es la mediana, seguido de la M de Huber, y por último la media (el estimador original del algoritmo). Estas mejoras se producirán indistintamente del porcentaje de *missing data* y serán mucho más pronunciadas para el mecanismo MCAR que para el MAR. Además, si comparamos estos resultados con los de la tabla 4.5 o los de la tabla 4.8 se puede apreciar que combinando CMI con el estimador de la mediana se obtienen mejores resultados

que incluso para el algoritmo kNN. Siendo así este nuevo método, al que bautizamos como *Class Median Imputation* el que mejores resultados proporciona para la imputación de variables continuas en conjuntos de datos pequeños.

Tipo de Variable	Mecanismo	Método de imputación		
		Media	Mediana	M de Huber
continua	MCAR	1.095	0.766	0.849
	MAR	0.640	0.607	0.614

Tabla 4.11: MMREs de CMI utilizando los estimadores media, mediana y M de Huber para mecanismos de datos perdidos MAR y MCAR.

4.5 Coste computacional de los algoritmos

Equipo	Lenovo G500
Memoria RAM	8 GB
Procesador	Intel Core i7-3612QM2.10 GHz
Sistema operativo	Debian 8 Jessie
versión de R	3.2.2
versión de Rstudio	1.0.336

Tabla 4.12: Especificaciones del equipo de trabajo y del Software empleado.

En la tabla 4.12 se puede observar el hardware y el software que se ha utilizado para llevar a cabo el proyecto. La figura 4.16 muestra el tiempo de ejecución medio en segundos de cada uno de los tres algoritmos utilizados, es importante destacar que esta medida es una media y obviamente el tiempo de ejecución es superior para porcentajes de datos ausentes más altos. En el gráfico de barras se aprecia que el coste computacional de CMI es mucho más bajo que el de los otros dos algoritmos, incluso parece que el tiempo de ejecución sea de 0 segundos, no obstante, es del orden de milisegundos. MINI es ligeramente más costosos computacionalmente que kNN, como era de esperar.

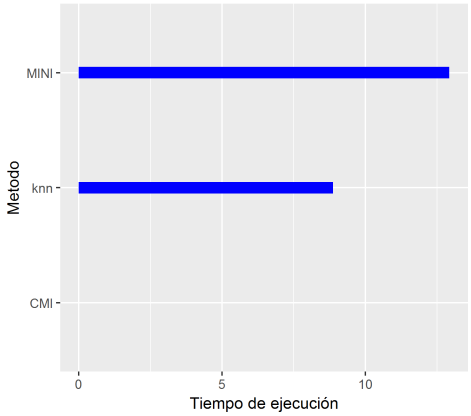


Figura 4.16: Tiempo de ejecución para CMI, kNN y MINI en segundos

Capítulo 5

Conclusiones, limitaciones y futuro trabajo

5.1 Conclusiones

En la actualidad existe gran variedad de técnicas para abordar el problema de los datos faltantes. Dependiendo de la situación a la que nos enfrentemos deberemos escoger la que mejor se adapte a nuestras necesidades. Por ejemplo, en el caso de una encuesta en la que se haya perdido los datos en el proceso de recolección, antes que imputar los datos ausentes, la mejor solución consiste en intentar contactar con los participantes y repetir las preguntas. Si reducimos el problema a las situaciones en la que es necesario imputar los datos perdidos, o a aquellas situaciones en las que no siendo necesario, la imputación es la mejor opción, los métodos reconocidos como más efectivos son las técnicas de múltiple imputación o las de Máxima Similitud [20] (*Maximum likelihood*). No obstante, estos métodos se comportan mejor con grandes bases de datos y además requieren hacer presuposiciones sobre la distribución de los datos que no tienen por qué cumplirse. El repositorio del ISBSG es una base de datos de naturaleza heterogénea con alta presencia de *outliers* y el comportamiento de cada variable es considerablemente diferente, motivo por el cuál es difícil realizar estas presuposiciones.

El principal objetivo de este proyecto era implementar el algoritmo MINI y analizar su comportamiento, en la sección B.1 se encuentra la función que implementa el algoritmo y su comportamiento se analiza en el capítulo 4. El objetivo secundario consistía en la selección de variables relevantes y la elección del número de clases, el objetivo secundario se ha abordado utilizando el algoritmo C4.5 [17], este algoritmo trata de clasificar un conjunto de datos de entrenamiento en base al ratio de ganancia, de manera que se van dividiendo los datos y creando subconjuntos en forma de árbol de decisión. A partir del árbol de decisión se puede obtener una versión simplificada de los datos donde se pueden distinguir las variables más relevantes. Aplicando C4.5, se ha llegado a la conclusión de que el método de discretización de la variable de clase que reporta un árbol de decisión cuya clasificación es más precisa es el de frecuencia y que el número de intervalos que mejores resultados proporciona es siete. Y se han seleccionado como variables relevantes las siguientes: Industry Sector, Development Type, Development Platform, Langua-

ge Type, Primary Programing Language, Functional Size, Adjusted Functio Points, Normalised Work Effort Level 1, Project Elapsed Time, X1st Data Base System y Used Methodology.

El caso concreto de estudio en este trabajo es el de la imputación de valores perdidos en subconjuntos reducidos (100 y 150 casos) de la base de datos del ISBSG. Para abordar el problema se ha replicado el algoritmo MINI, propuesto en [2]. Este método se ha comparado con los algoritmos kNN y CMI [5]. La investigación del comportamiento de estos algoritmos se ha llevado a cabo escogiendo conjuntos de datos que no presentaban ningún valor perdido. Posteriormente se ha simulado la presencia de datos ausentes para diferentes porcentajes de perdida, para patrones univariante y bivariante, en variables tanto continuas, nominales o mezcla de ambas y utilizando los mecanismos de datos perdidos MCAR y MAR. A continuación, se han imputado los datos faltantes y se han comparado los valores obtenidos con los originales, utilizando como medidas de error el MMRE y el error de imputación nominal. Con el objetivo de evitar que los resultados fueran poco representativos del comportamiento general de los algoritmos propuestos se ha repetido el experimento 50 veces.

La principal aportación de MINI según sus autores originales es la reducción de la varianza. No obstante, si establecemos como criterio de selección del mejor algoritmo de imputación la reducción del MMRE para variables continuas y la reducción del error para nominales, a partir de los resultados obtenidos se concluye que el mejor de los tres métodos es kNN para la mayoría de situaciones de estudio.

Adicionalmente se han realizado pruebas del algoritmo CMI utilizando estimadores robustos y se ha llegado a la conclusión de que si modificamos el algoritmo *Class Mean Imputation* (algoritmo que apenas se ha estudiado) por un nuevo algoritmo, al que llamaremos *Class Median Imputation* será éste el que ofrezca los mejores resultados de los tres planteados en términos del MMRE para variables continuas.

5.2 Limitaciones

Es importante destacar que la generación del árbol de decisión con el conjunto de datos completo sobre el que posteriormente generamos datos perdidos nos ofrece una posición de ventaja. En caso de que la perdida de los datos no fuese simulada, la generación del árbol tendría que hacerse con un *dataset* incompleto, produciéndose como consecuencia peores resultados.

En la publicación [2] proponen el algoritmo MINI y al igual que en este proyecto lo comparan con CMI y kNN. Ha sido una publicación muy citada, pero cuyo algoritmo no se ha replicado muchas veces. Sus resultados revelan que MINI es ligeramente superior a los otros dos algoritmos planteados. Aun desconociendo las circunstancias por las cuales en su estudio MINI ofrece los mejores resultados frente a kNN y CMI, se sugieren diferentes motivos por los que sus resultados experimentales son distintos a los obtenidos en este trabajo:

1. En el artículo [2] no aportan información del número de clases que utilizan para llevar a cabo los algoritmos, ni del método utilizado para clasificar los datos, así como del proceso de discretización de las variables continuas. En esta publicación tampoco aportan información acerca de las variables sobre las que llevan a cabo los procesos de imputación. Y los resultados serán completamente diferentes dependiendo de las variables sobre las que se generen datos perdidos. Por ejemplo, la variable nominal *Language Type* puede tomar tres estados, mientras que *Primmary Programming Language* puede tomar 23, motivo que hará que el segundo atributo sea mucho más difícil de imputar.

2. En la publicación [2] mencionan que utilizan el algoritmo *Class Mean Imputation* pero apenas hablan de él y no describen su funcionamiento. Casi con total certeza no aplican estimadores robustos para su implementación.

3. El cálculo de la ganancia de información y del ratio de ganancia es un proceso muy delicado, sobre todo para el caso del tratamiento de variables continuas, por ejemplo, en una prueba realizada en la fase inicial del proyecto se ve como los paquetes FSeletor [14] y RWeka [30] del repositorio de R CRAN proporcionan resultados diferentes para el mismo cálculo de la ganancia de información con el mismo conjunto de datos.

4. El algoritmo C4.5 [17] trabaja con ratio de ganancia, y el algoritmo de selección de variables relevantes de MINI trabaja con ganancia de información. Podría haberse empleado ID3, pero no tolera variables continuas, por lo que se ha optado por utilizar C4.5. No obstante, en la publicación [2] utilizan ganancia de información a la hora de describir el algoritmo de FSS.

5. En la publicación [2] realizan el experimento un total de 5 iteraciones, éstas son muy pocas y el algoritmo podría no haber llegado a converger, siendo así los resultados poco representativos del comportamiento natural de los algoritmos.

5.3 Líneas de futuro trabajo

En este proyecto se ha comparado los algoritmos MINI, kNN y CMI en términos del MMRE para variables cuantitativas y del error para atributos cualitativos a la hora de imputar datos perdidos en *datasets*. Futuros trabajos tendrán en cuenta otras medidas de error para consagrar los resultados obtenidos. Estos resultados muestran que el mejor método para el tratamiento de variables nominales es kNN, mientras que para las continuas es CMI modificado (*Class Median Imputation*), en el cual se sustituye la media por el estimador robusto de la mediana. Futuros proyectos que se centren en el estudio de los mismos algoritmos, podrían contrastar las conclusiones obtenidas empleando otros bancos de datos. Así mismo, también podría replicarse el experimento utilizando diferentes números de vecinos, y otros métodos para la selección de variables relevantes, como por ejemplo los árboles de decisión ID3 [16] o CART. En el caso de ID3 será necesario aplicar un proceso de discretizado previo a todas las variables continuas.

Otro aspecto a mejorar es el análisis de *outliers* del tipo multivariable [23] y la decisión de si estos casos reales deberían ser eliminados, y en caso de una respuesta afirmativa, bajo qué circunstancias se tendrían que eliminar.

Los scripts utilizados pueden optimizarse. Un problema, es que los algoritmos de imputación (MINI, kNN y CMI) solo están pensados para imputar una única variable simultáneamente, por lo que las llamadas a las funciones de imputación se han de realizar diversas veces. Un aspecto a mejorar en los scripts es la imputación en paralelo de varias variables.

Una sugerencia para futuras investigaciones es que puede resultar de interés ponderar las variables en el cálculo de las distancias entre casos para los algoritmos kNN y MINI, utilizando por ejemplo para ello la ganancia de información o el ratio de ganancia [13] de cada variable con respecto a la variable a imputar.

Este estudio se centra en imputar datos ausentes en variables que no son el esfuerzo, con el objetivo de contar con bases de datos completas que puedan ser utilizadas para estimar posteriormente el esfuerzo y mejorar el rendimiento, la planificación y la productividad en proyectos

de software. Por lo que futuras investigaciones que se centren en esta temática, así como casos reales de aplicación de empresas u organizaciones que pretendan hacer una estimación del coste de sus proyectos podrán utilizar este trabajo como referente.

s

Referencias bibliográficas

[1] N. Landwehr, M. Hall y E. Frank, «Logistic Model Trees», *Machine Learning*, vol. 59, n.º 1, págs. 161-205, 2005, ISSN: 1573-0565. DOI: 10.1007/s10994-005-0466-3 (vid. pág. 13).

[2] Q. Song y M. Shepperd, «A new imputation method for small software project data sets», *Journal of Systems and Software*, vol. 80, n.º 1, págs. 51-62, 2007, ISSN: 0164-1212. DOI: 10.1016/j.jss.2006.05.003 (vid. págs. 1-3, 7-9, 14, 17, 18, 24, 32, 34, 35, 37, 39, 41, 58, 59).

[3] J. Méndez y R. Morales, *Inteligencia artificial. Técnicas, métodos y aplicaciones*. McGraw-Hill Interamericana de España S.L., 2008, ISBN: 9788448156183 (vid. págs. 2, 25).

[4] P. Jonsson y C. Wohlin, «An evaluation of k-nearest neighbour imputation using Likert data», en *10th International Symposium on Software Metrics, 2004. Proceedings.*, sep. de 2004, págs. 108-118. DOI: 10.1109/METRIC.2004.1357895 (vid. págs. 3, 17, 18).

[5] Q. Song, M. Shepperd y M. Cartwright, «A Short Note on Safest Default Missingness Mechanism Assumptions», *Empirical Software Engineering*, vol. 10, n.º 2, pág. 235, 2005 (vid. págs. 3, 14, 17, 18, 34, 55, 58).

[6] Q. Song, M. Shepperd, X. Chen y J. Liu, «Can k-NN imputation improve the performance of C4.5 with small software project data sets? A comparative evaluation», *Journal of Systems and Software*, vol. 81, n.º 12, págs. 2361-2370, 2008, Best papers from the 2007 Australian Software Engineering Conference (ASWEC 2007), Melbourne, Australia, April 10-13, 2007Australian Software Engineering Conference 2007, ISSN: 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2008.05.008 (vid. págs. 3, 17).

[7] F. González Ladrón de Guevara y M. Fernández Diego, «Potential and limitations of the {ISBSG} dataset in enhancing software engineering research: A mapping review», *Information and Software Technology*, vol. 56, n.º 6, págs. 527-544, 2014, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2014.01.003 (vid. págs. 3, 24, 25, 34).

[8] F. González Ladrón de Guevara, M. Fernández Diego y C. Lokan, «The usage of ISBSG data fields in software effort estimation: A systematic mapping study», *Journal of Systems and Software*, vol. 113, págs. 188-215, 2016. DOI: 10.1016/j.jss.2015.11.040 (vid. págs. 3, 25, 34, 65, 67).

[9] Z. Pan, «Comparison between K-NN, Median and Mean imputation methods to deal with missing data in effort estimation based on regression models», 2015 (vid. págs. 3, 17, 25).

[10] M. Templ, A. Alfons, A. Kowarik y B. Prantner, *CRAN: package VIM*, 201 (vid. págs. 3, 4, 18, 32).

[11] A. P. Serrano, «Evaluación del algoritmo KNN para imputación de datos», 2016 (vid. págs. 4, 17, 18, 30, 37, 40, 41).

[12] M. A. Hall y L. A. Smith, «Practical feature subset selection for machine learning», 1998 (vid. págs. 5, 7).

[13] C. Shannon, «A mathematical theory of communication, bell System technical Journal 27: 379-423 and 623–656», *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948 (vid. págs. 6, 59).

[14] P. Romanski y L. Kotthoff, *CRAN: package FSelector* (vid. págs. 8, 10, 32, 45, 59).

[15] J. G. García, J. P. Albaladejo y J. A. M. Fernández, «Métodos de inferencia estadística con datos faltantes. Estudio de simulación sobre los efectos en las estimaciones», *Estadística española*, vol. 48, n.º 162, págs. 241-270, 2006 (vid. pág. 8).

[16] J. Quinlan, «Induction of Decision Trees», *Machine Learning*, vol. 1, n.º 1, págs. 81-106, 1986, ISSN: 1573-0565. DOI: 10.1023/A:1022643204877 (vid. págs. 9, 35, 59).

[17] ——, *C4.5: Programs for Machine Learning*, ép. Morgan Kaufmann series in machine learning. Elsevier Science, 1993, ISBN: 9781558602380 (vid. págs. 12, 16, 35, 43, 45, 57, 59).

[18] M. E. Yahia y B. A. Ibrahim, «K-nearest neighbor and C4.5 algorithms as data mining methods: advantages and difficulties», en *ACS/IEEE International Conference on Computer Systems and Applications, 2003. Book of Abstracts.*, jul. de 2003, págs. 103-. DOI: 10.1109/AICCSA.2003.1227535 (vid. págs. 13, 17).

[19] R. J. Quinlan, «Learning with Continuous Classes», en *5th Australian Joint Conference on Artificial Intelligence*, Singapore: World Scientific, 1992, págs. 343-348 (vid. pág. 13).

[20] D. B. Rubin, «Inference and missing data», *Biometrika*, págs. 581-592, 1976 (vid. págs. 14, 15, 17, 57).

[21] C. Enders, *Applied Missing Data Analysis*, ép. Methodology in the social sciences. Guilford Press, 2010, ISBN: 9781606236390 (vid. págs. 15-17).

[22] F. M. O. Peinado, «Tratamiento estadístico de outliers y datos faltantes», *Técnicas estadísticas en Nutrición y Salud*, (vid. pág. 18).

[23] J. L. P. D. de los Rios, «Identificación de outliers en muestras multivariantes», Tesis doct., Facultad de Matemáticas de la Universidad de Sevilla (vid. págs. 19, 48, 59).

[24] A. M. Committee, «Robust statistics: a method of coping with outliers», *Royal Society of Chemistry*, 2001 (vid. págs. 19, 20, 48, 55).

[25] «The Comprehensive R Archive Network» (vid. pág. 26).

[26] *GNU General Public License, version 3*, http://www.gnu.org/licenses/gpl.html, Last retrieved 2012-05-10, jun. de 2007 (vid. pág. 26).

[27] R. Stallman, J. Gay y F. Foundation, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Free Software Foundation, 2002, ISBN: 9781882114986 (vid. pág. 26).

[28] N. Matloff, *The art of R programming: A tour of statistical software design*. No Starch Press, 2011 (vid. págs. 26-28, 30).

[29] U. Groemping, B. Amarov y H. Xu, *CRAN: package base*, 0.29, sep. de 2016 (vid. págs. 31, 55).

[30] K. Hornik, *CRAN: package RWeka* (vid. págs. 32, 35, 36, 43, 59).

[31] M. Hahsler, C. Buchta, B. Gruen, K. Hornik y C. Borgelt, *CRAN: package arules* (vid. págs. 32, 36, 45).

[32] H. Wickham y W. Chang, *CRAN: package ggplot2*, 2016 (vid. pág. 32).

[33] R. Calaway y S. Weston, *CRAN: package foreach*, 2015 (vid. pág. 32).

[34] R. Calaway, S. Weston y D. Tenenbaum, *CRAN: package doParallel*, 2015 (vid. pág. 32).

[35] M. Maechler, P. Rousseeuw, C. Croux, V. Todorov, A. Ruckstuhl y M. Salibian-Barrera, *CRAN: package robustase*, 0.92-6, dic. de 2016 (vid. págs. 32, 55).

[36] M. Hlavac, *CRAN: package stargazer*, 2015 (vid. pág. 32).

[37] M. Fernández-Diego y J.-M. Torralba-Martínez, «Discretization methods for nbc in effort estimation: an empirical comparison based on isbsg projects», en *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ACM, 2012, págs. 103-106 (vid. pág. 36).

[38] S. Weston, *Nesting Foreach Loops*, Revolution Analytics, oct. de 2015 (vid. pág. 38).

Apéndice A

ISBSG

Un repositorio de datos de ingeniería de software se define como un conjunto de datos bien definidos, útiles y pertinentes del mundo real relacionados con proyectos de software, llamados *datasets*, que incluyen variables cuantitativas y cualitativas sobre recursos, productos, procesos, técnicas, gestión, etc. Estos datos son recogidos por organizaciones reconocidas, así como por las organizaciones de software individuales e investigadores para diversos fines. En la mayoría de las disciplinas científicas y de ingeniería, estos datos son útiles para realizar evaluaciones comparativas, y estudios expermientales.

El *International Software Benchmarking Standards Group* mantiene un repositorio de datos que es referido en este trabajo como la base de datos del ISBSG. La versión de la base de datos utilizada en este proyecto es la 12. La base de datos cuenta con 125 variables organizadas por grupos, según puede verse en la tabla A.1. Por otra parte en la tabla A.2 pueden observarse las 20 variables más utilizadas para la estimación del esfuerzo, según la publicación [8].

Grupo	Variables
Rating	Data Quality Rating, UFP Rating
Software Age	Year of Project„
Major Grouping	Industry Sector, Organisation Type, Application Group, Application Type, Development Type, Development Platform, Language Type, Primary Programming Language
Sizing	Functional Size, Relative Size, Adjusted Function Points, Value Adjustmenet Factor
Effort	Normalised Work Effort Level 1, Normalised Work Effort, Summary Work Effort
Productivity	Normalised,Level 1 PDR (ufp), Normalised,PDR (ufp), Pre 2002 PDR,(afp)
Other metrics	Defect Density, Spped of Delivery, Manpower Delivery Rate
Schedule	Project Elapsed Time, Project Inactive Time, Implementation Date, ...
Quality Delivered	Minor Defects, Major Defects, Extreme Defect, Total Defects Delivered
Grouping Attributes	Business Area Type, Software Process CMM, Software Process ISO, ...
Architecture	Architecthrue, Client Roles, Server Roles, Type of Server...
Documents & Techniques	Development Techniques, JAD Method Used, Protyping Used, ...
1st Platform	1st Hardware, Integrated Developmente Enviroment, 1st Language, 1st Operating Systm, 1St Data Base System, ...
2nd Platform	1st Hardware, 1st Language, 1st Operating Systm, 1St Data Base System, ...
Project Attributes	CASE Tool Used, Used Methodology, How Methodology Acquired, ...
Effort Attributes	Team Size Group, Max Team Size, Avarage Time Size, Percentage of Uncolected Work Effort...
Size Attributes	Input Count, Output Cont, Enquiry count, Interface count, Changed Count, Deleted Count, Cosmic Entry, Cosmic Exit, COSMIC Read, COSMIC Write
Other Size	Lines of Code, Lines of Code not Statements, Other Size Units

Tabla A.1: Variables de la base de datos del ISBSG clasificadas por grupos

Posición	ID	Variable	Descripción
1	FS	Functional Size	Tamaño del proyecto en Function Points sin ajustar.
2	DT	Development Type	Establece si es un nuevo diseño, un rediseño o una mejora de una aplicación previa.
3	LT	Language Type	Define el nivel de abstracción del lenguaje empleado: 3GL, 4GL.
4	DP	Development Platform	La plataforma a la que va dirigida el desarrollo. PC, Main Frame, Multiplataforma, etc.
5	AFP	Adjusted Function Points	Tamaño del proyecto en Function Points ajustado por un factor de ajuste de valor.
6	MTS	Max Team Size	Número máximo de desarrolladores que se emplearon en el proyecto en algún momento. Se refiere solo a la fase de desarrollo.
7	OT	Organisation Type	Describe el tipo de organización que desarrolló el proyecto.
8	PPL	Primary Programming Language	Lenguage de programación utilizado. Java, C++, etc
9	PET	Project Elapsed Time	Tiempo transcurrido para completar el proyecto en meses.
10	AT	Application Type	Identifica el tipo de aplicación al que se ha enfocado el proyecto. Control de procesos, sistema de información, transacciones, etc.
11	BAT	Business Area Type	Dentro del Tipo de organización (OT), el área de negocio al que se enfoca el proyecto.
12	EC	Enquiry Count	Function Points (UFP) referentes a External Enquiry.
13	FC	File Count	Function Points (UFP) referentes a Internal Logical Files.
14	IFC	Interface Count	Function Points (UFP) referentes a External Interface.
15	OC	Output Count	Function Points (UFP) referentes a External Output.
16	INC	Input Count	Function Points (UFP) referentes a External Input.
17	1DBS	1st Database System	Si la hay, identifica la tecnología de base de datos que se emplea en el proyecto.
18	RL	Resource Level	Nivel de especialización de los técnicos envueltos en el desarrollo del proyecto. Variable nominal ordinal.
19	UM	Used Methodology	Describe si usa algún tipo de metodología para desarrollar el proyecto.
20	ATS	Average Team Size	Número promedio de desarrolladores que se emplearon durante el proyecto. Se refiere sólo a la fase de desarrollo.

Tabla A.2: 20 variables más usadas del ISBSG, `fuente:` The usage of ISBSG data fields in software effort estimation: A systematic mapping study. [8]

Apéndice B

Código fuente

B.1 Funciones.R

```
1
2  # Función que implementa diferentes tipos de discretización mediante diferentes
3  # métodos de discretización y diferentes nnúmeromeros de clases, posteriormente
4  # genera diferentes árboles de decisión C4.5 para los diferentes tipos de
5  # discretización, y con diferentes tipos de instancias por hoja. Finalmente
6  # almacena los resultados ordenados en un dataset de salida que servirá para
7  # ver cual es el método óptimo de discretización, y los mejores valores,
8  # para el nnúmeromero de clases y para el nnúmeromero de instancias por hoja
9  # Entradas:
10   # data- dataframe sobre el que se trabaja
11   # metodos- un vector difrenetes formas de discretizacion del paquete arules
12   # Ms - un vector con los diferentes valores del parametro M (minimo numero
13      # de instancias por hoja) de Weka_control(), sirve para limitar el
14      # tamaño del árbol.
15   # Nc - vector con las clases para las que se quiere realizar la prueba
16 # Salida:
17   # Output- dataframe donde se han generado los datos perdidos.
18
19 Feature.Subset.Selection<-function(data,metodos,Ms,Nc){
20   Output = data.frame(Metodo.Discretizacion= character(), Numero.Clases=numeric
         (),
21   M=numeric(), kappa= numeric(), RAE= numeric(),MAE=numeric(),
22   RMSE=numeric(),RRAE=numeric(), stringsAsFactors = FALSE)
23   # data.d es una copia de ISBSG, para poder realizar la discretización mas de
         una vez
24   data.d<-data
25   for (i in metodos){
26     for(j in Ms){
27       for (k in Nc){
28         data.d$Normalised.Work.Effort.Level.1<-discretize(data$Normalised.Work.
               Effort.Level.1, method=i, categories=k)
29         m<- J48(Normalised.Work.Effort.Level.1~., data=data.d, control=Weka_
               control(M=j))
30         sm<-summary(m)
```

```
31          Output<-rbind(Output,data.frame(Metodo.Discretizacion=i,Numero.Clases=k
                ,M=j,kappa=sm$details[4],RAE=sm$details[7],MAE=sm$details[5],RMSE=
                sm$details[6],RRAE=sm$details[8]))
32        }
33      }
34    }
35    return(Output)
36  }
37
38  #-----------------------------------------------
39
40  # Función que genera datos perdidos en una o varias variables segun el
        mecanismo MCAR
41  # Entradas:
42    #dataset- Dataframe donde se generan los datos perdidos.
43    #probabilidad- Un númeromero del 0 al 100 que indica el procentaje de datos
44             #perdidos sobre el nnúmeromero de celdas de las variables donde se
45             #generan dichos datos.
46    #variables - Un vector con las variables donde se quieren generar los datos
47            #perdidos.
48  # Salida:
49    #dataset- Dataframe donde se han generado los datos perdidos.
50
51  generaMCAR<-function(data, probabilidad, variables){
52  ### G
53    # Este if tiene como motivo que al llamar la función
54    # desde generaMAR, al multiplicar probabilidad por n.partes en ocasiones
55    # puede dar un valor ligeramente superior a 100. cuando la probabilidad es 40
56    if(probabilidad>100){
57      probabilidad<-100
58    }
59    columnas.incompletas<-data[variables]
60  ### G
61    ncols<-ncol(columnas.incompletas)
62    nrows<-nrow(columnas.incompletas)
63    longitud<-ncols*nrows
64    total.NAs<-floor(longitud*probabilidad/100)
65  ### G
66    if(total.NAs>0){
67  ### G
68      indices<-sample(1:longitud, total.NAs, replace=FALSE)-1
69      fila<-indices %/% ncols+1
70      columna<-indices %% ncols+1
71      for (n in 1:total.NAs){
72        columnas.incompletas[fila[n],columna[n]]<-NA
73      }
74  ### G
75      data[variables]<-columnas.incompletas
76
77    } #end if
78  ### G
79    return(data)
80  }
81
82
83  #-----------------------------------------------
84
85  # Función que genera datos perdidos en una o varias variables segun el
86  # mecanismo MAR  tal como se describe en la publicación de MINI. Se
87  # ordena el dataset segnún una variable y se divide en partes. Se calculan
88  # las medias de cada parte y se reparten los NAs en cada parte segnúmeron
89  # esas medias. Es decir, a mayor media aritmática mayor cantidad de NA.
```

```
# De esta forma te aseguras que el mecanismo es MAR.

# Entradas:
  # data- dataframe donde se generan los datos perdidos.
  # probabilidad- Un númeromero del 0 al 100 que indica el procentaje de datos
          #  perdidos sobre el numero de celdas de las variables donde
          # se generandichos datos.
  # n.partes- Numero de partes en las que se divide del dataframe
  # variables.datos.perdidos - un vector con las variables donde se quieren
                # generar los datos perdidos.
  # variable.orden - variable mediante la cual se divide el dataframe en partes
           # por frecuencia.
# Salida:
  # resultado - dataframe donde se han generado los datos perdidos. Está
        # desordenado.

generaMAR<-function(data, probabilidad, n.partes, variables.datos.perdidos,
    variable.orden){
  # índices ordenados
  indices <- order (data[[variable.orden]] ,decreasing = FALSE )
  # dataset ordenado
  data.ordenada<-data[indices,]
  # Se inicia el vector de medias
  medias<-rep(0, n.partes)
  num.filas.por.grupo<-ceiling(nrow(data.ordenada)/n.partes)

  # Esto genera una lista. Cada elemento de la lista tiene los índices
  # de cada parte
  filas.por.grupo<-split(seq_along(data.ordenada[[variable.orden]]),
  ceiling(seq_along(data.ordenada[[variable.orden]])/num.filas.por.grupo))
  # Se calculan las medias de cada parte
  for(i in 1:n.partes){
    medias[i]<-mean(data.ordenada[filas.por.grupo[[i]],variable.orden])
  }
  # Se calcula la probabilidad de cada parte
  probabilidades<-probabilidad*medias/sum(medias)

  # Me guardo la columna a imputar

  # Por último, en cada parte se genera MCAR con su probabilidad
  # correspondiente
  for (i in 1:n.partes){
    #generaMCAR funciona sin la columna a imputar
### G
    datos <- generaMCAR(data.ordenada [filas.por.grupo[[i]],],
    probabilidades[i]*n.partes, variables.datos.perdidos)
### G
    if (i==1) {
      resultado<-datos
    } else {
      resultado<-rbind(resultado, datos)
    }
  }
### G
  resultado<-resultado[row.names(data),]
### G
  return (resultado)
}


#-------------------------------------------------
# Calcula el MMRE
```

```
# Entradas:
  # imputados :vector de datos imputados
  # noimputados: vector de datos originales.
# Salida:
  # Mean magnitud of Relative Error

calculaMMRE<-function(imputados, noimputados){
  MMRE<-mean(abs(noimputados-imputados)/noimputados)
  return(MMRE)
}

# Calcula el MdRE
# Entradas:
  # imputados :vector de datos imputados
  # noimputados: vector de datos originales.
# Salida:
  # Mean magnitud of Relative Error

calculaMdRE<-function(imputados, noimputados){
  MMRE<-median(abs(noimputados-imputados)/noimputados)
  return(MMRE)
}


# Calcula el error para variables nominales
# Entradas:
  # imputados :vector de datos imputados
  # noimputados: vector de datos originales.
# Salida:
  # Error para variable nominales

calculaError<-function(imputados,noimputados){
  Ncor<-sum(imputados==noimputados)
  error<-1-(Ncor/length(imputados))
  return(error)
}


#-----------------------------------------------

# Función que se utiliza para imputar datos mediante el metodo Class Mean
    Imputation
# Entradas:
  # column.data - vector con los datos donde se quiere realizar la imputación
      mediante CMI.
  # clases - vector de la misma longitud que column.data donde se encuentra la
      variable de clase.
# Salida:
  # column.data- datos imputados

CMI<- function(column.data,clases) {
  cells.with.missing.data<-is.na(column.data)
  moda.media<-vector()
  if ( is.factor (column.data) | is.ordered (column.data)| is.character(column.
      data)){
    # levels(clases) es un vector de chars, por eso los hago numáricos
    for (i in as.numeric(levels(clases))){
      # Esta variable se llama modamedia porque será la moda para variables
          categoricas
      # y la media para variables continuas
      moda.media[i] <- names(sort(table(column.data[clases==i]), decreasing=
          TRUE))[1]
```

```
207     }
208
209   } else{
210     for(i in levels(as.factor(clases))){
211       moda.media[i]<-mean(column.data[clases==i],na.rm=TRUE)
212     }
213   }
214
215   for(i in which(cells.with.missing.data==TRUE)){ #Debería intentar hacer este
          bucle con ifelse
216     column.data[i]<-moda.media[clases[i]]
217   }
218   return(column.data)
219 }
220
221
222 #-----------------------------------------------
223 # Función para calcular la distancia entre un caso y una clase
224 # Solo funcionará adecuadamente cuando la última columna del dataframe
225 # sea la columna de clase
226 # Entradas:
227   # caso - caso a imputar
228   # datos - dataframe con los datos a imputar
229   # numerical.columns- columnas nnúmeromericas del dataset
230   # factor.columns- columnas nominales del dataset
231   # multifactor.columns- Estos son las variables del dataset que
232   # incluyen varios valores separados por ";" (variable sep)
233
234 # Salida:
235   # which.min(Distancia)- Entero con el valor de la clase mas cercana
236   # al caso a imputar
237
238 library(foreach)
239 library(doParallel)
240
241 min.dis.caso.clase<-function(caso,datos, Nc, numerical.columns, factor.columns,
        multifactor.columns){
242   Distancia<-vector()
243   no.cores <- detectCores ()
244   registerDoParallel(cores=no.cores -1)
245   Distancia<-foreach(i=1:Nc, .combine="c", .export="distanciaGower") %dopar% {
246     return(sum(distanciaGower(rbind(caso,datos[datos$Clase==as.character(i),])
          , fila =1 ,numerical.columns, factor.columns, multifactor.columns))/
          nrow(datos[datos$Clase==as.character(i),]))
247   }
248   stopImplicitCluster()
249   return(which.min(Distancia))
250 }
251
252 #-----------------------------------------------
253 # Función que implementa el algoritmo MINI:
254 # Entradas:
255   # data: dataframe a imputar
256   # variable: variable a imputar
257   # k: valor del parámetro k. El valor por defecto es 3
258   # multifactorColumns: variables que deben ser tratadas como columnas
259   #                     multifactor. Se le puede pasar un vector con los
260   #                     índices de las columnas o con el nombre.
261   # catFun: Función que se llamará para calcular el valor imputado
262   #         entre los k valores posibles en el caso de variables nominales
263   # numFun: Función que se llamará para calcular el valor imputado
264   #         entre los k valores posibles en el caso de variables cuantitativas
```

```
265   # distancias.param: TRUE o FALSE. Indica si al hacer la llamada a la
266   #                    función catFun o numFun también se tiene que pasar
267   #                    el valor de las distancias o no para que se realice la
268   #          ponderación además de los k valores posibles.
269   # Nc: Número de clases.
270 # Salida:
271   # data - dataset imputado.
272
273
274 MINI<- function(data,variable,k =3, multifactorColumns=NULL,
275 catFun=moda, numFun=mean,distancias.param=FALSE, Nc){
276   # Comprueba que k es un valor válido (> 0 y < que el número de filas)
277   if ((k<1) | (k>(nrow(data)-1))) {
278     stop (" Valor de k no válido")
279   }
280   funcion.k<-NULL
281   numerical.columns<-vector()
282   factor.columns<-vector()
283   multifactor.columns<-vector()
284   if (!is.null(multifactorColumns)){
285     # El parámetro multifactor contiene un valor no nulo. Hay que
286     # comprobar si es un vector con los índices de las columnas o con los
287     # nombres de las columnas. Trabajaremos con los índices
288     if (is.character(multifactorColumns)){
289       # Si es un vector de nombres me quedo con los índices
290       multifactor.columns<-which(colnames(data) %in %multifactorColumns)
291     } else{
292       # Si es un vector numérico no tengo que hacer nada
293       if (is.numeric(multifactorColumns)){
294         multifactor.columns <- multifactorColumns
295       } else{
296         # Si es un vector de cualquier otro tipo... error
297         stop ( " El parámetro \"multifactor \" tiene que ser un vector con los
                índices o con los nombres de las columnas multifactor " )
298       }
299     }
300   }
301
302 ### G
303   for (j in 1:ncol(data)-1) { #El -1 es por la variables Clase
304 ### G
305     if (is.numeric(data[,j])){
306       numerical.columns<- c(numerical.columns, j)
307     }
308     # Con Gower, las columnas que son factores ordenados se tratan como
309     # numáricas así que los convertimos y los añadimos al conjunto de
310     # columnas numáricas. OJO! Esto hay que hacerlo antes de valorar
311     # las columnas factores (porque R considera los factores ordenados
312     # como factores también (is.factor(ordenado)==TRUE)
313     if (is.ordered(data[,j])){
314       data[,j]<- as.numeric(data[,j])
315       numerical.columns<-c(numerical.columns,j)
316     }
317     if (is.factor(data[,j])) {
318       factor.columns<-c(factor.columns,j)
319     }
320     if (is.character(data[,j])){
321       data [,j] <- as.factor(data[,j])
322     factor.columns<- c(factor.columns,j)
323     }
324   }
325   # Por si las moscas al final decido pasar los índices como parámetro o
```

```
326   # por si los ordenados también pasan como factores hago una comprobación
327   factor.columns<-factor.columns[!factor.columns %in% numerical.columns]
328
329   # Si la variable a imputar es numárica se llamará a numFun, de otra
330   # forma se llamará a catFun
331   if (is.numeric (data[[variable]])){
332     funcion.k<-numFun
333   } else {
334     funcion.k<-catFun
335   }
336   nrows<-nrow(data)
337   #No tiene sentido calcular distancias a casos donde la variable a
338   #imputar tiene un NA(ya que no se podría usar para imputar)
339   data.sin.NAs<-data[!is.na(data[,variable]),]
340   for (fila in 1:nrows) {
341     if  (is.na(data[fila, variable]))  {
342
343 ### G
344       clase<-min.dis.caso.clase(data[fila,], data, Nc, numerical.columns,
345       factor.columns, multifactor.columns)
346
347       # Si hay que imputar un valor coloco esa fila la primera y calculo
348       # las distancias
349       data.testeo<-rbind(data[fila,], data.sin.NAs[data.sin.NAs$Clase==clase,]
              )
350 ### G
351       #SI NO FUNCIONA LA COMPUTACIÓN EN PARALELO SE PUEDE SUSTITUIR
352       #LA FUNCIÓN distanciaGowerPar POR distanciaGower
353
354       distancias <- distanciaGowerPar(data.testeo, fila =1 ,numerical.columns,
355       factor.columns  ,multifactor.columns)
356       # Ordenamos los índices de las filas de menor a mayor distancia
357       filas.ordenadas<-order(distancias, decreasing = FALSE)
358       # La fila que estoy comprobando siempre tendrá distancia 0. Estará la
359       # primera en el data.testeo.
360       filas.ordenadas<-filas.ordenadas[-1]
361
362       # Si distancias.param es verdadero, a la función para calcular el valor
363       # imputado, se le tienen que pasar dos parámetros, los k valores y sus
364       # respectivas distancias
365       if (distancias.param == TRUE ){
366         distancias.ordenadas<-distancias[filas.ordenadas][-1]
367         distancias.ordenadas<-distancias.ordenadas [1: k ]
368         valor.a.imputar<-funcion.k(data.testeo[filas.ordenadas[1:k], variable],
                distancias.ordenadas)
369       } else {
370         # De lo contrario solo se pasa los k valores
371         valor.a.imputar<-funcion.k(data.testeo[filas.ordenadas[1:k], variable])
372       }
373       # Finalmente se sustituye el NA por el valor imputado
374       data[fila ,variable]<-valor.a.imputar
375     }
376   }
377   return (data)
378 }
379
380 #-----------------------------------------------
381 # Función que sirve para identificar el patrón de datos perdidos,
382 # tiene como entrada  una lista, que contiene las variables donde se
383 # van a simular los datos perdidos y se va a realizar la imputación
384
385 funcion.patron<-function(variable){
```

```
  l<-length(variable)
  if(l<1){
    patron<-NA
  } else if(l==1){
    patron<-"Univariate"
  } else if (l==2){
    patron<-"Bivariate"
  } else if (l>2){
    patron<-"Multivariate"
  }
  return(patron)
}


#-----------------------------------------------
# Función que sirve para identificar el tipo de variable (Continua, Nominal o Mixta),
# tiene como entrada un elemento de una lista, que contiene las variables donde se
# van a simular los datos perdidos y se va a realizar la imputacion.
# Es necesario que la base de datos del ISBSG está cargada en las variables de entorno globales y se llame ISBSG.

funcion.tipo.variable<-function(variables){
  var.Continua<-FALSE
  var.Nominal<-FALSE
  for(j in 1:length(unlist(variables))){
    c<-class(eval(parse(text=paste("ISBSG",unlist(variables)[j],sep="$"))))
    #print(class(eval(parse(text=paste("ISBSG",unlist(variables[i])[j],sep="$")))))
    if(c=="numeric" |c=="integer"){
      var.Continua<-TRUE
    } else if(c=="factor"|c=="ordered"|c=="character"){
      var.Nominal<-TRUE
    }
  }
  if(var.Continua==TRUE & var.Nominal==TRUE){
    tipo.variable<-"Mixta"
  } else if(var.Continua==TRUE){
    tipo.variable<-"Continua"
  } else if(var.Nominal==TRUE){
    tipo.variable<-"Nominal"
  }
  return(tipo.variable)
}


source("Funciones_preliminares.R")
```

B.2 Funciones_preliminares.R

```



# La siguiente función admite dos strings o incluso dos vectores de strings,
# las separa en dos arrays según un separador űsepű y por último devuelve
# true si alguno de los dos arrays tienen alguna palabra en común

tokenize.and.compare <- function (str1,str2 ,sep=";") {
  #Nos aseguramos que las variables de entrada son texto
  if (!is.character(str1))str1<-as.character(str1)
  if (!is.character(str2))str2<-as.character(str2)
  # No voy a comprobar que str1 y str2 sean de distinto tamaño porque como
  # van a ser filas del mismo dataset no puede ocurrir.
  # Crea un vector de booleanos de determinado tamaño
  resultado<-logical(length(str1))
  for(j in 1:length(str1)){
    comparacion<-any(unlist(strsplit(str1[j], sep, fixed = TRUE)) %in% unlist(
        strsplit(str2[j], sep, fixed=TRUE)))
    resultado[j]<-comparacion
  }
  # NA %in % NA da TRUE pero da igual
  return (resultado)
}

#űűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűű
# La siguiente función calcula la moda de un vector de valores

moda<-function(valores){
  if (!is.factor(valores)){
    valores<-as.factor(valores)
  }
  # Ordena las frecuencias de cada nivel del factor y se queda con la
  # primera. Para calcular las frecuencias se utiliza el comando űtableű para
  # hacer una tabla de contingencia. Para ordenarlas se usa el comando űsortű.
  # Esto hará que si hay empate cogerá la primera que aparezca en el factor,
  # que es justo lo que queremos. (El orden de aparición en table(), es el
  # orden de aparición en el factor)
  moda<-names(sort(table(valores),decreasing=TRUE))[1]
  return(moda)
}

#űűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűű
# La siguiente función calcula la media ponderada del vector űvaloresű,
# según un segundo vector de distancias

media.ponderada.distancias<-function(valores,distancias){
  # Se calculan los pesos de la ponderación, que tienen que ser inversamente
  # proporcionales a las distancias,(a mayor distancia menor peso en la
  # ponderación). Se tiene que sumar 1 en el denominador, porque si hay dos
  # filas iguales, la distancia sera cero
  pesos <-1/(1+distancias)
  return(weighted.mean(valores,pesos))
}

#űűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűű
# Cálculo de la distancia Gower en un dataset. Esta función es llamada
# internamente
# @param A: dataset en el que se calcularán las distancias
# @param fila: las distancias se calculan entre esta fila y el resto
# @param numerical.columns: columnas númericas del dataset
```

```
# @param factor.columns: columnas nominales del dataset
# @param multifactor.columns: Estos son las variables del dataset que
# incluyen varios valores separados por ű;ű (variable sep)
# @param weights: En ocasiones se le quiere dar más importancia a unas
# columnas que a otras. Se puede modelar esto con este parámetro. Pasando
# un vector numérico del tamaño el número de columnas.
# @param sep: Las columnas multifactor se tienen que interpretar como
# cadenas de caracteres separadas por este valor

distanciaGower<-function(A,fila =1,numerical.columns = vector () ,
  factor.columns = vector (),multifactor.columns=vector(),
  weights =rep(1 ,ncol(A)),sep=";"){
  # En las columnas numéricas se calcula el rango, el máx menos el mín. Se
  # usa luego en la fórmula de gower
  rangos<-vector()
  if (length(numerical.columns)>0) {
    rangos<-apply(as.data.frame(A[,numerical.columns]), 2,range,na.rm=TRUE)
    rangos<-rangos[2,]-rangos [1,]
    #Si el rango es cero ( vmax - vmin = 0 ) luego se dividirá entre cero
    #y dará error por eso lo sustituyo por 1
    rangos<-ifelse(rangos==0,1,rangos )
  }
  # Variable Resultado
  distancias<-numeric(nrow(A))
  for (i in 1:nrow(A)) {
    # Esta variable acumula los sumandos que aparecen en la ecuación de Gower
    # en el numerador
    parciales.numerador<- 0
    # Esta variable acumula los sumandos que aparecen en la ecuación de Gower
    # en el denominador
    parciales.denominador<-0
    if (i==fila) {
      # Si estoy estimando la fila=fila en cuestion la distancia es 0
      distancias[i]<-0
    } else {
      # Fórmula de Gower para valores numéricos
      if(length(numerical.columns)>0){
        Dij.numerical<-unlist(abs(A[fila,numerical.columns]-A[i,numerical.
            columns]))/rangos
        wk.delta.numerical <- ifelse(is.na(Dij.numerical),0,
        weights [numerical.columns ])
        parciales.numerador<-sum(wk.delta.numerical*Dij.numerical,na.rm=TRUE )
        parciales.denominador<-sum(wk.delta.numerical)
      }
      # Fórmula de Gower para variables nominales
      if (length(factor.columns)>0) {
        # identical si los dos son NA devuelve TRUE, no me interesa
        Dij.nominales <- !(A[i,factor.columns]== A[fila,factor.columns])
        wk.delta.nominales<-weights[factor.columns]*!(is.na(A[i,factor.columns
            ])
        | is.na (A[ fila , factor.columns ]))
        parciales.numerador <- sum(parciales.numerador, wk.delta.nominales *
        Dij.nominales,na.rm=TRUE )
        parciales.denominador <- sum(parciales.denominador,wk.delta.nominales,
        na.rm=TRUE )
      }
      # Fórmula de Gower para variables multifactor. Si dos valores multifactor
      # contienen algún valor en común la distancia es cero
      if ( length ( multifactor.columns ) > 0 ) {
        # identical si los dos son NA devuelve TRUE, no me interesa
        Dij.multifactor <- !tokenize.and.compare(A[fila,multifactor.columns ],
        A [i , multifactor.columns], sep=sep)
```

```
        wk.delta.multifactor <- weights [ multifactor.columns ] *
        !(is.na(A[i, multifactor.columns ]) | is.na(A[fila, multifactor.columns
            ]) )
        parciales.numerador <- sum(parciales.numerador,wk.delta.multifactor*
        Dij.multifactor,na.rm = TRUE )
        parciales.denominador <- sum(parciales.denominador,
        wk.delta.multifactor,na.rm=TRUE )
      }
      # anado al vector de distancias la distancia a la fila i
      distancias[i]<-parciales.numerador/parciales.denominador
    }
  }
  return (distancias)
}


#ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ
# Función knn:
# @param data: dataframe a imputar
# @param variable: variable a imputar
# @param k: valor del parámetro k. El valor por defecto es 3
# @param multifactorColumns: variables que deben ser tratadas como columnas
# multifactor. Se le puede pasar un vector con los índices de las columnas
# o con el nombre
# @param catFun: Función que se llamará para calcular el valor imputado
# entre los k valores posibles en el caso de variables nominales
# @param numFun: Función que se llamará para calcular el valor imputado
# entre los k valores posibles en el caso de variables cuantitativas
# @param distancias.param: TRUE o FALSE. Indica si al hacer la llamada a la
# función catFun o numFun también se tiene que pasar el valor de las
# distancias o no para que se realice la ponderación además de los k
# valores posibles.
#

knn <- function (data , variable , k =3 , multifactorColumns=NULL ,
catFun=moda , numFun=mean,distancias.param=FALSE ) {
  # Comprueba que k es un valor válido (> 0 y < que el número de filas)
  if ( (k <1) | (k > ( nrow ( data ) -1) ) ) {
    stop ( " Valor de k no válido " )
  }
  funcion.k<-NULL
  numerical.columns<-vector()
  factor.columns<-vector()
  multifactor.columns <- vector()
  if (!is.null(multifactorColumns)){
    # El parámetro multifactor contiene un valor no nulo. Hay que
    # comprobar si es un vector con los índices de las columnas o con los
    # nombres de las columnas. Trabajaremos con los índices
    if (is.character (multifactorColumns)) {
      # Si es un vector de nombres me quedo con los índices
      multifactor.columns <- which(colnames(data) %in %multifactorColumns)
    } else{
      # Si es un vector numérico no tengo que hacer nada
      if (is.numeric(multifactorColumns ) ) {
        multifactor.columns<-multifactorColumns
      } else{
        # Si es un vector de cualquier otro tipo... error
        stop ("El parámetro \"multifactor \" tiene que ser
        un vector con los índices o con los nombres de
        las columnas multifactor ")
      }
    }
```

```
181   }
182
183   for (j in 1: ncol(data)-1) { #-1 es para eliminar la variable clase
184     if (is.numeric(data[,j])) {
185       numerical.columns <- c(numerical.columns,j)
186     }
187     # Con Gower, las columnas que son factores ordenados se tratan como
188     # numéricas así que los convertimos y los añadimos al conjunto de
189     # columnas numéricas. OJO! Esto hay que hacerlo antes de valorar
190     # las columnas factores (porque R considera los factores ordenados
191     # como factores también (is.factor(ordenado)==TRUE)
192     if (is.ordered (data[,j])) {
193       data [,j]<-as.numeric(data[,j])
194       numerical.columns<-c(numerical.columns,j)
195     }
196     if (is.factor(data[,j])){
197       factor.columns<-c(factor.columns,j)
198     }
199     if (is.character(data[,j])){
200       data[,j]<-as.factor ( data[,j])
201     factor.columns<- c(factor.columns,j)
202     }
203   }
204   # Por si las moscas al final decido pasar los índices como parámetro o
205   # por si los ordenados también pasan como factores hago una comprobación
206   factor.columns<-factor.columns[!factor.columns %in% numerical.columns ]
207
208   # Si la variable a imputar es numérica se llamará a numFun, de otra
209   # forma se llamará a catFun
210   if ( is.numeric (data[[variable]])) {
211     funcion.k<-numFun
212   } else{
213     funcion.k<-catFun
214   }
215   nrows<-nrow(data)
216   # No tiene sentido calcular distancias a casos donde la variable a
217   # imputar tiene un NA(ya que no se podría usar para imputar)
218   data.sin.NAs<- data [!is.na(data[,variable ]),]
219   for (fila in 1:nrows ) {
220     if (any(is.na(data[fila, variable]))) {
221       # Si hay que imputar un valor coloco esa fila la primera y calculo
222       # las distancias Gower
223
224       data.testeo<-rbind(data[fila ,],data.sin.NAs )
225
226       # SI NO FUNCIONA LA COMPUTACIÓN EN PARALELO SE PUEDE SUSTITUIR
227       # LA FUNCIÓN distanciaGowerPar POR distanciaGower
228
229       distancias<-distanciaGowerPar(data.testeo, fila =1, numerical.columns,
230        factor.columns, multifactor.columns)
231       # Ordenamos los índices de las filas de menor a mayor distancia
232       filas.ordenadas<-order(distancias, decreasing=FALSE )
233       # La fila que estoy comprobando siempre tendrá distancia 0. Estará la
234       # primera en el data.testeo.
235       filas.ordenadas<-filas.ordenadas[-1]
236
237       # Si distancias.param es verdadero, a la función para calcular el valor
238       # imputado, se le tienen que pasar dos parámetros, los k valores y sus
239       # respectivas distancias
240       if ( distancias.param==TRUE) {
241         distancias.ordenadas<-distancias [ filas.ordenadas][-1]
242         distancias.ordenadas<-distancias.ordenadas [1:k]
```

```
        valor.a.imputar<-funcion.k ( data.testeo
        [filas.ordenadas[1:k],variable],distancias.ordenadas)
      } else {
        # De lo contrario solo se pasa los k valores
        valor.a.imputar<-funcion.k(data.testeo[filas.ordenadas[1:k], variable])
      }
      # Finalmente se sustituye el NA por el valor imputado
      data [fila,variable]<-valor.a.imputar
    }
  }
  return (data)
}


#űűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűűű
# Cálculo de la distancia Gower en un dataset. VERSÍÓN COMPUTACIÓN EN
# PARALELO. Es la misma que distanciaGower pero ésta manda cada cálculo de
# la distancia a un core distinto del procesador.

distanciaGowerPar<-function(A,fila=1, numerical.columns=vector(),
factor.columns=vector(), multifactor.columns=vector(),
weights=rep(1,ncol(A)),sep =";") {

  rangos<-vector()
  if (length(numerical.columns )>0) {
    rangos <- apply(as.data.frame (A[,numerical.columns]), 2, range, na.rm=TRUE
      )
    rangos <- rangos[2,]-rangos[1,]
    #Si el rango es cero (vmax - vmin = 0) luego se dividirá entre cero y
    #dará error. Lo sustituyo por 1
    rangos<-ifelse(rangos==0, 1, rangos)
  }
  no.cores<-detectCores()
  registerDoParallel(cores=no.cores-1)

  # Resultado
  result<-foreach(i =1:nrow(A), .combine = "c" , .export='tokenize.and.compare'
    ) %dopar% {
  # Esta variable acumula los sumandos que aparecen en la ecuación de
  # Gower en el numerador
  parciales.numerador<-0
  # Esta variable acumula los sumandos que aparecen en la ecuación de
  # Gower en el denominador
  parciales.denominador<-0
  if (i==fila) {
    # Si estoy estimando la fila=fila en cuestion la distancia es 0
    distancia <- 0
  } else {
    if ( length ( numerical.columns ) >0 ) {
      Dij.numerical<-unlist(abs(A[fila,numerical.columns] - A[i,numerical.
        columns]))/rangos
      wk.delta.numerical<-ifelse(is.na(Dij.numerical), 0, weights[numerical.
        columns])
      parciales.numerador <- sum(wk.delta.numerical*Dij.numerical, na.rm = TRUE
        )
      parciales.denominador<-sum ( wk.delta.numerical )
    }
    if(length(factor.columns)>0){
      # identical si los dos son NA devuelve TRUE, no me interesa
      Dij.nominales<-!(A [i ,factor.columns]==A[fila,factor.columns ] )
      wk.delta.nominales <-weights[factor.columns]*!(is.na(A[i ,factor.columns
        ])
```

```
299       | is.na (A[fila,factor.columns]) )
300       parciales.numerador <-sum(parciales.numerador, wk.delta.nominales *
301       Dij.nominales, na.rm=TRUE )
302       parciales.denominador<-sum(parciales.denominador,
303       wk.delta.nominales, na.rm=TRUE )
304     }
305     if(length(multifactor.columns)>0) {
306       # identical si los dos son NA devuelve TRUE, no me interesa
307       Dij.multifactor<-!tokenize.and.compare( A[fila,multifactor.columns],
308       A[i,multifactor.columns],sep =sep)
309       wk.delta.multifactor<-weights [multifactor.columns] * !(is.na (A[i,
310       multifactor.columns]) | is.na(A[fila, multifactor.columns]))
311       parciales.numerador<-sum(parciales.numerador , wk.delta.multifactor*Dij.
              multifactor, na.rm = TRUE )
312       parciales.denominador<-sum(parciales.denominador, wk.delta.multifactor,
              na.rm = TRUE )
313     }
314     # añado al vector de distancias la distancia a la fila i
315     distancia<-parciales.numerador/parciales.denominador
316   }
317   return (distancia)
318   }
319   stopImplicitCluster()
320   return (result)
321 }
```