Máster en Matemática Computacional

Trabajo Final de Máster

Diseño de un paquete R para el Análisis Estadístico Implicativo

Autor:
Xavier Valls Pla

Supervisor:
Pablo Gregori Huerta

Fecha de lectura: 28 de Octubre de 2014
Curso académico 2013/2014

Certifico que el presente trabajo ha sido realizado por el alumno Xavier Valls Pla, bajo mi supervisión, y se presenta para que constituya el Trabajo de Fin de Máster del Máster de Matemática Computacional.

Firmado: Pablo Gregori Huerta

Resumen

El Análisis Estadístico Implicativo nació hace 35 años y sin embargo ha tenido una difusión muy gradual. Existen dos proyectos que implementan sus técnicas, una aplicación privativa en Windows (CHIC), y un paquete R (RCHIC) que comparten código fuente escrito en C++ y demasiado complejo para su público objetivo.

Para solventar esto e impulsar la adopción y aprendizaje del ASI, se desarrolla un paquete completamente escrito en R, experimentando con Reference Classes –su última implementación de la Programación Orientada a Objetos– y comparando los resultados con los de las alternativas mencionadas.

Los resultados obtenidos son correctos pero tienen un coste temporal considerablemente mayor, por lo que se reflexiona sobre R, su manejo de memoria y su eficiencia tanto espacial como temporal; y se aplica el profiling sobre la aplicación con la finalidad de señalar áreas de mejora en la implementación realizada. Para finalizar, se apuntan posibles mejoras e implementaciones de cara al desarrollo futuro del paquete.

Palabras clave

R, análisis implicativo, análisis de cohesiones, análisis de similaridades, CHIC.

Keywords

R, implicative analysis, cohesitive analysis, similarity analysis, CHIC.

Índice general

Capítulo 1

Introducción y motivación

Hoy en día, y cada vez más, la sociedad genera datos de manera exponencial y se requiere de herramientas, procedimientos y nuevos métodos de análisis que permitan extraer el conocimiento subyacente a dichos datos y ajustarlos a modelos comprensibles por el ser humano, que permitan predecir información futura a partir de información observada previamente.

En este contexto, la minería de datos acaba convirtiéndose en una herramienta cada vez más indispensable para tratar de comprender y predecir el entorno. El Análisis Estadístico Implicativo surge como una serie de herramientas que se pueden enmarcar en la minería de datos que pretende arrojar luz sobre las implicaciones entre los sucesos y variables de un conjunto de datos de la Didáctica de las Matematicas.

El Análisis Estadístico Implicativo tiene una implementación computacional en el programa informático CHIC, propietario y únicamente disponible en el sistema operativo Windows. Pese a surgir hace ya varias décadas, el Análisis Estadístico Implicativo ha tenido una adopción muy limitada, lo que provoca que se busquen nuevas formas de difundir su teoría y sus aportaciones a la minería de datos. Para ello se decide portar su implementación computacional en forma de paquete (librería) para el lenguaje de programación R: libre, multiplataforma y con una amplia base de usuarios con perfil estadístico que facilite dicha difusión.

En este capítulo se realiza una introducción a los conceptos sobre los que girará el trabajo. En la Sección 1.1 se realiza una pequeña descripción de la Minería de Datos y su función, así como el papel que representa el Análisis Estadístico Implicativo

dentro de ella. En la Sección 1.2 se presenta su soporte computacional a día de hoy y, a continuación, en la Sección 1.3 se introduce el lenguaje de programación R.

Posteriormente, en la Sección 1.4, se expone el propósito del presente trabajo así como los objetivos que se pretenden conseguir.

Finalmente se explican tanto el método seguido en el desarrollo del proyecto (Sección 1.5) como la estructura del presente trabajo (Sección 1.6).

1.1. La Minería de datos y el Análisis Estadístico Implicativo

La **Minería de datos** recoge una amplia cantidad de métodos y algoritmos procedentes de las Matemáticas y Ciencias de la Computación que tiene como objetivo analizar un gran volumen de datos y extraer información para convertirla en estructuras comprensibles y manejables posteriormente.

Dicho análisis se realiza segmentando los datos, clasificándolos y buscando patrones y reglas que puedan sacar a la luz implicaciones, dependencias o cualquier otro tipo de relación entre ellos.

El **Análisis Estadístico Implicativo** (**ASI**, de *Analyse Statistique Implicative* en francés) fue inventado por Régis Gras para tratar de descubrir reglas $a \Rightarrow b$ sobre un conjunto de datos, cuantificando su calidad con la probabilidad condicional multivariante como base.

Estas implicaciones (inductivas, no simétricas) intentan responder a la pregunta *¿Cómo de probable es que suceda b habiendo sucedido a?* Es decir, no modelamos la implicación de una forma determinista, si no que cuantificamos la cuasi-implicación, buscando medir la calidad inductiva e informativa de la misma.

Se cuenta con una serie de índices e indicadores que nos permiten cuantificar su calidad, estabilidad o contribución de uno de los sucesos que comprende la regla al cumplimiento de la misma y define representaciones gráficas en forma de árbol jerárquico o grafo para una mejor visualización de las interrelaciones entre variables.

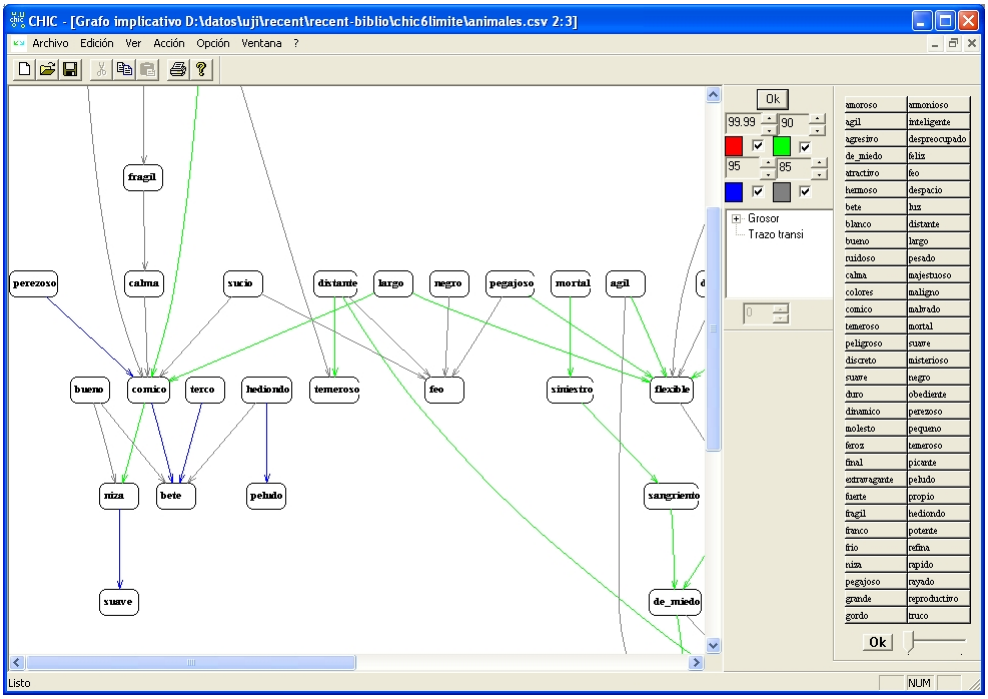

Figura 1.1: Captura de pantalla de CHIC

1.2. CHIC

CHIC (*Classification Hiérarchique Implicative et Cohésitive*) es un programa informático que trata de llevar a la computación el ASI. Creado a petición del mismo Régis Gras, cuenta con una versión privativa para Windows programada en C++. Hoy en día sigue en desarrollo a cargo de Raphaël Couturier, siendo Saddo Ag Almouloud y Harrisson Ratsimba-Rajohn sus primeros responsables.

Además de realizar los cálculos necesarios muy eficientemente, proporciona visualización gráfica e interactiva de los resultados en forma de grafos de similaridad o árboles jerárquicos.

En la Figura 1.1 se puede observar el aspecto de la aplicación.

1.3. The R Project for Statistical Computing

El lenguage de programación R[1] nació como una implementación libre del lenguage S de IBM.

Creado por la comunidad como software libre (proceso en el que participaron los creadores de S), desde su aparición y su posterior unión al proyecto GNU en 1995, su aceptación y su uso no ha dejado de crecer, convirtiéndose poco a poco en el lenguaje de referencia en el ámbito estadístico, financiero y biológico con comunidades tan grandes como R-Finance[2] o Bioconductor[3].

Con una amplia comunidad alrededor, R continua en desarrollo continuo consiguiendo que su base de usuarios crezca muy rápidamente, principalmente debido a su sencillez comparado con otros lenguajes, facilidad de uso y extensibilidad mediante paquetes.

1.4. Objetivo y propósito

El objetivo de este trabajo es proporcionar un paquete R escrito completamente en dicho lenguaje para aplicar las técnicas del ASI de forma que, además de proporcionar todas las funcionalidades que CHIC proporciona, pueda ser extensible en el lenguaje que su público objetivo domina sin suponer un gran sacrificio en tiempo de computación.

El ASI se ha caracterizado por estar vinculado, desde su nacimiento, geográficamente a Francia y su área de influencia lingüística y cultural, y temáticamente al análisis de datos de Didactica de las Matemáticas. Con el paso de los años la difusión geográfica ha pasado a países vecinos como Italia, Grecia, y España, desde donde ha saltado hacia Latinoamérica en los últimos 10 años. La ampliación en temáticas de aplicación ha transcurrido paralelamente a la geográfica, cubriendo campos como la sociología, psicología, educación, así como la biología. Sin embargo, la difusión a la literatura anglosajona es muy reciente y gradual. Con este paquete lo que se pretende es proporcionar las herramientas necesarias a la comunidad para acelerar el proceso de adopción.

[1] What is R?: http://www.r-project.org/about.html
[2] R Finance: Applied Finance with R: http://www.rinfinance.com/
[3] Bioconductor:Open Source Software for Bioinformatics: http://www.bioconductor.org/

El trabajo nace como una ramificación del proyecto RCHIC[4]. RCHIC es una interfaz R del software CHIC original para Windows, intentando aprovechar todo el código existente y únicamente cambiando la representación gráfica de los resultados.

Tras constatar la rigidez, la extrema complejidad y la poca extensibilidad del código se toma la decisión de reescribirlo completamente en R, intentando con ello facilitar la comprensión y el desarrollo colaborativo. Para ello se toman una serie de decisiones y medidas explicadas con detalle en el Capítulo 3.

1.5. Método y proceso de desarrollo

En el desarrollo de todo programa informático, y en especial haciendo uso de Programación Orientada a Objetos, hay que tener en cuenta la importancia de la fase de diseño de la arquitectura de software de la aplicación: estructuras de datos, clases, algoritmos, flujo del programa, interdependencias... Pese a todo, se debe mantener cierta flexibilidad y conseguir que los cambios sean lo menos dolorosos posible. El programa se desarrollará por partes, en las que los cambios se irán introduciendo gradualmente y será comprobada su corrección. Una vez todas las funcionalidades se hayan implementado, se deben valorar los resultados.

Esta valoración tiene dos fases. Primero se comprueba que los resultados no se desvían demasiado de los esperados, que dependiendo de la implementación, podrían incluso ser idénticos. A continuación se comparan los resultados con los de la aplicación existente en R (RCHIC) y se analizan las mejoras o desventajas respecto a ella. Mantener los tests lo más sencillos posible facilitará la identificación de potenciales problemas y causas de diferencias en los resultados.

Para el desarrollo del programa se decide empezar por la aproximación sobre datos binarios, desarrollando por partes y centrándolo en los modelos basados en las distribuciones binomial y Poisson de los datos.

El desarrollo del paquete R ha seguido el siguiente proceso, todo ello descrito en profundidad en el Capítulo 3.

- **Diseño de la arquitectura:** diseño de algoritmos, estructuras de datos y dependencias.

[4] Rchic en GitHub: https://github.com/rchic/Rchic

- **Desarrollo del análisis clasificatorio:** cálculo de similaridades, unión de clases, matrices de clases y demás estructuras y funciones fundamentales necesarias para generar el árbol de similaridad.
- **Cálculo de los nodos significativos**.
- **Implementación de métricas avanzadas:** par genérico, distancia, contribución, tipicalidad.
- **Desarollo de la herramienta de informe y su representación gráfica**.

Así pues, fuera del proyecto quedan entonces las siguientes características:

- **Aplicación a datos no binarios.**
- **Implementación de las aproximaciones Normal e Hipergeométrica.**
- **Análisis implicativo.**
- **Adaptación al análisis de cohesiones.**
- **Representación gráfica de los resultados.**

1.6. Estructura del trabajo

Un breve vistazo a la estructura del proyecto:

- El Capítulo 2 introduce al lector en la teoría e historia del ASI y presenta su soporte computacional CHIC.
- El Capítulo 3 realiza una introducción a la programación en R y justifica la elección de dicho lenguaje como la más idónea.
- El Capítulo 4 describe el proceso de diseño del paquete y expone algunas de las decisiones de implementación que se han debido tomar durante el desarrollo del mismo, así como la experimentación llevada a cabo y la exposición de los resultados obtenidos.
- En el Capítulo 5 se valora el "impacto"del paquete. Se proponen también mejoras, cambios y soluciones a los problemas surgidos durante la implementación y a la vista de los resultados.

Capítulo 2

Análisis Estadístico Implicativo

En este capítulo se busca familiarizar al lector con el Análisis Estadístico Implicativo en el contexto de la minería de datos y su historia y teoría. La mayor parte de los conceptos se introducen sin excesiva profundización, pero se aportan las referencias necesarias en cada caso.

Así, en la Sección 2.1 se introduce al lector en el campo de la minería de datos: qué es, qué supone y en qué contextos es útil. En la Sección 2.4 se describe el nacimiento del Análisis Estadístico Implicativo y su función, mientras que en la Sección 2.5 se definen sus bases desde el punto de vista teórico.

2.1. La Minería de datos

La Minería de datos como disciplina se refiere a una amplia variedad de métodos automáticos de exploración, análisis y modelización de grandes repositorios de datos intentando identificar nuevos, útiles y comprensibles patrones. La Minería de datos implica la búsqueda de algoritmos que exploran los datos para crear y desarrollar un modelo de los mismos que proporcione una infraestructura que permita descubrir en ellos patrones previamente desconocidos para utilizarlos en el análisis y la predicción.

La accesibilidad y la abundancia de datos hoy en día hace de la minería de datos una disciplina de considerable importancia y necesidad. Dado el crecimiento que está teniendo el campo, no sorprende que los investigadores y practicantes tengan

cada vez más herramientas a su disposición para allanar su camino en la búsqueda de toda esa cantidad de información que los conjuntos de datos modernos pueden ofrecer.

Entre sus procedimientos, tres son de interés para el presente trabajo: la segmentación o **clustering** es utilizada para agrupar los datos, definiendo perfiles para adaptar las soluciones a esos datos, mientras que las **reglas de asociación** y la **clasificación** son utilizadas como método de comprensión de mecanismos o patrones y predicción de resultados.

2.2. Reglas de asociación

Una regla de asociación es una declaración implicativa que tiene como objetivo encontrar relaciones entre variables, en función de cómo responden los individuos a esas variables dentro de una base de datos enorme, esto es, observar los sucesos y reunir pares de variables en relación a sus ocurrencias.

En otras palabras, una regla de asociación tiene como finalidad encontrar valores conjuntos de las variables $X = \{X_1, X_2, ..., X_p\}$ que aparezcan habitualmente en la base de datos. Más generalmente, el objetivo de las reglas de asociación es encontrar conjuntos de valores del vector X de forma que la densidad probabilística $Pr(v_i)$ evaluada en cada uno de dichos valores sea relativamente grande[10]. El ejemplo prototipo es el análisis de la cesta de la compra[1].

Habitualmente se utilizan los indicadores confianza y soporte para medir la calidad o validez de una regla de asociación e identificar las más importantes o representativas. La confianza es proporcional al número de veces que la regla ha sido verificada. El soporte es una medida de la frecuencia de aparición de los elementos en la base de datos.

Por ejemplo, un supermercado ha expedido 10000 tickets de compra, de modo que *cereales* aparece en 1050 tickets, *leche* aparece en 6200 tickets, y ambos ítems aparecen conjuntamente en 800 tickets. En ese caso, la regla *cereales* $\rightarrow$ *leche* tendría un soporte de $800/10000 = 0{,}08$, y una confianza de $800/1050 = 0{,}7619048$. El 8 % de los tickets llevan *leche* y *cereales*, y el 76,2 % de los tickets que llevan *cereales*, también llevan *leche*. Por tanto la *leche* y *cereales* van asociados con frecuencia (8 %), pero la regla que se cumple es *cereales* $\rightarrow$ *leche*, puesto que la confianza de la regla *leche* $\rightarrow$ *cereales* es de solo $800/6200 = 0{,}1290323$ (casi el

13 %).

El Análisis Estadístico Implicativo propone otra forma de medir la calidad de las reglas de asociación, justificada en el cálculo de probabilidades de los contrajemplos que se pueden encontrar, además de destacar grupos de individuos de la muestra, por su contribución a la calidad de las reglas encontradas, o por ser típicos de las mismas (Sección 2.5.3).

Apriori

Apriori[2] es un algoritmo de que determina reglas de asociación sobre sucesos frecuentes. Se caracteriza por empezar por la identificación de los individuos frecuentes en el conjunto de datos e ir extendiéndo los conjuntos y expandiéndolos siempre y cuando dichos conjuntos aparezcan suficientemente a menudo. Estos conjuntos frecuentes buscados previamente (o "a priori") son los utilizados para buscar reglas de asociación que sirvan para modelizar las tendencias del conjunto de datos.

CHIC utiliza una versión modificada del algoritmo *apriori* para realizar tanto el cálculo de similaridades como la búsqueda de reglas de asociación.

2.3. Clustering

El clustering, tambien llamado segmentación de datos, es una técnica utilizada en la exploración de conjuntos de datos para determinar si pueden ser condensados en un número relativamente pequeño de grupos o clusters de objetos o individuos similares entre ellos y que son diferentes en algunos aspectos de los individuos pertenecientes a otros grupos[6].

Podemos describir un objeto con un conjunto de medidas o por su relacion con otros objetos. Además, algunas veces se intenta descubrir algún tipo de jerarquía en los datos para clasificar los clusters. Esto implica agrupar sucesivamente los clusters de forma que en cada nivel de la jerarquía, los clusters del mismo grupo son más similares entre ellos que respecto a los pertenecientes a otros grupos.

El concepto central del clustering y en el que se basa todo es la llamada **similaridad** entre los objetos que se están agrupando. Un método de clustering agrupa

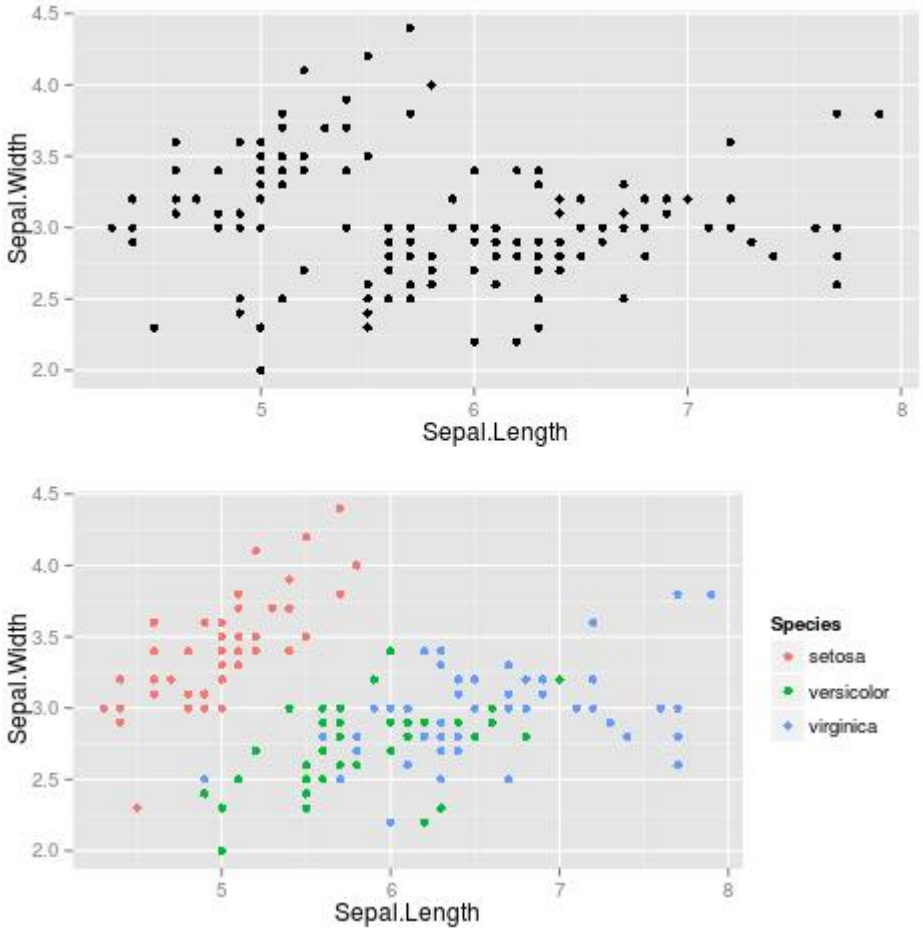

Figura 2.1: Ejemplo de algoritmo de clustering (k-means) usando un conjunto de datos clásico, de clasificación de flores, implementado en R.

los objetos basándose en la noción o función de similaridad que le es provista. Es decir, la agrupación va a depender de consideraciones tomadas fuera de los datos.

El análisis cluster también puede ser utilizado en estadística descriptiva para establecer si un conjunto de datos consta de subgrupos o, en cambio, no se aprecian diferencias sustanciales entre los objetos para poder agruparlos. Esto, por supuesto dependerá del rango de libertad que demos a la similaridad.

Entre las técnicas de clustering más utilizadas encontramos K-means[15], Canopy Clustering[17], Fuzzy c-means[5]... En la Figura 2.1 se puede observar un ejemplo de la aplicación de estos algoritmos.

2.4. Evolución del Análisis Estadístico Implicativo

El Análisis Estadístico Implicativo nace por parte de Régis Gras como un método de la minería de datos para la *modelización estadística de la cuasi-implicación*. Es decir, intenta cuantificar cómo de probable es que suceda la variable b si se ha observado la variable a en la población.

Gras, en su tesis doctoral [7] sobre didáctica, trata de encontrar una estructura de "orden" entre los aprendizajes más que una estructura de aprendizajes similares. En ese momento, Lerman (que forma parte de su tribunal), decide comentarle su método de análisis de similaridades[14], que se diferencia de las técnicas de análisis cluster que se practicaban en el momento por medir las distancias (las *similaridades*) de una forma totalmente diferente, mediante el uso de probabilidades.

A Gras parece interesarle la idea y planea aplicarlo para medir las relaciones de implicación que buscaba en su tesis. En 1981, Gras, Lerman y Rostam publican el artículo[11] que sienta las bases de su análisis implicativo para datos en formato binario: la definición de φ, los modelos que puede seguir... junto con otro artículo aplicando su nueva teoría a los datos de su tesis[12].

La siguiente evolución aparece con la definición del análisis de cohesiones[8], similar al análisis cluster, pero aplicado a las implicaciones y dirigido (es decir, no indica similaridades entre implicaciones sino implicaciones entre implicaciones) y la extensión del uso de φ en datos categóricos y frecuenciales.

Finalmente van apareciendo sucesivamente los conceptos de nodos significativos, par genérico y cuantificadores como la tipicalidad y la contribución, tanto de individuos como de variables suplementarias.

Pese a su denominación, el ASI engloba 3 procedimientos distintos e independientes:

- **Análisis de similaridades o clasificatorio**: *Cluster analysis* con una forma original de medir distancias, las similaridades de Lerman. Se forma un árbol jerárquico similar a un dendrograma (cluster jerárquico ascendente).
- **Análisis Implicativo**: Genera una matriz con todas las implicaciones $a \rightarrow b$ encontradas en los datos y forma un grafo implicativo, con flechas relacionando

las variables con las implicaciones más fuertes.

- **Análisis de cohesiones**: Construye un árbol jerárquico parecido al árbol de similaridades, en el que se marcan de nuevo las variables con implicaciones más fuertes mediante flechas.

2.5. Formalización teórica del Análisis Estadístico Implicativo

En la Sección 2.5.1 se expone el núcleo teórico del análisis implicativo. A continuación, en la Sección 2.5.2 se explica la clasificación a realizar sobre el conjunto de datos de entrada hasta obtener un árbol de similaridad. Finalmente, en la Sección 2.5.3 se concretan los conceptos de tipicalidad y contribución, responsables de la determinación cuantitativa de los descriptores responsables de las estructuras anteriores. El análisis de cohesiones no será tratado en el presente trabajo.

2.5.1. Principios del Análisis Estadístico Implicativo

Sean A_i, A_j individuos del grupo individuos I que verifican las variables a_i y a_j, y la regla $a_i \rightarrow a_j$. El ASI compara el número de contraejemplos $n_{a_i \wedge \overline{a_j}}$ a dicha regla observados en $A_i \cap \overline{A_j}$ con el número de contraejemplos obtenidos tras una extracción aleatoria e independiente de dos subconjuntos X_i y X_j de I con los mismos cardenales que A_i y A_j respectivamente [9] y [13].

Formalizando, se considera un conjunto I compuesto por n individuos y otro conjunto $A = \{a_1, a_2, ..., a_p\}$ formado por p características, suponiendo además que

$$A_i = \{x \in I : a_i(x) = 1\},\ Card(I) = n,\ Card(A_i) = n_{a_i}\ y\ Card(\overline{A_i}) = n_{\overline{a_i}}.$$

Sabiendo que X e Y han de ser elegidas al azar, y suponiendo la independencia de a y b, se pueden elegir diversos modelos de distribución probabilística. En este trabajo se supone el uso de los dos más habituales: el de Poisson y el binomial, dependiendo de si la muestra es resultado de un proceso azaroso o si fue establecida a priori [4].

Sabiendo esto, se definen los tres conceptos básicos del análisis implicativo[18]

Definición 1. *Llamamos* ***índice de implicación*** $q(a_i, \overline{a_j})$ *al indicador de la no-implicación de* a_i *sobre* a_j*.*

$$q(a_i, \overline{a_j}) = \frac{n_{a_i \wedge \overline{a_j}} - \frac{n_{a_i} n_{\overline{a_j}}}{n}}{\sqrt{\frac{n_{a_i} n_{\overline{a_j}}}{n}}}$$

Es la tipificación del número de contraejemplos, usando la media y desviación típica de la distribución de Poisson.

Definición 2. *La* ***intensidad de la implicación*** $\varphi(a_i, a_j)$ *mide la admisibilidad de la regla* $a_i \to a_j$ *o, lo que es lo mismo, la* ***calidad inductiva*** *de* a_i *sobre* a_j*.*

En este caso, para $n_{a_i} \leq n_{a_j} < n$ se define $\varphi(a_i, a_j)$ a partir del índice de implicación $q(a_i \wedge \overline{a_j})$ aproximado al modelo normal:

$$\varphi(a_i, a_j) \begin{cases} 1 - Pr\left[Card(X \cap \overline{Y}) \leq q(a_i, \overline{a_j})\right] = \frac{1}{\sqrt{2\pi}} \int_{q(a_i, \overline{a_j})}^{\infty} e^{-\frac{t^2}{2}} dt & \text{si } n_{a_j} \neq n \\ 0 & \text{si } n_{a_j} = n \end{cases} \tag{2.1}$$

Por supuesto, si no se utiliza la aproximación al modelo normal, se obtendrá un resultado distinto con cada modelo.

Definición 3. *Decimos que la implicación* $a_i \to a_j$ *es* ***admisible al nivel de confianza*** $1 - \alpha$ *si y sólamente si* $\varphi(a_i, a_j) \geq 1 - \alpha$*.*

Comparado con otros métodos estadísticos, el ASI se distingue por el hecho de que utiliza una medida no lineal que satisface diversos criterios:

- Se basa en el número de contraejemplos $n_{a_i \wedge \overline{a_j}}$.
- Tiene capacidad predictiva y sus resultados son extrapolables a distintos tamaños de muestras.
- Encuentra sucesos raros, que pasarían desapercibidos a medidas como el soporte y la confianza, por restringirse a sucesos frecuentes.
- Desoye lo trivial: El grado de implicación mide la sorpresa inherente a una regla. Así pues, se suprimen las reglas triviales, que son potencialmente evidentes y conocidas por el experto.

Bodín mostró un ejemplo demoledor contra esta aproximación[4], en el que demostraba que para muestras grandes la intensidad de la implicación *toma valores poco discriminantes* y obligó a redefinir φ.

Tras la constatación, se complementa el índice de implicación con uno nuevo, llamado índice de implicación-inclusión o versión entrópica de la implicación, definido utilizando la entropía de Shannon. En este caso, la medida no tiene sólo en cuenta la validez de la regla sino la de su contrarecíproca. El nuevo índice, utilizado para muestras grandes, se denomina ψ.

Aunque hoy en día el ASI soporta variables de distintos tipos (binario, modal, frecuencial o de intervalo, incluso fuzzy) este trabajo y su implementación se centra en el caso binario. Es decir, los sucesos únicamente podrán tomar como valor 1 y 0 para cada individuo.

2.5.2. Análisis clasificatorio

El análisis clasificatorio es una técnica de *clustering* con la que se busca encontrar agrupaciones de objetos similares dentro del conjunto de datos. Para ello se busca una forma de medir dichas similaridades entre ellos y estructurar las variables en un árbol de similaridad mediante jerarquía ascendente atendiendo a su semejanza.

El análisis clasificatorio se diferencia del análisis implicativo en que el primero se basa en el número de co-presencias (a, b) en el conjunto de datos de forma que se pueda establecer alguna relación de similaridad, mientras que el segundo busca el número de contraejemplos para cuantificar la sorpresa de la regla. En el Cuadro 2.1 se resumen las similaridades y diferencias entre ambos análisis.

La variable de interés ahora es $Card(X_i \cap X_j)$, la cantidad de individuos que poseen característica a_i y la característica a_j al mismo tiempo. Lo visto anteriormente para la variable $Card(X_i \cap \overline{X}_j)$ es válido para $Card(X_i \cap X_j)$.

Árbol de similaridad

Para formar el árbol de similaridad se calculan los **índices de proximidad** definidos por Lerman[14] tal que:

	φ **de Gras**	**Similaridad de Lerman**
Contraste de hipótesis	$\begin{cases} H_0: \texttt{a y b indep.} \\ H_1: \texttt{a} \rightarrow \texttt{b} \end{cases}$	$\begin{cases} H_0: \texttt{a y b indep.} \\ H_1: \texttt{a similar a b} \end{cases}$
Estad. contr.	$T := N_{a\bar{b}}$ =Número de contraejemplos $(a, \bar{b})$	$T := N_{ab}$ =Número de copresencias (a, b)
Proced. contr.	Rechazar H_0 si $T \leq c$	Rechazar H_0 si $T \geq c$
Muestra	resumida con $t = n_{a\bar{b}}$	resumida con $t = n_{ab}$
Def.	$\varphi(a \rightarrow b) := 1 - p-$valor de t	$s(a, b) := 1 - p-$valor de t

Cuadro 2.1: Comparativa Análisis implicativo-Análisis de similaridad.

$$s(a_i, a_j) = Pr\left[Card(X_i \cap X_j) \leq K\right] \tag{2.2}$$

donde $K := Card(A_i \cap A_j)$ son las copresencias observadas entre a_i y a_j y el cálculo Pr dependerá de la ley de probabilidad asumida.

Centrando la atención en las distribuciones Poisson y Binomial y aproximando estas a la normal, las similaridades de cada pareja (a_i, a_j) al nivel cero de la jerarquía se obtienen mediante la siguiente fórmula:

$$s(a_i, a_j) = Pr\left[\frac{Card(X_i \cap X_j) - \frac{n_{a_i} * n_{a_j}}{n}}{\sqrt{\frac{n_{a_i} * n_{a_j}}{n}}} \leq K_c\right] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{K_c} e^{-\frac{1}{2}x^2} dx \tag{2.3}$$

donde $K_c := \frac{K - \frac{n_{a_i} * n_{a_j}}{n}}{\sqrt{\frac{n_{a_i} * n_{a_j}}{n}}}$ es el número de copresencias, tipificado por la media y desviación típica de la distribución.

El nivel raíz del árbol está formado por todas las variables que aparecen en los datos de entrada, a las que se les aplica el cálculo de los índices de similaridad a nivel cero dos a dos para almacenarlos en una matriz bidimensional.

A partir de este punto y para cada nivel se buscan los nuevos índices al combinar la clase (a_i, a_j) con mayor índice en el nivel anterior con:

- cada variable aislada:

$$s((a_i, a_j), a_k) = [\text{máx}\{s(a_i, a_j); s(a_i, a_k)\}]^2$$

- con clases previamente formadas:

$$s((a_i, a_j), (a_{k1}, a_{k2}, a_{k3})) = [\text{máx}\{s(a_i, a_{k1}), s(a_i, a_{k2}), s(a_i, a_{k3}), s(a_j, a_{k1}), s(a_j, a_{k2}), s(a_j, a_{k3})\}]^{2x3}$$

- y en general, siendo C_1, C_2 clases previamente formadas:

$$s(C_1, C_2) = [\text{máx}\{s(a_j, a_k) : a_j \in C_1, a_k \in C_2\}]^{\text{Card}(C_1)\times\text{Card}(C_2)}$$

Nodos significativos

Un calculo de interés a aplicar al árbol de similaridad es la búsqueda e identificación de nodos significativos. Un nodo significativo es un nodo que se quiere destacar de los demás, por reunir en su seno a variables que mantienen una similaridad (cuando se mira por parejas) mejor que en otros niveles.

Para ello primero necesitamos definir los conceptos de preorden inicial y global, nivel significativo y nodo significativo como[18]:

Definición 4. *Se llama preorden inicial y global Ω sobre $A \times A$ al preorden inducido por la aplicación S (similaridad) sobre $A \times A$.*

$$G_s(\Omega) = \{((a, b); (c, d)) : s(a, b) < s(c, d)\} \tag{2.4}$$

Sean $S\Pi_k$ y $R\Pi_k$ el conjunto de pares separados y reunidos respectivamente al nivel k del árbol de similaridades. En este caso, los pares de parejas que al nivel k respetan el preorden inicial forman $G_s(\Omega) \cap [S\Pi_k \times R\Pi_k]$.

Al cardinal de $G_s(\Omega) \cap [S\Pi_k \times R\Pi_k]$ se le asocia un indice aleatorio $G_s(\Omega^*) \cap [S\Pi_k \times R\Pi_k]$, donde Ω^* es un preorden aleatorio en general con probabilidad uniforme de todos los preordenes del mismo tipo cardinal que Ω.

Este índice tiene:

- por esperanza: $\frac{1}{2}s_k r_k$
- por varianza $\frac{s_k r_k (s_k + r_k + 1)}{12}$

siendo $Card(S\Pi_k) = s_k$ y $Card(R\Pi_k) = r_k$. El índice tipificado se define como:

$$S(\Omega, k) = \frac{Card[G(\Omega) \cap [S\Pi_k \times R\Pi_k]] - \frac{1}{2}s_k r_k}{\sqrt{\frac{s_k r_k (s_k + r_k + 1)}{12}}} \tag{2.5}$$

Definición 5. *Se denomina* ***nivel significativo*** *a aquel que corresponda a un máximo local de $S(\Omega, k)$ durante la construcción de la jerarquía. En este caso se dirá que la división Π_k está en* ***resonancia parcial*** *con Ω. Si además $G(\Omega) \cap [S\Pi_k \times R\Pi_k] = [S\Pi_k \times R\Pi_k]$, diremos que la división Π_k está en* ***resonancia total*** *con Ω.*

Definición 6. *Se identifica como* ***nodo significativo*** *cualquier nodo formado a un nivel que corresponde a un máximo local de $v(\Omega, k)$, donde*

$$v(\Omega, k) = S(\Omega, k) - S(\Omega, k-1) \tag{2.6}$$

2.5.3. Tipicalidad y contribución

Para completar el análisis se dispone de medidas o cuantificadores avanzados para tratar de medir la relación de los individuos con la regla $a \to b$, buscando saber los responsables de su creación, los más significativos en su formación y cuáles no aportan información sobre la misma.

De este modo, la **tipicalidad** es un índice porcentual que cuantifica cómo un individuo concreto se comporta en relación a la regla, cuán "típico" es. Del mismo modo, la **contribución**, nos proporcionará una forma de cuantificar cómo ha contribuido un determinado individuo a formar la calidad de la regla $a \to b$.

De un modo formal[18]:

Definición 7. *Se define como* ***sujeto típico*** *aquel que verifica todas las implicaciones que poseen mayor intensidad de implicación en la formación de las clases.*

Definición 8. *El par (a, b) tal que: $\psi(a, b) \geq \psi(i, j)\ \forall i \in A$ y $\forall j \in B$ es denominado* ***par genérico de la clase*** *C. Llamaremos* ***Implicación Genérica de*** *C al número $\psi(a, b)$.*

Dado el par genérico (a, b), para cada individuo x de la muestra, se define $\psi_x(a, b)$ como:

$$\begin{array}{lll} \psi_x(a,b) = 1 & \text{si } b(x) = 1 & \\ \psi_x(a,b) = 0 & \text{si } a(x) = 1 & \text{y } b(x) = 0 \\ \psi_x(a,b) = p & \text{si } a(x) = 0 & \text{y } b(x) = 0 \end{array}$$

En el caso de que la clase C tuviera g subclases (anidadas), se puede tomar cada uno de los g pares genéricos. Se define la distancia de un individuo x a la clase C:

$$d^2(x, C) = \frac{1}{g} \sum_{i=1}^{g} \frac{[\psi_i - \psi_{x,i}]^2}{1 - \psi_i} \tag{2.7}$$

donde para cada i que indica cada subclase de C, ψ_i denota la implicación genérica de esa subclase y $\psi_{x,i}$ el valor de la implicación genérica para el individuo.

Definición 9. *La **Tipicalidad** del individuo x se define como*

$$\gamma_T(x, C) = 1 - \frac{d(x, C)}{\text{máx}_{y \in I}\, d(y, C)} \tag{2.8}$$

Si x es un sujeto típico verifica todas las implicaciones que poseen mayor intensidad de implicación en la formación de las clases, esto es, $\psi_{x,i} = \psi_i, i = 1, ..., g$. Por tanto, $d(x, C) = 0$ y $\gamma(x, C) = 1$.

Por contra, cuando x es el que más en desacuerdo está con C, significa que $d(x, C) = \text{máx}_{y \in I}\, d(y, C)$ y, por tanto, $\gamma_T(x, C) = 0$.

Definición 10. *Llamamos **grupo óptimo** a aquel grupo de individuos con las mayores tipicalidades.*

Definición 11. *La **Contribución** de un individuo x a la clase C se define como*

$$\gamma_C(x, C) = 1 - \tilde{d}(x, C), \tag{2.9}$$

donde

$$\tilde{d}^2(x, C) = \frac{1}{g} \sum_{i=1}^{g} [1 - \psi_{x,i}]^2$$

es otra distancia del individuo x a la clase C.

Si x es óptimo $\gamma_C(x, C) = 1$, en cuyo caso $\psi_{x,i} = 1 \; \forall$ regla i.

2.6. CHIC: *Classification Hiérarchique Implicative et Cohésitive*

CHIC es un programa informático desarrollado exclusivamente para el estudio y la aplicación del ASI.

En sus incicios, tanto CHIC como el ASI sólo trataba con variables binarias, enriqueciéndose más tarde con variables modales y frecuenciales.

A lo largo de su ciclo de vida, se han ido incorporando todos los análisis que han ido surgiendo en torno al ASI, incluyendo los tres principales: el análisis clasificatorio, el análisis implicativo y el análisis de cohesiones. Para cada uno de estos análisis proporciona también la posibilidad de realizar los cálculos tanto en la versión clásica de la implicación o en su versión entrópica, elección que influirá en gran medida en las reglas producidas.

Además de realizar dichos análisis y generar un informe con los resultados, CHIC proporciona tres métodos de representación gráfica:

- **Árbol de similaridad:** Resultado del análisis clasificatorio basado en las similaridades de Lerman. Figura 2.2. Destaca los nodos significativos (trazos rojos gruesos) e intensidades implicativas del análisis de cohesiones, como se puede ver en la Figura 2.4.
- **Grafo implicativo:** Grafo resultado del análisis implicativo que representa las intensidades de implicación. Permite seleccionar al usuario las reglas y variables que desea visualizar. Figura 2.3.
- **Árbol jerárquico orientado:** Resultado de la aplicación del ASI. Marca los nodos significativos del mismo modo que el árbol de similaridad. Figura 2.4.

A parte de los modos de representación descritos, CHIC proporciona la posibilidad de calcular la contribución y tipicalidad de una regla para cada método de representación.

Debido a las restricciones temporales del proyecto, se decide centrar el trabajo en la reproducción del comportamiento de CHIC para variables binarias generando un informe, dejando la implementación de los otros tipos de variables y la representación de los árboles y el grafo para más adelante.

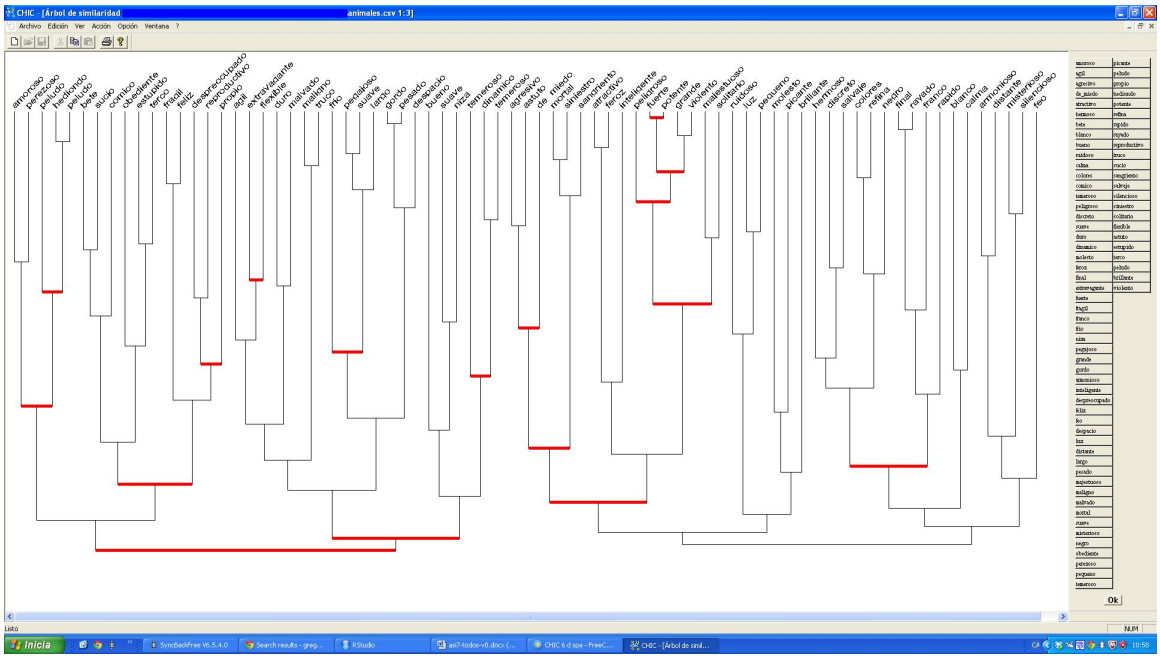

Figura 2.2: Árbol de similaridades.

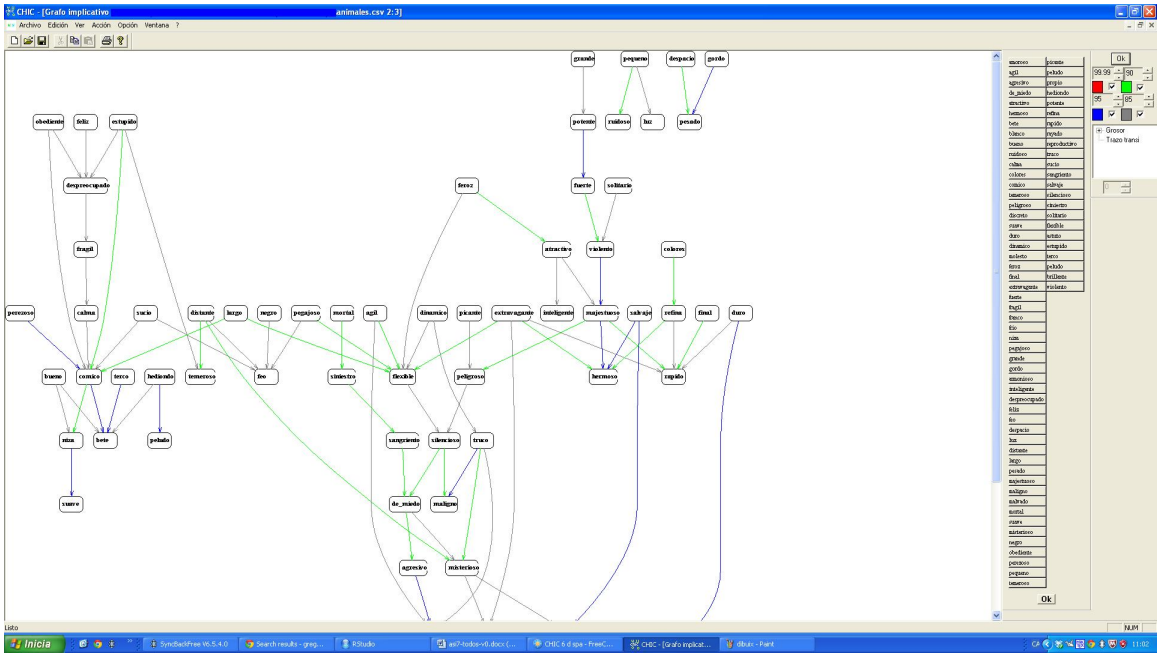

Figura 2.3: Grafo implicativo.

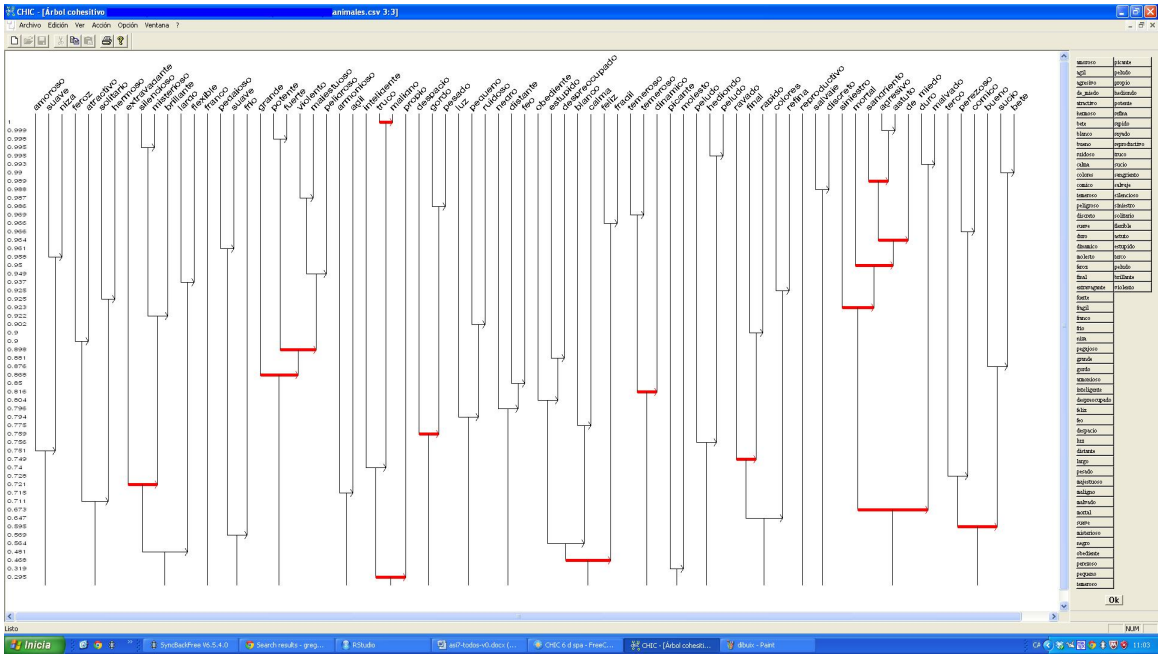

Figura 2.4: Árbol de cohesiones.

Capítulo 3

Programación en GNU R

El lenguaje de programación R se ha convertido prácticamente en un estándar en el ámbito estadístico. A su amplia –y creciente en número– variedad de librerías o extensiones, se une una comunidad de desarrolladores de perfil técnico muy activa y una relativa facilidad de desarollo.

Sin embargo, estas justificaciones no son suficientes para tomar la decisión en cuanto a la programación de la librería. ¿Es R el lenguaje ideal para el desarrollo del programa? ¿Qué limitaciones podemos encontrar? ¿Cómo se estructura un paquete?

En este capítulo se describe el proceso de programación del paquete R, empezando con el razonamiento inicial que lleva a la elección de R como lenguaje de programación (Sección 3.1) y continuando con la descripción del concepto de paquete y su programación (Sección 3.2).

3.1. ¿Por qué R?

Desde nuestro punto de vista R tiene muchas ventajas:

- **Amplia base de usuarios de perfil estadístico:** R nació como un lenguaje orientado al trabajo estadístico y fue rápidamente adoptado por la comunidad investigadora como una alternativa gratuita y libre a S. Esta comunidad no ha parado de crecer y por el camino ha ido desarrollando y ampliando la

funcionalidad de R adaptándola a sus necesidades, lo que a su vez ha seguido atrayendo a más usuarios necesitados de dichas funcionalidades y capacitados para seguir contribuyendo. Esto ha resultado históricamente muy útil a la hora de corregir, mejorar e incluso avanzar en la investigación de cualquiera de estos campos.

- **Flexibilidad:** Todas las herramientas estándar usadas en el ámbito de la estadística están implementadas en R: desde el acceso a los datos en diferentes formatos hasta su manipulación, pasando por tanto los modelos estadísticos tradicionales como los más modernos. Todo esto incluido en una infraestuctura orientada a objetos que hace sencillo obtener los resultados concretos buscados en lugar de cortar y pegar de un informe estático.

- **Acceso a las últimas técnicas de análisis:** Los investigadores y académicos utilizan R para implementar los últimos métodos estadísticos, de aprendizaje automático y modelización predictiva. Hay extensiones que siguen creciendo en ámbitos como las finanzas, la genómica y docenas de otros campos. Hoy en día hay más de 2000 paquetes R disponibles para su descarga, número que crece cada día.

- **Plug as you need:** No hay necesidad de instalar todo el conjunto de paquetes y librerías existentes para R. Su modularidad permite que en cada ocasión se instalen únicamente los paquetes a utilizar, que sólo necesitan ser referenciados desde el que se está desarrollando y el mismo R se encarga del proceso de instalación.

- **Extensibilidad:** La característica modulable de R hace del mismo un lenguaje extremadamente extensible, permitiendo ampliar su funcionalidad de una manera sencilla.

3.1.1. Limitaciones. R contra C++

- **Lenguaje interpretado. Lento:** R es un lenguaje interpretado y, por tanto, más lento que uno compilado. Pese a todo se valora su base de usuarios y su sencillez sobre el rendimiento. En la Figura 3.1 se puede observar una tabla comparativa respecto a otros lenguajes utilizados en el ámbito científico y en la Figura 3.2 puede observarse esta comparativa en una gráfica, obtenidas ambas del sitio web del lenguaje de programación Julia[1][2].

[1] http://julialang.org/
[2] http://julialang.org/benchmarks/

	Fortran gcc 4.8.1	Julia 0.2	Python 2.7.3	R 3.0.2	Matlab R2012a	Octave 3.6.4	Mathe- matica 8.0	JavaScript V8 3.7.12.22	Go go1
fib	0.26	0.91	30.37	411.36	1992.00	3211.81	64.46	2.18	1.03
parse_int	5.03	1.60	13.95	59.40	1463.16	7109.85	29.54	2.43	4.79
quicksort	1.11	1.14	31.98	524.29	101.84	1132.04	35.74	3.51	1.25
mandel	0.86	0.85	14.19	106.97	64.58	316.95	6.07	3.49	2.36
pi_sum	0.80	1.00	16.33	15.42	1.29	237.41	1.32	0.84	1.41
rand_mat_stat	0.64	1.66	13.52	10.84	6.61	14.98	4.52	3.28	8.12
rand_mat_mul	0.96	1.01	3.41	3.98	1.10	3.41	1.16	14.60	8.51

Figura 3.1: Tabla comparativa del rendimiento de distintos lenguajes de programación respecto a C (C=1). Cada fila representa una tarea diferente como benchmark: ordenación, multiplicación de matrices, sumatorios, cálculo de series de Fibonacci...

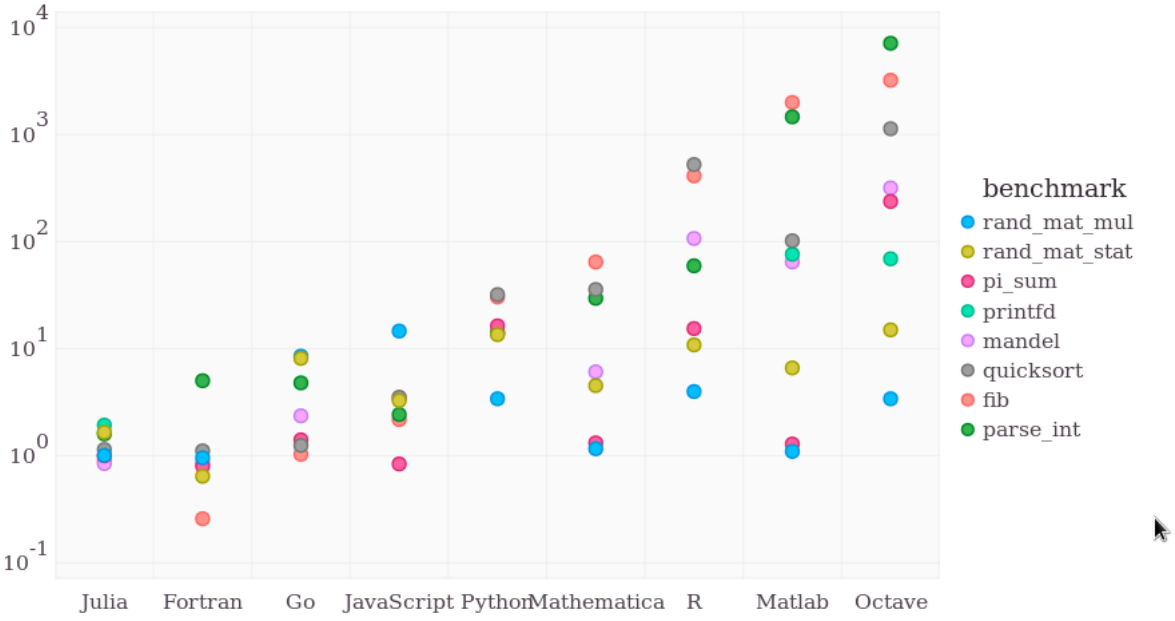

Figura 3.2: Gráfico comparativo del rendimiento de distintos lenguajes de programación respecto a C (C=1). Cada color representa una tarea diferente como benchmark: ordenación, multiplicación de matrices, sumatorios, cálculo de series de Fibonacci...

- **Auge de Python en el ámbito científico:** Pese a que R sigue siendo la referencia en lenguajes orientados a la estadística, Python se ha ido erigiendo en los últimos años como el lenguaje a utilizar en el ámbito científico y ha provocado una tendencia migratoria hacia él desde lenguajes como C++, R y, sobre todo, MATLAB.

- **Lenguaje funcional, técnico, eminentemente orientado al prototipado:** No proporciona programación orientada a objetos pura. Como se explicará en la Sección 3.2, se toma la decisión de experimentar con ello en el desarrollo de este paquete.

- **C++ es más rápido, pero complejo:** C++ es mucho más rápido que R. Sin embargo, tal y como CHIC (ver Sección 3.1.1) está programado, mucho más difícil de entender y modificar. En la Sección 4.2 se comparan las dos aproximaciones (RCHIC y su reescritura en R) y se valora la decisión tomada.

- **Librerías gráficas no orientadas a la interactividad. Complejidad:** Pese al gran abanico de opciones que ofrece R en la producción de representaciones gráficas, no hay ningún paquete sencillo de utilizar orientado al manejo de grafos interactivos y árboles jerárquicos de gran tamaño. Requisito de Raphaël Couturier como persona a cargo del proyecto para RCHIC, no se ha alcanzado dicha etapa en la implementación del paquete sobre el que versa el trabajo.

RCHIC

La primera propuesta surgida fue portar el código del CHIC a R haciendo uso únicamente de una interfaz que conectara el programa R y la respresentación gráfica de los resultados con el código original en C++. Con ello nació el paquete RCHIC.

Sin embargo el uso de una interfaz y el hecho de que el código C sea tan complejo impide tanto su inspección como su extensión. Esto choca con el propósito del proyecto: la difusión de la teoría y aplicaciones del Análisis Estadístico Implicativo.

Se acuerda intentar reescribirlo desde el inicio intentando reproducir los resultados del CHIC original.

Actualmente el paquete RCHIC original sigue en desarrollo por parte de Raphaël Couturier centrándose en los resultados gráficos, al tomar la mayor parte de su implementación del programa CHIC original.

```
Package: pkgname
Version: 0.5-1
Date: 2004-01-01
Title: My First Collection of Functions
Authors@R: c(person("Joe", "Developer", role = c("aut", "cre"),
                    email = "Joe.Developer@some.domain.net"),
             person("Pat", "Developer", role = "aut"),
             person("A.", "User", role = "ctb",
                    email = "A.User@whereever.net"))
Author: Joe Developer and Pat Developer, with contributions from A. User
Maintainer: Joe Developer <Joe.Developer@some.domain.net>
Depends: R (>= 1.8.0), nlme
Suggests: MASS
Description: A short (one paragraph) description of what
  the package does and why it may be useful.
License: GPL (>= 2)
URL: http://www.r-project.org, http://www.another.url
BugReports: http://pkgname.bugtracker.url
```

Figura 3.3: Formato del archivo DESCRIPTION

3.2. Programación de un paquete R

En el momento de programar el paquete R se deberá tener en cuenta la estructura general a seguir y tomar las decisiones de implementación adecuadas. Como se ha indicado con anterioridad, R consta de muchos paquetes que extienden su funcionalidad básica, por lo que se busca no replicar funcionalidades en la medida de lo posible. Lamentablemente, ninguno de los paquetes existentes relacionados con la minería de datos y clustering implementan los métodos necesarios.

Se ha decidido experimentar con Programación Orientada a Objetos (POO) así que deben tenerse en cuenta las diferentes aproximaciones implementadas por R y elegir la más adecuada remarcando pros y contras.

Se repasa la estructura de un paquete R en la Sección 3.2.1 y se introduce al lector a las diferentes implementaciones de la POO en R justificando la decisión tomada en la Sección 3.2.2.

3.2.1. Estructura de un paquete R

- Archivo **DESCRIPTION**: Este archivo contiene la información básica del paquete. Como se observa en la Figura 3.3
- Archivo **NAMESPACE**: Contiene el espacio de nombres del paquete, esto es, el nombre de las funciones exportadas del paquete y reconocidas por el intérprete.

- Subdirectorios:
  - **R**: Contiene el código fuente en lenguaje R.
  - **data**: Almacena los datos a utilizar tanto por el programa como por sus ejemplos incluidos en la carpeta demo.
  - **demo**: Incluye las demostraciones del funcionamiento del paquete.
  - **inst**: Autogenerado durante la instalación del paquete.
  - **man**: Contiene la documentación necesaria relativa al funcionamiento del paquete y sus funciones. Los diversos archivos se combinan para formar código fuente LaTeX.
  - **test**: Incluye los archivos de test a ejecutar durante la compilación y creación del paquete.
  - **src**: Almacena si lo hubiera el código de otros lenguajes necesario para la ejecución del paquete.

3.2.2. *Reference Classes* y programación orientada a objetos en R

La **Programación Orientada a Objetos** es un paradigma de programación surgido en los años 90 que se caracteriza por usar **objetos** en sus interacciones. El comportamiento de estos objetos se describe por una **clase** a través de los atributos de los mismos y su relación con otras clases.

Herencia

La **herencia** es un mecanismo de la POO que permite crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La nueva clase hereda los métodos (comportamiento) de la clase padre, pero no necesariamente sus atributos.

Una **clase abstracta** es una clase base no instanciable que define métodos pero no tiene por qué implementarlos. Las clases que heredan de una clase abstracta completarán su implementación o, por contra, serán declaradas como clases abstractas. Una clase abstracta no puede ser instanciada.

Llamamos **interfaz** a la clase abstracta que en lugar de contener datos y código define el comportamiento y las cabeceras de las funciones a heredar sin implementarlas. Se dice que una clase implementa a una interfaz cuando contiene los datos e implementa el código para cada uno de los métodos correspondientes a dicha interfaz.

S3, S4 y Reference Classes

R ofrece 3 aproximaciónes a la POO: S3, S4 y Reference Classes [20].

- **S3**: Toma la aproximación a la orientación a objetos del uso de funciones genéricas[3], donde una función especial llamada *función genérica* es la que decide, en lugar de hacerlo la clase como es habitual en los lenguajes de programación más conocidos, a que métodos se debe llamar.
- **S4**: Funciona como S3, pero añade otro nivel de formalidad. Las principales diferencias entre S3 y S4 son que S4 define las clases de una manera formal, describiendo los mecanismos de herencia en el proceso; y que S4 tiene métodos especiales llamados helpers que definen métodos y clases genéricas.
- **Reference classes**: Implementa la programación orientada a objetos desde el punto de vista del paso de mensajes[4], por lo que los métodos son propiedad de las clases, y no de las funciones. Además, los objetos de la clase son mutables. Es decir, no utiliza la aproximación tradicional en R (copiar los objetos cada vez que hay una modificación) sino que permite modificar el mismo objeto evitando la operación de copia. Esto permite un ahorro de memoria importante en determinados casos, y añade mayor flexibilidad de la que podemos obtener con S3 y S4.

Además de la mutabilidad de los objetos, la mayor parte de los lenguajes de programación utilizados hoy en día (Java, C++, C#) implementan la orientación a objetos por paso de mensajes, por lo que la más que posible familiaridad del usuario hacia esta aproximación influye también en la selección de Reference Classes como la aproximación a tomar en nuestra implementación.

[3] Generic Functions (Wikipedia.org): http://en.wikipedia.org/wiki/Generic_function.
[4] Message Passing (Wikipedia.org): http://en.wikipedia.org/wiki/Message_passing.

Capítulo 4

Diseño e implementación del paquete ReRCHIC

Todo programa requiere de un diseño previo al desarollo de la aplicación. Este diseño planifica, identifica posibles áreas de dificultad y evita rectificaciones, rediseños constantes y cambios costosos durante el proceso de implementación.

Por supuesto, el diseño es flexible, y, aunque se consiga tener en cuenta la mayoría de los escenarios posibles, durante el proceso de implementación se suelen encontrar dificultades que suponen cambios en dicho diseño.

Este capítulo describe tanto el proceso de diseño (Sección 4.1) como la implementación llevada a cabo (Sección 4.2), explicando detalladamente cada una de las estructuras y algoritmos resultantes, así como los problemas encontrados que provocan cambios en el diseño inicial.

Finalmente, en la Sección 4.3 se describe el proceso de experimentación y generación de resultados y se realiza una comparación de los resultados obtenidos por el paquete desarrollado con los obtenidos por el original RCHIC programado en C++ y adaptado mediante interfaces de llamada a R.

4.1. Diseño

El objetivo de esta sección es mostrar las decisiones tomadas en el proceso de diseño e implementación del proyecto. Se prioriza un diseño que facilite la comprensión y el desarrollo sin dejar de lado la eficiencia computacional.

Históricamente las aplicaciones orientadas a entornos científicos se han centrado en el rendimiento. Los avances tecnológicos actuales han permitido que las computadoras puedan procesar varios millones más de operaciones por minuto y el código ha ido aumentando en tamaño y complejidad hasta hacer patente la necesidad de establecer unas guías para hacer más sencillo su desarrollo y su mantenimiento.

Esto provoca que se busque inspiración en el área de la Ingeniería del Software y se decida explorar su camino para observar su estado actual en lenguajes como R y qué limitaciones podemos encontrar en ellos.

En la Sección 4.1.1 se expone el proceso de diseño seguido y cómo nuevos requisitos han provocado cambios en el mismo.

4.1.1. Diseño de la arquitectura

Hoy en día y cada vez más, incluso los científicos de perfil más teórico están involucrados en procesos de desarrollo de software informático. La programación ha pasado de ser una herramienta auxiliar a ser una parte fundamental del trabajo diario, con proyectos que se han ido alargando en el tiempo, aumentando de complejidad y cambiando de manos habitualmente. No es extraño pues que cada vez más se centre la atención en el proceso de desarrollo del mismo para hacer viable y más secillo su mantenimiento y modificación a lo largo de su ciclo de vida.

Este proyecto sirve también para estudiar, practicar, experimentar y aplicar en la medida de lo posible el estudio del diseño de software en un ámbito puramente científico y con un lenguaje técnico como es R. Se han pretendido seguir para ello las recomendaciones habituales para el diseño y desarrollo del software, en concreto las descritas por Robert C. Martin en Clean Code [16] y su serie online Clean Coders[1].

Las siguientes secciones se centran en el proceso de diseño de la arquitectura.

[1] http://cleancoders.com/

La deuda técnica

¿Por qué darle tanta importancia al diseño y a una correcta implementación del software? Ward Cunningham propuso el concepto de deuda técnica[2] como metáfora de la deuda financiera.

En el momento en el que se contrae una deuda financiera, incluso aunque se pague a tiempo, se incurre en pago de intereses, por lo que se está pagando más de lo recibido a cambio. Si no se paga la deuda irá creciendo y sus intereses aumentando. Si se deja crecer la deuda el tiempo suficiente se volverá impagable.

Análogamente, en el momento en el que retrasan las decisiones de diseño y se toma el camino rápido se está incurriendo en pago de intereses, que aparecerán en forma de esfuerzo extra en el futuro desarrollo del programa. En ese momento quedan dos opciones: se puede continuar pagando el interés o se puede reducir la deuda principal refactorizando el diseño rápido y sucio en un buen diseño. Aunque la segunda opción es más costosa, acaba resultando más beneficiosa por la reducción de los intereses a pagar en el futuro.

Pesea a todo, en parte por limitaciones temporales que acompañan este tipo de proyectos y en parte por la todavía hoy ignorancia del autor acerca de su adecuada implementación, se omiten prácticas como Test Driven Development[3][3].

Modularidad

El paquete será programado de forma modular. La modularización del software consiste en un conjunto de técnicas que permiten (1) escribir un módulo del programa con poco conocimiento del código en otro módulo y (2) reensamblar y sustituir módulos del programa sin necesidad de reensablar todo el sistema [19].

Los beneficios que obtenemos al decidir implementar el paquete con estas técnicas vienen dados en términos de:

- **Mantenimiento**: El tiempo de desarrollo se acorta al no necesitar información de los otros módulos y, por ejemplo en el caso de que varios grupos de trabajo

[2] Debt metaphor(YouTube.com):https://www.youtube.com/watch?v=pqeJFYwnkjE
[3] TDD (Wikipedia.org): http://en.wikipedia.org/wiki/Test-driven_development

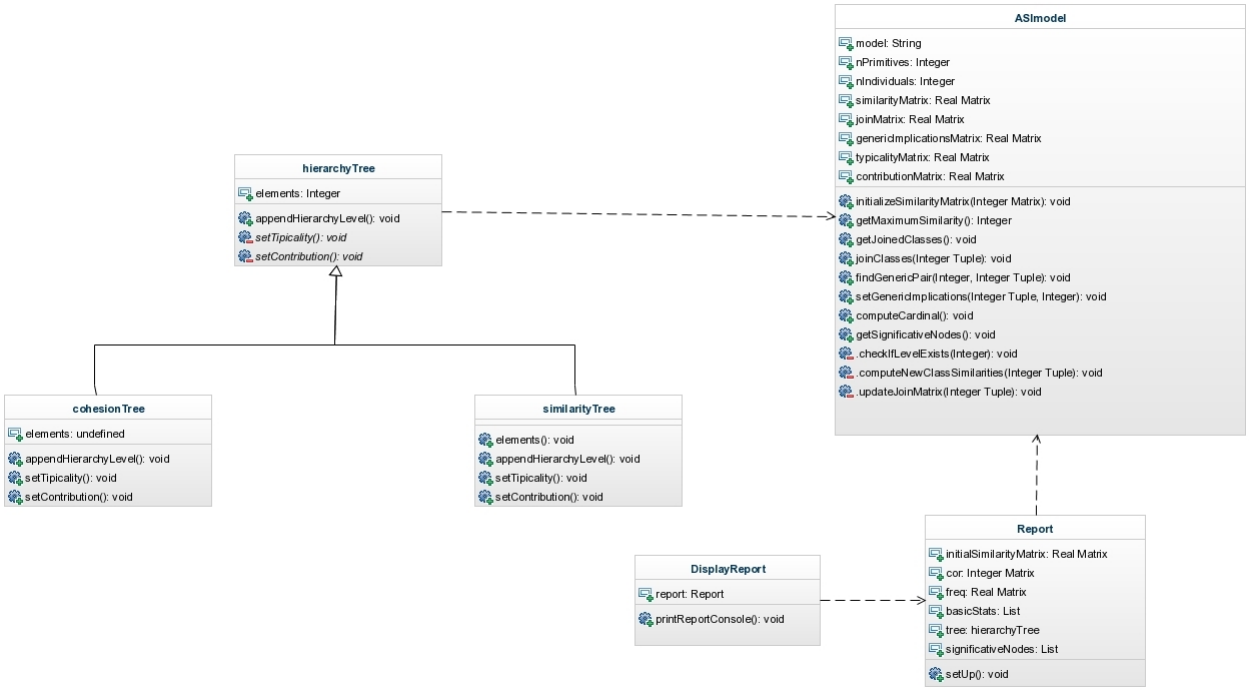

Figura 4.1: Diagrama de clases del diseño inicial.

se encarguen de diferentes módulos, se reduce la comunicación.

- **Flexibilidad**: Se pueden realizar cambios drásticos en un módulo sin necesidad de cambiar los otros.
- **Comprensibilidad**: Debe ser posible estudiar el sistema módulo a módulo. El sistema, por tanto, puede diseñarse mejor debido a esta facilidad de comprensión.

Diseño inicial

Tras analizar el problema, se observan y deciden implementar tres estructuras bien definidas: El modelo que va a implementar todas las operaciones, al que llamaremos *ASImodel*; Una interfaz árbol jerárquico (*hierarchyTree*) que será implementada por los diferentes tipos de árboles a crear que contengan la información necesaria para los resultados (*similarityTree, cohesionTree*); y una serie de funciones que sirvan como interfaz entre el usuario y el algoritmo que llamará al modelo.

Tras la primera implementación del programa, se solicita que se añada la funcio-

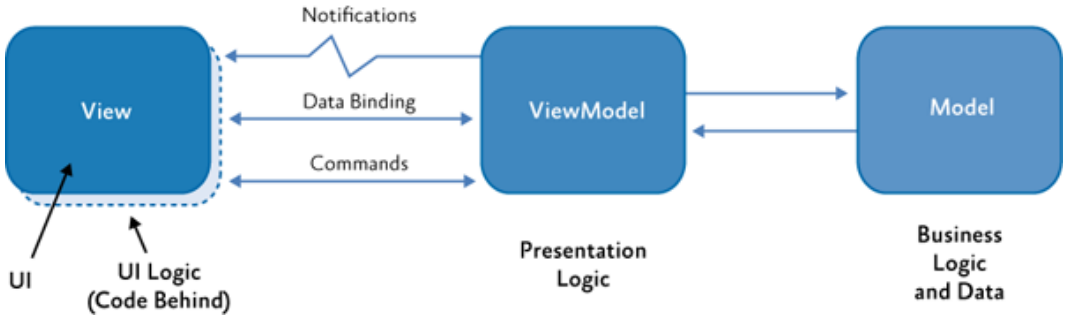

Figura 4.2: Esquema del patrón MVVM.

nalidad de generar un informe similiar al generado por CHIC.

Al cambiar los requisitos se vuelve a analizar el diseño y se intuye como posible mejor opción de implementación de un patrón de arquitectura MVVM[4] (Figura 4.2), con la clase *report* como databinder o view-model y una nueva clase *displayReport* como presentador o vista.

Esta misma estructura podrá ser aprovechada para la futura representación gráfica de la clase árbol. En la Figura 4.1 se puede observar el diagrama de clases inicial del proyecto.

4.1.2. Estructuras de datos

En este diseño inicial se implementa el algoritmo como una función que interactua con el modelo principal, *ASImodel*, que se encarga de las clasificaciones y los análisis.

Mientras, el tratamiento de los árboles y características del modelo a obtener (tipicalidad, contribución) se define e implementa a partir de las clases *similarityTree* y *cohesionTree* que implementan la interfaz dada por la clase abstracta *hierarchyTree*.

Árbol jerárquico

Inicialmente se diseña una interfaz **hierarchyTree** que define la estructura árbol jerárquico y consta de tres métodos y un atributo:

[4]Model View ViewModel: http://en.wikipedia.org/wiki/Model_View_ViewModel

```
> tree
[1] "( atractivo , amoroso )"
[2] "( ( atractivo , amoroso ) , blanco )"
[3] "( ( ( atractivo , amoroso ) , blanco ) , agil )"
[4] "( de_miedo , agresivo )"
[5] "( ( de_miedo , agresivo ) , ( ( ( atractivo , amoroso ) , blanco ) , agil ) )"
[6] "( bete , hermoso )"
[7] "( ( bete , hermoso ) , ( ( de_miedo , agresivo ) , ( ( ( atractivo , amoroso ) , blanco ) , agil ) ) )"
```

Figura 4.3: Estructura tree en formato Newick o dendograma.

- El atributo **elements** es un vector de tuplas que contiene las parejas de elementos ordenados del árbol, ordenados por nivel.

- El método **appendHierarchyLevel** es el corazón de la clase y, en un principio el único que debería ser público. Es el que se encarga de llamar a los otros dos métodos e introducir el nuevo nivel en el atributo elements. En él se resume toda la funcionalidad de la clase.

- El método **computeSignificativeLevels** obtiene los pares significativos en cada uno de los niveles.

- El método **setTipicallity** es el encargado de calcular la tipicalidad.

- El método **setContribution** calcula la contribución.

Las clases ***similarityTree*** y ***cohesionTree*** implementan los métodos de esta interfaz a su medida.

Durante la implementación del algoritmo y, en especial del modelo *ASImodel*, surgen problemáticas y dependencias entre métodos del modelo y métodos de la clase *hierarchyTree* difíciles de resolver.

Esto provoca revisar el diseño y, ya con un mayor conocimiento de los requisitos al tener el programa avanzado, se observa que la implementación de una interfaz es innecesaria e incluso excesiva si se integran el cálculo de niveles significativos, tipicalidad y contribución dentro del modelo principal.

De esta forma, el árbol pasa a ser una lista de cadenas, representando los diversos niveles en formato dendograma. Puede observarse un ejemplo de esta estructura árbol final en la Figura 4.3.

Modelo ASImodel

Esta es la clase que define el modelo y las operaciones con las que trabaja el algoritmo. Debe implementar todas y cada una de las operaciones necesarias para la manipulación de los datos de entrada, así como los atributos para almacenar los datos a consultar por las funciones.

Una corta descripción de los atributos de la clase ***ASImodel***:

- **model**: Modelo de distribución probabilística correspondiente a los sucesos. Únicamente las distribuciones Poisson y binomial han sido implementadas por el momento.
- **nPrimitiveClasses**: Número de clases primitivas.
- **nIndividuals**: Número de Individuos.
- **similarityMatrix**: Matriz de similaridades. Muta en cada iteración de la construcción del árbol jerárquico.
- **joinMatrix**: Matrix de uniones. Señala qué clases están unidas entre sí y a qué nivel del árbol jerárquico se unen.
- **genericImplicationsMatrix**: Matriz de implicaciones genéricas.
- **typicalityMatrix**: Matriz de tipicalidad de los individuos a cada nivel.
- **contributionMatrix**: Matriz de contribucción de los individuos a cada nivel.

y de sus funciones públicas:

- **initialize**: Constructor de la clase.
- **initializeSimilarityMatrix**: Calcula las similaridades entre los datos de entrada.
- **getMaximumSimilarity**: Devuelve la máxima similaridad de la matriz de similaridades.
- **getJoinedClasses**: Devuelve las clases que han sido unidas con otras en un nivel determinado, por defecto el último.

- **joinedWithClass**: Obtiene las clases unidas con una en particular en un nivel determinado, por defecto el último.
- **joinClasses**: Une un par de clases determinadas por una Tupla.
- **findGenericPair**: Devuelve el par genérico del nivel indicado por argumento, por defecto el último.
- **setGenericImplications**: Asigna los valores de la matriz de implicaciones genéricas para un par genérico y un nivel pasados por argumento
- **getSimilarityMatrix**: Devuelve la matriz de similaridades del nivel especificado, por defecto el último.
- **computeCardinal**: Calcula el cardinal de un nivel determinado, por defecto el último.
- **computeSignificativeLevels**: Obtiene los pares significativos en cada uno de los niveles.
- **distance2**: Calcula la distancia d^2 requerida para el cálculo de la tipicalidad. (ver Sección 2.5.3).
- **distanceTilde2**: Calcula la distancia $\tilde{d}^2$ requerida para el cálculo de la contribución (ver Sección 2.5.3).
- **setTipicallity**: Encargado de calcular la tipicalidad.
- **setContribution**: Calcula la contribución.

View-Model report

Posteriormente se requiere la inclusión de una herramienta de informe sobre los resultados. Esta debe contener la misma información que el informe generado por CHIC.

En este caso se opta por una estructura lista con campos nombrados. Así, los campos que contiene son:

- **dims:** Individuos y variables de los datos de entrada.
- **initialSimilarityMatrix:** Matriz de similaridades inicial.

- **cor:** Correlación entre variables o clases primitivas.
- **freq:** Frecuencias de las variables.
- **basicStats:** media y desviación típica de las clases primitivas.
- **tree:** Árbol de similaridades en formato Newick o dendograma.
- **significativeNodes:** Nodos significativos.

Se toma la decisión de convertir la generación del informe en opcional mediante un campo booleano en la llamada al algoritmo.

Vista displayReport

Esta función es la que se encarga de mostrar los resultados del informe en un formato agradable para el usuario.

Se decide implementar la impresión en consola en primera instancia, dejando para más adelante una posible representación en Markdown, LaTeX, o HTML mediante el paquete `knitr`[5].

Se puede consultar la documentación del paquete, incluida en el Apéndice A, para más información sobre las llamadas y argumentos de cada una de estas estructuras.

4.1.3. Algoritmo

Durante el desarrollo del programa y de las funciones se ha tenido muy en cuenta la refactorización, la modularidad, el principio DRY (don't repeat yourself), y se han seguido en la medida de lo posible las recomendaciones usuales para el diseño de software.

Aunque en este caso únicamente se ha implementado el análisis de similaridades, el algoritmo principal del programa es el mismo independientemente del análisis que se desee realizar, pues es el modelo el que decide las operaciones a realizar en el cuerpo de la función:

[5]knitr: http://yihui.name/knitr/

$model \leftarrow newASImodel()$
$tree \leftarrow newlist$
for each level **do**
  $(c1, c2) \leftarrow model.getMaximumSimilarity()$
  $model.joinClasses(c1, c2)$
  $GP \leftarrow model.findGenericPair()$
  $model.setGenericImplications(GP)$
  $model.setTypicality(GP)$
  $model.setContribution(GP)$
  $tree \leftarrow format(c1, c2)$
end for
if *report* **then**
  $newReport \leftarrow buildReport(model)$
  $displayReport(newReport)$
end if

4.2. Implementación y resultados

El programa se desarolla por módulos y funciones iterativamente, implementando nuevas características a medida que se necesitan.

Así, por ejemplo, primero se programa la inicialización de la matriz de similaridades, continuando con el cálculo de nuevas similaridades, de donde surge la necesidad de tener un método para calcular la cardinalidad de los individuos.

Una vez finalizada la implementación de estos dos cálculos y se comprueba su validez se pasa a desarrollar la unión de dos clases, lo que implica la necesidad de obtener la máxima similaridad y mantener los índices en la matriz de uniones y en la lista de unidos.

Así se sigue iterando hasta añadir por último la posibilidad de seleccionar el nivel en el que se quiere aplicar cada una de las funciones anteriores, al considerarla una característica útil que proporciona más flexibilidad al usuario para construir sus propios algoritmos a medida.

La implementación también permite observar diversas incompatibilidades con el diseño que provocan cambios en el mismo:

- Se toma la decisión de no implementar por el momento un índice para el par de clases unidas en cada nivel (al ser necesarias para un único cálculo), lo que provoca la dependencia del cálculo de tipicalidades y contribuciones de la clase principal, cálculos que en el diseño principal habían recaído sobre la estructura *hierarchyTree*.

- Las estructuras de árboles jerárquicos pasan a convertirse en una lista de cadenas donde cada posición representa un nivel. Se toma esta decisión al depender las funciones *setTypicality* y *setContribution* del algoritmo principal y de la estructura *ASImodel* y perder la clase *hierarchyTree* su objetivo, pudiéndose convertir en una estructura más sencilla.

El informe (clase report en el diseño inicial) se construye como una lista de campos nombrados, en los que se almacenan cada una de las matrices y campos a representar. Las características propias de las listas en R permiten simplificar esta estructura como tal en lugar de implementarla como *ReferenceClass*.

Se consideran necesarios tres nuevos atributos en la clase *ASImodel* para simplificar el código y el desarrollo: el array de cadenas *joinedClasses*, que almacena las clases ya unidas a cada nivel; una matriz almacén de tipicalidades (*typicalityMatrix*) y una matriz donde almacenar las contribuciones (*contributionMatrix*).

La lista definitiva de campos y el diagrama de clases pasan a ser los observables en la Figura 4.4.

Se añaden a la clase tres demostraciones, una con la llamada más simple al algoritmo, otra como ejemplo de algoritmo creado por el usuario interactuando directamente con el modelo y, por último una con el ejemplo utilizado para la experimentación.

También se proporcionan tres bases de datos con el paquete:

- **5x10Individuals:** Base de datos pequeña generada de forma aleatoria. Útil para controlar la corrección de los resultados.

- **animales:** Encuesta realizada a individuos en edad escolar que relaciona adjetivos con animales.

- **animalesSmall:** Versión recortada de la tabla anterior.

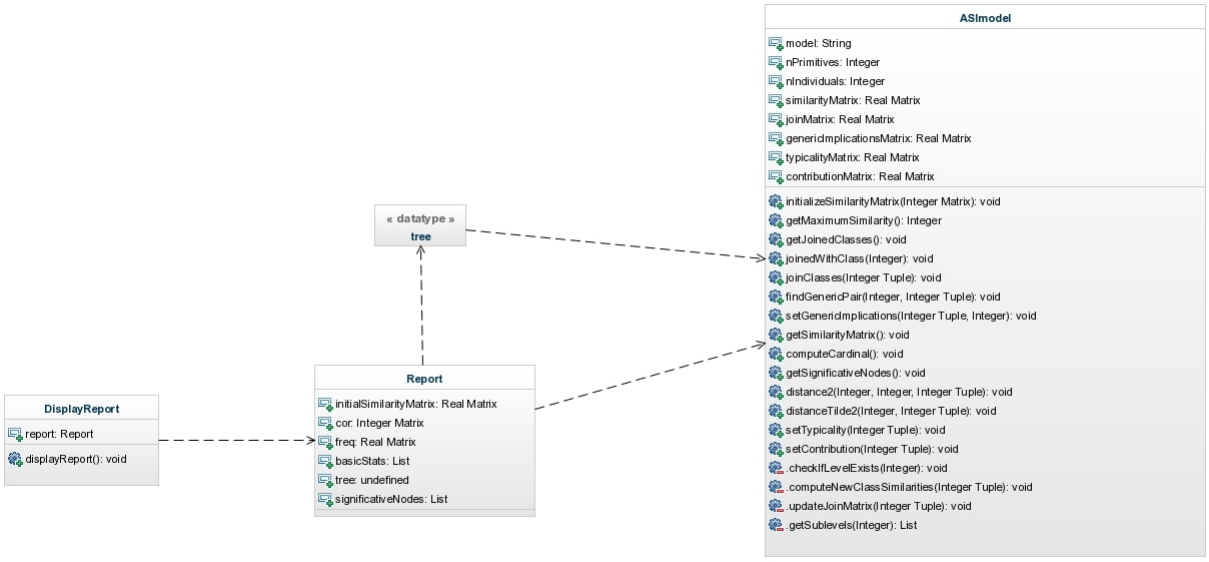

Figura 4.4: Diagrama de clases del diseño final.

4.3. Experimentación

Para analizar el comportamiento de nuestro algoritmo y compararlo con el original en RCHIC se realizarán tres experimentos: uno con la base de datos animales proporcionada con nuestro paquete, con la que se compararán los resultados; y dos con versiones recortadas de la base de datos Adult del paquete `arules`, aumentando en cada uno de los ejemplos el tamaño de los datos con los que trabajar para medir los diferentes tiempos de ejecución.

Datos utilizados

Se utiliza la base de datos Adult del paquete `arules`:

```
1 The "Adult" database was extracted from the census bureau
    database found at <URL: http://www.census.gov/ftp/pub/DES/
    www/welcome.html> in 1994 by Ronny Kohavi and Barry Becker
    , Data Mining and Visualization, Silicon Graphics. It was
    originally used to predict whether income exceeds USD 50K/
    yr based on census data. We added the attribute 'income'
    with levels 'small' and 'large' (>50K).
```

Máquina utilizada

Todos los experimentos se realizan sobre una computadora HP pavilion dm4000 con procesador intel i5 de cuatro núcleos a 2.4 GHz, 4GB de memoria RAM y distribución Ubuntu Linux 13.08. Descripción detallada en el Apéndice C.

4.3.1. Resultados

Los resultados son correctos y coinciden con los obtenidos por RCHIC. Se puede observar el árbol de similaridades construido en la demostración *smallAnimalsDemo* incluida en el paquete y representado gráficamente mediante el uso del paquete `ape` en la Figura 4.5. Se puede comparar el mismo con la salida de CHIC observable en la Figura 4.6.

4.3.2. Tiempos

Se toman las medidas con diferentes números de variables tomadas en la base de datos `Adult` del paquete `arules` y se obtienen los resultados mostrados en la Tabla 4.1 y la Figura 4.7, donde paquete valora la ejecución del algoritmo sin calcular la contribución, los pares genéricos y la tipicalidad; y paqueteTipCon realiza todos los cálculos posibles.

	RCHIC	**PaqueteTipCon**	**Paquete**
100 variables	1.7 seg	8.11 seg	3.86 seg
200 variables	6.68 seg	29.43 seg	21.49 seg
300 variables	15.16 seg	72.86 seg	58.31 seg
400 variables	26.06 seg	159.92 seg	131.95 seg
500 variables	46.82 sec	280.45 seg	238.71 seg

Cuadro 4.1: Comparación de tiempos entre RCHIC y el paquete desarrollado.

La diferencia de tiempos obtenidos varía en cada uno de los casos, pero resulta evidente que los resultados en término de tiempo no son los esperados, y que la comparación no es buena para el algoritmo programado. Era de esperar que la implementación C++ fuera más rápida que la implementación R aunque la lectura/escritura en disco compensara un poco, pero los resultados han sido mucho peores de lo esperado.

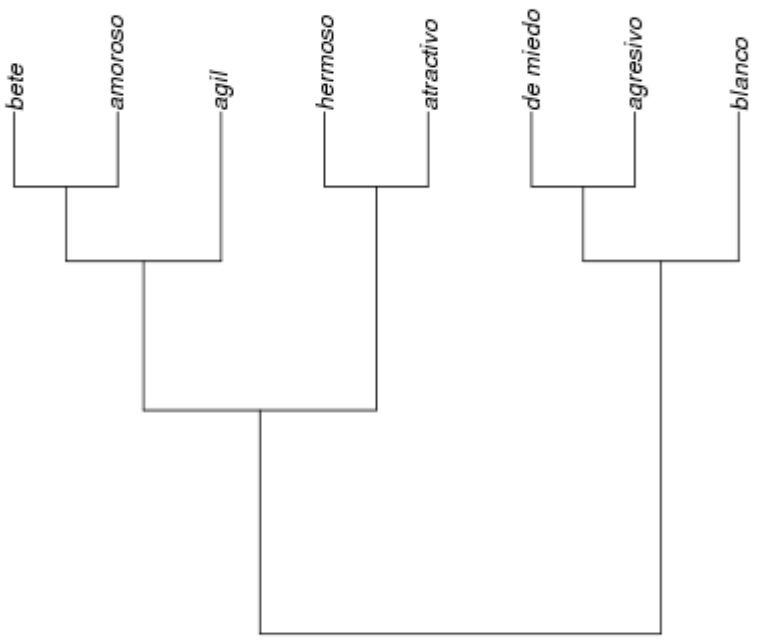

Figura 4.5: Árbol de similaridades de la base de datos *animalesSmall* generado.

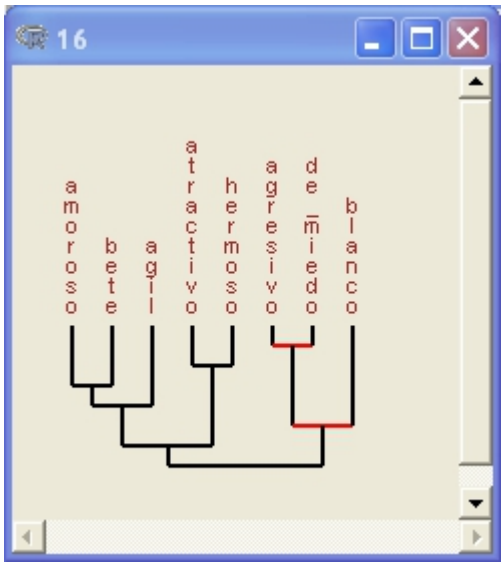

Figura 4.6: Árbol de similaridades de la base de datos *animalesSmall* producido por CHIC.

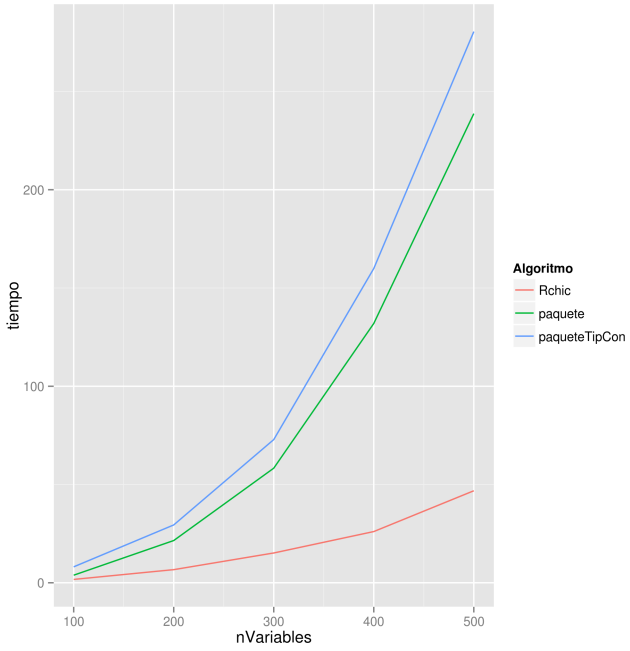

Figura 4.7: Gráfica comparativa de los tiempos de ejecución.

En el siguiente capítulo se analizan las causas de obtener estos tiempos y posibles soluciones para mejorarlos.

Capítulo 5

Conclusiones y desarrollo futuro

Los resultados no son los esperados, por lo que este capítulo se dedica a analizar las causas, las posibles mejoras y el desarrollo del paquete en el futuro y se compara lo ofrecido por nuestro paquete con las dos alternativas disponibles, CHIC y RCHIC.

Primero se comparan las ventajas e inconvenientes del paquete respecto al programa CHIC original en la Sección 5.1 y respecto al programa RCHIC en la Sección 5.2.

A continuación, en la Sección 5.3, se proponen soluciones para mejorar los tiempos de ejecución y el uso de memoria y se recapacita sobre los errores cometidos en el desarrollo del paquete.

Para finalizar, en la Sección 5.4 se reflexiona sobre el desarrollo futuro del paquete y se proponen una serie de próximas tareas a realizar.

5.1. Comparación con CHIC

CHIC es el producto que intentamos reemplazar tanto con nuestro paquete como con RCHIC. Por ello conviene analizar las ventajas que nuestro paquete supone respecto al programa original, así como exponer los puntos donde CHIC supera a nuestra librería.

Como ventajas:

- **Código abierto:** Que sea de codigo abierto permite que la comunidad pueda observarlo para así añadir sus propias mejoras e incluso identificar errores más rápidamente. Contribuye también de forma más significativa en la difusión de la teoría.
- **Multiplataforma:** R se puede ejecutar tanto en Windows como en Linux, MacOS o incluso OpenBSD.
- **Resultados manipulables estadísticamente:** El formar parte de un paquete R permite que tanto el algoritmo como los resultados se puedan manipular posteriormente e incluso desarrollar a medida, ya que se proporcionan también llamadas públicas a todas las operaciones de las que hace uso el algoritmo.
- **Libre**[1] **y gratuito.**

y en cuanto a las desventajas:

- **Tiempo de ejecución:** Los tiempos de ejecución son considerablemente máyores que en las versiones programadas en C++.
- **Poco desarollado:** Resultados únicamente del análisis implicativo por el momento, posponiendo el desarrollo del análisis de cohesiones y el análisis implicativo.

5.2. Comparación con RCHIC

Comparado con RCHIC nuestro paquete ofrece el código en R, que es el lenguaje que domina su público objetivo, lo que lo hace más comprensible y más útil de cara al futuro desarrollo del proyecto.

Por contra, en este caso los resultados están limitados de momento al análisis de similaridades y, sobre todo, la diferencia en tiempo de ejecución crece de manera notable a medida que aumenta el tamaño de los datos, aunque es probable que estos tiempos sean mejorables.

[1] Free vs Opensource: http://askubuntu.com/a/79316

Por esto, y para obtener un análisis más detallado de los resultados temporales, se recurre al profiling.

5.2.1. Profiling

El perfilado o **profiling** de una aplicación es una técnica que permite medir la complejidad temporal de un programa, el uso de determinadas instrucciones o la frecuencia de las llamadas a funciones[2] para, entre otras cosas, identificar cuellos de botella en la aplicación.

Con el fin de analizar el bajo rendimiento del algoritmo se realiza el profiling sobre el algoritmo y se observan las operaciones a las que dedica más tiempo:

```
$by.total
                                total.time total.pct self.time
"callASIAlgorithm"                  176.02    100.00      0.02
"<Anonymous>"                       175.76     99.85      1.92
"standardGeneric"                   115.98     65.89     55.14
"%in%"                               97.74     55.53      2.16
"apply"                              60.90     34.60      0.98
"FUN"                                59.26     33.67      1.34
".computeNewClassSimilarities"       51.88     29.47      9.54
"match"                              51.42     29.21     23.62
"which"                              47.46     26.96      3.48
"loadMethod"                         40.22     22.85      7.40
".getSubLevels"                      27.66     15.71      2.40
"joinedWithClass"                    19.72     11.20      0.32
".updateJoinMatrix"                  16.88      9.59      5.94
"distance2"                          15.58      8.85      0.44
"rbind"                              15.48      8.79     10.02
"assign"                             14.80      8.41     14.80
"distanceTilde2"                     14.28      8.11      0.34
```

[2]Profiling: http://en.wikipedia.org/wiki/Profiling_(computer_programming)

5.3. Aspectos mejorables del paquete actual

A la vista de los resultados y el profiling, en la Sección 5.3.1 se buscan soluciones para reducir la diferencia en cuanto a tiempos así como el coste espacial del programa.

Además, y aunque se ha desarrollado de forma correcta, durante el proceso de desarrollo también se han cometido errores, que se deben anotar para futura referencia y son expuestos en la Sección 5.3.2.

5.3.1. Posibles soluciones

Analizando la salida del profiling se puede pensar en diversas mejoras en la implementación del paquete. En este caso, se observa un especial consumo de tiempo en operaciones sobre matrices (*apply*, filtrado *%in %*, *.computeNewClassSimilarities...*), lo que sugiere la necesidad de controlar el tamaño excesivo de las mismas y reducir el número de operaciones sobre ellas en la medida de lo posible.

Cada vez que se asigna un elemento a una variable, R copia el elemento entero en lugar de realizar la operación sobre el mismo elemento. Por ejemplo, el comando $a \leftarrow a + 5$ crea el elemento $a + 5$ y lo copia después a la variable a. Esta forma de actuar se repite también al añadir elementos a una lista, array(`c()`) o matriz (`rbind, cbind,...`).

La estructura Reference Class ofrece el operando de asignación $<< -$, que permite evitar la copia en asignación, pero no si R entiende que estamos asignando un elemento diferente, como sucede cuando añadimos nuevas filas o columnas a las matrices de similaridades y uniones.

Para ello, se debe **fijar el tamaño de las matrices del modelo**. Nótese que gracias a buenas prácticas en la implementación, esto sólo requiere modificar una función por matriz.

El siguiente paso será reducir el filtrado, quizas con un nuevo método de marcar las uniones de clases que no requiera del mismo. Una posible implementación a valorar podría ser almacenar un array de listas con las clases a las que se ha unido cada entrada de la lista. Esto reduciría el coste espacial de almacenar la matriz de uniones y eliminaría el filtrado pero, por contra, implicaría un mayor coste espacial

asociado a las operaciones de copia resultantes de añadir elementos a la lista.

Obsérvese que, en el estado actual, tanto la matriz de similaridad como la de uniones son simétricas. Con un tamaño tan grande, reducirlas a la mitad mejoraría en mucho el consumo de memoria y reduciría las operaciones de filtrado de una forma importante.

5.3.2. Errores detectados en el proceso

Se han observado los siguientes errores, que de no haber cometido, hubieran permitido un desarrollo más rápido y seguro de la librería:

- **Incorrecta definición de los objetivos**, procesos y secciones a desarrollar desde el inicio: cambios en los requisitos iniciales provocan cambios costosos en el código.
- **Big Design up front**: Pese a ser útil en ocasiones (y mientras sea correcto), en este caso se definen e implementan estructuras de datos que han sido eliminadas en el proceso de refactorización y simplificación del código al no ser necesarias (no hasta este punto).
- **No desarrollar iterativamente haciendo uso de grandes bancos de datos**: El desarrollo se lleva a cabo con datos de tamaño controlable, lo que provoca que al aplicarlo sobre datos de tamaño considerable al final se encuentren múltiples errores que no aparecían anteriormente. Si tras cada etapa, y en especial tras el análisis clasificatorio se hubiera probado el algoritmo contra bancos de datos de gran tamaño se hubieran detectado los errores más velozmente.
- **Subestimar el coste espacial**: Al trabajar con datos de gran tamaño y debido a las características de operaciones como *crossproduct* que aumentan considerablemente la necesidades espaciales del progama se debería haber tenido más cuidado con la elección de las estructuras de datos y operaciones a realizar.

  Como ejemplo, el banco de datos del experimento para una elección de 1000 variables: si no se recorta, se generan 3 matrices de 17.8 GB cuando resulta imposible guardar una de 8 GB en RAM.
- **Uso de *demo()* como test para validar los resultados**, en lugar de desarrollar una suite de tests en la carpeta test del paquete.

5.4. Aspectos a tener en cuenta para el desarrollo futuro del paquete

Además de la evolución lógica mencionada anteriormente (implementar análisis implicativo, de cohesiones, extender tipos de datos de entrada soportados), durante el desarrollo del proyecto se detectan las siguientes situaciones y mejoras posibles:

- **Desacople**. El análisis de similaridades debería estar totalmente desacoplado del modelo ASI, para una implementación más sencilla del análisis de cohesiones y del análisis implicativo. Idealmente, ASImodel debería ser una interfaz/clase abstracta que dichos análisis implementen a posteriori.
- **Mejora en la visualización del informe**. Trabajar con el paquete `knitr` y generar otros formatos (LaTeX, HTML, Markdown) más amigables para el usuario.
- **Visualización de los datos**. Se deberá unir la visualización a la del paquete RCHIC. Sin embargo, R ya no está atado a su ventana gráfica como único método para representar sus resultados.

  La opción de RCHIC pasa por el uso de librerias gráficas externas como *tcltk* (no disponible por defecto en Windows y MacOS) y la instalación de paquetes como *Rgraphviz* dependientes de los repositorios de *Bioconductor*. Dado que hoy en día cualquier computador dispone de un navegador web y la existencia de suficientes librerías gráficas para la representación de datos en ellos, HTML y javascript podría ser considerada como una mejor opción para la representación de resultados.
- **Formato de los datos**. Se justifica anteriormente la elección de la matriz binaria en formato numérico para los datos de entrada por cuestiones de eficiencia espacial al depender las estructuras de datos definidas de la creación de múltiples matrices de tamaño considerable. Sin embargo, una estructura `data.frame` debería tomarse en consideración al tener más sentido en términos de utilización, programación y comprensión del código.
- **Valorar la escritura y lectura en disco** y decidir el formato, ofreciendo una interfaz para abstraer al usuario del manejo del mismo.
- **Implementar privacidad** en las funciones y campos de la clase ASImodel en los que se considere necesaria.
- **Desarrollo de una suite de tests** que permita un desarollo seguro.

- **Mejorar la documentación** añadiendo ejemplos, incluyendo nuevas demostraciones explorando las posibilidades que ofrece el paquete y mejorando la descripción de las funciones.

Bibliografía

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Kent Beck. *Test-driven development : by example.* Addison-Wesley, Boston, 2003.

[4] A. Bodin. Modèles sous-jacents à l'analyse implicative et outils complémentaires. *Cahiers du séminaire de didactique de l'IRMAR de Rennes*, 1997.

[5] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[6] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis.* Wiley Publishing, 4th edition, 2009.

[7] R. Gras. *Contribution à l'étude expérimentale et à l'analyse de certaines acquisitions cognitives et de certains objectifs didactiques en mathématiques.* PhD thesis, Thèse d'Etat, Université de Rennes 1, 1979.

[8] R. Gras and A. Larher. L'implication statistique, une nouvelle méthode d'analyse de données. *Mathématiques, Informatique et Sciences Humaines*, (120):5–31, 1993.

[9] Régis Gras and Pascale Kuntz. Discovering r-rules with a directed hierarchy. *Soft Computing, a Fusion of Foundations, Methodologies and Applications*, 10(5):453–460, 2006.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[11] R. Gras I.C. Lerman and H.R. Rostam. Élaboration et évaluation d'un indice d'implication pour des données binaires i. *Mathématiques, Informatique et Sciences Humaines*, (74):5–35, 1981.

[12] R. Gras I.C. Lerman and H.R. Rostam. Élaboration et évaluation d'un indice d'implication pour des données binaires ii. *Mathématiques, Informatique et Sciences Humaines*, (75):5–47, 1981.

[13] Ludovic Lebart, Marie Piron, and Alain Morineau. *Statistique exploratoire multidimensionnelle : visualisations et inférences en fouille de données*. Sciences sup. Dunod, Paris, 2006. Autres tirages : 2008, 2012.

[14] I.C. Lerman. Sur l'analyse des données préalable à une classification automatique (proposition d'une nouvelle mesure de similarité). *Mathématiques, Informatique et Sciences Humaines*, (32):5–15, 1970.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[16] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[17] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[18] P. Orus, L. Zamora, and P. Gregori, editors. *Teoria y Aplicaciones del Analisis Estadistico Implicativo*. 2009.

[19] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972.

[20] Hadley Wickham. *Advanced R*. R Series. Chapman and Hall, 2014.

Apéndice A

Documentación

A continuación se incluye la documentación generada al crear el paquete ReRCHIC.

Package 'rerchic'

October 6, 2014

Type Package

Title R Package for Implicative Statistical Analysis

Version 0.2

Date 2014

Author Xavier Valls

Maintainer Xavier Valls<xvalls@uji.es>

Description This package is intented to implement all the features of Statistical Implicative Analysis. This theory aims at building association rules of the form : if an object has the property A it also tends to have the property B. The SIA builds confidence of rules by computing the number of counter examples of such an rule and then by measuring the surprise (or the probability) to have so few counter examples compared to the number of counter examples we would have with two random variables have the same sizes (than A and B). With Rchic you can build a similirity tree (using similarity analysis), an implicative graph and a hierarchical tree.

Depends R (>= 2.14.0)

Suggests testthat,ape

License GPL

URL https://github.com/xvallspl/ReRchic/

R topics documented:

animales *Animals dataset.*

Description

A dataset polling children perception of animals characteristics.

Author(s)

Xavier Valls<xvalls@uji.es>

References

""

animalesSmall *Small animals dataset.*

Description

A (cropped) dataset polling children perception of animals characteristics.

Author(s)

Xavier Valls<xvalls@uji.es>

References

""

ASImodel *A Reference Class to represent the Statistical Implicative Analysis model.*

Description

A Reference Class to represent the Statistical Implicative Analysis model.

Fields

`data` External data.

`model` Probability model to use. Binom or pois.

`joinMatrix` Matrix of joint classes

`similarityMatrix` Matrix of bivariant similarities.

`genericImplications` Matrix Matrix of generic Implications

`nPrimitiveClasses` Number of primitive classes

Methods

`findGenericPair(level, Tuple)` Finds the generic Pair of a derived class

`getMaximumSimilarity(at.level = levels)` Gets the maximum similartity at the specified level. By default the last one

`getSignificativeNodes()` Gets the significative nodes of the hierarchical tree

`getSimilarityMatrix(at.level = levels)` Returns the similarity matrix at the specified level of the hierarchical tree

`joinClasses(Tuple)` Joins two classes into a new node of the hierarchical tree

`setContribution(genericPair, at.level = levels)` Computes the contribution of each individual for a certain class

`setGenericImplications(genericPair, level, p = 0.5)` Sets the generic implication of each individual for a given derived class (level)

`setTypicality(genericPair, at.level = levels)` Computes the typicality of each individual for a certain class

---

`callASIAlgorithm` *Functions that performs the algorithm for the Statistical implicative analysis.*

Description

Functions that performs the algorithm for the Statistical implicative analysis.

Usage

```
callASIAlgorithm(data, model, report = FALSE)
```

Arguments

`data`	External data.
`model`	Probability model to use. Binom or pois.
`report`	If true, displays a report

Value

report Report: algorithm results, hierarchical tree, basic statistic analysis.

newickTree Hierarchical tree in Newick's format.

Author(s)

Xavier Valls <xvalls@uji.es>

`data` *5x10 dataset.*

Description

A randomly generated dataset with 10 individuals and 5 binary variables.

Author(s)

Xavier Valls<xvalls@uji.es>

Apéndice B

Código

B.1. ASImodel

```
#' A Reference Class to represent the Statistical Implicative
    Analysis model.
#' @name ASImodel
# @import methods
#' @export ASImodel
#' @exportClass ASImodel
#' @field data   External data.
#' @field model Probability model to use. Binom or pois.
#' @field joinMatrix Matrix of joint classes
#' @field similarityMatrix Matrix of bivariant similarities.
#' @field genericImplicationsMatrix Matrix of generic
   Implications
#' @field nPrimitiveClasses Number of primitive classes

ASImodel <- setRefClass("ASImodel",
  fields = c("data", "model", "levels", "joinMatrix", "
     similarityMatrix","genericImplicationsMatrix", "
     joinedClasses","nPrimitiveClasses", "contributionMatrix"
     ,"typicalityMatrix","nIndividuals"),
  methods = list(

      initialize = function( externalData, model = "pois" )
      {
        if(!is.matrix(externalData))
```

```
20            stop("Provided␣data␣should␣be␣a␣matrix")
21          if(!model %in% c("pois", "binom"))
22            stop("Specified␣probability␣distribution␣model␣not␣
                implemented")
23
24          nIndividuals <<- nrow(externalData)
25          nPrimitiveClasses <<- ncol(externalData)
26          data<<- externalData
27          .self$model <<- model
28          initializeSimilarityMatrix(externalData)
29          joinMatrix <<- matrix(NaN, nPrimitiveClasses,
                nPrimitiveClasses)
30          diag(joinMatrix)<<-(Inf)
31          genericImplicationsMatrix <<- matrix(1, nIndividuals,
                nPrimitiveClasses-1)
32          contributionMatrix <<- matrix(NaN, nIndividuals,
                nPrimitiveClasses-1)
33          typicalityMatrix <<- matrix(NaN, nIndividuals,
                nPrimitiveClasses-1)
34          joinedClasses<<-list()
35          levels <<- 0
36        },
37
38        initializeSimilarityMatrix = function(externalData){
39          nBinaryPresences <- apply(externalData,2,'sum')
40          nBinaryCopresences <- crossprod(externalData,
                externalData)
41          probabilityOfCopresence <- tcrossprod(
                nBinaryPresences, nBinaryPresences)/nIndividuals^2
42
43          if(model == 'pois'){
44            similarityMatrix <<- ppois( q = nBinaryCopresences,
                  lambda = nIndividuals*probabilityOfCopresence )
45          }else if(model == 'binom'){
46            similarityMatrix <<- pbinom( q = nBinaryCopresences
                , size = nIndividuals, prob =
                probabilityOfCopresence )
47          }
48          diag(similarityMatrix)<<-0
49        },
50
51        getMaximumSimilarity = function(at.level = levels){
```

```
52          "Gets␣the␣maximum␣similartity␣at␣the␣specified␣level.
               ␣By␣default␣the␣last␣one"
53          simMat <- getSimilarityMatrix(at.level)
54          joined <- getJoinedClasses(at.level)
55          M <- which( similarityMatrix == max(simMat), arr.ind
               = TRUE )
56          MnotJoined <- apply( M, 1, function(x){all(!(x %in%
               joined))})
57          nodes <- M[MnotJoined,][1, ]
58          names(nodes)<-colnames(similarityMatrix)[nodes]
59           return(list(nodes = nodes, value = max(simMat)))
60        },
61
62        getJoinedClasses = function(at.level=levels,
             primitivesOnly = FALSE){
63          .checkIfLevelExists(at.level)
64          if(at.level==0)
65            ret=c()
66          else{
67            ret<-joinedClasses[[at.level]]
68            if(primitivesOnly) ret<-ret[ret<=nPrimitiveClasses]
69          }
70          return(ret)
71        },
72
73        joinedWithClass = function(class, at.level=levels,
             primitivesOnly = FALSE){
74          .checkIfLevelExists(at.level)
75          if(!primitivesOnly){
76            ret <- class
77            n   <- nPrimitiveClasses+at.level
78          }
79          else{
80            ret <- c()
81            n   <- nPrimitiveClasses
82          }
83          joined <- which(!(joinMatrix[class, 1:n]  %in% NaN) &
84                       joinMatrix[class, 1:n] <=at.level)
85           return(c(ret, joined))
86        },
87
88       joinClasses = function(Tuple){
```

```
89          "Joins␣two␣classes␣into␣a␣new␣node␣of␣the␣
               hierarchical␣tree"
90          if(ncol(similarityMatrix) == (2*nPrimitiveClasses-1))
               {
91                stop("You're␣already␣at␣the␣last␣level!")
92          }
93           .computeNewClassSimilarities(Tuple)
94           .updateJoinMatrix(Tuple)
95           levels <<- levels+1
96           if(levels>1)
97           {
98             joinedClasses[[levels]]<<-c(joinedClasses[[levels
                  -1]],Tuple)
99           }
100          else joinedClasses[[levels]]<<-Tuple
101       },
102
103       .computeNewClassSimilarities = function(Tuple){
104         newClassRow <- newClassCol <- array(0,ncol(
               similarityMatrix))
105         joinedWithFirst  <- joinedWithClass(Tuple[1])
106         joinedWithSecond <- joinedWithClass(Tuple[2])
107         joinedWithTuple <-c(joinedWithFirst,joinedWithSecond)
108         primitivesJoinedWithTuple <- joinedWithTuple[
               joinedWithTuple<(nPrimitiveClasses+1)]
109         for( i in 1:ncol(similarityMatrix)){
110           if(!(i %in% joinedWithTuple))
111           {
112             classOutsideTuple <- which(!(joinMatrix[i, 1:ncol
                  (similarityMatrix)] %in% NaN))
113             newClassCol[i] <- max(similarityMatrix[
                  classOutsideTuple, primitivesJoinedWithTuple])
                  ^(length(primitivesJoinedWithTuple)*length(
                  classOutsideTuple))
114             newClassRow[i] <- max(similarityMatrix[
                  primitivesJoinedWithTuple, classOutsideTuple])
                  ^(length(primitivesJoinedWithTuple)*length(
                  classOutsideTuple))
115           }
116         }
117         similarityMatrix <<- rbind(cbind(similarityMatrix,
               newClassCol), c(newClassRow,0))
```

```
118         colnames(similarityMatrix)[ncol(similarityMatrix)]<<-
                paste("(", colnames(similarityMatrix)[Tuple[1]],"
               ,", colnames(similarityMatrix)[Tuple[2]],")")
119       },
120
121       .updateJoinMatrix = function(Tuple){
122         newClass <- array(NaN, ncol(joinMatrix))
123         joinedWithFirst  <- joinedWithClass(Tuple[1])
124         joinedWithSecond <- joinedWithClass(Tuple[2])
125         joinMatrix[joinedWithFirst,joinedWithSecond] <<-
               levels+1
126         joinMatrix[joinedWithSecond,joinedWithFirst] <<-
               levels+1  #Simmetry
127         newClass[c(joinedWithFirst, joinedWithSecond)] <-
               levels+1
128         joinMatrix <<- rbind(cbind(joinMatrix, newClass), c(
               newClass, Inf))
129         colnames(joinMatrix)[ncol(joinMatrix)] <<-levels+1
130       },
131
132       findGenericPair = function(level, Tuple){
133         "Finds␣the␣generic␣Pair␣of␣a␣derived␣class"
134         joinedWithFirst  <- joinedWithClass(Tuple[1], level,
               primitivesOnly=TRUE)
135         joinedWithSecond <- joinedWithClass(Tuple[2], level,
               primitivesOnly=TRUE)
136
137         maxPhi <- max(similarityMatrix[joinedWithFirst,
               joinedWithSecond])
138         simMat <- similarityMatrix[1:nPrimitiveClasses, 1:
               nPrimitiveClasses]
139         maxPhiInd <- which( simMat == maxPhi, arr.ind=T)[1,]
140
141         return(list(phi = maxPhi, pos = maxPhiInd))
142       },
143
144       setGenericImplications = function(genericPair, level, p
              = 0.5){
145         "Sets␣the␣generic␣implication␣of␣each␣individual␣for␣
               a␣given␣derived␣class␣(level)"
146         for(i in 1:nIndividuals)
147         {
148           if(data[i, genericPair[2]]!=1)
```

```
149             if(data[i, genericPair[1]])
                   genericImplicationsMatrix[i, level]<<-0
150             else genericImplicationsMatrix[i, level]<<-p
151           }
152         },
153
154         getSignificativeNodes = function(){
155           "Gets␣the␣significative␣nodes␣of␣the␣hierarchical␣
                 tree"
156           centeredIndices <- rep(NA, ncol(similarityMatrix)-
                 nPrimitiveClasses)
157           for( i in 1:(ncol(similarityMatrix)-nPrimitiveClasses
                 ))
158           {  c <- computeCardinal(i)
159             centeredIndices[i]  <- (c$cardinal- 1/2 * c$nSep *
                   c$nJoin)/ sqrt(c$nJoin*c$nSep*(c$nJoin+c$nSep+1)
                   /12)
160           }
161           v <- diff(diff(centeredIndices))
162           localMaximumPositions <- which(diff(sign(v))!=0)
163           return(localMaximumPositions)
164         },
165
166         computeCardinal = function(at.level=levels){
167           primJoined <- getJoinedClasses(at.level,
                 primitivesOnly = TRUE)
168           primSepparated <- (1:nPrimitiveClasses)[-primJoined]
169           SimilaritiesJoined     <- similarityMatrix[primJoined
                 , primJoined]
170           SimilaritiesSepparated <- similarityMatrix[
                 primSepparated, primSepparated]
171           # Mann - Whitney
172           cardinal <- sum(which(sort(unique(c(
                 SimilaritiesJoined, SimilaritiesSepparated))) %in%
                  SimilaritiesSepparated)) - (length(
                 SimilaritiesJoined)*(length(SimilaritiesJoined)+1)
                 )/(2*2)
173           return(list( cardinal= cardinal, nJoin = length(
                 SimilaritiesJoined)/2, nSep = length(
                 SimilaritiesSepparated)/2))
174         },
175
176         getSimilarityMatrix = function(at.level=levels){
```

```
177         "Returns␣the␣similarity␣matrix␣at␣the␣specified␣level
               ␣of␣the␣hierarchical␣tree"
178         .checkIfLevelExists(at.level)
179         omit <-getJoinedClasses(at.level)
180         if(!length(omit))
181           simMat <- similarityMatrix[1:nPrimitiveClasses, 1:
                 nPrimitiveClasses]
182         else
183           simMat <- similarityMatrix[1:(nPrimitiveClasses+at.
                 level), 1:(nPrimitiveClasses+at.level)][-omit, -
                 omit]
184         return(simMat)
185       },
186
187       .checkIfLevelExists = function(level){
188         if((level+nPrimitiveClasses)> ncol(similarityMatrix))
189           stop("Wrong␣level␣or␣level␣still␣not␣computed")
190       },
191
192       distance2 = function(individual, level, genericPair){
193         classSubClasses <- .getSubLevels(level)
194         return( 1/length(classSubClasses) * sum((genericPair$
                 phi-genericImplicationsMatrix[individual,
                 classSubClasses])^2 / (1-genericPair$phi)))
195       },
196
197       .getSubLevels = function(class){
198         joinedWithClass <- joinedWithClass(nPrimitiveClasses+
                 class, class)
199         classSubClasses <- joinedWithClass[which(joinMatrix[
                 nPrimitiveClasses+class,joinedWithClass] == class)
                 ]
200         return(c(class, classSubClasses[classSubClasses >
                 nPrimitiveClasses]-nPrimitiveClasses))
201       },
202
203       distanceTilde2 = function(individual, level){
204         classSubclasses <- .getSubLevels(level)
205         return( 1/length(classSubclasses) * sum((1-
                 genericImplicationsMatrix[individual,
                 classSubclasses])^2))
206       },
207
```

```
208       setTypicality = function(genericPair, at.level=levels )
              {
209         "Computes␣the␣typicality␣of␣each␣individual␣for␣a␣
                certain␣class"
210         dist<-array(NaN, nIndividuals)
211         for(i in 1:nIndividuals)
212         {
213           dist[i]<-sqrt(distance2(i, at.level, genericPair))
214         }
215         typicalityMatrix[,at.level]<<- 1 - dist/ max(dist)
216       },
217
218       setContribution = function(genericPair, at.level=levels
              ){
219         "Computes␣the␣contribution␣of␣each␣individual␣for␣a␣
                certain␣class"
220         for(i in 1:nIndividuals)
221         {
222           contributionMatrix[i,at.level]<<-(1 - sqrt(
                  distanceTilde2(i, at.level)))
223         }
224       }
225   )
226 )
```

B.2. similarityClassification

```
1  #' Functions that performs the algorithm for the Statistical
       implicative analysis.
2  #'
3  #' @name callASIAlgorithm
4  #'
5  #' @param data    External data.
6  #' @param model   Probability model to use. Binom or pois.
7  #' @param report  If true, displays a report
8  #'
9  #' @return report      Report: algorithm results,
       hierarchical tree, basic statistic analysis.
10 #' @return newickTree  Hierarchical tree in Newick's format.
11 #'
12 #' @author Xavier Valls \email{xaviervallspla@@gmail.com}
13 #' @export
```

```
14
15 callASIAlgorithm <-function( data, model, report=FALSE){
16   aData <- ASImodel$new(data)
17   tree<-array("",ncol(data)-1)
18   for( i in 1:( ncol(data)-1))
19   {
20     S <- aData$getMaximumSimilarity()
21     aData$joinClasses(S$nodes)
22     genericPair<- aData$findGenericPair(i, S$nodes)
23     aData$setGenericImplications(genericPair$pos, i)
24     aData$setTypicality(genericPair, at.level=i)
25     aData$setContribution(genericPair, at.level=i)
26     S$nodes[S$nodes>ncol(data)] = tree[S$nodes[S$nodes>ncol(
         data)]-ncol(data)]
27     names<-names(S$nodes)
28     tree[i]<- paste(colnames(aData$similarityMatrix)[aData$
         nPrimitiveClasses+i])
29   }
30
31   newickTree = paste(tree, ';', sep="")
32   significativeNodes<-aData$getSignificativeNodes()
33
34   if(report){
35     report = list()
36     report$dims <- c(ncol(data), nrow(data))
37     report$initialSimilarityMatrix<-aData$
         getSimilarityMatrixAtLevel(0)
38     report$cor <- cor(data)
39     report$freq <- NULL
40     report$basicStats <- cbind(apply(data,2,sum), apply(data,
          2, mean), apply(data,2, sd))
41     colnames(report$basicStats) <- c("freq","mean","sd")
42     report$tree <- newickTree
43     report$significativeNodes <- significativeNodes
44     return(report)
45   }
46   return(list(tree = newickTree, significativeNodes=
       significativeNodes))
47 }
48
49 displayReport <- function(report)
50 {
```

```
51   print(paste("ncol:", report$dims[1], ", nrow:", report$dims
       [2]))
52   print(report$basicStats)
53   print("Bivariant frequency:")
54   print(report$freq)
55   print("Correlation coefficient:")
56   print(report$cor)
57   print("Similarity indices:")
58   print(report$initialSimilarityMatrix)
59
60   for(i in 1:ncol(length(report$tree)))
61   {
62     print(paste("Classification at level ", i))
63     print(report$tree[i])
64   }
65 }
```

Apéndice C

Información de la computadora

Todos los experimentos se realizan sobre una computadora HP pavilion dm4000 con procesador intel i5 de cuatro núcleos a 2.4 GHz y 4GB de memoria RAM y distribución Ubuntu Linux 13.08.

```
vendor_id : GenuineIntel
cpu family : 6
model : 37
model name : Intel(R) Core(TM) i5 CPU       M 450  @ 2.40GHz
stepping : 5
microcode : 0x2
cpu MHz : 1199.000
cache size : 3072 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 2
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
bogomips : 4800.17
clflush size : 64
```

```
cache_alignment : 64
address sizes : 36 bits physical, 48 bits virtual
```