Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

Redes bayesianas en R: análisis de los paquetes software disponibles

Autor: Luis Eduardo Angulo Montes

Director: Bojan Mihaljevic

Director: María Concepción Bielza Lozoya

Madrid, julio 2020

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Ciencia de datos
Título: Redes bayesianas en R: análisis de los paquetes software disponibles
Julio 2020

Autor: Luis Eduardo Angulo Montes

Director:
Bojan Mihaljevic
María Concepción Bielza Lozoya
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En la actualidad existen gran cantidad de paquetes *software* o lenguajes de programación que facilitan la modelización de redes bayesianas. En esencia son herramientas que permiten el aprendizaje de la estructura y parámetros de la red, así como el respectivo proceso de inferencia probabilística que se puede ejecutar sobre las mismas. El entorno de R dispone de una gran variedad de paquetes destinados a estas actividades pero no despliega una guía que oriente a los usuarios sobre cuál de ellos utilizar dadas las particularidades del conjunto de datos objeto de estudio o del contexto general. Este último es el propósito del presente trabajo, el cual pretende generar una guía básica a partir del análisis exhaustivo de los paquetes de R más populares. En total se analizan nueve: **catnet, sparsebn, pcalg, deal, bnclassify, bnstruct, bnlearn, gRain** y **BayesNetBP**.

Se estudian las funcionalidades, algoritmos y métodos implementados en cada uno de éstos, haciendo especial énfasis en sus limitaciones y factores diferenciadores; también se identifican los escenarios o casos especiales en los que éstos brindan mejores resultados en comparación con los otros paquetes. Lo anterior se realizó a partir del estudio de los manuales de uso y de funciones dispuesto por cada uno de los paquetes objeto de análisis, así como de la bibliografía adicional relacionada con éstos. De igual forma, se realizó un experimento donde se probaron en el entorno de R las funcionalidades disponibles en los paquetes sobre diferentes conjuntos de datos estándar o que comúnmente son utilizados en la literatura sobre redes bayesianas para efectuar comparaciones. Específicamente se implementó el código o funcionalidades propias de cada paquete para la modelización de redes y a partir de ésto se pudo comparar tiempos de ejecución, calidad de los resultados, limitaciones y valor añadido, entre otros.

Abstract

Nowadays, there are a large number of software packages or programming languages that facilitate the Bayesian network modeling process. In essence, these tools allow the structure and parameter learning of the network, as well as to perform the respective probabilistic inference methods that can be executed on them. The R environment has a wide variety of packages for these activities, but it does not provide a guide to lead users to which one to apply, given the particularities of the dataset they have available or the general context. The latter is the purpose of this work, which aims to generate a basic guide from the exhaustive analysis of the most popular packages. In total, nine of them were analyzed: **catnet, sparsebn, pcalg, deal, bnclassify, bnstruct, bnlearn, gRain** and **BayesNetBP**.

The functionalities, algorithms, and methods implemented in each of them are studied, with special emphasis on their limitations and differentiating factors; the scenarios or special cases in which each provides better results compared to the other packages are also identified. This was done based on the study of the user's and functions manuals available for each of the packages under analysis, as well as the additional bibliography related to the topic. Likewise, an experiment was carried out in the R environment, the idea was to test the functionalities available in each package using different datasets employed in the Bayesian network literature to make comparisons. Specifically, the code or functionalities of each package were implemented for network modeling, and from this, it was possible to compare execution times, quality of the results, limitations, and added value, among others.

Tabla de contenidos

1 Introducción

En el ámbito del aprendizaje automático es común encontrar algoritmos o metodologías que facilitan el procedimiento de predicción sobre nuevas instancias u observaciones sin entrar en detalles en torno a la interpretabilidad del modelo construido, razón por la cual no es posible acceder a información que ayude al entendimiento del método utilizado para abordar el problema o de su respectiva solución. No obstante, la utilización de redes bayesianas solventa el anterior impase al facilitar la representación gráfica del conocimiento sobre un área dada a través de la codificación de las variables involucradas como nodos en una estructura cuyas aristas muestran las relaciones de dependencia probabilística entre éstos.

Una de las finalidades subyacentes en torno a la utilización de las redes bayesianas consiste en describir e identificar las relaciones de dependencia o relevancia entre variables para descubrir conocimiento o para clasificar nuevas observaciones, entre otras. La anterior tarea esgrime gran importancia debido a que proporciona nuevo conocimiento bajo esquemas de incertidumbre, facilitando a su vez la toma de decisiones y el razonamiento por medio del uso de la teoría de probabilidad.

Al vislumbrar los beneficios potenciales de la implementación de esta técnica en el campo del aprendizaje automático, a lo largo de los últimos años se han desarrollado una gran cantidad de paquetes *software* o herramientas en diferentes lenguajes de programación que implementan funcionalidades para llevar a cabo el aprendizaje de la estructura de la red, la identificación de las relaciones de dependencia derivadas de la distribución de probabilidad conjunta con que se modela la incertidumbre propia de las variables o los procesos de inferencia probabilística que se pueden ejecutar sobre las mismas.

En este sentido, en la literatura sobre redes bayesianas se pueden encontrar trabajos como el de Scanagatta, Salmerón y Stella (2019), quienes realizan una revisión de los principales algoritmos implementados para el aprendizaje de redes bayesianas y además despliegan una exploración enfocada en listar las herramientas de *software* que se utilizan habitualmente para la modelización de redes bayesianas; o como el de Mahjoub y Kalti (2011), quienes se enfocan en realizar un análisis comparativo de los diferentes paquetes *software* disponibles en su momento para la misma actividad. Por su parte, Scutari y Denis (2014) en su libro *Bayesian Networks: with examples in R* presentan los paquetes más populares implementados específicamente en el entorno de R para la modelización de redes bayesianas, haciendo especial énfasis en el paquete **bnlearn** desarrollado por Scutari. De igual forma, Nagarajan, Scutari y Lèbre (2013), muestran cómo se pueden ejecutar en R las diferentes tareas propias del uso de redes bayesianas a partir de funciones, código o ejemplos provenientes de diversos paquetes.

Hoy en día existe una gran cantidad y variedad de herramientas de *software* desarrolladas para ejecutar la modelización de redes bayesianas, se pueden encontrar alternativas comerciales creadas utilizando lenguajes de programación como Java, C++ u otros, algunas de ellas son: Hugin, BayesiaLab, Bayes Server y Netica, entre otros. Del mismo modo, es posible hallar herramientas de código abierto o licencia de uso libre como: Bayesian Network tools in Java (BNJ), Bayesian Network Inference with Java objects (BANJO) y

GeNIe and SMILE (BayesFusion), entre otros. Así mismo, es posible trabajar con redes bayesianas en librerías de Python, R o directamente en WEKA u Octave/MATLAB.

Considerando lo anterior, el presente documento acota su análisis al entorno de R, el cual se configura como uno de los más populares hoy en día para la implementación de técnicas estadísticas y aprendizaje automático, entre otros. Así, el alcance de éste es el análisis de los principales paquetes disponibles en dicho entorno relacionados con el aprendizaje e inferencia sobre redes bayesianas.

Si bien ya existen aproximaciones a esta labor, como las mencionadas anteriormente, en ambos casos el estudio de los paquetes no llega hasta un nivel de detalle profundo en cuanto a las funcionalidades implementadas, lo cual a su vez no permite identificar cuáles son las metodologías, algoritmos, puntajes, pruebas de independencia, tipos de datos y utilidades para la manipulación de las redes, entre otras, disponibles.

Por consiguiente, este trabajo busca alcanzar dos objetivos prioritarios. El primero pretende determinar las principales funcionalidades, métodos, algoritmos o procedimientos implementados en los paquetes más utilizados en R para la modelización de redes bayesianas; el segundo procura identificar la aplicabilidad bajo diferentes contextos, los factores diferenciadores y las limitaciones de los paquetes objeto de análisis. En ambos casos, se procederá a realizar un estudio exhaustivo tanto de los manuales de uso como de los manuales de funciones de los paquetes, así como de la bibliografía relacionada con los mismos y con la temática en general. También se ejecutará en R el respectivo código que permita experimentar con las funciones habilitadas por cada uno de éstos.

Específicamente, se probarán y ejecutarán en el entorno de R las funcionalidades dispuestas por cada paquete para la modelización de redes bayesianas, haciendo uso de diferentes conjuntos de datos estándar o de prueba, los cuales comúnmente son utilizados en la literatura sobre redes bayesianas para realizar comparaciones; se implementará el código que permita explorar y valorar los métodos o algoritmos dispuestos por los paquetes para el aprendizaje de la red o de los parámetros, para el proceso de inferencia y para la manipulación o representación gráfica de la misma, entre otros. Adicionalmente, lo anterior facilitará la comparación del tiempo de ejecución, calidad de los resultados, limitaciones y valor añadido de los diferentes paquetes.

Este trabajo aspira a servir como una guía básica que oriente a usuarios interesados o investigadores sobre cuáles son los paquetes de los que disponen en R para acometer la modelización de redes bayesianas, dándoles luces sobre cuál utilizar teniendo en cuenta la tarea que se quiera ejecutar, el algoritmo que se desee implementar o el tipo de dato que se tenga a disposición, entre otros. El resultado del mismo consiste en unas tablas resúmenes donde se exhibe toda la información recopilada. Éstas se encontrarán disponibles en la cuenta GitHub del autor y podrán ser editadas, ajustadas, complementadas o actualizadas por cualquier persona interesada.

El resto del documento se divide en cuatro capítulos. En el capítulo 2 se esbozan los conceptos teóricos relacionados con la temática de redes bayesianas. En el capítulo 3, se describen y analizan cada uno de los paquetes identificados,

también se describen algunas herramientas complementarias o sustitutas de éstos. En el capítulo 4 se muestra la mayoría de las tablas creadas a modo de guía, por lo que éste se constituye como un resumen de las principales aplicaciones y funciones incorporadas en los paquetes. En el capítulo 5 se presentan las consideraciones finales. Antes de pasar al capítulo 2, esta introducción continúa con una breve reseña sobre el entorno de R.

1.1 ¿Qué es el entorno de R?

R es considerado un lenguaje y entorno computacional diseñado para el análisis estadístico, el cual está disponible para el público a través de un *software* libre bajo una licencia pública general de GNU sobre su código fuente. En esencia, R permite dar instrucciones a nuestro equipo de cómputo para la aplicación de una gran variedad de técnicas estadísticas y/o para la generación de gráficos (The R Foundation, Sin fecha).

La administración y mantenimiento tanto de los derechos de autor del *software* como de la documentación están a cargo de los miembros del "*R Development Core Team*" por medio de la Fundación R (entidad sin ánimo lucro). Entre los objetivos de esta fundación se encuentran el apoyo continuo al desarrollo de R, la exploración e implementación de nuevas metodologías y la enseñanza o capacitación sobre computación estadística, entre otras.

El entorno de R consiste en una distribución base, desarrollada y mantenida por "*R Development Core Team*", donde se pueden encontrar las funcionalidades principales del *software*, como son: manipulación de datos, creación de gráficos, análisis de distribución de probabilidades, análisis estadístico de datos y utilidades para importar o exportar datos en diferentes formatos, entre otros. A esta distribución base se le pueden añadir funcionalidades a partir de paquetes creados por desarrolladores independientes. Éstos deben ser enviados a una red de servidores llamada "*The Comprehensive R Archive Network*" (CRAN), la cual se encarga de realizar la respectiva verificación de calidad de éstos y de servir como repositorio o distribuidor de las diferentes versiones o actualizaciones, tanto de la distribución base como de los diferentes paquetes producidos. Algunas herramientas que pueden ser agregadas a través de paquetes están enfocadas en aprendizaje automático, minería de texto, visualización interactiva de datos y procesamiento de imágenes, entre otros.

Cabe destacar que CRAN pone a disposición de los usurarios de R una guía llamada "*task view",* donde se pueden encontrar los paquetes que están disponibles según la temática que se pretenda abordar, adicionalmente éstos son sub-agrupados de acuerdo con la tarea o actividad a desarrollar en relación con dicha temática. Por ejemplo, si el usuario está interesado en utilizar R para trabajar en la temática de redes bayesianas, podría ingresar al *task view* y buscar el tópico de modelos gráficos (gR - *gRaphical Models in* R, https://cran.r-project.org/web/views/gR.html). Al ingresar en éste podrá encontrar todos los paquetes desarrollados con la intención de brindar herramientas para tratar dicho tema.

2 Redes bayesianas

A partir del auge de las ciencias computacionales, la configuración subyacente sobre cualquier campo del conocimiento ha podido ser codificada de manera más sencilla y rápida por medio de modelos, los cuales a fin de cuentas nos brindan una representación simplificada de la realidad. Estos modelos en muchas ocasiones son construidos con la finalidad de generar conocimiento que permita entender cuáles son las variables que participan en la temática que se intenta abordar, cómo interactúan entre ellas y/o para resolver preguntas relacionadas con dicho tema.

Para la construcción de estos modelos no siempre se tiene información completa o el verdadero estado del sistema, por lo que las observaciones con las que contamos son parciales y están sujetas a un estado de incertidumbre inherente a ellas. Es por esta razón que la teoría de la probabilidad coadyuva a su aprendizaje a través de datos y a su vez los denomina como "modelos probabilísticos". Se resalta que los mismos también pueden ser articulados mediante conocimiento de una persona experta en el tema o una combinación entre conocimiento experto y aprendizaje a partir de los datos.

Los modelos probabilísticos pueden ser representados de diversas formas, a saber: gráficamente, por listas de relaciones de independencia o utilizando un conjunto de funciones de probabilidad condicionada. En este documento nos centraremos en estudiar los modelos probabilísticos obtenidos a partir de gráficos, especialmente aquellos donde sus relaciones de dependencia están codificadas a través de aristas con una dirección específica, en otras palabras, modelos gráficos probabilísticos conocidos como redes bayesianas.

En relación con lo anterior, una red bayesiana se define como un modelo probabilístico que se representa por medio de un gráfico dirigido acíclico (DAG, por sus siglas en inglés), donde los nodos son variables aleatorias relacionadas con la temática que se está abordando y las aristas se constituyen como las relaciones de dependencia probabilística de un nodo sobre otro. En estas últimas se codifican las restricciones sobre la independencia condicional propias de la distribución conjunta de las variables. En este sentido, la estructura de la red está conformada por los nodos y sus respectivas aristas o arcos, mientras que las independencias condicionales de cada variable se pueden deducir de los arcos y de la distribución de probabilidad conjunta.

Cada variable o nodo en el modelo está asociado con su distribución de probabilidad condicional (CPD por sus siglas en inglés) dados sus nodos padres en el grafo, excepto los nodos sin padres quienes en lugar de la CPD tienen asociada la probabilidad marginal. Por lo tanto, la CPD especifica la distribución de las probabilidades que toma la variable, dada cada asignación de valores que pueden tomar las otras variables que figuran como nodos padres.

De acuerdo con Koller y Friedman (2009), el gráfico que simboliza a una red bayesiana puede ser visto de dos maneras diferentes:

- Como una estructura de datos que provee el esqueleto (grafo no dirigido) para representar la distribución de probabilidad conjunta compactada en su versión factorizada.

- Como una representación de un conjunto de supuestos sobre la independencia condicional de la distribución de probabilidad.

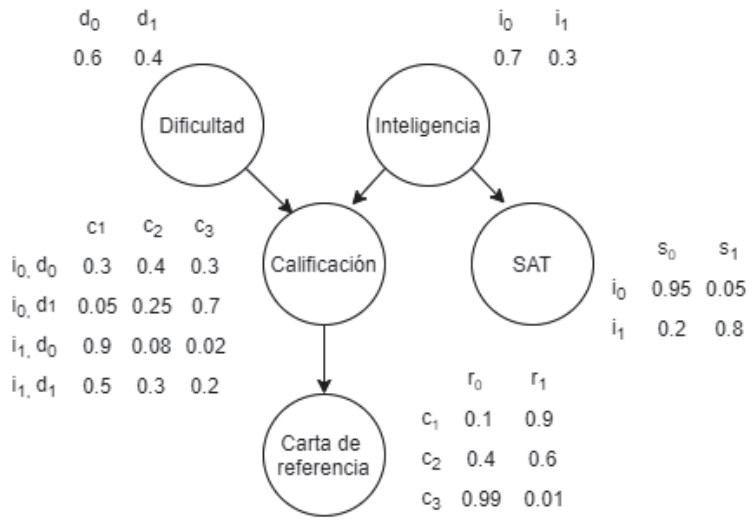
En todo caso, ambas pueden ser consideradas como análogas.

2.1 Representación e independencia gráfica

Formalmente, se puede definir una red bayesiana como una pareja $(G,\ \theta)$ que representa la distribución de probabilidad conjunta de un grupo finito $\chi = \{\chi_1, \dots, \chi_\eta\}$ de variables, donde $G = \{V,\ E\}$, es un DAG, cuyos nodos (V) corresponden a las variables definidas en χ y las aristas E se configuran como las dependencias probabilísticas directas entre ellas; mientras que θ hace referencia a las distribuciones de probabilidad condicionada que definen el valor de la probabilidad de cada variable χ_i dado sus padres Π_i en el grafo.

En la figura 1 se puede observar un ejemplo de una red bayesiana sencilla tomado de Koller y Friedman (2009). En ésta se evidencia que cada nodo es una variable, las aristas determinan la relación de dependencia de acuerdo con su dirección y también se muestran las tablas de probabilidad condicionada para cada variable dado sus padres. Simplificando, la representación gráfica consiste en el caso de un estudiante que necesita una carta de recomendación de su profesor para poder acceder a un trabajo. Como se refleja en el gráfico, la calidad de dicha carta dependerá de la calificación obtenida en la materia, la cual a su vez está supeditada por el grado de dificultad de la misma y por el nivel de inteligencia del estudiante, este último también figura como factor determinante del puntaje en las pruebas SAT (examen estandarizado para la admisión universitaria en E.E.U.U). Por ejemplo, si quisiéramos conocer la probabilidad de que el estudiante obtenga una calificación intermedia dado que el nivel de dificultad de la materia es alto pero su nivel de inteligencia es bajo, bastaría con computar la siguiente probabilidad condicional, $P(c_2 \mid i_0, d_1) = 0.25$. Una de las mayores utilidades de una red bayesiana es la inferencia probabilística, es decir que por medio de ésta se pueda calcular la probabilidad posterior $P\ (\chi|\gamma)$ de un evento χ dada la evidencia disponible de otras variables $\boldsymbol{\gamma}$.

Figura 1. Representación gráfica de una red bayesiana, caso del estudiante.



Fuente: Koller y Friedman (2009)

Recapitulando, el DAG exhibe las relaciones de dependencia condicional entre las variables del modelo y factoriza la distribución de probabilidad global en un conjunto de distribuciones de probabilidad local, una para cada variable. La forma de esta factorización viene dada por la propiedad de Markov, la cual establece que en una red bayesiana cada nodo χ_i es condicionalmente independiente de sus nodos no-descendientes dados sus padres. Así se tiene que:

$$P(\chi) = P(\chi_1, \ldots, \chi_\eta) = \prod_i P\,(\chi_i | \,\Pi_i)$$

Cabe resaltar que la representación completa de la tabla conjunta de probabilidad tomaría un espacio exponencial de acuerdo con el número de variables n, razón por la cual se hace uso de la propiedad de Markov para evitar esta dificultad.

En esta misma línea, un concepto clave en la representación e independencia gráfica de redes bayesianas es el manto de Markov, éste se define como el subconjunto de nodos que hacen que un nodo χ_i sea independiente del resto de la red. Para χ_i dicho manto está compuesto por sus padres, sus hijos y los otros nodos con los cuales χ_i comparta un hijo (esposas), los dos primeros a su vez se configuran como los vecinos de χ_i. Se subraya que la identificación del manto de Markov se puede generalizar como un problema de selección de variables (Koller y Friedman, 2009).

2.2 Mapas de independencias y dependencias

Los criterios de separación gráfica coadyuvan a entender cómo pueden codificarse las relaciones de (in)dependencia condicional en un grafo debido que facilitan la identificación y representación gráfica de éstas al comprobar cuáles, de entre todas las posibles relaciones de independencia condicional existentes en los datos son satisfechas por el grafo o viceversa.

Así las cosas, considérese a M como el modelo de dependencia de la distribución de probabilidad de χ, la cual es el conjunto de relaciones de independencia condicional que conecta cualquier tripleta **A, B, C** de subconjuntos de χ. Un DAG G se considera como un mapa de independencia (I-mapa) de la estructura de dependencia M, si existe una correspondencia uno a uno entre los nodos V pertenecientes a G y las variables aleatorias χ, tal que para todos los subconjuntos disjuntos **A, B, C** en χ, se tenga: $\mathrm{A} \perp\!\!\!\perp_g B \mid C \implies \mathrm{A} \perp\!\!\!\perp_p B \mid C$, es decir si todas las relaciones de dependencia derivadas de G, son verificadas por M. Análogamente, G es un mapa de dependencia (D-mapa) de M, si: $\mathrm{A} \perp\!\!\!\perp_p B \mid C \implies \mathrm{A} \perp\!\!\!\perp_g B \mid C$ y G es un mapa perfecto de M, si: $\mathrm{A} \perp\!\!\!\perp_p B \mid C \iff \mathrm{A} \perp\!\!\!\perp_g B \mid C$.

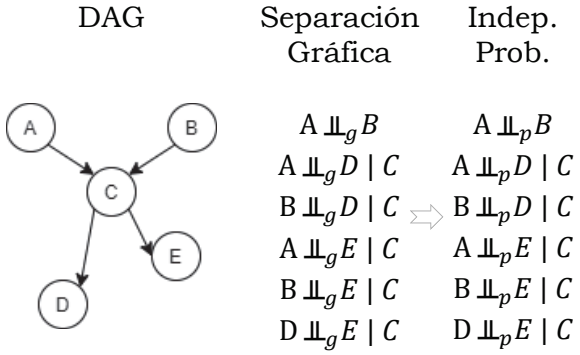
En el caso de los D-mapas, la distribución de probabilidad de χ determina cuáles son las aristas que estarán presentes en el gráfico G. Por ejemplo, los nodos que están conectados en el DAG G obligatoriamente corresponderán a variables que son dependientes entre sí en χ, sin embargo, nodos que estén separados en G no necesariamente serán variables condicionalmente independientes en χ.

Por otro lado, los I-mapas se caracterizan por el hecho de que la presencia de una arista en el grafo determina qué variables son condicionalmente

dependientes en χ, por lo tanto, nodos separados en G son variables condicionalmente independientes en χ pero nodos que sí están conectados en G, no necesariamente corresponden a variables dependientes en χ (Scutari y Denis, 2014).

La figura 2 muestra un ejemplo de lo mencionado anteriormente, a partir de ésta se observa un DAG, el cual puede definirse como un I-mapa de la distribución de probabilidad de χ, con una representación gráfica ($\perp\!\!\!\perp_g$) que implica independencia probabilística ($\perp\!\!\!\perp_p$), toda vez que las posibles relaciones de independencia condicional existentes en el grafo son satisfechas por el conjunto de datos.

Figura 2. Representación de la separación gráfica, independencia probabilística y DAG



Fuente: Scutari (2017)

2.3 D-separación

Al construir una red bayesiana a partir de los datos, un aspecto clave consiste en entender los criterios que se utilizan para determinar la independencia condicional, los cuales a su vez se derivan de los criterios de separación analizados anteriormente y son establecidos utilizando el concepto de D-separación.

En otro sentido, este concepto permite identificar las (in)dependencias condicionadas establecidas por la red y se define de acuerdo con Castillo, Gutiérrez y Hadi (1997), como: *sean **X, Y** y **Z** tres subconjuntos disjuntos de un DAG G, entonces se dice que Z D-separa a X e Y si y sólo si a lo largo de todo el camino no dirigido entre cualquier nodo de X y cualquier nodo de Y existe un nodo intermedio A tal que, o bien*

- *A es un nodo de aristas convergentes en el camino y ni A ni sus descendientes están en Z, o bien*
- *A no es un nodo de aristas convergentes en el camino y A está en Z.*

Cuando Z D-separa X e Y en G, se escribe $I(X,Y|Z)_G$ para indicar que la relación de independencia vine dada por el DAG G; en caso contrario, se escribe $D(X,Y|Z)_G$ para indicar que X e Y son condicionalmente dependientes dado Z en el DAG G.

La anterior definición es especialmente útil para cuatro escenarios donde X e Y pueden estar conectados por un nodo intermedio Z; las dependencias o independencias condicionales para estos casos dependerán de si Z es observado o no, es decir, si tenemos evidencia sobre el valor que toma dicha variable (ver figura 3). En este sentido, para los tres primeros casos X e Y serán dependientes condicionalmente dado Z si y sólo si el valor de Z es desconocido, mientras que el hecho de observar a Z haría que X no brinde información adicional sobre Y, y viceversa, por lo que se tendría que $X \perp\!\!\!\perp_g Y \mid Z$ o que $I(X, Y|Z)_G$. Para el cuarto caso la situación es diferente. Intuitivamente al ser Z desconocido X e Y son independientes, pero cuando se tiene evidencia de Z o de alguno de sus descendientes directos, estas dos se correlacionan debido a que al observar Z la creencia que se tenga de X puede ser influenciada por Y. Esta última representación se conoce como V-estructura y se caracteriza por su esqueleto en forma de V, así como por el hecho de que las aristas de los nodos X e Y convergen en Z sin que exista una arista adicional entre ellos; siempre que se presente ésta, se escribirá que $D(X, Y|Z)_G$ o $X \not\perp\!\!\!\perp_g Y \mid Z$.

Figura 3. Escenarios de D-separación

2.4 Aprendizaje de redes bayesianas

Una de las principales tareas que se debe desarrollar al momento de trabajar con redes bayesianas consiste en su construcción, actividad que puede ser realizada a partir de conocimiento previo acerca de la temática que se pretender abordar, por medio de la aplicación de ciertos algoritmos sobre un conjunto de datos relacionado con el fenómeno de interés o a través de una combinación de éstas. En la mayoría de las ocasiones la creación de la red que involucra el conocimiento previo o experto es ejecutada de forma manual, por lo que puede llegar ser problemática o confusa; mientras que la construcción de la misma por medio de datos es mucho más sencilla y rápida, en parte, gracias al auge de las ciencias computacionales. Específicamente, este procedimiento se conoce como aprendizaje de la red y se divide en dos etapas, a saber: aprendizaje de la estructura y aprendizaje de los parámetros. El primero se refiere a la identificación del DAG o estructura gráfica de la red, es decir, determinar la existencia y dirección de las aristas que deberán conectar dos nodos (variables) con el fin de estudiar las relaciones de dependencia entre éstos; mientras que el segundo consiste en la estimación de las distribuciones de probabilidad locales de cada nodo derivadas del gráfico aprendido.

De acuerdo con Scutari y Denis (2014), el procedimiento de aprendizaje de una red bayesiana puede ser formalizado así,

$$\underset{\text{Aprendizaje}}{\Pr(B|D) = \Pr(G, \theta|D)} \quad \rightarrow \quad \underset{\substack{\text{Aprendizaje de la} \\ \text{estructura}}}{\Pr(G|D)} \quad * \quad \underset{\substack{\text{Aprendizaje de los} \\ \text{parámetros}}}{\Pr(\theta|G, D)}$$

donde se tiene un conjunto de datos D y una red bayesiana $B = (G,\ \theta)$.

Como se mencionó anteriormente, el aprendizaje de una red bayesiana puede ejecutarse al combinarse conocimiento previo y la aplicación de un algoritmo sobre el conjunto de datos. En esencia, la información previa sobre el modelo se configura como la certeza que se tiene sobre la presencia o ausencia de aristas entre los nodos. En la mayoría de los casos, la inserción de dicho conocimiento se realiza a través de argumentos incluidos dentro de las funcionalidades implementadas por los *softwares* o herramientas elegidas para trabajar y se conocen como *whitelist* o *blacklist.* El primero de éstos garantiza la presencia de una arista entre dos nodos, mientras que el segundo se asegura de que una arista se encuentre ausente. Dichas restricciones se pueden combinar considerando cualquier pareja de nodos, por ejemplo: se podría establecer una *whitelist* para incluir una arista entre los nodos $A \rightarrow B$ pero también se podría determinar que $B \rightarrow A$ debe estar presente, en ese caso, el arista existirá entre los dos nodos pero la dirección la determinará el algoritmo (Scutari y Denis, 2014). Se destaca que en algunos *softwares* se presentan funciones o argumentos que siguen la misma idea pero ha sido desarrollados bajo otros nombres.

Para concluir, es importante resaltar que las redes bayesianas pueden ser construidas a partir de variables cuantitativas (discretas o continuas), cualitativas (variables categóricas) o una mezcla de ellas. Aquellas derivadas de conjuntos de datos compuestos por variables discretas o cualitativas (nominales u ordinales) se conocen como redes bayesianas (discretas), mientras que las provenientes de variables continuas con asunciones de normalidad se denominan redes bayesianas gaussianas y, por último, como redes bayesianas gaussianas condicionadas linealmente son conocidas las que proceden de conjuntos de datos mixtos (en la mayoría de los casos).

2.4.1 Aprendizaje de la estructura

En esencia, el procedimiento de aprendizaje de la estructura de una red bayesiana equivale a establecer el conjunto de aristas directas E para el DAG $G = \{V, E\}$. Dicho de otro modo, el objetivo consiste en determinar los nodos padres (Π_i) de cada variable o nodo χ_i.

Existen en la literatura académica sobre redes bayesianas una gran cantidad y variedad de métodos o algoritmos para acometer esta tarea a partir de los datos, no obstante, éstos pueden ser agrupados en tres categorías principales, a saber: algoritmos de aprendizaje basados en restricciones, algoritmos de aprendizaje basados en búsqueda y puntaje, y finalmente métodos híbridos.

2.4.1.1 Algoritmos de aprendizaje basados en restricciones

Este tipo de algoritmos llevan a cabo el aprendizaje estructural de la red por medio de pruebas estadísticas de independencia condicional sobre el conjunto de datos con el fin de identificar las relaciones de dependencia entre las variables del modelo. Las (in)dependencias encontradas se utilizan para construir las separaciones gráficas de la red y a su vez para esbozar un gráfico que satisfaga los criterios de separación correspondientes. En otras palabras, las pruebas estadísticas de independencia determinan la existencia de las aristas y su dirección. Se destaca que este tipo de algoritmos asumen que el DAG debe ser un mapa perfecto para exhibir la correcta estructura de la red.

A continuación, se presentarán los algoritmos basados en restricciones habilitados por los paquetes de R objeto de análisis:

- *Peter y Clark* (PC - estable): el algoritmo original, llamado PC, fue introducido por Spirtes, Glymour y Scheines (2000) con la desventaja de que el orden de las variables era importante para conseguir resultados estables; Colombo y Maathuis (2014) prestaron una versión mejorada que soslayaba dicha dificultad, al cual llamaron PC - estable. En este sentido, el algoritmo inicialmente define un grafo dirigido parcialmente completo en el cual se van eliminando aristas (o direccionando) si la prueba estadística de independencia muestra que existe independencia entre las variables y continúa haciendo este proceso hasta que todas las pruebas son satisfechas en el grafo o hasta cuando ha considerado todos los pares de nodos.

A continuación se muestra el pseudocódigo del algoritmo de PC, esto debido a que es uno de los más populares y además, en algunas ocasiones, figura como base de otros algoritmos; lo cual denota su importancia.

Pseudo código algoritmo PC
Entrada: conjunto de aristas (E), información de independencia condicional, nivel de significancia α **Salida:** CPDAG $\hat{G}$(grafo dirigido parcialmente completo, es decir que algunas aristas no están orientadas), conjunto de separaciones $\hat{S}$ **Tipo de aristas:** $\rightarrow, -$ 1. Construye el grafo completo no dirigido a partir del conjunto de aristas (E). 2. Realiza pruebas de independencia condicional entre nodos adyacentes y dado el nivel de significancia α elimina aristas si encuentra evidencia estadística de que éstos son independientes condicionalmente. 3. Orienta las V-estructuras 4. Orienta las aristas restantes

- *Fast Casual Inference* (FCI): diseñado como una extensión del algoritmo PC para inferir las independencias condicionales y la información causal entre variables, su valor añadido consiste en que tolera y en ocasiones descubre variables latentes u ocultas. FCI construye un grafo causal al iniciar con una estructura no directa totalmente conectada e ir eliminando aristas que unen a nodos condicionalmente independientes, posteriormente orienta las aristas al identificar las D-separaciones o V-estructuras y las posibles variables ocultas. Su principal limitación radica en que es computacionalmente costoso en redes grandes (Spirtes, Glymour y Scheines, 2000).
- *Really Fast Casual Inference* (RFCI): versión mejorada de FCI propuesta por Colombo, Maathuis, Kalisch y Richardson (2012). En comparación con FCI este algoritmo emplea una menor cantidad de pruebas de independencia condicional al evitar evaluar subconjuntos posiblemente D-separados, por lo que trabaja más rápido y es computacionalmente menos complejo.
- *Fast Casual Inference Plus* (FCIPLUS): este algoritmo toma como referencia o base la implementación de FCI pero con una modificación en la etapa de búsqueda de las D-separaciones. Puntualmente, utiliza la información de independencia derivada de etapas previas para guiar la

búsqueda de las D-separaciones, razón por la cual el tiempo empleado en las evaluaciones de independencia mejora (Claassen, Mooij y Heskes, 2013).

- *Grow-Shrink* (GS): construye la red al recuperar la estructura local de cada nodo, la cual a su vez es derivada de la estimación del manto de Markov. Posteriormente, direcciona los arcos por intermedio de pruebas de independencia condicional. El manto de Markov es identificado a través del proceso de *"Growing"* y *"Shrinking"*: en el primero se tiene un conjunto vacío en el cual van ingresando variables siempre y cuando tengan una relación de dependencia con χ_i dadas las demás variables que están incluidas en el conjunto. En la segunda parte, se identifican y eliminan las variables que realmente estaban por fuera del manto (Margaritis, 2003).
- *Incremental Association Markov Blanket* (IAMB): este algoritmo es estructuralmente similar a GS, es decir, la construcción de la red y el direccionamiento de las aristas siguen el mismo método. No obstante, sus autores consideran que existe una diferencia importante entre estos dos, la cual consiste en que GS para llevar a cabo la inclusión de las variables en la primera fase realiza un único ordenamiento de éstas de acuerdo con su grado de asociación con χ_i dado el conjunto vacío, por lo que la siguiente variable en dicho orden puede no ser condicionalmente independente de χ_i dado el conjunto; mientras que IAMB reordena el conjunto de variables cada vez que una de éstas ingresa al manto, generando así la disminución del número de falsos positivos que producen, razón por la cual en la siguiente fase son menos las eliminaciones que se deben hacer. (Tsamardinos, Aliferis y Statnikov, 2003).
- *Interleaved Incremental Association* (Inter-IAMB): se considera como una extensión de IAMB que espera ser más eficiente al reducir la cantidad de pruebas de independencia condicional a realizar. En este sentido, Inter-IAMB utiliza dos métodos para disminuir dicho número de pruebas; en el primero va intercalando el proceso de inclusión de variables y el proceso de eliminación de falsos positivos con la idea de mantener la dimensión del manto lo más pequeña posible mientras se ejecuta el algoritmo. El segundo cambio consiste en que para determinar la dirección de las aristas implementa el algoritmo PC (Tsamardinos, Aliferis y Statnikov, 2003).
- *Fast Incremental Association* (Fast-IAMB): implementado por Yaramakala y Margaritis (2005) como una versión computacionalmente más rápida de IAMB; la intención era reducir el número de pruebas de independencia al añadir más de una variable al manto de Markov después de cada reordenamiento (fase 1). Cabe destacar que al igual que IAMB e Inter-IAMB, Fast-IAMB después de cada iteración en el proceso de inclusión de variables, reorganiza las mismas de manera descendiente de acuerdo con el nivel de dependencia; no obstante, a diferencia de las dos anteriores utiliza la versión apropiada de la prueba G^2 de independencia condicional en lugar de la información mutua. En esencia, Fast-IAMB añade una o más variables de forma especulativa sin tener la necesidad de hacer un reordenamiento después de cada iteración, ejecutando múltiples pruebas en lugar de una a la vez. Con esto espera incluir más de un verdadero miembro del manto a la vez, reduciendo así la cantidad de falsos positivos a eliminar.

- *Incremental Association with FDR Correction* (IAMB-FDR): modificación del algoritmo IAMB que busca controlar la tasa de falsos positivos (FDR por sus siglas en inglés). Específicamente, Peña (2008) modifica IAMB para que los nodos que formen parte del manto de Markov sean aquellos cuya prueba de hipótesis nula sea rechazada al utilizar el procedimiento BY, al nivel de significación α con respecto a la hipótesis nula de $X \perp Y \mid MB \backslash Y$.
- *Max-Min Parent and Children* (MMPC): se ejecuta en dos fases, en la primera se seleccionan las posibles variables que figuran como padres o hijos de χ_i, se crea entonces un subconjunto de padres e hijos (CPC por sus siglas en inglés), donde la primera variable que ingresa es aquella que tenga mayor asociación con χ_i, las siguientes son incluidas debido a que exhiben la asociación máxima con $\chi_i$condicionada al subconjunto CPC que logra la asociación mínima posible para esta variable. En la segunda fase se eliminan los falsos positivos incluidos en la primera etapa al realizarse pruebas de independencia condicional para determinar si χ_i puede estar D-separada de una variable del subconjunto CPC dados todos los posibles subconjuntos de éste. Se destaca que éste construye un grafo no dirigido (Tsamardinos, Aliferis y Statnikov, 2003).
- *Semi-Interleaved Hiton-PC* (SI-HITON-PC): determina la frontera de Markov (manto de Markov minimal) de χ_i, siempre y cuando éste exista en la distribución de probabilidad conjunta P, tal que todos sus miembros sean marginalmente dependientes de χ_i y a su vez sean condicionalmente dependiente de éste. Se destaca que produce un grafo no dirigido (Aliferis, et al. 2010)
- *Hybrid Parents & Children* (HPC): busca identificar los padres e hijos de un nodo χ_i a través de tres subrutinas, a saber: determinar el superconjunto de padres e hijos, establecer el superconjunto de esposas (padres de los hijos) y finalmente utiliza *Incremental Association with FDR Correction* para controlar los falsos positivos. Se resalta que éste construye un grafo no dirigido (Gasse, Aussem y Elghazel, 2014).

Se subraya el hecho de que en algunas ocasiones los algoritmos basados en restricciones no necesariamente aprenden o identifican un DAG, en estos casos devuelven un grafo no dirigido. Por otro lado, la mayoría de estos algoritmos son implementados en R por medio del paquete **bnlearn** y en menor medida por el paquete **pcalg.** En secciones posteriores se especificará cuál de éstos permite utilizar cada uno de los algoritmos.

Como se mencionó anteriormente, el aprendizaje basado en restricciones se fundamenta en pruebas estadísticas de independencia condicional para determinar las restricciones de independencia existentes en los datos y así plasmar éstas en la estructura de la red. Razón por la cual es importante tener en cuenta cuáles son las pruebas que están habilitadas en los paquetes objeto de análisis. Se listan a continuación.

- Información mutua: mide la reducción en la incertidumbre de χ debido a que γ es observada o se tiene conocimiento sobre su valor (o viceversa, ya que es una relación simétrica). Si χ y γ son independientes la información mutua toma un valor de cero.

  Definida como:

$$\mathrm{MI}(X, Y \mid \boldsymbol{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{C}\sum_{k=1}^{L}\frac{n_{ijk}}{n}\log\frac{n_{ijk}n_{++k}}{n_{i+k}n_{+jk}}$$

donde, las frecuencias observadas $\{n_{ijk}, i = 1, \ldots, \mathrm{R}, j = 1, \ldots, \mathrm{C}, k = 1, \ldots, \mathrm{L}\}$ para las variables aleatorias X y Y y todas las configuraciones de la variable condicionante $\boldsymbol{Z}$.

- Test de chi-cuadrado de independencia para tablas de contingencia: se utiliza para medir la asociación (dependencia) entre dos variables cualitativas o categóricas; se basa en la comparación de las frecuencias observadas con respecto a las frecuencias esperadas. El valor del estadístico resultante mide la discrepancia entre las frecuencias mencionadas anteriormente y debe ser comparado con el valor crítico (p-valor) para determinar si las variables objeto de análisis son independientes o no.

  Definida como:

$$\mathrm{X}^2(X, Y \mid \boldsymbol{Z}) = \sum_{i=1}^{R}\sum_{j=1}^{C}\sum_{k=1}^{L}\frac{(n_{ijk} - m_{ijk})^2}{m_{ijk}}, \qquad m_{ijk} = \frac{n_{i+k}n_{+jk}}{n_{++k}}$$

- *Shrinkage estimator* para información mutua: propuesto por Hausser y Strimmer (2009), como un procedimiento para estimar la entropía y/o la información mutua en muestras pequeñas al emplear la contracción James-Stein. Se configura como una prueba chi-cuadrado asintótica mejorada para la información mutua.
- *Jonckheere-Terpstra*: es una prueba que se utiliza para determinar la existencia de una tendencia estadísticamente significativa entre dos variables, donde una de ellas es de tipo ordinal.
- Correlación lineal de Pearson: medida de dependencia lineal entre dos variables cuantitativas continuas. Toma valores en el intervalo [-1,1], siendo cero evidencia de que no existe relación lineal entre las variables.
- Transformación de la Z de *Fisher*: se puede utilizar para probar hipótesis sobre la correlación entre variables. La prueba es una transformación de la correlación lineal donde todas las distribuciones son normales y con varianza conocida, evitando así el sesgo.

  Definida como:

$$\mathrm{Z}(X, Y \mid \boldsymbol{Z}) = \frac{1}{2}\sqrt{n - |\mathbf{Z}| - 3}\log\frac{1 + \mathrm{P}_{XY|Z}}{1 - \mathrm{P}_{XY|Z}}$$

Las anteriores pruebas deben ser usadas teniendo en cuenta la naturaleza de los datos y por ende el tipo de red bayesiana que se está construyendo. En este sentido la tabla 1 resume qué prueba se puede utilizar en cada escenario.

Tabla 1. Uso pruebas de independencia según tipo de dato

Pruebas de (in)dependencia	Red bayesiana discreta		Red bayesiana Gaussiana	Red bayesiana mixta
	Variables categóricas	Variables ordinales		
Información mutua	x		x	x
Correlación de Pearson para tablas de contingencia	x			
Shrinkage estimator para información mutua	x		x	x
Jonckheere-Terpstra		x		
Correlación lineal de Pearson			x	
Transformación de la Z de Fisher			x	

Al igual que sucede con los algoritmos, la mayoría de estas pruebas están disponibles en el paquete **bnlearn.**

2.4.1.2 Algoritmos de aprendizaje basados en búsqueda y puntaje

En pocas palabras, la tarea que desarrollan estos algoritmos consiste en encontrar el mejor DAG teniendo en cuenta un puntaje que califica los distintos modelos de acuerdo con la calidad con que se ajusten éstos a los datos. Dado lo anterior, el aprendizaje de la red bayesiana se convierte en un problema de maximización del puntaje, donde se tiene un espacio hipotético que contiene todas las posibles estructuras gráficas que se derivan de los datos y un método para calcular el puntaje que el algoritmo asignará a cada red potencial. Considerando ésto, intentará maximizar dicho puntaje a partir de una técnica de optimización heurística por medio de una búsqueda sobre el espacio de posibles soluciones. Formalmente, el objetivo es: $argmax\ puntaje\ (G, D)$.

Cabe destacar que la mayoría de las tareas asociadas al aprendizaje y a la inferencia sobre redes bayesianas se configuran como problemas *NP-Hard* (no pueden ser resueltos en tiempo polinómico de acuerdo con el número de variables), por lo que el aprendizaje basado en puntaje y búsqueda no es la excepción. En este último caso, el espacio hipotético que contiene todas las posibles estructuras gráficas crece súper-exponencialmente de acuerdo el tamaño del conjunto de datos y el número de padres que puede tener un nodo, por lo tanto, aun tratando de maximizar un puntaje no es seguro que se encuentre la red con el puntaje más alto. Lo anterior fue evidenciado por Chickering (1996), quien demostró que para uno de los puntajes bayesianos más utilizados (BDeu), identificar la estructura con el puntaje más elevado en conjuntos de datos pequeños es difícil, incluso cuando cada nodo es restringido a tener como máximo dos padres. Posteriormente, Chickering, Heckerman y Meek (2004), obtuvieron conclusiones similares a partir de conjuntos de datos grandes.

A continuación, se presentarán los algoritmos de búsqueda habilitados por los paquetes de R objeto de análisis:

- *Greedy search - Hill Climbing* (HC): el algoritmo inicia con la selección de una red bayesiana, puede ser una estructura vacía (sin ninguna arista),

una construida a partir de conocimiento previo o una elección aleatoria sobre el espacio hipotético de estructuras gráficas. Posteriormente, se calcula el puntaje inicial de dicha red y se inicia un proceso de iteración donde se intenta mejorar este valor al eliminar, añadir o redireccionar una arista a la vez. Después de cada una de las posibles perturbaciones se calcula el puntaje de la nueva red para finalmente decantarse por la estructura que obtuvo el valor más alto. Se destaca que en cada iteración se evalúa la red o el estado actual versus el siguiente estado, denominado "vecino"; el proceso se mueve al siguiente vecino y finaliza cuando no se evidencian mejoras en el puntaje.

A continuación se muestra el pseudocódigo del algoritmo de *Greedy search - Hill Climbing*, esto debido a que es uno de los más populares y además, en algunas ocasiones, figura como base de otros algoritmos; lo cual denota su importancia.

Pseudo código algoritmo *Greedy search* – HC

1. Selecciona una estructura G del espacio hipotético de estructuras gráficas, por lo general, una vacía (aunque no necesariamente).
2. Calcula el puntaje de G, denotado como $Puntaje_G = Puntaje\ (G)$.
3. Establece $puntaje\ máximo = Puntaje_G$.
4. Repite los siguientes pasos siempre y cuando el *puntaje máximo* aumente:
    a. Para cada posible adición, eliminación o redireccionamiento de un arco que no resulte en una red cíclica:
        i. Calcula el puntaje de la red modificada G^*, $Puntaje_{G^*} = Puntaje(G^*)$.
        ii. Si $Puntaje_{G^*} > Puntaje_G$, establece $G = G^*$ y $Puntaje_G = Puntaje_{G^*}$.
    b. Actualiza $puntaje\ máximo$ con el nuevo valor de $Puntaje_G$.
5. Devuelve el DAG G.

- *Greedy search with random restarts* (GSRR): la principal desventaja del algoritmo HC consiste en que puede quedarse atascado en un punto conocido como máximo local o en una zona denominada *plateaux* (zona de alto puntaje con muchos valores iguales, es decir, muy plana) al no tener una visión completa del espacio, por lo que podría seleccionar puntajes que en realidad no maximizan la función global. Dado ésto, se implementan reinicios aleatorios en el procedimiento de búsqueda con la idea de moverse de estos puntos en caso de haber caído en ellos (ver Koller y Friedman, 2009).
- *Tabu:* este algoritmo sigue una metodología similar a la de *Greedy Search* - HC, no obstante, brinda la posibilidad de escapar de los puntos de máximo local y *plateaux* al conservar un listado de las últimas modificaciones o perturbaciones realizadas sobre la estructura y no le permite al algoritmo volver a operar sobre ellas, obligándolo así a moverse hacía adelante en la búsqueda de la red que maximice el puntaje (Glover, 1989).
- *Simulated Annealing* (SA): cuenta con una composición parecida a la del algoritmo *Greedy Search* - HC, en el sentido de que evalúa algunos vecinos de la red actual y decide entre efectuar una transición a la nueva red o no. No obstante, SA se diferencia de una *Greedy Search* al ejecutar el aprendizaje de la estructura por medio de movimientos aleatorios y en el uso de probabilidades para la transición o no a la nueva red; por

ejemplo, si uno de los movimientos aleatorios muestra que coadyuvó a la mejora del puntaje, es aceptado. De lo contrario, el algoritmo puntúa el movimiento con una probabilidad inferior a uno; el valor de esta probabilidad es castigado considerando la mala calidad del movimiento (ver Russell y Norvig, 2009).

- *Greedy Equivalence Search* (GES): es un algoritmo codicioso (*greedy*) que busca sobre el espacio de DAGs equivalentes (dos grafos son equivalentes si y sólo si tiene el mismo esqueleto y V-estructuras). En general, una *Greedy Search* en cada iteración evalúa la red o el estado actual versus el siguiente estado o vecino y selecciona a quien tenga el mayor puntaje si eso conlleva a una mejora del mismo. El conjunto de vecinos de cada estado en la búsqueda define la conectividad del espacio. GES se divide en dos fases, en la primera se realiza una *Greedy Search* (eliminando o añadiendo aristas) sobre el espacio de DAGs equivalentes usando una conectividad particular entre éstos. Una vez se alcanza el máximo local, una segunda fase procede desde el anterior máximo local usando una segunda conectividad, cuando esta segunda fase alcanza su máximo local, el grafo equivalente es devuelto (Chickering, 2002).
- *Greedy Interventional Equivalence Search* (GIES): este algoritmo se constituye como una generalización de GES que permite trabajar sobre datos experimentales. En esencia siguen la misma metodología, la única diferencia es que GIES introduce una fase donde no se elimina ni se añaden aristas sino que se ejecuta el redireccionamiento de una sola de éstas para determinar si dicha acción mejora el puntaje de la red (Hauser y Bühlmann, 2012).
- *Silander-Myllymäki* (SM): se constituye como una solución a problemas de aprendizaje cuando se tienen aproximadamente veinticinco variables, como máximo.
- *Regularized Maximum-Likelihood Estimation:* algoritmo propio del paquete **sparsebn** que se basa en la siguiente función:

$$\min_{B \in D} \ell(G; \chi) + \rho_\lambda(G)$$

donde el objetivo es minimizar el programa anterior, el cual se compone por el conjunto de observaciones χ, la *log-likelihood* negativa ℓ, el grafo G y un regularizador ρ_λ, este último se incluye para evitar estructuras densas (está compuesta por una mayor cantidad de aristas en comparación con la cantidad de nodos). El resultado es un conjunto de DAGs ordenado de acuerdo con el rango establecido (por defecto o provisto por el usuario) para ρ_λ (Aragam, Gu y Zhou, 2019).

- *Exhaustive Search Dynamic Programming* (SearchOrder)*:* este algoritmo es propio del paquete **catnet** y se basa en programación dinámica (método utilizado cuando la solución de un problema requiere ser trabajada a través de subproblemas que ocurren varias veces) para buscar en el espacio hipotético de redes teniendo en cuenta el orden de los nodos provisto por el usuario. El resultado es un conjunto DAGs ordenado de acuerdo con su nivel de complejidad (especifica el número de parámetros necesarios para definir la distribución de probabilidad de la red). En otras palabras, cada DAG resultante es exactamente el *Maximum-Likelihood Estimation* (MLE) para cada nivel de complejidad (Balov y Salzman, 2020).

Como bien se sabe, para realizar el aprendizaje de la estructura de la red no basta con seleccionar el algoritmo de búsqueda, también es fundamental definir el puntaje que se desea maximizar teniendo en cuenta la composición del conjunto de los datos con el que se desea trabajar. Este puntaje nos permitirá determinar el grado en que los datos se ajustan al modelo y por ende nos ayudará a resolver el problema de selección del mismo.

Los puntajes comúnmente utilizados en los paquetes objeto de estudio son:

- *Log-Likelihood score:* se busca el DAG y los parámetros del modelo que maximizan el logaritmo de la probabilidad de los datos. En otras palabras, dado $(G,\ \theta)$ se puede calcular la probabilidad del conjunto de datos D y por ende encontrar el grafo con el puntaje del logaritmo de la probabilidad más alto para D.

  Se busca:

$$\max_{G,\theta} L\ (\langle G,\ \theta\rangle\ :\ D) = \max_{G}[\max_{\theta} L\ (\langle G,\ \theta\rangle\ :\ D)]$$
$$= \max_{G}\ [L(\langle G,\ \theta\rangle\ :\ D)]$$

  Por ende, el puntaje se define:

$$Puntaje_L\ (G{:}D) = \ \ell(\hat{\theta}{:}D),$$

  donde, $\ell(\hat{\theta}{:}D)$ es el logaritmo de la función de probabilidad y $\hat{\theta}$ son los parámetros de máxima probabilidad para G (ver Koller y Friedman, 2009).
- *Predictive Log-Likelihood score:* se utiliza con la idea de evaluar el modelo a través de la precisión de sus predicciones.
- *Akaike Information Criterion* (AIC): se entiende como una medida de la bondad de ajuste de un modelo dado un conjunto de datos pero desestima el sobreajuste al incluir una función de penalización creciente en torno al número de parámetros.

$$Puntaje_{AIC}\ (G{:}D) = \ \ell\big(\hat{\theta}{:}D\big) - Log\ n * \ Dim[G]$$

- *Bayesian Information Criterion* (BIC): medida de bondad de ajuste estrechamente relacionada con el AIC, de hecho, utiliza exactamente la misma estructura con la diferencia de que el BIC incluye una penalización mayor por lo que tiende a seleccionar modelos un poco más sencillos.

$$Puntaje_{BIC}\ (G{:}D) = \ \ell\big(\hat{\theta}{:}D\big) - \frac{Log\ n}{2}\ Dim[G]$$

- *Bayesian Dirichlet equivalent* (BDeu): puntaje bayesiano de Dirichlet equivalente en verosimilitud. Asume uniformidad sobre las probabilidades de cada χ_i y sus padres, es decir sobre la distribución a priori debido a la falta de información previa. Produce el mismo valor para DAGs que contienen una distribución de probabilidad equivalente (mismas independencias condicionales). (Scutari, 2016).
- *K2 Score* (K2): la metodología para el cálculo de este puntaje es análoga a la de BDeu, la única diferencia radica en que K2 hace uso de parámetros a priori iguales a uno.
- *Bayesian Dirichlet sparse score* (BDs): se constituye como alternativa a BDeu al proponer un conjunto de supuestos adicionales en torno a la

distribución de probabilidad a priori y la probabilidad marginal para evitar la aparición de aristas espurias (Scutari, 2016).

- *Bayesian Dirichlet score with Jeffrey's prior* (BDJ): sigue una metodología similar a BDeu, no obstante, como probabilidad a priori utiliza el enfoque de *Jeffrey's* en lugar del supuesto de uniformidad (Suzuki, 2016).
- *Modified Bayesian Dirichlet equivalent score* (MBDe)*:* desarrollado por Cooper y Yoo (1999) como una variación simple del puntaje BDeu que permite trabajar sobre conjuntos de datos que combinen información experimental y observacional.
- *Locally averaged BDe score* (BDla): diseñado con la intención de mitigar el inconveniente relacionado con los supuestos que toma BDeu en torno a la distribución de los parámetros. El puntaje BDla, en lugar de usar la distribución global de éstos, introduce un método de marginalización local que permite modelar por separado distribuciones de parámetros realmente complejas, asemejándose así un poco más a la realidad (Cano, et al. 2013).
- *Equivalent Gaussian posterior density* (BGe): puntaje equivalente a BDeu que se implementa en datos continuos gaussianos (Kuipers, Moffa, y Heckerman, 2014).

Análogamente a las pruebas de independencia en los algoritmos basados en restricciones, el puntaje que se utilizará para determinar cuál es la estructura que mejor se ajusta a los datos debe ser seleccionado teniendo en cuenta la naturaleza de éstos y por consiguiente el tipo de red que se está construyendo. Considerado lo anterior, la tabla 2 resume qué puntaje se puede utilizar en cada escenario.

Tabla 2. Uso del puntaje según tipo de dato

Puntajes	**Red bayesiana discreta**	**Red bayesiana Gaussiana**	**Red bayesiana mixta**
	Variables categóricas/discretas		
Log-Likelihood	x	x	x
Bayesian Dirichlet equivalent (BDeu)	x		x
Akaike Information Criterion (AIC)	x	x	x
Bayesian Information Criterion (BIC)	x	x	x
Predictive Log-Likelihood	x	x	
Sparse Dirichlet posterior density (BDs)	x		
Bayesian Dirichlet score with Jeffrey's prior (BDJ)	x		
Modified Bayesian Dirichlet equivalent score (MBDe)	x		
Locally averaged BDe score (BDla)	x		
K2 score	x		
Equivalent Gaussian posterior density (BGe)		x	

Se destaca que, nuevamente, la mayoría de éstos son implementados por el paquete **bnlearn.**

2.4.1.3 Algoritmos de aprendizaje híbridos

Como su nombre permite suponer, en este tipo de algoritmos se compaginan los procedimientos utilizados por los métodos basados en restricciones y por aquellos basados en puntaje-búsqueda. El objetivo es complementar ambas técnicas para potenciar sus fortalezas y mitigar sus debilidades. Estos algoritmos trabajan al alternar dos fases; en la primera, reducen el espacio hipotético de estructuras candidatas al determinar las restricciones de independencia condicional, posteriormente toman esta menor cantidad de redes e intentan identificar o aprender el DAG que maximice el puntaje seleccionado y que a su vez satisfaga las independencias provenientes de los datos. En otras palabras, en primera instancia se utilizan métodos basados en restricciones y seguidamente se emplean métodos basados en búsqueda y puntaje.

Estos algoritmos son implementados sólo por unos cuantos paquetes de R, siendo los que se listan a continuación los más utilizados:

- *Max-Min Hill Climbing* (MMHC): este algoritmo fue desarrollado por Tsamardinos, Brown y Constantin (2006), hace uso de *Max-Min Parent and Children* (MMPC) para identificar la estructura inicial o esqueleto de la red y luego orienta las aristas por medio del método HC.
- *Hybrid HPC* (H2PC): haciendo uso del algoritmo HPC restringe el espacio de búsqueda al identificar los padres e hijos de cada variable, por ende limita el esqueleto de la red; posteriormente direcciona las aristas a través del método HC (Gasse, Aussem y Elghazel, 2014).
- *General 2-Phase Restricted Maximization* (RSMAX2): se divide en dos fases, una llamada restricción y otra maximización. En la etapa de restricción, se identifican las variables con mayor posibilidad de fungir como padres de una variable χ_i. Dado ésto se acota la búsqueda a redes donde se cumpla dicha condición. En la fase de maximización, simplemente se intenta determinar cuál es la red que maximiza el puntaje y a su vez satisface las restricciones establecidas (Friedman, Nachman y Pe'er, 1999).

De acuerdo con la literatura existente en relación con la temática de redes bayesianas, este tipo de algoritmos son generalmente más rápidos que los dos anteriores, computacionalmente hablando, al partir de un espacio de búsqueda más pequeño.

2.4.1.4 Aprendizaje de clasificadores bayesianos

En algunas ocasiones la motivación subyacente para desarrollar el procedimiento de aprendizaje de una red bayesiana consiste en solucionar un problema de clasificación supervisada, es decir, ejecutar la construcción de una red que permita predecir o determinar la categoría a la cual pertenece una variable clase (γ) dadas ciertas características representadas en un conjunto de variables predictoras (χ). En este escenario el proceso de aprendizaje es abordado de manera similar a como se efectuaría un procedimiento de *Machine Learning*, por lo que sería necesario, a grandes rasgos, entrenar un modelo,

evaluarlo, ajustarlo, volver a evaluar hasta reducir al máximo la función de pérdida y finalmente realizar las predicciones.

Los algoritmos implementados para esta labor en los paquetes objeto de análisis son:

- *Naive Bayes* (NB): se constituye como el modelo más sencillo de clasificación con redes bayesianas; considerando que la estructura es fija sólo debe ejecutarse la estimación de los parámetros. En este caso solo se requiere la distribución de probabilidad a priori de la clase y las probabilidades condicionales de cada atributo dada la clase. Se basa en el supuesto de que todos los atributos son independientes entre sí dada la clase o variable γ.
- *Tree Augmented naive Bayes* (TAN) - *Chow-Liu for ODEs* (CL – ODE): se entiende como una extensión de NB que flexibiliza el supuesto de independencia condicional en las variables predictoras al permitir la presencia de aristas o dependencia entre ellas, lo anterior se limita a que cada variable predictora tendrá como máximo dos padres: el nodo que representa a la clase y cualquier otra variable, generándose así una estructura de árbol. Friedman, Geiger y Goldszmidt (1997) propusieron una modificación al algoritmo de Chow y Lui para la estimación de un TAN, basándose en el criterio de información mutua condicionada.
- *Hill-Climbing Tree Augmented Naive Bayes* (HC-TAN): este algoritmo facilita la estimación de un TAN a través de la utilización del método de búsqueda HC. Se emplea dicho método con el objetivo de determinar la presencia de las aristas entre los nodos al intentar maximizar el puntaje establecido. En el proceso de búsqueda se van añadiendo conexiones hasta el punto en que no se genera ninguna mejora en la precisión (*accuracy*) del clasificador, la evaluación de este indicador se realiza por medio validación cruzada luego de cada posible iteración (Keogh y Pazzani, 2002).
- *Hill-Climbing Super-Parent Tree Augmented Naive Bayes* (HC-SP-TAN): versión extendida del HC-TAN donde la idea consiste en establecer un nodo como súper-padre y evaluar el efecto en el desempeño del modelo cuando se prueba cada variable disponible en dicho rol. En este sentido, el nodo súper-padre será hijo de la variable clase y padre de los demás nodos, quienes a su vez tendrán como padres a la clase y al nodo súper-padre. De esta forma, el espacio de búsqueda se reduce, por lo que se implementa el método HC para determinar la dirección de las aristas (Keogh y Pazzani, 2002).
- *Averaged one-dependence estimators* (AODE): se basa en la idea de los nodos súper-padres para construir un conjunto de redes bayesianas donde en cada una de las redes de éste funge como súper-padre una variable diferente. El clasificador se determina al promediar el resultado generado por todas las redes del conjunto (Webb, Boughton y Wang, 2005).
- *Backward Sequential Elimination and Joining* (BSEJ): este procedimiento ejecuta la selección del clasificador al considerar una estructura tipo NB completa, sobre la cual, de forma secuencial se van creando nuevos atributos que contienen la combinación de un par de variables utilizadas por el clasificador y donde adicionalmente se eliminan las variables originales consideradas por éste. Se utiliza una estimación de validación cruzada para evaluar cada estructura. Si las operaciones anteriores no

suscitan mejoras en el clasificador se selecciona la estructura actual. Cabe destacar que el presente procedimiento no asume independencia entre variables (Pazzani, 1996). La estructura final se denomina modelo semi-NB.

- *Forward Sequential Selection and Joining* (FSSJ): inicia con una estructura donde sólo se tiene a la variable clase. Secuencialmente se añaden variables en la red al considerarlas una por una y evaluar si ésta es independiente de las otras consideradas en el modelo y/o si está relacionada con las variables pertenecientes a un subconjunto de variables relacionadas (Pazzani, 1996). Es otro procedimiento para obtener un semi-NB.
- *Hill-Climbing K-Dependence Bayesian Classifier* (K-DB): en este método se le brinda la posibilidad a la red de que cada nodo tenga un máximo de K padres, exceptuando a la variable clase. La selección de los nodos padres de una variable o la identificación de la presencia de aristas entre ellos se realiza por medio del método de búsqueda HC, donde de forma iterativa se añaden aristas que derivan en mejoras de la red dado el puntaje seleccionado. El proceso continúa hasta que no se generan cambios significativos.

La ventaja de este tipo de clasificadores sobre otros existentes en la literatura de aprendizaje automático radica en la fácil interpretabilidad de los mismos, no se configuran como "cajas negras" donde es difícil, por no decir imposible, entender cómo funciona el modelo.

2.4.1.5 Aprendizaje en presencia de datos incompletos

Cabe destacar que la mayoría de los algoritmos presentados anteriormente asumen completitud de los datos, por lo que si se desarrolla el procedimiento de aprendizaje de la red en presencia de valores faltantes la solución puede resultar inestable o incluso el algoritmo puede llegar a no funcionar. Por lo general, es necesario efectuar un proceso previo donde se imputen los datos o eliminen los casos perdidos. Varios paquetes de R disponen de métodos de imputación pero se deben desarrollar con anterioridad a la ejecución de los algoritmos de aprendizaje.

No obstante, en la literatura es posible encontrar unos pocos algoritmos que permiten el aprendizaje de la red a partir de un conjunto de datos con valores perdidos, sin la necesidad de ejecutar ningún tratamiento o procesamiento sobre éstos. El descrito a continuación es el de mayor uso.

- *Structural EM* (SEM): propuesto por Friedman (1998) siguiendo la idea del algoritmo *Expectation Maximization,* el cual fue desarrollado por Dempster, Laird y Rubin (1977) para el aprendizaje de los parámetros de la red en presencia de valores faltantes. Por lo tanto, al igual que el método anterior, SEM consta de dos etapas; la primera, denominada paso E, emplea para la imputación de los datos el valor esperado de éstos al derivarlo de la red identificada. La segunda etapa, llamada paso M, desarrolla el aprendizaje de la red utilizando la información del paso E para asumir que la información está completa. En esencia, el algoritmo ejecuta de forma iterativa un proceso de imputación de datos y aprendizaje de estructura hasta la convergencia.

Además del algoritmo SEM y la posibilidad de la imputación de datos, algunos paquetes le brindan la opción al usuario de trabajar con el procedimiento de "análisis de casos disponibles" (*available cases analysis*), el cual como su nombre lo indica incluye para el aprendizaje de la red sólo a aquellos registros que no tienen ningún dato incompleto y deja por fuera todo aquel registro con valores perdidos, disminuyendo así el número de filas del conjunto de datos.

2.4.2 Aprendizaje de los parámetros

El principal interés de esta etapa consiste en estimar los parámetros de las distribuciones de probabilidad local, las cuales a su vez son el resultado del procedimiento de factorizar la distribución global que representa la probabilidad conjunta de las variables (nodos) dada la estructura identificada en la etapa anterior. En este sentido, estos parámetros determinan el tamaño del efecto de dependencia entre los nodos padres e hijos.

Si bien esta tarea puede realizarse vía el aprendizaje de los parámetros de la distribución global de probabilidad, utilizar las distribuciones locales implican a un menor número de variables y el cálculo de menos parámetros ya que se realizaría una estimación a la vez (por cada variable), evitando así problemas con la dimensionalidad de los datos.

Existen en la literatura diferentes alternativas para desarrollar la labor de aprendizaje de los parámetros, siendo las más comunes:

- *Maximum Likelihood Estimation* (MLE): la idea detrás de este método consiste en encontrar los parámetros que maximicen la probabilidad de los datos.

  La función *Likelihood* se representa así:

$$P(D|\theta) = \prod_{i=1}^{N} P\ (\mathrm{X} = \chi_i|\theta) = \prod_{k=1}^{r} \theta_k^{N_k}$$

  donde, N es el número de variables y N_k es el número de casos del conjunto de datos para los cuales $X = k$.

- *Bayesian Estimation* (BE): como su nombre indica, se basa en la estadística bayesiana al tratar a los parámetros como variables aleatorias, lo cual a su vez genera incertidumbre que debe ser representada por medio de distribuciones de probabilidad. En este sentido, el problema de aprendizaje se transforma en uno de inferencia, donde el objetivo es estimar la probabilidad a posteriori de los parámetros considerando la función de probabilidad y la distribución a priori de éstos.

  La estimación bayesiana se representa a través de la media de la probabilidad a posteriori, así:

$$\theta_k^* = \frac{N_k + a_k}{N + \sum_{i=1}^{r} a_i}$$

- *Maximum a Posteriori Estimation* (MAP): se utiliza cuando la estimación de los parámetros por medio de una solución bayesiana es impráctica. En esencia, MAP busca los parámetros que maximizan la distribución de probabilidad posterior.
- *Expectation Maximization* (EM): procedimiento utilizado para estimar los parámetros en presencia de valores pérdidos. Consiste en un proceso iterativo donde el algoritmo inicia con la estimación de los valores perdidos a través de un conjunto de parámetros establecidos (paso E) y luego utiliza el nuevo conjunto de datos completos para reestimar los parámetros (paso M), (Dempster, Laird, y Rubin, 1977).

2.5 Inferencia

Una de las finalidades de construir una red bayesiana consiste en la posibilidad de realizar razonamiento probabilístico (inferencia) sobre ésta , es decir, efectuar consultas sobre la red encaminadas a identificar variaciones en la distribución de probabilidad dada nueva información o evidencia en relación con las variables del modelo. En muchas ocasiones, lo anterior se lleva a cabo a través de un mecanismo comúnmente conocido como propagación de evidencias o creencias (*belief propagation*), el cual realiza la actualización de la probabilidad de las variables en función de las nuevas observaciones.

En este sentido, la evidencia o nueva información se denota como un subconjunto de variables cuyos valores son conocidos u observados; en la literatura generalmente se dividen en dos tipos:

- *Hard evidence:* consiste en la especificación de nuevos valores sobre una o más variables de la red. Podría ser un solo elemento del conjunto de variables (χ_i) o el conjunto completo (χ).
- *Soft evidence:* consiste en la especificación de una nueva distribución de probabilidad sobre una o más variables de la red.

Por otro lado, el tipo de consultas que se pueden ejecutar sobre la red están asociadas a la distribución de probabilidad conjunta y por ende a la distribución de probabilidad condicional de las variables gracias a la propiedad de descomposición analizada anteriormente. En esencia, se tiene que estas consultas o preguntas se agrupan en dos clases:

- *Conditional Probability Queries* (CPQ): la tarea consiste en estimar la probabilidad condicional de la variable χ_i dado el valor observado o la evidencia $E = e$, es decir, $P(X_i|E = e)$.
- *Maximum a Posteriori Queries* (MAP): la idea radica en encontrar una asignación y que maximice la probabilidad posterior de las variables contenidas en el subconjunto $\boldsymbol{Y}$ dada la evidencia $E = e$. Es decir, busca encontrar una asignación que tenga la probabilidad más alta. Lo anterior se resumiría como $argmax_y\, P(Y = y|E = e)$.

De igual forma, es importante resaltar que existen dos tipos de algoritmos para la ejecución del mecanismo de propagación de evidencia, a saber:

- Algoritmos de inferencia exacta: como su nombre indica, realiza el cálculo de las probabilidades de forma exacta, es decir, por medio de la combinación de repetidas aplicaciones del teorema de Bayes junto con

computaciones locales. Estos algoritmos son comúnmente utilizados en estructuras pequeñas debido a que su costo computacional crece en función del número de variables (Scutari y Denis, 2014).
- Algoritmos de inferencia aproximada: surgieron como alternativa cuando los algoritmos de inferencia exacta no podían ser aplicados. Se basan en la utilización de técnicas de simulación para realizar un muestreo de la distribución global de las variables con el objetivo de estimar valores aproximados de la probabilidad.

Estos algoritmos a su vez se agrupan en diferentes métodos o técnicas que se utilizan para propagar la nueva información y realizar el cálculo o estimación de la probabilidad. Los implementados en los paquetes de R objeto de análisis son:

- LS *algorithm - Message Passing*: propuesto por Lauritzen y Spiegelhalter (1988), es un método exacto que se basa en la construcción de grafos no dirigidos agrupados, en los cuales se forman grupos de nodos de acuerdo con el orden de eliminación de éstos y/o teniendo en cuenta los nodos cercanos con la intención de facilitar la transmisión del mensaje entre grupos vecinos con información sólo de la variable de interés o que tengan en común, sin duplicarla.
- *Logic Sampling*: la idea detrás de algoritmo es generar un muestreo de los valores contenidos en el modelo de probabilidad y luego realizar una estimación de la probabilidad deseada. Específicamente, *Logic Sampling* selecciona una muestra al simular el valor de cada variable en el modelo dado un orden topológico (padres van antes que hijos) donde al momento en que la variable χ_i es simulada su nodo padre ya ha sido considerado (Henrion, 1986).
- *Likelihood Weighting*: sigue un enfoque similar a *Logic Sampling* con la diferencia de que utiliza un ponderador para cada prueba de la simulación, el cual por lo general son las probabilidades locales de la red (Fung y Chang, 1989).

Más adelante se detallará cuáles son los paquetes de R desarrollados con la finalidad de ejecutar el proceso de inferencia y sus respectivas funcionalidades, adelantando aquí que todos ellos se centran en resolver consultas del tipo *Conditional Probability Queries* (CPQ).

2.6 Redes bayesianas causales

Si bien el análisis de redes bayesianas causales se encuentra por fuera del alcance del presente trabajo, vale la pena realizar una breve introducción a las mismas ya que varios de los paquetes objeto de estudio implementan algoritmos o métodos que permiten realizar el aprendizaje o manipulación de éstas.

Hasta el momento se ha considerado que las redes bayesianas son construidas a partir de modelos de probabilidad, por lo que las aristas que conectan a dos nodos no tienen por qué tener un significado más allá de la correlación o relación de dependencia entre éstos. No obstante, en la literatura académica sobre esta temática es común encontrar que dicha relación de dependencia también podría ser interpretada como una relación de causa-efecto. Este tipo de redes bayesianas son conocidas como modelos de causalidad y se basan en el hecho de que las aristas codifican dicho tipo de relaciones, además de que

permiten las inclusión de intervenciones en el modelo al distinguir entre variables observacionales (sólo se observa su comportamiento) y experimentales (se manipula o interviene asignándole un valor).

Un claro ejemplo de la importancia que reviste la interpretación de las aristas en los modelos causales se puede observar en el arco que conecta $X \rightarrow Y$, ésta sugiere que X causa a Y pero si se redirecciona la arista el significado es totalmente contrario, es decir, $Y \rightarrow X$, Y causa a X, configurando así dos redes bayesianas totalmente diferentes. Esta situación no se observa en los modelos probabilísticos toda vez que $X \rightarrow Y$ y $Y \rightarrow X$ serían consideradas como redes equivalentes.

Por otro lado, el hecho de analizar las relaciones de causalidad entre variables facilita el estudio de situaciones donde se presentan intervenciones que modifican el curso natural de los eventos. En este sentido, los modelos probabilísticos se limitan a observar los valores que toma una variable, mientras que los modelos causales se enfocan en las situaciones donde se pueden tomar acciones para manipular dichos valores. Específicamente, este tipo de modelos permiten encontrar la distribución sobre las variables Y, al establecer para el conjunto Z un valor de z y observar los valores x para las variables en X, a esto se le conoce como *Intervention Queries* (Koller y Friedman, 2009).

Por último, se subraya que para el aprendizaje de este tipo de redes es importante que no existan variables latentes u ocultas (variables no observadas que influencian a otras que sí están presentes) en el modelo, esto debido a que introducirían relaciones espurias entre las variables observadas y sesgarían el modelo de causalidad.

3 Paquetes disponibles en R

Actualmente en el *task view* de modelos gráficos ubicado en el repositorio CRAN de R se encuentran disponibles más de 30 paquetes destinados a tareas como la representación, manipulación o aprendizaje, entre otras, tanto de redes bayesianas como de redes de Markov (no dirigidas). En este sentido, para el presente trabajo se acotó el análisis de los paquetes disponibles a aquellos enfocados a la modelización de redes bayesianas. Específicamente se estudiaron los más utilizados o populares de acuerdo con la literatura para tareas como el aprendizaje de la estructura, el aprendizaje de los parámetros y la inferencia probabilística.

En total se analizaron nueve paquetes con la intención de identificar su aplicabilidad, factores diferenciadores y limitaciones sobre redes bayesianas de diferentes tamaños y tipos de variables. Asimismo, se construyó un inventario de funcionalidades, algoritmos y métodos implementados por cada uno de los paquetes, esto con el fin de crear una guía que pueda ser útil para investigadores interesados y usuarios no expertos al momento decidir sobre con qué paquete trabajar dada la actividad y características del conjunto de datos que tengan a su disposición.

La tabla 3 muestra los paquetes estudiados y su respectiva versión, así como otra información de interés (información completa, ver anexo tabla A.1). En el presente documento, en algunas ocasiones se muestran tablas resúmenes debido a la extensión de las mismas. La información completa puede ser

consultada en los anexos o en el repositorio GitHub del autor donde también podrán ser editadas, corregidas y/o actualizadas por cualquier usuario interesado (https://github.com/LuisEduardoAngulo/BN-software-review-in-R).

Tabla 3. Paquetes objeto de estudio

Paquete	Versión analizada	Año publicación	Año versión actual	Descarga histórica*	Descarga semestral**
catnet	1.15.7	2010	2020	49.603	6.858
sparsebn	0.1.0	2016	2019	12.116	3.179
pcalg	2.6.10	2006	2020	89.004	13.598
deal	1.2.39	2002	2018	53.857	6.358
bnclassify	0.4.5	2015	2020	27.325	5.528
bnstruct	1.0.6	2016	2019	27.226	4.717
bnlearn	4.5	2007	2019	259.676	40.783
gRain	1.3.4	2008	2020	114.351	22.196
BayesNetBP	1.4.0	2017	2018	13.833	3.106

Las descargadas fueron obtenidas usando el paquete **cranlogs** de R, el cual se configura como una API al repositorio "CRAN" específicamente a la base de datos de descargas de paquetes desde RStudio (IDE de R).
*desde octubre de 2012 hasta marzo de 2020
**desde octubre de 2019 hasta marzo de 2020

3.1 Descripción de los paquetes

A continuación se presenta una breve descripción de las características propias y principales funcionalidades implementadas por los paquetes objeto de estudio.

catnet

El objetivo principal del paquete **catnet** es el de brindarle al usuario herramientas para el aprendizaje de la estructura de la red haciendo uso de algoritmos de búsqueda-puntaje. En este sentido dispone de dos métodos para desarrollar dicha tarea; en el primero se requiere conocimiento previo acerca de la red, específicamente resulta necesario conocer el orden de los nodos y el nivel de complejidad de ésta. Teniendo estos parámetros en cuenta se implementa un algoritmo de búsqueda exhaustiva, vía programación dinámica, que explora todas las posibles redes y devuelve un listado con las estructuras que se ajustan a los datos de acuerdo con el criterio de máxima verosimilitud. En el segundo escenario no se precisa conocimiento previo y el aprendizaje se desarrolla a partir de un método de búsqueda estocástica como el algoritmo SA. En ambos casos el resultado es un listado con las mejores redes encontradas, por lo que **catnet** facilita diferentes criterios para la selección de la estructura final, como lo son los puntajes de AIC y BIC, además del nivel de complejidad o de forma manual, de acuerdo con las preferencias del usuario. Estas funcionalidades dispuestas para el proceso de selección de la red se configuran como el factor diferenciador en comparación con otros paquetes. Cabe destacar que **catnet** solo puede implementarse en redes bayesianas cuyos nodos sean de tipo discreto y que además posibilita trabajar con valores faltantes a través de la imputación de datos (Balov y Salzman, 2020).

sparsebn

El paquete **sparsebn** se especializa en el aprendizaje de la estructura y los parámetros de modelos gráficos con grandes dimensiones; especialmente redes bayesianas dispersas (compuesta por una menor cantidad de aristas en comparación con la cantidad de nodos) en las cuales a partir de su base de datos se puede observar una mayor cantidad de variables en comparación con la cantidad de registros. A diferencia de otros paquetes, **sparsebn** fue específicamente diseñado para trabajar sobre grandes volúmenes de datos (con cientos/miles de variables), bien sean experimentales y/u observacionales y de tipo discreto o continuo. Para el aprendizaje de la estructura de la red se implementa un algoritmo de búsqueda-puntaje basado en una estimación de máxima verosimilitud regularizada, en la cual se emplea un regularizador para evitar estructuras demasiado densas o con muchas aristas. El valor añadido de **sparsebn** consiste en que para la creación y funcionamiento de sus algoritmos se incorporaron algunos trucos provenientes de la literatura con el fin de aumentar la velocidad del proceso de aprendizaje sobre redes con grandes cantidades de variables o, lo que es lo mismo, disminuir el tiempo de éste; en detalle, las ideas incorporadas fueron: *warm starts, active set iterations, block updates* y *sparse data structures*, los cuales permiten: usar la estimación de una red (previa) como inicio de la siguiente iteración, ajustar los parámetros distintos de cero resolviendo a lo sumo *p* problemas de regresión penalizada, ajustar los parámetros en bloques en lugar de hacerlo individualmente y almacenar internamente haciendo uso de estructuras de datos dispersos para representar DAGs, respectivamente (Aragam, Gu y Zhou, 2019).

pcalg

El análisis y la medición de las relaciones de causalidad entre variables se configuran como los principales intereses del paquete **pcalg**. Si bien éste puede ser utilizado para el aprendizaje de la estructura de la red, su factor diferenciador radica en la posibilidad de estimar el tamaño de la causalidad entre nodos una vez se tiene la estructura, especialmente después de una intervención. Con el fin de cuantificar dichos efectos sobre redes provenientes de información observacional, se asume la no presencia de variables latentes o de variables de selección y considerando esto el paquete implementa el método IDA (*Intervention calculus when the DAG is Absent*), el cual fue desarrollado por Maathuis, Kalisch y Bühlmann (2009) combinando el algoritmo PC junto con el criterio *Pearl´s backdoor* (Pearl, 1993). En cuanto al aprendizaje de la estructura, dispone de algoritmos como PC, FCI, RFCI y GES para dicha tarea sobre datos observacionales, mientras que para datos experimentales utiliza GIES. De igual forma que con el método IDA, para la utilización de uno u otro algoritmo se debe cumplir con ciertos supuestos, por ejemplo: presencia o no de variables ocultas, consistencia en entornos de grandes dimensiones (el número de variables crece con el tamaño de la muestra) y tipo de dato (experimental/observacional), entre otros (Kalisch, et al. 2014).

deal

Este paquete incluye herramientas orientadas a facilitar el aprendizaje de la estructura y de los parámetros de redes conformadas por nodos mixtos (discretos y continuos). El proceso de aprendizaje de la estructura es desarrollado por medio del método de búsqueda-puntaje, en el cual primero se calcula el puntaje de la red haciendo uso de la probabilidad relativa con el fin

de determinar el grado en que un grafo representa las independencias condicionales entre variables. Posteriormente, evalúa los grafos con los mejores valores de puntaje (BDeu, en este caso) utilizando una estrategia de *greedy search* o de GSRR para encontrar las representaciones sobresalientes y finalmente las compara a través del factor Bayes (alternativa bayesiana a la prueba de hipótesis clásica). Por otro lado, el aprendizaje de los parámetros se ejecuta teniendo en cuenta el enfoque bayesiano: como *local prior parameter* para nodos discretos se usa la distribución de Dirichlet, mientras que para nodos continuos se emplea la distribución gamma-inversa Gaussiana. Cabe destacar que **deal** incorpora herramientas para conectar sus resultados con el programa Hugin y que adicionalmente forma parte del proyecto gR (Boettcher y Dethlefsen, 2003).

bnclassify

Si bien **bnclassify** puede ser incluido dentro el listado de paquetes de R enfocados en el proceso de aprendizaje de la red, su auténtica finalidad es, a partir de los datos, construir una red bayesiana que pueda ser utilizada para una tarea de clasificación, es decir, para predecir la clase de una variable discreta dado un conjunto de variables predictoras (también discretas). Dentro de las funcionalidades dispuestas para el usuario, **bnclassify** permite lo siguiente: el aprendizaje de la estructura red y sus respectivos parámetros, analizar o evaluar el modelo construido y/o implementar éste para predecir la clase de nuevas observaciones. El procedimiento de aprendizaje de la estructura de la red de clasificación dispone de variaciones del método *greedy hill-climbing search* como: HC-TAN, HC-SP-TAN, BSEJ, FSSJ, K-DB; o la adaptación del algoritmo de Chow-Liu para el TAN, entre otros. Por su parte, para el aprendizaje de los parámetros hace uso de estimadores bayesianos o máxima verosimilitud para datos completos y del procedimiento de análisis de casos disponibles para datos incompletos, entre otros. **bnclassify** incorpora herramientas para conectar sus resultados con otros paquetes del entorno de R, a saber: **bnlearn, gRain** y **graph**. De igual forma puede importar funciones de los paquetes **mlr** y **caret** para complementar el proceso de aprendizaje automático (Mihaljevic, Bielza y Larrañaga, 2018).

bnstruct

En esencia, el paquete **bnstruct** ha sido desarrollado para implementar algoritmos de aprendizaje, tanto de la estructura como de los parámetros de la red en presencia de valores faltantes, lo cual se constituye como su valor añadido en comparación con otros paquetes. Para realizar este procedimiento dispone de dos alternativas; en primera instancia se podrían imputar los datos haciendo uso del algoritmo de los K vecinos más cercanos incorporado en el paquete por medio de una función propia o se podría utilizar el método SEM. Adicionalmente, incluye una funcionalidad para realizar el aprendizaje de la estructura usando la técnica de re-muestreo *Bootstrap* (Efron, 1979) con la finalidad de estimar un nivel de confianza asociado a la presencia de las aristas en el grafo original. Cabe destacar que el paquete también brinda herramientas para hacer inferencia sobre la red a través del mecanismo de propagación de evidencias o creencias. Para aprender la estructura dispone de algoritmos basados en restricciones y algoritmos basados en búsqueda-puntaje, ellos son: SM, MMPC, HC y MMHC. Los puntajes habilitados para calificar los modelos durante el proceso de búsqueda son: BDeu, AIC y BIC. En relación con el

aprendizaje de parámetros, la estimación se realiza a partir del método *Maximunm-A-Posteriori* (MAP), (Sambo y Franzini, 2019).

bnlearn

Por medio de la utilización de este paquete el usuario dispone de una gran variedad de algoritmos enfocados al aprendizaje de la estructura de la red, sea ésta compuesta por nodos discretos, continuos o mixtos. De igual forma, **bnlearn** implementa herramientas para el aprendizaje de los parámetros y la realización de inferencia aproximada sobre ésta. El paquete le brinda la posibilidad al usuario de combinar el aprendizaje ejecutado a partir de datos y el conocimiento previo/experto a través de funciones que facilitan la modificación de la estructura aprendida o la incorporación de información previa en torno al modelo o los datos. La característica principal de **bnlearn** consiste en que permite realizar todas las actividades relacionadas con la modelización de redes bayesianas, a saber: preprocesamiento, aprendizaje de la estructura, aprendizaje de los parámetros, inferencia, evaluación y construcción de clasificadores, entre otros; poniendo a disposición del usuario diferentes opciones de métodos y/o una extensa variedad de algoritmos para cada una de estas tareas. Por ejemplo, para aprender la estructura el paquete despliega distintos tipos de algoritmos, entre ellos: basados en restricciones, basados en búsqueda-puntaje, híbridos, *local discovery* (algoritmos que realizan el aprendizaje de un grafo no dirigido conocido como el esqueleto de la red) e incluso para estimar clasificadores de redes bayesianas. Éstos son: PC, GS, IAMB, Fast-IAMB, Inter-IAMB, IAMB-FDR, MMPC, SI-HITON-PC, HPC, HC, Tabu, MMHC, H2PC, RSMAX2, *Chow-Liu* (Chow y Liu, 1968), *ARACNE* (Margolin, et al. 2006), NB y TAN. Por otra parte, los puntajes habilitados para calificar los modelos durante el proceso de búsqueda son: *Log-likelihood,* AIC, BIC, *predictive log-likelihood,* BDeu, BDs, BDJ, BDla, MBDe, BGe y K2. Del mismo modo, habilita diferentes pruebas para analizar las independencias condicionales cuando se hace uso de métodos de aprendizaje basado en restricciones, siendo algunas de ellas: información mutua, χ^2 de Pearson, Z de Fisher y correlaciones lineales, entre otros. Para el aprendizaje de los parámetros utiliza máxima verosimilitud o estimadores bayesianos. Por último, para los procesos de inferencia sobre la red dispone de los métodos de *Logic Sampling* y *Likelihood Weighting*, ambos enfocados en la inferencia aproximada. Se subraya el hecho de que el paquete permite trabajar con datos incompletos o valores faltantes por medio de la imputación de los mismos o a través del algoritmo SEM (Scutari, 2010).

gRain

Desarrollado con el objetivo de permitir al usuario la ejecución del proceso de inferencia probabilística sobre redes bayesianas discretas por medio del algoritmo de propagación de Lauritzen y Spiegelhalter (LS). En **gRain** el proceso de inferencia inicia con la especificación de la red. Ésta puede ser por medio de la definición manual de la tabla de probabilidades condicionales o a partir de la importación de un DAG y sus respectivos datos. Luego se introduce la evidencia observada para que la misma se propague por toda la estructura y así poder ejecutar las consultas de interés, las cuales podrían estar relacionadas con la distribución de probabilidad marginal, conjunta o condicional de las variables. El paquete cuenta con un conjunto de herramientas adicionales para utilizar sobre las redes, por ejemplo, podría realizarse la predicción de una variable

respuesta (clase) teniendo en consideración un grupo de variables explicativas o trabajar con archivos del software Hugin, entre otros (Højsgaard, 2012).

BayesNetBP

BayesNetBP es un paquete que ha sido creado con el objetivo de permitir la realización de razonamiento probabilístico sobre una red bayesiana utilizando el método de propagación de evidencias o creencias. Puntualmente, por medio de un algoritmo de inferencia exacta, en este caso LS - *Message Passing*. Dado lo anterior, admite consultas sobre la distribución de probabilidad marginal, conjunta o condicional derivadas de la inclusión de nueva evidencia. Según los autores, hasta el momento, es el único paquete gratuito destinado a la inferencia que permite trabajar sobre nodos discretos, continuos o mixtos; además de ser el primero en trabajar sobre nodos continuos o mixtos sin utilizar dependencias de software comercial. Asimismo, cuenta con soporte para realizar la visualización de la red y los resultados del proceso de inferencia a través de Shiny. Es compatible con otros paquetes enfocados al aprendizaje de la estructura como: **catnet, bnlearn, deal, pcalg** y **RHugin**, entre otros; o bien, la estructura de la red puede ser definida manualmente (Yu, Moharil y Blair, 2019).

Finalmente, la tabla 4 muestra un resumen donde se puede observar en paralelo los paquetes objeto de análisis y sus principales funcionalidades asociadas con la modelización de redes bayesianas. A partir de ésta, se identifican las actividades o tareas que pueden ser desarrolladas con cada paquete y los tipos de dato que soporta. Cabe la pena destacar que el paquete **bnlearn** es el único que dispone de herramientas para acometer todas las tareas relacionadas con redes bayesianas y que también permite trabajar con cualquier tipo de información/dato.

Tabla 4. Funcionalidades de los paquetes objeto de estudio

Funcionalidad	**Paquete**								
	catnet	**sparsebn**	**pcalg**	**deal**	**bnclassify**	**bnstruct**	**bnlearn**	**gRain**	**BayesNetBP**
Aprendizaje estructura	sí	sí	sí	sí	sí	sí	sí	no	no
Aprendizaje parámetros	sí	sí	sí	sí	sí	sí	sí	no	no
Aprendizaje de clasificadores de BN	no	no	no	no	sí	no	sí	no	no
Inferencia	no	no	no	no	no	sí	sí	sí	sí
Datos discretos	sí	sí	sí	sí	sí	sí	sí	sí	sí
Datos continuos	no	sí	sí	sí	no	sí	sí	no	sí
Datos mixtos	no	no	no	sí	no	no	sí	no	sí
Datos incompletos	sí	no	no	no	sí	sí	sí	no	no

3.2 Herramientas disponibles para el aprendizaje

Del total de paquetes analizados se evidenció que siete de éstos implementan metodologías o algoritmos para el aprendizaje de la estructura de la red y de sus respectivos parámetros. Ellos son: **catnet**, **sparsebn**, **pcalg**, **deal**,

bnclassify, **bnstruct** y **bnlearn**. Cada uno cuenta con su peculiaridad o rasgo distintivo y en muy pocas ocasiones la aplicabilidad de alguno se solapa con la de otro, es decir, cada paquete ha sido desarrollado con una finalidad específica o para abordar un escenario dado. En cuanto a ésto, se entrará en detalles en el siguiente capítulo.

Resulta interesante el hecho que todos los paquetes disponen de por lo menos un algoritmo basado en búsqueda y puntaje para ejecutar el aprendizaje de la estructura de la red, mientras que solo tres implementan algoritmos basados en restricciones, a saber: **pcalg**, **bnstruct** y **bnlearn**. Asimismo, los dos últimos se constituyen como los únicos paquetes que disponen de algoritmos híbridos para desarrollar dicha actividad.

Por otro lado, se tiene el caso del paquete **bnclassify** el cual busca aprender la estructura y parámetros de la red con la finalidad de hacer clasificaciones de nuevas instancias. Ningún otro paquete tiene este mismo propósito a excepción de **bnlearn** que habilita un par de funciones para esta tarea.

La tabla 5 especifica cuáles son los métodos implementados por cada paquete para el aprendizaje de la estructura de la red, así como los algoritmos, pruebas de dependencia y puntajes que ponen a disposición del usuario. De acuerdo con esto, **bnlearn** se constituye como la herramienta que brinda la mayor cantidad de alternativas en cada uno de los aspectos mencionados. Los demás paquetes ofrecen las soluciones clásicas correspondientes a los problemas o escenarios que abordan. Por ejemplo, **pcalg**, cuyos autores afirman que puede ser utilizado para el aprendizaje de estructuras causales, implementa algoritmos desarrollados para dicha labor que permiten trabajar con datos observacionales y/o experimentales.

Tabla 5. Métodos, algoritmos, puntajes, pruebas de independencia y funciones en R para el aprendizaje de la estructura de la red habilitados en los paquetes objeto de análisis

Paquetes	Método	Algoritmo	Puntaje	Prueba de independencia	Función en R - Aprendizaje o selección de DAG
catnet	Búsqueda y puntaje	SearchOrder SA	AIC BIC	-	cnSearchOrder() – busca todas las redes posibles dado el orden de los nodos cnSearchSA() – busca las redes posibles usando SA cnFind() – encuentra la mejor red dada la complejidad cnFindBIC () – encuentra la mejor red usando BIC cnFindAIC ()-encuentra la mejor red usando AIC
sparsebn	Búsqueda y puntaje	*Regularized MLE*	*Log-likelihood*	-	estimate.dag() - estima los posibles DAGs (máx. 20) select() – selecciona la red
pcalg	Restricciones Búsqueda y puntaje	PC FCI RFCI FCIPLUS GES GIES SM	AIC BIC	Información mutua Transformación de la Z de Fisher	pc() – implementación algoritmo PC fci() - implementación algoritmo FCI rfci() - implementación algoritmo RFCI fciplus() – implementación algoritmo FCIPLUS ges () – implementación algoritmo GES gies () – implementación algoritmo GIES simy() – implementación algoritmo SM
deal	Búsqueda y puntaje	*Greedy search* GSRR	BDeu	-	networkfamily() – realiza el aprendizaje de todas las redes posibles autosearch() – implementación de *greedy search* heuristic() - implementación de *greedy search with randon restarts*
bnclassify	Búsqueda y puntaje	NB CL-ODE HC-TAN HC-SP-TAN BSEJ FSSJ K-DB AODE	AIC BIC *Log-likelihood*	-	nb() – aprende una estructura NB tan_hc() – aprende una estructura HC-TAN tan_hcsp() – implementación de HC-SP-TAN bsej() – aprende una estructura usando BSEJ fssj()-aprende una estructura usando FSSJ kdb() – aprende un K-DB tan_cl() – implementación de CL-ODE aode() – aprende un AODE

bnstruct	Restricciones Búsqueda y puntaje	MMPC *Greedy search* - HC SM MMHC SEM	AIC BIC BDeu	NRI	A diferencia de los demás paquetes, en **bnstruct** cada algoritmo de aprendizaje no tiene su propia función. Este paquete dispone de una función única donde se debe especificar el algoritmo que se desea utilizar. Dicha función está compuesta de varios argumentos, no obstante, para definir el método de aprendizaje se debe especificar el argumento “algo”. learn.network() – realiza el aprendizaje de la estructura. Para definir con qué algoritmo trabajar se debe especificar el argumento “algo”. Ejemplo: SM_asia.bnstruct <- learn.network(asia_bnstruct, asia_bnstructObj, algo = "sm", scoring.func = "BDeu")

bnlearn	Restricciones Búsqueda y puntaje	PC GS IAMB Fast-IAMB Inter-IAMB IAMB-FDR MMPC SI-HITON-PC HPC *Greedy search* - HC Tabu MMHC H2PC RSMAX2 Chow-Liu ARACNE NB TAN SEM	*Log-likelihood* AIC BIC pred. loglikelihood BDe BDs BDJ BDla MBDe K2	Información mutua Correlación de Pearson para tablas de contingencia Shrinkage estimator para información mutua Jonckheere-Terpstra Correlación lineal de Pearson Transformación de la Z de Fisher	pc.stable() – implementación algoritmo PC gs() – implementación algoritmo GS iamb() – implementación algoritmo IAMB fast.iamb() – implementación algoritmo fast-IAMB inter.iamb() -implementación algoritmo inter-IAMB iamb.fdr() – implementación algoritmo IAMB-FDR mmpc() -implementación algoritmo MMPC si.hiton.pc() – implementación algor. SI-HITON-PC hpc() – implementación algoritmo HPC hc() – implementación algoritmo HC tabu() – implementación algoritmo Tabu mmhc() – implementación algoritmo MMHC h2pc() – implementación algoritmo H2PC rsmax2() – implementación algoritmo RSMAX2 chow.liu() – implementación algoritmo Chow-Liu aracne() – implementación ARACNE naive.bayes() - implementación NB tree.bayes() – implementación TAN structural.em() – implementación SEM

Entrando en detalle, la tabla 6 facilita la identificación y cotejo de los algoritmos implementados por paquete. Como se mencionó anteriormente, la aplicabilidad de los mismos no se solapa toda vez que ningún algoritmo se encuentra disponible en más de dos herramientas. Dado ésto, es posible aseverar que los más comunes o utilizados son: SM, *Greedy search*-HC, GSRR, PC, MMPC, MMHC, SEM, NB y TAN; CL – ODE. Los paquetes **bnclassify**, **pcalg** y **bnstruct**, en su orden, le siguen a **bnlearn** en torno al número de alternativas que proveen para el aprendizaje de la estructura.

Tabla 6. Algoritmos habilitados en cada paquete objeto de análisis

Método	**Algoritmos**	**Paquetes**						
		catnet	**deal**	**sparsebn**	**pcalg**	**bnstruct**	**bnlearn**	**bnclassify**
Búsqueda y puntaje	SearchOrder	x						
	SA	x						
	Regularized MLE			x				
	GES				x			
	GIES				x			
	SM				x	x		
	Greedy search - HC		x			x		
	GSRR - HC		x				x	
	Tabu						x	
Restricciones	PC				x		x	
	FCI				x			
	RFCI				x			
	FCIPLUS				x			
	GS						x	
	IAMB						x	
	Fast-IAMB						x	
	Inter-IAMB						x	
	IAMB-FDR						x	
	MMPC					x	x	
	SI-HITON-PC						x	
	HPC						x	
Híbridos	MMHC					x	x	
	H2PC						x	
	RSMAX2						x	
Otros	SEM					x	x	
Clasificadores	NB						x	x
	TAN; CL – ODE						x	x
	HC – TAN							x
	HC – SP – TAN							x
	BSEJ							x
	FSSJ							x
	K-DB							x
	AODE							x

En relación con las pruebas de independencia dispuestas por los paquetes que implementan algoritmos basados en restricciones, se observa que las más comunes son: la información mutua, la cual cabe recordar que puede ser utilizada para cualquier tipo de conjunto de datos, sean éstos continuos, discretos o mixtos; y la transformación de la Z de Fisher, la misma sólo puede

ser usada con datos continuos. El paquete **bnlearn** es el único que dispone de más de dos opciones como se exhibe en la tabla 7.

Llama la atención el caso del paquete **bnstruct** debido a que en comparación con **pcalg** y **bnlearn** habilita sólo un algoritmo basado en restricciones, a saber: MMPC. Éste permite el aprendizaje de un DAG parcialmente completo (CPDAG – por sus siglas en inglés) pero no se documenta qué prueba de independencia se ejecuta para establecer las relaciones de dependencia entre variables o qué metodología se sigue para identificar la presencia de las aristas entre los nodos, razón por la que su columna en la tabla 7 se muestra vacía.

Tabla 7. Pruebas de independencia disponibles en los paquetes que implementan algoritmos basados en restricciones

Método	**Pruebas de independencia**	**Paquetes**		
		pcalg	**bnstruct**	**bnlearn**
Restricciones	Información mutua	x		x
	Correlación de Pearson para tablas de contingencia			x
	Shrinkage estimator para información mutua			x
	Jonckheere-Terpstra			x
	Correlación lineal de Pearson			x
	Transformación de la Z de Fisher	x		x

Por otro lado, al analizar cuáles son los puntajes que pone a disposición cada paquete en el marco del proceso de selección o identificación de la red, se evidenció que la mayoría de estos habilitan más de uno para que el usuario sea quien elija con cuál desea trabajar, exceptuando **sparsebn** donde solo se implementa *log-likelihood*. Los más comunes o utilizados por la mayor parte de paquetes son: AIC, BIC, *log-likelihood* y BDeu. Nuevamente, **bnlearn** se configura como el paquete que ofrece la mayor cantidad de opciones, siendo gran parte de estas variantes o modificaciones del puntaje Dirichlet Bayesiano (tabla 8).

Tabla 8. Puntajes disponibles en los paquetes que implementan algoritmos basados en búsqueda y puntaje

Métodos	**Puntajes**	**Paquetes**						
		catnet	**deal**	**sparsebn**	**pcalg**	**bnstruct**	**bnlearn**	**bnclassify**
Búsqueda y puntaje	Log-likelihood		x	x			x	x
	AIC	x			x	x	x	x
	BIC	x			x	x	x	x
	BDeu		x			x	x	
	Pred. log-likelihood						x	
	BDs						x	
	BDJ						x	
	MBDe						x	
	BDla						x	
	K2						x	
	BGe						x	

Para finalizar, la tabla 9 muestra los métodos, aproximaciones y funciones en R que utilizan los paquetes objeto de estudio para el respectivo aprendizaje de los parámetros de la red. En líneas generales, los métodos empleados se circunscriben a MLE, BE y EM, donde cada paquete puede emplear aproximaciones derivadas de uno o varios de éstos. En este sentido, cada paquete implementa diferentes aproximaciones que llevan a distintas maneras de realizar la estimación de los parámetros.

Tabla 9. Métodos, aproximación y función en R para el aprendizaje de los parámetros en los paquetes objeto de análisis

Paquete	Método	Aproximación	Función en R
catnet	NRI	NRI	NRI
deal	BE	Distribución a priori para variables discretas: Dirichlet Distribución a priori para variables continuas: *Gaussian inverse gamma*	learn()
sparsebn	MLE	Para variables discretas: *multi-log regression* Para variables continuas: *least squares regression*	estimate.parameters()
pcalg	NRI	NRI	NRI
bnstruct	MLE EM	MAP EM	learn.network() – incluye aprendizaje parámetros learn.params () – sólo para aprendizaje de parámetros
bnclassify	MLE BE	Dirichlet	lp()
bnlearn	MLE BE	NRI	bn.fit()

NRI = No Registra Información

3.3 Herramientas disponibles para la inferencia

En comparación con la cantidad de herramientas desarrolladas para abordar el proceso de aprendizaje de la red, los paquetes disponibles en R para realizar inferencia probabilística sobre éstas son relativamente menos. Entiéndase lo anterior como la cantidad de paquetes habilitados que se dedican única y exclusivamente a implementar algoritmos enfocados a dar respuestas sobre razonamientos probabilísticos. De hecho los más utilizados en la literatura para este fin son **gRain**, **rbmn** y recientemente **BayesNetBP**, siendo el primero de los mencionados la solución estándar o más popular. Estos paquetes disponen de diversas funcionalidades para importar, inicializar o especificar redes aprendidas en otros paquetes como los analizados en la sección anterior y así, una vez la red haya sido introducida correctamente ejecutar las consultas de interés.

Es importante señalar que si bien los paquetes **bnlearn** y **bnstruct** se concentran en ofrecer diferentes alternativas para acometer el proceso de aprendizaje de la red, también implementan algoritmos básicos que permiten ejecutar consultas sobre la probabilidad condicional de las variables. De igual forma, se tiene el caso del paquete **pcalg** cuya aplicabilidad no se extrapola a

procedimientos de inferencia pero permite la estimación de la magnitud del efecto causal de una intervención sobre una variable.

En cuanto a las funcionalidades implementadas por los paquetes identificados para el desarrollo del proceso de inferencia, la tabla 10 permite observar que todos facilitan el tipo de consulta CPQ, incluyendo preguntas sobre la probabilidad marginal, conjunta o condicional de las variables pero ninguno implementa soluciones para ejecutar consultas tipo MAP. Asimismo, el algoritmo que se despliega en la mayoría de los casos es el propuesto por Lauritzen y Spiegelhalter (1988), comúnmente conocido como *Message Passing*, exceptuando a **bnlearn** donde se utiliza *Logic Sampling* y *Likelihood Weighting*.

Se resalta el hecho que los paquetes **BayesNetBP** y **gRain** son los únicos que facilitan la inclusión de *hard evidence* y *soft evidence* sobre la red, por lo que considerando ésto y los tipos de consulta habilitados, se puede concluir que los paquetes dedicados única y exclusivamente a labores de inferencia habilitan una mayor cantidad de funcionalidades y/o alternativas en torno al tipo de algoritmo, de consulta y de evidencia con los cuales pueda trabajar el usuario, mientras que los paquetes **bnlearn** y **bnstruct** se limitan a aspectos básicos en relación con esta tarea.

Vale la pena mencionar que el paquete **rbmn** fue considerado y se realizó la respectiva consulta bibliográfica pero no se encontró documentación suficiente para desarrollar el análisis exhaustivo que permitiera el entendimiento de las funcionalidades dispuestas en el mismo.

Tabla 10. Tipo de algoritmo, algoritmo, tipo de consulta y tipo de evidencia habilitados en los paquetes destinados para la inferencia probabilística sobre redes bayesianas

Paquete	Tipo de algoritmo	Algoritmo	Tipo de consulta	Tipo de evidencia
BayesNetBP	Exacta	*Lauritzen y Spiegelhalter – LS or message passing*	Marginal, conjunta y condicional	*Hard and soft evidence*
bnstruct	Exacta	*Lauritzen y Spiegelhalter – LS or message passing*	Condicional, Marginal	*Hard evidence*
bnlearn	Aproximada	*Likelihood weighting* *Logic sampling*	Marginal, conjunta y condicional	*Hard evidence*
gRain	Exacta	*Lauritzen y Spiegelhalter – LS or message passing*	Marginal, conjunta y condicional	*Hard and soft evidence*

3.4 Utilidades y *wrappers*

En esta sección se realiza una breve descripción de las utilidades y *wrappers* más utilizados en la literatura para la modelización de redes bayesianas. Por lo general, dichas utilidades están enfocadas en la representación o manipulación de las redes, mientras que los *wrappers* se configuran como *softwares* comerciales o conjunto de funciones desarrollados por fuera del entorno de R que cuentan con extensiones que incluyen funcionalidades básicas y/o limitadas para trabajar en dicho entorno.

3.4.1 Utilidades

Algunas de las más populares son:

shinyBN

Consiste en una aplicación de Shiny de código abierto y gratuita donde por medio de una interfaz gráfica de usuario se facilita la modelización de redes bayesianas, así como el respectivo proceso de inferencia y representación gráfica o visualización de la misma. El principal valor agregado de esta herramienta radica en que todos los procesos descritos anteriormente pueden realizarse a través de pocos *clicks* y sin la necesidad de utilizar código o de conocer un lenguaje de programación específico. La aplicación puede ser utilizada en línea o en el entorno de R. Hace uso de las funcionalidades implementadas en los paquetes **bnlearn, gRain, visNetwork, pROC** y **rmda** para las tareas de: aprendizaje de la estructura y parámetros, inferencia, visualización, análisis con curva ROC y bosquejo de la curva DCA (*decisión curve analysis*), respectivamente. Como archivos de entrada puede recibir conjuntos de datos, tablas con la estructura de la red u objetos propios del paquete **bnlearn.** Solamente soporta variables discretas (Chen, et al. 2019).

BayesianNetwork

Se constituye como una aplicación de Shiny similar a **shinyBN**, donde a través de una interfaz gráfica de usuario bastante sencilla se puede realizar toda la modelización de redes bayesianas sin necesidad de implementar código. Para el aprendizaje de la red, de los parámetros y para el proceso de inferencia se basa en el paquete **bnlearn**, mientras que para la representación gráfica de la red hace uso de **networkD3**. El principal factor diferenciador con respecto a **shinyBN** consiste en que **BayesianNetwork** dispone de un módulo para la manipulación de las redes, específicamente para la identificación de los nodos padres, de los nodos hijos, del manto de Markov, de las aristas entrantes y salientes, entre otros. De igual forma, contiene un módulo para ejecutar simulaciones sobre la red aprendida (Govan, Sin fecha).

3.4.2 Wrappers

Algunos de los *wrappers* más populares en la literatura son:

RHugin

RHugin es un paquete de R que provee una interfaz de programación (API por sus siglas en inglés) del software comercial Hugin, específicamente de la herramienta llamada *Hugin Decision Engine* (HDE), la cual permite realizar el aprendizaje de la estructura de la red y el proceso de inferencia sobre la misma. En Hugin la inferencia no es ejecutada directamente en la red objeto de análisis, en su lugar, el dominio representado en ésta es fragmentado en un conjunto de subdominios llamados *belief universes* sobre los cuales se puede propagar la nueva evidencia de dos formas; la primera es conocida como *Distribute Evidence* y es usada cuando la evidencia de un *belief universe* debe ser propagada a todo el sistema, mientras que en *Collect Evidence* se desarrolla el proceso contrario, la evidencia se propaga de todo el sistema a un universo en concreto.

TETRAD

Tetrad es un programa no comercial que ha sido diseñado por fuera del entorno de R con el objetivo de otorgarle a sus usuarios herramientas para realizar aprendizaje y/o inferencia sobre modelos gráficos (continuos, discretos o mixtos) usando una interfaz simple, en la cual no es necesario tener conocimientos avanzados sobre programación o estadística. Se destaca que cuenta con un módulo en R que permite implementar los algoritmos y funcionalidades propias del programa. En Tetrad, el aprendizaje de la estructura de la red se puede ejecutar por medio de algoritmos de búsqueda como *Fast Greedy Search* o *Greedy Fast Casual Inference*, a partir de conocimiento previo (manualmente) o usando un generador aleatorio de grafos. Por otra parte, para el aprendizaje de los parámetros en modelos con variables continuas hace uso de Modelos de Ecuaciones Estructurales (permiten probar y estimar relaciones causales a partir de datos estadísticos y suposiciones cualitativas sobre la causalidad), mientras que para variables discretas utiliza modelos bayesianos (Scheines, et al. 1998).

Grappa

Su autor, Green (2005), lo define como un conjunto de funciones en R que sirven para realizar propagación probabilística sobre modelos gráficos discretos. Específicamente, las funciones pueden ser usadas para calcular la distribución de probabilidad marginal y/o condicional en variables con espacio finito.

4 Aplicación y comparativa

En este capítulo se presenta un resumen de las principales funciones incorporadas en los paquetes objeto de análisis para la manipulación de las redes aprendidas. Así mismo, pretende generar una guía básica encaminada a orientar a los usuarios no expertos o investigadores sobre qué paquete utilizar para la modelización de redes bayesianas considerando el escenario o la información que ellos tengan disponible.

Como se ha mencionado en otras secciones de este documento, en aras de alcanzar los objetivos trazados y como soporte a lo mostrado en el presente capítulo, se procedió a probar e implementar en el entorno de R cada uno de los paquetes objeto de análisis siguiendo sus manuales de uso y de funciones. Se desplegaron, por paquete, uno a uno los algoritmos destinados al aprendizaje de la estructura y de los parámetros, así como las funciones para su manipulación y representación gráfica, entre otros. Para este ejercicio se utilizaron diversos conjuntos de datos estándar o de prueba de acuerdo con el tamaño y el tipo de dato que contenía, la mayoría de éstos fueron descargados del repositorio de **bnlearn** (https://www.bnlearn.com/bnrepository/), a saber: discretos (ASIA, ALARM, HAILFINDER), continuos (MARKS, NOVARTIS – propio del paquete **catnet**) y mixto (KSL – propio del paquete **deal**). En relación con tamaño de éstas, se siguió la siguiente categorización: pequeña (menos de 20 variables), mediana (entre 20 y 50 variables) y grande (más de 50 variables). La ejecución de las funcionalidades encaminadas a tratar con datos incompletos se realizó con el conjunto de datos ASIA al modificarlo para que exhibiera valores faltantes. Por último, en cuanto a los paquetes destinados a la inferencia se realizó un ejercicio similar, donde se introdujo nueva evidencia para observar un evento de interés.

4.1 Funcionalidades de los paquetes objeto de análisis

Si bien cada paquete implementa diferentes funciones alineadas con el objetivo con que sus autores pretenden que se utilice, éstos por lo general contienen una línea base de comandos que permiten la manipulación de las redes generadas o de aquellas seleccionadas para analizar. La tabla 11 muestra las funcionalidades básicas que de acuerdo con el autor del presente documento son necesarias o útiles al momento de trabajar con redes bayesianas. Las mismas son transversales a la tarea que se intente abordar, es decir, la mayoría de las funciones pueden ser utilizadas bien sea para labores de aprendizaje, clasificación o inferencia.

En este sentido, se observa cómo la función encaminada a la generación de muestras aleatorias sobre el conjunto de datos es la más implementada, ya que se encuentra disponible en todos los paquetes menos en **bnclassify.** Ésta reviste gran importancia para la modelización de redes bayesianas, toda vez que facilitaría la validación de cualquier procedimiento desplegado. Curiosamente, sólo el paquete **catnet** dispone de la misma funcionalidad con el agregado de posibilitar la inclusión de perturbaciones sobre la asignación aleatoria de valores.

Por ejemplo:

- En cualquier paquete el muestreo aleatorio generaría un conjunto de datos donde sus variables tomen valores aleatorios sobre el subconjunto de valores posibles. En **catnet** sería:

```
> samples2 <- cnSamples(object = cnet1, numsamples = 100, output =
"frame")
> head(samples2,10)
   N1 N2 N3 N4
1  C3 C2 C4 C1
2  C3 C1 C3 C3
3  C2 C4 C1 C3
4  C1 C3 C3 C3
5  C1 C4 C2 C3
6  C3 C1 C3 C3
7  C1 C4 C2 C3
8  C1 C3 C3 C3
9  C2 C4 C4 C4
10 C3 C3 C3 C4
```

En la anterior instrucción se le pide al paquete **catnet** que genere un conjunto de datos con 100 registros, donde cada variable puede tomar aleatoriamente 4 valores, a saber: C1, C2, C3 y C4.

- En la muestra aleatoria con perturbaciones propias del paquete **catnet** se obligaría a una o más variables a tomar el valor asignado por el usuario.

```
> samples3 <- cnSamples(object = cnet1, numsamples = 10,
perturbations = c(0,0,1,2))
> samples3
   N1 N2 N3 N4
1  C3 C4 C1 C2
2  C2 C3 C1 C2
3  C2 C2 C1 C2
4  C3 C3 C1 C2
5  C2 C4 C1 C2
6  C2 C4 C1 C2
7  C2 C4 C1 C2
8  C3 C1 C1 C2
9  C2 C3 C1 C2
10 C1 C3 C1 C2
```

En la anterior instrucción se le pide al paquete **catnet** que genere un conjunto de datos con 10 registros, donde las variables N1 y N2 toman valores aleatorios peros a las variables N3 y N4 se les asignan los valores C1 y C2, respectivamente.

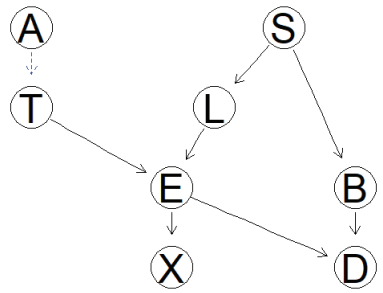
Considerando lo anterior, se podría aducir que una funcionalidad que permita la comparación entre redes complementaría el proceso de generación aleatoria de muestras y posterior validación de las mismas. Los paquetes **catnet, pcalg, bnstruct, bnlearn** y **BayesNetBP** implementan uno o varios comandos para esta labor. En la mayoría de los casos se hace uso de *Hamming distance* (número de aristas diferentes entre dos estructuras) como medida de comparación, aunque también se emplea el número de arcos identificados como falsos-positivos, falsos-negativos y verdaderos-positivos. De igual forma, algunos de estos paquetes habilitan el proceso de confrontación entre redes a partir de sus representaciones gráficas, mostrando por medio de diversos colores la ausencia o presencia de aristas de una red a otra. Este caso es propio del proceso de aprendizaje de la estructura. La comparación en el proceso de inferencia

también se puede desarrollar gráficamente al mostrar por medio de colores las variaciones suscitadas en las distribuciones marginales de probabilidad gracias a la inclusión de nueva evidencia sobre la red.

Por ejemplo:

- Uno de los paquetes que incorpora la comparación gráfica de redes es **bnlearn**, donde las aristas de color rojo o azul denotan la ausencia o presencia de ésta al comparar el grafo de referencia con el grafo aprendido.

```
> graphviz.compare(true.asiadag, bn_hcBIC)
>
```



En este caso la arista punteada de color azul representa a un arco que se encuentra presente en el grafo de referencia, el cuál es el verdadero DAG, pero se encuentra ausente en el grafo aprendido usando el algoritmo HC y el puntaje BIC.

Otra funcionalidad que puede resultar de mucha utilidad, sobre todo cuando se trabaja con redes grandes consiste en la representación textual del modelo, ya que cuando las redes contienen muchos nodos (más de 30), en el cuadrante de gráficos de R es difícil distinguir qué nodo representa a cada variable, por lo que entender las relaciones de dependencia entre éstos resulta una tarea titánica. La representación textual del modelo se configura como una solución a dicho impase gracias a que exhibe el orden topológico de los nodos y sus respectivas relaciones de dependencia. Los paquetes que implementan esta función son **deal, bnstruct** y **bnlearn.**

Por ejemplo:

- En el paquete **deal** en lugar de realizar la representación gráfica de la red se podría observar su representación textual, por lo que tendríamos lo siguiente:

```
> deal::modelstring(object = ksl.s)

[1] "[FEV|Smok:Sex:Year][Kol|Sex:Year][Hyp|FEV:logBMI][logBMI|Kol:Sex]
[Smok|Sex:Year][Alc|Smok:Sex:Year][Work|Alc:Sex:Year][Sex][Year]"
```

donde, la variable *FEV* depende de *Smok, Sex* y *Year*, mientras que la variable *Kol* depende de *Sex* y *Year*, así sucesivamente. Si bien este caso sería fácilmente representable y entendible gráficamente, el ejemplo se encamina a mostrar cómo se mostraría la representación textual del modelo.

Cabe destacar que paquetes como **sparsebn, pcalg** y **bnclassify** no facilitan la importación de objetos gráficos como graphNEL (paquete graph, Gentleman, et al. 2020) o igraph (paquete igraph, Csardi y Nepusz, 2006) y tampoco la lectura de archivos en formatos como BIF, DSC o NET, por lo que toda la manipulación y análisis que se deba realizar sobre la red tiene que ejecutarse sobre estructuras aprendidas por éstos. Vale la pena mencionar que el paquete **deal** sólo permite la lectura de archivos en formato NET, el cual corresponde a estructuras provenientes del *wrapper* **RHugin** o de su *software* comercial.

Una función poco implementada pero que a criterio del autor de este documento brinda un valor añadido en relación con el proceso de aprendizaje de la red, consiste en la utilización de técnicas de re-muestreo para ejecutar la anterior actividad, por ejemplo *Bootstrap*; ésta se encuentra disponible sólo en dos paquetes: **bnstruct** y **bnlearn.** Al aplicar re-muestreo en el aprendizaje de la estructura se estima el nivel de confianza de la presencia de una arista en la misma, es decir, se identifica una medida de precisión con respecto a la presencia de dicho arco en la red original. Puntualmente, en **bnstruct** el resultado no sería un DAG sino un DAG ponderado parcialmente (WPDAG), el cual consiste en una matriz de adyacencia cuyas celdas (*i, j*) toman como valor el número de veces en el que una arista que va del nodo *i* al nodo *j* aparece en la red obtenida por medio de re-muestreo. Sobre este no se podría ejecutar el aprendizaje de los parámetros (Sambo y Franzini, 2019).

Por ejemplo:

- En el paquete **bnstruct** el aprendizaje de una red por medio de la aplicación de re-muestreo usando *Bootstrap* se ejecuta de la siguiente forma:

```
> asia_bootstrap <- bootstrap(object = asia_bnstructObj1, num.boots =
200)
> asia_net_bot <- learn.network(object = asia_bootstrap, algo = "hc",
scoring.func = "BDeu", bootstrap = TRUE)
> asia_net_bot@variables

[1] "A" "S" "T" "L" "B" "E" "X" "D"
> asia_net_bot@wpdag

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    8   73   13   22    9    7    5
[2,]   11    0    6   98   89   10    0    0
[3,]   75   19    0   27   53  181    0   58
[4,]   10  183   25    0   36  182    0   71
[5,]   25  175   26   22    0   31    4  170
[6,]   11    9   29   36   41    0  191  134
[7,]   11    0    0    0    4   79    0    0
[8,]   17    1   24   30   62   34    1    0
```

En las instrucciones anteriores se le pide a **bnstruct** que genere un re-muestro *Bootstrap* de 200 muestras sobre el conjunto de datos, luego se realiza el aprendizaje de la red utilizando el algoritmo HC y el puntaje BDeu; también se devuelven por consola el nombre de las variables del modelo y finalmente se muestra la matriz de adyacencia WPDAG. En esencia, dicha matriz permite observa que en 191 de las 200 muestras ejecutadas existe una arista entre los nodos X y E, o que en 181 de las 200 muestras se evidencia una arista entre E y T.

Tabla 11. Funciones básicas disponibles en cada paquete objeto de análisis

Funciones básicas	catnet	sparsebn	pcalg	deal	bnclassify	bnstruct	bnlearn	gRain	BayesNetBP
Especificar manualmente la red	x			x		x	x	x	
Generar redes aleatoriamente	x	x	x				x		
Importar objetos gráficos	x			x		x	x	x	x
Mostrar número de aristas		x			x	x	x		
Matriz - Listado de aristas	x	x	x						
Matriz de adyacencia		x	x			x	x		
Representación textual del modelo				x	x		x		
Comparar redes	x		x			x	x		x
Generar muestras	x	x	x	x		x	x	x	x
Generar muestras con perturbaciones	x								
Re-muestreo aprendizaje de la red						x	x		
Aprendizaje con intervenciones		x	x						

Por otro lado, al analizar en detalle las características y funcionalidades de los paquetes objeto de estudio se concedió especial atención a atributos como: la facilidad para incluir conocimiento previo en relación con la presencia o ausencia de aristas, las conexiones incluidas desde y hacia otros paquetes, los formatos de visualización o representación gráfica de las redes, las restricciones en torno a los nodos padres y los formatos con los cuales se pueden importar o exportar objetos gráficos. Todos estos se encuentran resumidos en la tabla 12 (información completa, ver anexo tabla A.2).

Considerando lo expuesto anteriormente, la tabla 12 reafirma que los paquetes **sparsebn, pcalg** y **bnclassify** son los únicos que no facilitan la importación de objetos gráficos, siendo **pcalg** un caso atípico al no permitir ni la importación ni la exportación de redes. Por su parte, **sparsebn** y **bnclassify** habilitan funciones para convertir o exportar sus resultados a formatos populares como graphNEL (paquete graph), bn (paquete bnlearn) o gRain. En cuanto a conexiones dispuestas con otros paquetes se evidencia cómo la mayoría de éstos cuentan con funciones para esta labor. En muchas ocasiones dichas conexiones se dan entre los mismos paquetes objeto de análisis. Se destaca que **bnstruct** no exhibe ningún tipo de enlace hacia otro paquete.

La inclusión de conocimiento previo durante el aprendizaje de la red se configura como una funcionalidad necesaria para dicha etapa, toda vez que permite asegurar la presencia o ausencia de aristas que de antemano se conocen. Todos los paquetes enfocados en el aprendizaje de la red incluyen

dentro de las respectivas funciones donde se implementa el algoritmo para la identificación de la estructura, argumentos en los cuales el usuario puede establecer o garantizar la ausencia o presencia de aristas entre dos nodos. Un claro ejemplo son los argumentos de *blacklist* para el primero de los casos y *whitelist* para el segundo, los cuales son utilizados en los paquetes **sparsebn**, **bnlearn** y **pcalg**, aunque el último emplea otros nombres para éstos. El paquete **bnstruct** sólo habilita un argumento para determinar las aristas que deben mostrarse en el gráfico, mientras que **catnet** facilita la inclusión de conocimiento previo al permitir la especificación del subconjunto de padres de cada nodo o la probabilidad de existencia de una arista entre nodos. Se resalta la forma en que el paquete **deal** habilita la ejecución de esta labor: lo hace por medio de una interfaz gráfica muy amigable del tipo "*drag and drop*" en la cual sólo basta con conectar el par de nodos y seleccionar si la arista está permitida o prohibida.

En relación con la funcionalidad encaminada a restringir el número de padres de cada nodo, los paquetes que la implementan son **bnstruct**, **catnet** y **bnlearn**; lo hacen por medio de argumentos localizados en la misma función del algoritmo de aprendizaje.

Tabla 12. Funcionalidades avanzadas disponibles en cada paquete objeto de análisis

Paquete	Visualización	Inclusión conocimiento previo	Conexiones a	Restricciones en nodos padres	Importar - Exportar gráficos
catnet	No incluida. El usuario debe usar Graphviz por fuera de R	parentsPool parámetro fixedParents parámetro edgeProb parámetro dirProb parámetro	Graphviz	maxParentSet parámetro (no especifica valor)	archivos SIF archivos BIF
sparsebn	Incluida, la estructura puede ser representada por cualquiera de los siguientes objetos: graphNEL, igraph, network - plotDAG ()	whitelist parámetro blacklist parámetro specify.prior() - función	bnlearn pcalg	NRI	Conversión a objeto igraph Conversión a objeto graphNEL Conversión a objeto network Conversión a objeto bn
pcalg	Incluida por medio de RgraphViz o igraph	fixedGaps parámetro fixedEdges parámetro addBgKnowledge () - función	dagitty	NRI	NRI
deal	Incluida por medio de RgraphViz	specifygraph parámetro drawnetwork() - función banlist parámetro	Hugin	Nodos discretos sólo pueden tener otros nodos discretos como padres. Nodos continuos pueden tener nodos discretos o continuos como padres	Archivos NET
bnclassify	Incluida por medio de RgraphViz	NRI	caret mlr gRain graph bnlearn	NRI	Conversión a objeto gRain Conversión a objeto graphNEL

bnstruct	Incluida por medio de RgraphViz - plot()	layering parámetro mandatory.edges parámetro	NRI	max.fanin parámetro (no especifica valor) max.parents parámetro (no especifica valor) max.parents.layers parámetro (no especifica valor) layer.struct parámetro (no especifica valor)	archivos BIF archivos DSC archivos NET
bnlearn	Incluida por medio de RgraphViz o graph	whitelist parámetro blacklist parámetro	gRain graph igraph pcalg deal parallel	maxp parámetro (no especifica valor)	archivo BIF archivo DSC archivo NET archivo DOT Conversión a objeto gRain Conversión/Lectura a objeto pcalg
gRain	Incluida por medio de RgraphViz o graph	-	Hugin	-	archivo NET Lectura objeto bn Lectura objeto graphNEL
BayesNetBP	Incluida por medio de RgraphViz o graph	-	bnlearn pcalg catnet deal RHugin	-	Lectura objeto graphNEL

NRI = No Registra Información
No especifica valor = el parámetro no exhibe un valor mínimo o máximo de padres con el cual se puedan restringir los nodos

4.2 Factores diferenciadores y limitaciones de los paquetes objeto de análisis

Con el objetivo de identificar los principales factores diferenciadores y limitaciones de los paquetes objeto de estudio se realizó para cada uno de éstos su respectiva descarga y utilización en R. Se siguieron los documentos de referencia y manuales de uso dispuestos para las versiones analizadas. A continuación se presenta un resumen de los resultados de esta actividad para cada paquete.

4.2.1 Paquetes destinados al aprendizaje

catnet

Factores diferenciadores:

- Del proceso de aprendizaje se deriva un listado de redes potenciales de acuerdo con el nivel de complejidad de las mismas. Es necesario realizar la selección del modelo con el que se desea trabajar.
- En presencia de valores faltantes brinda buenos resultados (red aprendida es similar a la red verdadera).

Limitaciones:

- Algoritmo SearchOrder es computacionalmente costoso en redes medianas y grandes. En promedio, el aprendizaje de la red HAILFINDER tomó 102 segundos, mientras que el mismo proceso con otros algoritmos no superaba los 10 segundos (ver anexo tabla A.5).
- Algoritmo SA es computacionalmente costoso en redes medianas y grandes. En promedio, el aprendizaje de la red ALARM tomó 204 segundos, mientras que el mismo proceso con otros algoritmos no superaba los 6 segundos (ver anexo tabla A.4).
- Solo trabaja con nodos discretos (categóricos).
- Establecer el orden de los nodos en redes muy grandes puede resultar engorroso.

Observaciones:

- Es necesario conocer el orden de los nodos para la implementación de SearchOrder.

Ejemplo:

El aprendizaje de una red pequeña (ASIA, consta de 8 variables) se podría realizar por SearchOrder o por SA, teniendo en cuenta que para la primera opción es necesario conocer el orden de los nodos. SearchOrder se ejecutaría de la siguiente manera:

```
> nodeOrder <- c("A", "S", "T", "L", "B", "E", "X", "D")
> MLE_solution <- cnSearchOrder(data = asia, maxParentSet = 2,
nodeOrder = nodeOrder)
> MLE_solution

 Number of nodes    = 8,
 Sample size        = 5000,
 Number of networks = 20
 Processing time    = 3.460
```

Al establecer en primera instancia el orden de los nodos y posteriormente realizar el aprendizaje de la red por medio del método de búsqueda exhaustiva, se obtiene un listado de redes potenciales ordenadas de acuerdo con su nivel de complejidad, en este caso, serían 20 redes potenciales.

Considere ahora el caso donde se está trabajando con una red mediana o grande, por ejemplo, una con 40 nodos (variables), establecer el orden de éstos se configuraría como una tarea compleja.

Conclusión:

- Paquete útil para cuando se tiene conocimiento sobre el orden de los nodos y en general conocimiento previo en relación con la red. Dado este caso brinda buenos resultados (red aprendida es similar a la red verdadera).

sparsebn

Factores diferenciadores:

- Especializado en trabajar con bases de datos donde se observa mayor cantidad de variables en comparación con el número de registros.
- Del proceso de aprendizaje se deriva un listado de redes potenciales de acuerdo con el número de aristas por lo que es necesario realizar la selección del modelo con el que se desea trabajar.
- Diferentes opciones y utilidades para ejecutar la representación gráfica del modelo.

Limitaciones:

- Computacionalmente costoso cuando se trabaja con bases de datos que contiene menor número de variables en comparación con el número de registros. No está diseñado para trabajar es tipo de redes, puede realizar el proceso de aprendizaje pero será lento o demorado (ver anexo tabla A.3, tabla A.4 o tabla A.5).
- Computacionalmente costoso cuando se trabaja con redes que contienen sólo datos discretos y que adicionalmente son grandes, en el sentido de gran cantidad de registros (>10.000) pero menor número de variables (entre 50 y 100), comparativamente (ver anexo tabla A.3, tabla A.4 o tabla A.5).

Observaciones:

- Brinda especial importancia o atención al tipo de dato (experimental u observacional).

Ejemplo:

Aprendizaje de la red ASIA – Tiempo promedio de aprendizaje 109 segundos.

```
> asia_data <- sparsebnData(data = asia, type = "discrete")
> asia_dag <- estimate.dag(sparsebnData = asia_data)
> summary(asia_dag)

sparsebn Solution Path
 8 nodes
 5000 observations
 20 estimates for lambda in [0.166, 16.6029]
```

```
 Number of edges per solution: 0-1-2-2-2-2-5-6-7-8-8-8-10-11-14-14-15-
15-15-16

       lambda nedge
1  16.6029137     0
2  13.0293021     1
3  10.2248747     2
4   8.0240724     2
5   6.2969708     2
6   4.9416106     2
7   3.8779782     5
8   3.0432821     6
9   2.3882459     7
10  1.8741998     8
11  1.4707970     8
12  1.1542226     8
13  0.9057877    10
14  0.7108259    11
15  0.5578277    14
16  0.4377609    14
17  0.3435372    15
18  0.2695943    15
19  0.2115668    15
20  0.1660291    16
```

Para el aprendizaje en este paquete se debe crear en primera instancia un objeto de la clase *sparsebnData*. El resultado del aprendizaje es un listado de redes potenciales, 20 en este caso, las cuales están ordenadas descendentemente de acuerdo con el rango establecido para el parámetro regularizador pero ubicadas de forma ascendente de acuerdo con el número de aristas en cada red.

Conclusión:

- Paquete útil para cuando se tiene mayor cantidad de variables en comparación con la cantidad de registros. Funciona mejor con datos continuos.

pcalg

Factores diferenciadores:

- Implementa metodologías para calcular el efecto y tamaño de una intervención en el modelo.
- Proceso de aprendizaje con variables continuas deriva en buenos resultados y rápidos.
- Permite combinar cualquiera de los métodos basados en restricciones junto con cualquiera de los algoritmos basados en búsqueda y puntaje implementados.

Limitaciones:

- No trabaja con datos cualitativos - nominales (factores). Los mismos deben ser convertidos a enteros.
- Se deben especificar los "estadísticos suficientes" para el proceso de aprendizaje. No es una tarea compleja en redes pequeñas pero es absolutamente necesaria. En redes medianas y discretas es engorroso porque se debería establecer el tamaño de cada variable (número de valores que puede tomar), aunque no es absolutamente necesario pero recomendable.
- El proceso de aprendizaje basado en búsqueda y puntaje no cuenta con un puntaje para datos discretos por lo que no se puede realizar.

- No soporta trabajar con bases de datos medianas o grandes que contengan variables discretas.

Observaciones:

- Tiene en cuenta si las variables son observacionales o experimentales, así como cuándo existen variables latentes u ocultas en el modelo.

Ejemplo:

Aprendizaje de la red ASIA – Algoritmos basados en restricciones

```
> head(asia, 5)

   A   S   T  L   B   E   X   D
1 no yes  no no yes  no  no yes
2 no yes  no no  no  no  no  no
3 no  no yes no  no yes yes yes
4 no  no  no no yes  no  no yes
5 no  no  no no  no  no  no yes
```

Con este conjunto de datos donde las variables son cualitativas el paquete **pcalg** no realiza el aprendizaje, por consola sale un error o directamente se cae la sesión. Se deben convertir a enteros, puede realizarse de la siguiente forma:

```
> asia <-as.data.frame(coerce_discrete(asia))
> head(asia, 5)

  A S T L B E X D
1 0 1 0 0 1 0 0 1
2 0 1 0 0 0 0 0 0
3 0 0 1 0 0 1 1 1
4 0 0 0 0 1 0 0 1
5 0 0 0 0 0 0 0 1

> pc_asia.pcalg <- pc(suffStat = list(dm = asia, adaptDF = FALSE),
indepTest = binCItest, alpha = 0.05, labels = colnames(asia),
skel.method = "stable.fast")
```

Para el aprendizaje se deben definir "estadísticos suficientes", lo cual es bastante sencillo para una red donde las variables solo tomen dos valores (binario), también se debe establecer el nivel de significancia, si la prueba se realizará sobre datos discretos, binarios o continuos y el nombre de las columnas.

En un conjunto de datos discretos cuyas variables puedan tomar más de dos valores resulta necesario definir el tamaño de éstas, es decir, especificar cuántos valores puede tomar cada una de las variables que hacen parte del conjunto de datos. Por ejemplo, en una conjunto como ALARM se debería realizar lo siguiente:

```
> alarm <- as.data.frame (coerce_discrete(alarm))
> suffStat <- list(alarm, nlev = c(3,3,2,3,3,3,3,3,3,3,3,2,4,4,
                                   4,3,2,2,2,2,2,3,2,2,3,3,2,2,
                                   3,2,2,3,3,4,4,4,4), adaptDF = FALSE)
> pc_alarm.pcalg<- pc(suffStat = suffStat, indepTest = disCItest, alpha
= 0.01, labels = colnames(alarm))
```

No obstante, el aprendizaje en **pcalg** sobre conjuntos de datos de más de 5.000 registros y 30 variables tumba directamente la sesión de R, por lo cual es necesario utilizar una muestra.

Aprendizaje de la red MARKS – Algoritmos basados en búsqueda y puntaje

```
> score.BIC <- new(score = "GaussL0penObsScore", data = marksCon, lambda
= 0.5*log(nrow(marksCon)))
> ges_marksBIC.pcalg <- ges(score.BIC, verbose = TRUE)
```

Para el aprendizaje por búsqueda y puntaje en **pcalg** primero se debe definir el puntaje a utilizar, actualmente sólo cuenta con uno para datos continuos (GaussL0penObsScore). Posteriormente se realizar el aprendizaje con el algoritmo seleccionado.

Conclusión:

- Paquete útil para cuando se quieren analizar las relaciones de causalidad entre variables, así como para cuando se quiere estimar o cuantificar el tamaño de dichas relaciones.

deal

Factores diferenciadores:

- Interfaz gráfica tipo "*drag and drop*" para especificar manualmente la red o incluir conocimiento previo sobre la misma.
- Facilita la representación textual del modelo.
- Especializado en trabajar con conjuntos de datos mixtos, compuestos tanto por variables discretas como continuas. Dado este escenario brinda resultados estables y rápidos (ver anexo tabla A.9).

Limitaciones:

- Brinda soluciones densas e inestables cuando se trabaja con conjuntos de datos que sólo contienen datos discretos (pequeñas).
- No soporta el proceso de aprendizaje de redes donde el conjunto de datos sólo contiene datos continuos.
- No soporta el proceso de aprendizaje en redes de tamaño mediano o grande compuesta sólo por nodos discretos (>25 de nodos).
- El aprendizaje requiere la creación de diversos objetos previos.

Ejemplo:

Aprendizaje de la red KSL

```
> kslNet <- network(data = ksl)
> ksl_prior <- jointprior(nw = kslNet)
> kslNet <- learn(nw = kslNet, data = ksl, prior = ksl_prior)$nw
> ksl.s <- autosearch(nw = kslNet, data = ksl, prior = ksl_prior, trace
= FALSE, timetrace = TRUE)$nw
```

Para el aprendizaje de una red primero se debe crear un objeto de la clase *network* (nw), luego se debe calcular la distribución a *priori* de probabilidad conjunta, posteriormente se realiza la estimación de los parámetros propios de las distribuciones de probabilidad local y finalmente se emplea el algoritmo aprendizaje, en este caso *greedy search.*

Conclusión:

- Paquete útil para cuando se trabaja con redes que constan de variables discretas y continuas (conjunto de dato de carácter mixto). Funciona muy

bien para crear manualmente la red o incluir conocimiento previo a través de una interfaz gráfica sencilla.

bnstruct

Factores diferenciadores:

- Implementa diferentes alternativas en torno a metodologías para la imputación de datos (incluye *k-nn*, predicciones, casos completos y algoritmo SEM).
- Brinda gran cantidad de alternativas para la manipulación de las redes aprendidas.

Limitaciones:

- No trabaja con datos cualitativos - nominales (factores). Los mismos deben ser convertidos a enteros.
- Se debe definir el tipo de datos con que va a trabajar la red, si son continuos o discretos. En caso de ser discretos se debe establecer el tamaño de cada nodo, lo que supone más trabajo para el usuario.
- Para trabajar con datos continuos resulta necesario ejecutar una discretización antes del proceso de aprendizaje, lo que en redes grandes resulta engorroso.
- Algoritmo SM es computacionalmente costoso (ver anexo tabla A.5).
- El aprendizaje requiere la creación de diversos objetos previos.

Ejemplo:

Aprendizaje de la red ALARM

```
> head(sample(alarm,8),5)

      PAP HRBP   HRSA MINV   PMB      BP VLNG    LVV
1  NORMAL HIGH   HIGH HIGH FALSE NORMAL  LOW NORMAL
2  NORMAL HIGH   HIGH ZERO FALSE    LOW ZERO NORMAL
3  NORMAL HIGH   HIGH ZERO FALSE NORMAL ZERO NORMAL
4  NORMAL HIGH   HIGH ZERO FALSE    LOW ZERO NORMAL
5  NORMAL HIGH   HIGH ZERO FALSE    LOW ZERO NORMAL
```

Con este conjunto de datos donde las variables son cualitativas el paquete **bnstruct** no realiza el aprendizaje, por consola sale un error o directamente se cae la sesión. Se deben convertir a enteros, puede realizarse de la siguiente forma:

```
> alarm[1:37] <- sapply(alarm[1:37], as.integer)
> head(sample(alarm,8),5)

   VMCH VALV LVV MINV ACO2 ANES HIST CO
1     3    1   3    1    3    1    1  1
2     3    4   3    4    2    1    1  2
3     3    4   3    4    2    1    1  1
4     3    4   3    4    2    1    1  1
5     3    4   3    4    2    1    1  3

> alarm_bnstructObj <- BNDataset(data = alarm,
                         discreteness = rep(TRUE,37),
                         variables = colnames(alarm),
                         node.sizes = c(rep(3,2),2,rep(3,8),2,4,4,
                         4,3,rep(2,5),3,2,2,3,3,2,2,3,2,2,3,3,rep(4,4)))
> alarm_bnstruct <- BN(BNDataset = alarm_bnstructObj)
```

```
> MMPC_alarm.bnstruct <- learn.network(BN = alarm_bnstruct,
                        BNDataset = alarm_bnstructObj, algo = "mmpc",
                        alpha = 0.05)
> MMHC_alarm.bnstruct <- learn.network(BN = alarm_bnstruct,
                        BNDataset = alarm_bnstructObj,
                        algo = "mmhc", scoring.func = "AIC")
```

En relación con el proceso de aprendizaje, inicialmente se debe crear un objeto de la clase *BNDataset*, donde se debe definir el tipo de dato que contiene cada variable (*discreteness*) y el tamaño que toma cada una de éstas (*node.sizes*). Posteriormente, se crea un objeto de la clase *BN* y luego se procede al aprendizaje, en este ejemplo se ejecutó el aprendizaje a partir de dos algoritmos diferentes, uno basado en restricciones y otro híbrido, a saber: MMPC y MMHC, en su orden.

Aprendizaje de la red MARKS (ver anexo tabla A.7)

```
> marksCon <- marksCon %> mutate(
      MECH = cut(MECH, breaks = c(0,20,40,60,82), labels = c(1,2,3,4)),
      VECT = cut(VECT, breaks = c(0,20,40,60,82), labels = c(1,2,3,4)),
      ALG  = cut(ALG, breaks = c(0,20,40,60,82), labels = c(1,2,3,4)),
      ANL  = cut(ANL, breaks = c(0,20,40,60,82), labels = c(1,2,3,4)),
      STAT = cut(STAT, breaks = c(0,20,40,60,82), labels = c(1,2,3,4)))
> marksCon[1:5] <- sapply(marksCon[1:5], as.integer)
> marks_bnstructObj <- BNDataset(data = marksCon,
                              discreteness = rep("c",5),
                              variables = colnames(marksCon),
                              node.sizes = rep(4,5))
> marks_bnstruct <- BN(BNDataset = marks_bnstructObj)
> HC_marks.bnstruct <- learn.network(BN = marks_bnstruct,
      BNDataset = marks_bnstructObj, algo = "hc", scoring.func = "BIC")
```

En **bnstruct** el aprendizaje de redes bayesianas gaussianas demanda mayor trabajo para el usuario toda vez que se debe ejecutar una discretización de los datos. Como se observa en el código anterior, se da inicio al proceso al realizar dicha tarea, luego se convierten los datos a valores enteros, posteriormente se crean los objetos *BNDataset* y *BN* para finalmente efectuar el aprendizaje por medio del algoritmo HC y el puntaje BIC.

Cuando se trabaja con conjuntos de datos donde se presentan valores faltantes el proceso de imputación se ejecuta luego de crear los objetos *BNDataset* y *BN* pero previo a la implementación del algoritmo de aprendizaje.

```
##Imputación
> asia_missingObj <- bnstruct::impute(BNDataset = asia_missingObj1,
                                      k.impute = 5)
> MMPC_Masia.bnstruct <- learn.network(BN = Masia_bnstruct,
                              Imputed.data = asia_missingObj,
                              algo = "mmpc", alpha = 0.05)
```

Conclusión:

- Paquete ideal para trabajar cuando el conjunto de datos tiene valores perdidos, ya que incluye una funcionalidad para imputar datos a partir de *k-nn* y otra para trabajar sólo con datos completos, así como el algoritmo SEM.

bnlearn

Factores diferenciadores:

- Trabaja de forma estable y rápida sobre cualquier base de datos, sea grande o pequeña o con nodos de tipos discretos, continuos o mixtos.
- Implementa gran cantidad de alternativas para el aprendizaje de la red, así como de puntajes y pruebas de independencia.
- Brinda gran cantidad de alternativas para la manipulación de las redes aprendidas.
- Permite ejecutar todas las etapas del proceso de modelización con redes bayesianas, a saber: aprendizaje estructura, aprendizaje parámetros, inferencia y clasificación.

Limitaciones:

- El puntaje de *pred. Log-likelihood* sólo puede ser utilizado con valores enteros.
- Puntaje *log-likelihood* computacionalmente costoso en redes grandes (ver anexo tabla A.4).
- El proceso de imputación requiere una especificación previa de la estructura de la red.
- No permite inferencia exacta.

Observaciones:

- Se caracteriza por su sencillez, no requiere creación o definición de objetos previos al aprendizaje.

Ejemplo:

Aprendizaje red ASIA – basado en restricciones

```
> pc_asia.bnlearn<- pc.stable(database = asia, test = "x2", undirected
 = FALSE))
> iamb_asia.bnlearn<- inter.iamb(database = asia, test = "mi", undirec
ted = FALSE))
```

Como se observa en el código anterior, el aprendizaje de la red en el paquete **bnlearn** es directo y no requiere la creación de objetos previos. En el caso de utilizar algoritmos basados en restricciones sólo basta con definir la base de datos y la prueba de independencia con la que se desea trabajar. En los ejemplos anteriores se hizo uso de los algoritmos PC e Inter-IAMB y de las pruebas chi-cuadrado e información mutua.

Aprendizaje red ALARM – basado en búsqueda y puntaje, e híbridos.

```
> hc_alarm.bnlearn <- hc(database = alarm, score = "mbde", restart = 5
, perturb = 3)
> tabu_alarm.bnlearn <- tabu(database = alarm, score = "k2", tabu = 15
> mmhc_alarm.bnlearn <- mmhc(alarm)
```

Por su parte, los algoritmos basados en búsqueda y puntaje sólo requieren de la especificación de la base de datos y el puntaje de interés. Así como de los argumentos propios de cada algoritmo. En el ejemplo anterior se utiliza el algoritmo GSRR y Tabu, por lo que se define el número de reinicios, el número de perturbaciones y el número de movimientos que se incluirán en la lista Tabu, respectivamente.

De igual forma, para los algoritmos híbridos solo es necesario definir la base de datos con la que se desea trabajar.

Conclusión:

- Paquete ideal para trabajar bajo cualquier escenario, con una implementación rápida y estable de cualquier algoritmo de aprendizaje disponible.

Se destaca que todos los comentarios realizados en relación con los paquetes objeto de análisis son derivados por parte del autor del presente documento teniendo en cuenta la actividad desarrollada con cada uno de éstos. En ningún caso se pretende demeritar alguno de los paquetes o sus funcionalidades.

4.2.2 Paquetes destinados a la clasificación

En cuanto a los paquetes destinados para el aprendizaje de redes bayesianas enfocadas en la clasificación, sólo dos implementan funcionalidades para efectuar dicha tarea, ellos son: **bnclassify** y **bnlearn**. El primero se constituye como el punto de referencia para el desarrollo de este tipo de aprendizaje, provee diferentes alternativas para la estimación del clasificador (alrededor de ocho), las cuales pueden desplegarse de manera sencilla y directa, e incluye funcionalidades de estimación por validación cruzada y predicción. Las únicas limitaciones evidenciadas consistieron en la ausencia de una funcionalidad destinada al cálculo de la matriz de confusión y la ausencia de mayor número de alternativas en relación con la función de pérdida; la versión analizada sólo trabaja con el *"accuracy"* como medida de evaluación del modelo.

Por su parte, **bnlearn** provee sólo dos alternativas para la estimación del clasificador y funciones aisladas para la validación cruzada y predicción. Cabe destacar que el objetivo final de este paquete no es brindar soluciones para este tipo de aprendizaje a diferencia de **bnclassify**, el cual sí fue desarrollado exclusivamente para dicha labor. Como valor agregado se podría argumentar que **bnlearn** dispone de una variedad considerable de funciones de pérdida para la respectiva evaluación del modelo.

Ejemplo:

Aprendizaje red bayesiana ALARM para clasificación – paquete **bnclassify**

```
> nb_alarm <- nb(class = "HYP", dataset = alarm_sample) %>% lp(dataset
= alarm_sample, smooth = 0.01)
> cv(bnc_bn = nb_alarm, dataset = alarm_sample, k = 10
```

Al hacer uso del paquete **bnclassify** para el aprendizaje de una red de clasificación se deben especificar los siguientes argumentos: la variable que figura como clase y el conjunto de datos a utilizar. Posteriormente, se ejecuta el aprendizaje de los parámetros de la red por medio de la función *lp*. Se destaca que también cuenta con una función que permite realizar el aprendizaje de la estructura y de los parámetros en un solo paso. En el código anterior se incluyó una validación cruzada del aprendizaje de la red.

Aprendizaje red bayesiana ALARM para clasificación – paquete **bnlearn**

```
> nb_alarmbnlearn <- naive.bayes(dataset = alarm_sample, class = "HYP")
> bn.cv(dataset = alarm_sample, bn = nb_alarmbnlearn , loss = "pred-
exact", method = "k-fold", k = 10)
```

En **bnlearn** el procedimiento para aprender una red de clasificación es muy similar al evidenciado en el paquete **bnclassify**, tanto para el aprendizaje como para la validación cruzada, la única diferencia es que en **bnlearn** puedes hacer uso de diferentes alternativas en cuanto a la función de pérdida.

4.2.3 Paquetes destinados a la inferencia

En este caso se probaron los paquetes **bnlearn, bnstruct, gRain** y **BayesNetBP.** Los resultados más estables en relación con las consultas realizadas se evidenciaron en los dos últimos, ya que ambos siempre coincidieron en los valores de las probabilidades estimadas. **BayesNetBP** podría considerarse como un paquete más completo en comparación con **gRain**, toda vez que el último se limita a desarrollar el proceso de inferencia solamente sobre redes discretas, mientras que el primero puede ejecutarlo sobre cualquier tipo de red. No obstante, para la utilización de **BayesNetBP** resulta necesario ejecutar algunas especificaciones previas a la inclusión de la nueva evidencia, como son establecer el tipo de nodo (si son discretos o continuos) y precisar el nombre de los mismos. En **gRain** el proceso es mucho más sencillo y directo.

Ejemplo:

BayesNetBP

```
>    dg_bayesnet    <-as.graphNEL
(true.asiadagPar) #network import
> node.class <- rep(TRUE, 8) #type
of node
>       names(node.class)       <-
dg_bayesnet@nodes #node´s name
> tree_int <- Initializer(dag =
dg_bayesnet,    data    =    asia,
node.class    =    node.class)    #
initialization
>  tree_post  <-  AbsorbEvidence
(tree_int, vars = c("A", "D"),
values   =   list("yes",   "yes"))
#evidence inclusion
> FactorQuery(tree_post, c("B"),
mode = "conditional")#query 1

    B   S       prob
1  no  no 0.28946306
2 yes  no 0.71053694
3  no yes 0.08954115
4 yes yes 0.91045885


>  Marginals(tree_post,   c("B"))
#query 2

$marginals
$marginals$B
       no       yes
0.1582561 0.8417439
```

gRain

```
>    asia_grain    <-    as.grain
(true.asiadagPar) #network import
>    asia_ev    <-    setFinding
(asia_grain, nodes = c("A", "D"),
 states   =   c("yes",   "yes"))
#evidence inclusion
> querygrain(asia_ev, c("B","S"),
type = "conditional") #query 1

     B
S             no       yes
  no  0.28946306 0.7105369
  yes 0.08954115 0.9104588


>querygrain(asia_ev, c("B"), type
= "marginal") #query 2

$B
B
       no       yes
0.1582561 0.8417439
```

Como se puede observar en el código anterior, **BayesNetBP** requiere de tres pasos intermedios entre la importación de la red y la inclusión de evidencia, haciendo que el proceso no sea directo; en comparación con **gRain**. También, se evidencia la similitud en la respuesta de las consultas. Se destaca que

BayesNetBP sólo permite la consulta de probabilidades condicionadas entre nodos que sean padre-hijo (o descendiente) en la red, mientras que gRain lo permite para cualquier par de nodos.

Por su parte, los paquetes **bnlearn** y **bnstruct** muestran algunas discrepancias en los resultados en comparación con los dos anteriores. Si bien en **bnlearn** la introducción de nueva evidencia y consulta sobre la red puede efectuarse a partir de una sola instrucción, los resultados de la misma consulta siempre serán diferentes (si no se utiliza una semilla) debido a la metodología que se emplea durante el proceso de inferencia (aproximada). En cuanto al paquete **bnstruct,** la introducción de evidencia y posterior consulta es bastante compleja. Adicionalmente, la forma en que se puede acceder a los resultados no es directa; no implementa una funcionalidad para ésto sino que se debe ingresar al objeto derivado de dicho proceso mediante el mecanismo de "@".

Ejemplo:

Bnlearn

```
> true.asiadag <- model2network
("[A][S][T|A][L|S][B|S][D|B:E]
[E|T:L][X|E]") #true dag creation
> true.asiadagPar <- bn.fit
(true.asiadag, asia, method =
"mle") #parameter estimation true
dag
> cpquery(true.asiadagPar, event =
(B == "yes" & S =="yes"), evidence
= (A == "yes" & D == "yes")) #
evidence inclusion and query

[1] 0.5454545 #output 1

[1] 0.7777778 #output 2
```

Bnstruct

```
> dataset <- asia() #specification
of the dataset
> net <- learn.network(dataset)
#network learning
> obs <- list("observed.vars" = c
("Asia","Dyspnea"),"observed.val"
=        c("yes","yes"))#evidence
inclusion and query object
> engine <- InferenceEngine(net)
#inference engine object creation
> engine <- belief.propagation
(engine, obs) #propagation
> new.net <- updated.bn(engine)
#network updating
> new.net@cpts #results
```

En conclusión, para desarrollar tareas de inferencia los paquetes de mayor utilidad y con resultados estables son **gRain** y **BayesNetBP**, siendo **gRain** un paquete cuya operación o manipulación es mucha más sencilla que **BayesNetBP** pero limitado en términos del tipo de variables con las que puede trabajar. Se destaca que los paquetes objeto de estudio en este acápite mostraron tiempos de respuesta similares en torno las consultas realizadas (ver anexo tabla A.10).

4.3 Guía sobre conveniencia de cada paquete

A continuación se muestran las tablas creadas a modo de guía para orientar a los usuarios sobre cuál(es) paquete(s) utilizar teniendo en cuenta el objetivo que se pretende alcanzar, la técnica que se desea aplicar, el tamaño del conjunto de datos, el tipo de dato con que se cuenta y la presencia o no de valores faltantes, entre otros. Las diferentes combinaciones de éstas dieron como resultado más de 20 escenarios, para los cuales se marcó el/los paquetes que disponen de herramientas para trabajar sobre cada uno de éstos (ver tabla 13, 14 y 15).

Estas tablas también intentan resumir o unir la información dispuesta en capítulos anteriores. Se resalta que **bnlearn** es el único paquete que puede ser

utilizado para la mayor cantidad de escenarios identificados, evidenciando así que es el paquete más completo entre todos los analizados.

Un ejemplo de cómo se deben leer las siguientes tablas o establecer qué paquete utilizar dado un escenario, puede ser: suponiendo que se quiere realizar el aprendizaje de la estructura de la red por medio de un algoritmo basado en restricciones, se tiene un conjunto de datos mediano con valores faltantes y datos continuos, el usuario dispone de los paquetes **bnstruct** y **bnlearn** para ejecutar la tarea propuesta.

Tabla 13. Guía para el uso de paquetes destinados al aprendizaje de la estructura

Escenario					Paquetes						
Objetivo	**Método**	**Tipo de dato**	**Tamaño de la base**	**Datos completos**	**catnet**	**sparsebn**	**pcalg**	**deal**	**bnclassify**	**bnstruct**	**bnlearn**
Aprendizaje estructura	Restricciones	Discreto	Pequeña	sí			x			x	x
				no						x	x
			Mediana	sí						x	x
				no						x	x
			Grande	sí						x	x
				no						x	x
		Continuo	Pequeña	sí			x			x	x
				no						x	x
			Mediana	sí			x			x	x
				no						x	x
			Grande	sí			x			x	x
				no						x	x
		Mixto	Pequeña	sí							x
				no							x
			Mediana	sí							x
				no							x
			Grande	sí							x
				no							x
	Búsqueda y puntaje	Discreto	Pequeña	sí	x	x				x	x
				no	x					x	x
			Mediana	sí	x	x				x	x
				no	x					x	x
			Grande	sí	x	x				x	x
				no	x					x	x
		Continuo	Pequeña	sí		x	x			x	x
				no						x	x
			Mediana	sí		x	x			x	x
				no						x	x
			Grande	sí		x	x			x	x
				no						x	x
		Mixto	Pequeña	sí				x			x
				no							x
			Mediana	sí				x			x
				no							x
			Grande	sí				x			x
				no							x

	Método	Tipo de dato	Tamaño de la base	Datos completos	catnet	sparsebn	pcalg	deal	bnclassify	bnstruct	bnlearn
	Híbrido	Discreto	Pequeña	sí						x	x
				no						x	x
			Mediana	sí						x	x
				no						x	x
			Grande	sí						x	x
				no						x	x
		Continuo	Pequeña	sí						x	x
				no						x	x
			Mediana	sí						x	x
				no						x	x
			Grande	sí						x	x
				no						x	x
		Mixto	Pequeña	sí							x
				no							x
			Mediana	sí							x
				no							x
			Grande	sí							x
				no							x
	Clasificador	Discreto	Pequeña	sí					x		x
				no							x
			Mediana	sí					x		x
				no							x
		Continuo	Pequeña	sí							x
				no							x
			Mediana	sí							x
				no							x
		Mixto	Pequeña	sí							x
				no							x
			Mediana	sí							x
				no							x

Nota: definición del tamaño de las bases: pequeñas (menos de 20 variables), medianas (entre 20 y 50 variables), grandes (más de 50 variables).

Tabla 14. Guía para el uso de paquetes destinados al aprendizaje de los parámetros

Escenario				Paquetes						
Objetivo	**Tipo de dato**	**Tamaño de la base**	**datos completos**	**catnet**	**sparsebn**	**pcalg**	**deal**	**bnclassify**	**bnstruct**	**bnlearn**
Aprendizaje parámetros	Discreto	Pequeña	Sí		x		x	x	x	x
			No						x	x
		Mediana	Sí		x			x	x	x
			No						x	x
		Grande	Sí		x			x	x	x
			No						x	x
	Continuo	Pequeña	Sí		x				x	x
			No						x	x
		Mediana	Sí		x				x	x
			No						x	x
		Grande	Sí		x				x	x
			No						x	x
	Mixto	Pequeña	Sí				x			x
			No							x
		Mediana	Sí							x
			No							x
		Grande	Sí							x
			No							x

Nota: los paquetes **catnet** y **pcalg** no fueron marcados debido a que en sus manuales no se detalla la metodología o método que se sigue para la estimación de parámetros, adicionalmente no implementan una función específica para esta actividad.

Tabla 15. Guía para el uso de paquetes destinados al procedimiento de inferencia

Escenario				Paquetes			
Objetivo	**Tipo de dato**	**Tipo de algoritmo**	**Tipo de evidencia**	**bnstruct**	**bnlearn**	**gRain**	**BayesNetBP**
Inferencia	Discreto	Exacto	Hard	x		x	x
			Soft			x	x
		Aproximado	Hard		x		
			Soft				
	Continuo	Exacto	Hard	x			x
			Soft				x
		Aproximado	Hard		x		
			Soft				
	Mixto	Exacto	Hard				x
			Soft				x
		Aproximado	Hard		x		
			Soft				

5 Comentarios y conclusiones

Al realizar un análisis completo de los paquetes más utilizados en la literatura para la modelización de redes bayesianas en el entorno de R, específicamente de sus principales funcionalidades, valor añadido y limitaciones, entre otras; el presente trabajo se constituye como una guía básica que facilita la orientación de usuarios no expertos, investigadores o demás personas interesadas en la temática al momento de decidir sobre cuál paquete puede utilizar para ejecutar una determinada actividad.

Las redes bayesianas son modelos gráficos probabilísticos que permiten la representación y entendimiento de un fenómeno a partir de la identificación de las relaciones de dependencia entra las variables involucradas. Si bien la construcción de las mismas puede realizarse manualmente por medio de conocimiento experto, esta labor puede resultar complicada por lo que en su lugar se emplean algoritmos que la aprendan utilizando datos. Su modelización consiste en el aprendizaje de la estructura, el aprendizaje de los parámetros y posteriores procesos de inferencia probabilística.

Actualmente, existen en el entorno de R más de 20 paquetes destinados a tareas como la representación, manipulación, aprendizaje o inferencia de redes bayesianas. No obstante, aquellos enfocados exclusivamente en las dos últimas son un número reducido, bien sea que habiliten herramientas para una de éstas o para ambas, siendo el proceso de inferencia quien muestra menor cantidad de paquetes para su ejecución.

Los paquetes estudiados en el presente trabajo fueron: **catnet, sparsebn, pcalg, deal, bnclassify, bnstruct** y **bnlearn,** todos éstos encaminados a desarrollar el aprendizaje de la estructura y parámetros de la red, siendo **bnclassify** un caso especial al facilitar el aprendizaje de estructuras enfocadas a la clasificación de nuevas instancias. En cuanto a los procesos de inferencia, se analizaron los paquetes de **gRain**, **BayesNetBP**, **bnstruct** y **bnlearn**, estando los dos primeros destinados exclusivamente para esta labor, mientras que los restantes simplemente implementan algunas funcionalidades para desarrollarla. El análisis de estos paquetes se realizó al estudiar en detalle los manuales de uso y de funciones dispuestos por cada uno, así como la bibliografía adicional relacionada con éstos. De igual forma, se implementó en R el respectivo código que permitió experimentar con las funciones habilitadas.

A partir de los resultados observados se puede aseverar que el paquete **bnlearn** figura como el más completo, toda vez que permite realizar cada una las actividades relacionadas con la modelización de redes bayesianas; a saber: aprendizaje de la estructura y de los parámetros, inferencia, y aprendizaje de clasificadores, además implementa diferentes alternativas para su manipulación, validación, evaluación y representación. Del mismo modo, mostró resultados estables y rápidos para la mayoría de los algoritmos dispuestos por sus creadores. Se destaca que es de fácil implementación y no requiere especificaciones en relación con el conjunto de datos que se utiliza o la creación de objetos previos al proceso de aprendizaje.

Los paquetes **bnstruct** y **pcalg** le siguen a **bnlearn** en relación con el número de escenarios o actividades en los cuales pueden ser usados. Sin embargo, no ofrecen la misma cantidad de alternativas y funcionalidades; su implementación

resulta en mayor trabajo para el usuario debido a que es necesario la especificación de algunos parámetros o creación previa de objetos. Las principales ventajas de cada uno radican en la facilidad u opciones dispuestas para imputar datos faltantes y en la posibilidad de estimar el tamaño de las relaciones causales entre variables, respectivamente.

Por otra parte, se evidenció que **catnet**, **sparsebn** y **deal** son para uso específico o especializado de acuerdo con el escenario que se presente. Por ejemplo: **sparsebn** es ideal para trabajar cuando el conjunto de datos que se tiene a disposición contiene un mayor número de variables en comparación con el número de registros; **deal** mostró mejores resultados en términos de precisión de la red aprendida en comparación con otros paquetes cuando se utilizó un conjunto de datos con variables mixtas y **catnet** exhibió resultados sobresalientes cuando se tenía conocimiento previo relacionado con el orden de los nodos y el conjunto de datos estaba compuesto por variables categóricas.

En cuanto a los paquetes **gRain** y **BayesNetBP**, ambos ofrecen funcionalidades muy completas para el proceso de inferencia, facilitando la inclusión de evidencia *soft* y/o *hard* para la consulta de las probabilidades marginales, conjuntas o condicionales una vez dicha evidencia haya sido propagada. No obstante, las principales limitaciones de ambos paquetes consisten en que **gRain** sólo trabaja con datos discretos, mientras que **BayesNetBP** trabaja con cualquier tipo de dato, pero requiere especificaciones y creación de objetos previos a la introducción de evidencia en la red.

En términos generales, las funcionalidades implementadas por los paquetes enfocados en el aprendizaje de la estructura de la red y de sus respectivos parámetros cubren en buena medida todas las aproximaciones teóricas existentes. Así, se evidenció una cantidad considerable de alternativas para los algoritmos basados en restricciones, los algoritmos basados en búsqueda y puntaje, las pruebas de independencia y los puntajes. Sin embargo, a criterio personal considero que la mayoría de los paquetes carecen de alternativas para el aprendizaje de la red basándose en muchos modelos posibles, concretamente en *Bayesian Model Averaging*. Como aproximación de este método se observan las técnicas de re-muestreo implementadas en **bnlearn** y **bnstruct.**

Para el proceso de inferencia se exhiben mayores carencias, ya que son pocos los paquetes destinados exclusivamente para esta labor. En ningún caso se implementan funcionalidades encaminadas a responder consultas del tipo *Maximum a Posteriori Queries* (MAP). De igual forma, el escenario donde se cuenta con datos discretos, continuos o mixtos y se quiere implementar un algoritmo aproximado con evidencia *soft* no es cubierto por ningún paquete. En rasgos generales, el tipo de algoritmo aproximado es poco utilizado en estos paquetes, así como la evidencia de tipo *soft*.

Para finalizar, considero que trabajos próximos podrían enfocarse en extender este análisis a paquetes que no fueron cubiertos por el autor o a realizar las actualizaciones respectivas derivadas de nuevas versiones de los mismos. Así como también llevar a cabo un enfoque similar trascendiendo el entorno de R al llevarlo a herramientas como Python, WEKA, MATLAB u otros lenguajes de programación.

6 Bibliografía

- Aliferis, C., Statnikov, A., Tsamardinos, I., Subramani, M., y Koutsoukos, X. (2010). Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I: Algorithms and Empirical Evaluation. *Journal of Machine Learning Research*, *11*, 171–234.
- Aragam, B., Gu, J., y Zhou, Q. (2019). Learning Large-Scale Bayesian Networks with the sparsebn Package. *Journal of Statistical Software*, *91*(11), 1–38.
- Balov, N., y Salzman, P. (2020). "How to use the catnet package". Recuperado 30 de abril de 2020, de https://cran.r-project.org/web/packages/catnet/index.html.
- Boettcher, S., y Dethlefsen, C. (2003). deal: A Package for Learning Bayesian Networks. *Journal of Statistical Software, 8*(20), 1-40.
- Cano, A., Gómez-Olmedo, M., Masegosa, A.R., y Moral, S. (2013). Locally Averaged Bayesian Dirichlet Metrics for Learning the Structure and the Parameters of Bayesian Networks. *International Journal of Approximate Reasoning*, *54*, 526–540.
- Castillo, E., Gutiérrez, J., y Hadi, A.S. (1997). Sistemas expertos y modelos de redes probabilísticas, Madrid, España. Academia de Ingeniería.
- Chen, J., Zhang, R., Dong, X., Lin, L., Zhu, Y., He, J., Christiani, D., Wei, Y., y Chen, F. (2019). shinyBN: An online application for interactive Bayesian network inference and visualization. *BMC Bioinformatics*, *20,* 1-5.
- Chickering, D. (1996). Learning Bayesian networks is NP-complete. In D. Fisher and H. Lenz (Eds.), *Learning from data: Artificial intelligence and statistics V*, page 121–130. Springer-Verlag.
- Chickering, D. (2002). Optimal structure identification with greedy search. *Journal of Machine Learning Research, 3*(3), 507–554.
- Chickering, D., Heckerman, D., y Meek, C. (2004). Large-sample learning of Bayesian Networks is NP-hard. *Journal of Machine Learning Research, 5*, 1287-1330.
- Chow, C.K., y Liu, C.N. (1968). Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory, 3*,462–467.
- Claassen, T., Mooij, J., y Heskes, T. (2013). Learning Sparse Causal Models is not NP-hard. *In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, pages 172-181. AUAI Press, Corvallis.
- Colombo, D., Maathuis, M., Kalisch, M., y Richardson, T. (2012). Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, *40*(1), 294-321.
- Colombo, D., y Maathuis, M. (2014). Order-Independent Constraint-Based Causal Structure Learning. *Journal of Machine Learning Research, 15*, 3921-3962.
- Cooper, G., y Yoo, C. (1999). Causal Discovery from a Mixture of Experimental and Observational Data. *In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 116-125.
- Csardi, G., y Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695.

- Dempster, A., Laird, N., y Rubin, D. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*, 1-38.
- Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics, 7*(1), 1-26.
- Friedman, N., Geiger, D., y Goldszmidt, M. (1997). Bayesian Network Classifiers. *Machine Learning, 29*, 131-163.
- Friedman, N. (1998). The Bayesian Structural EM Algorithm. *In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 129-138. Morgan Kaufmann Publishers.
- Friedman, N., Nachman, I., y Pe'er, D. (1999). Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. *In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 206-215.
- Fung, R., y Chang, K. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. *In Proceedings of the 5th Conference on Uncertainty in Artificial Intelligence (UAI).* San Mateo, California. Morgan Kaufmann, pages 112-117.
- Gasse, M., Aussem, A., y Elghazel, H. (2014). A hybrid algorithm for Bayesian network structure learning with application to multi-label learning. *Expert Systems with Applications, 41*, 6755-6772.
- Gelman, A., Hwang, J., y Vehtari, A. (2013). Understanding predictive information criteria for Bayesian models. *Statistics and Computing, 24*, 997-1016.
- Gentleman, R., Whalen, E., Huber, W., y Falcon, S. (2020). *graph: A package to handle graph data structures*. R package version 1.66.0.
- Glover, F. (1989). Tabu Search: 1. *Operations Research Society of America (ORSA) journal on computing*, *1*(3), 190-206.
- Govan, P. (Sin fecha). BayesianNetwork. Recuperado 15 de mayo de 2020, de http://paulgovan.github.io/BayesianNetwork/
- Green, P. (2005). Grappa: R functions for probability propagation. Recuperado 19 de mayo de 2020 de https://people.maths.bris.ac.uk/~mapjg/Grappa/
- Hauser, A., y Buhlmann, P. (2012). Characterization and greedy learning of interventional Markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research, 13*, 2409–2464.
- Hausser, J., y Strimmer, K. (2009). Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks. *Journal of Machine Learning Research, 10*, 1469-1484.
- Henrion, M. (1986). Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. *In Proceedings of the 2nd Conference on Uncertainty in Artificial Intelligence (UAI),* pages 149–163.
- Hernández-Orallo, J., Ramírez Quintana, M. J., y Ferri Ramírez, C. (2004). Introducción a la minería de datos. Madrid, España. Pearson Educación S.A.
- Højsgaard, S. (2012). Graphical Independence Networks with the gRain Package for R. *Journal of Statistical Software*, *46*(10), 1–26.
- Højsgaard, S., Edwards, D., y Lauritzen, S. (2012). Graphical Models with R. Use R! series. Springer.
- Højsgaard, S. (2020). Bayesian Networks in R with the gRain package. Recuperado 25 de abril de 2020, de https://cran.r-project.org/web/packages/gRain/index.html

- Kalisch, M., Mächler, M., Colombo, D., Hauser, A., Maathuis, M., y Bühlmann, P. (2014). More Causal Inference with Graphical Models in R Package pcalg. Recuperado 10 de mayo de 2020, de https://cran.r-project.org/web/packages/pcalg/index.html
- Kayaalp, M., y Cooper, G. (2002). A Bayesian Network scoring metric that is based on globally uniform parameter priors. *In Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 251-258. Morgan Kaufmann Publishers.
- Keogh, E. J., y Pazzani, M. (2002). Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools, 11*(4), 587–601.
- Koller, D., y Friedman, N. (2009). Probabilistic Graphical Models: Principles and Techniques. MIT press.
- Kuipers, J., Moffa, G., y Heckerman, D. (2014). Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics. 42*(4), 1689-1691.
- Maathuis, M., Kalisch, M., y Bühlmann, P. (2009). Estimating High-Dimensional Intervention Effects from Observational Data." *The Annals of Statistics, 37*, 3133–3164.
- Mahjoub, M., y Kalti, K. (2011). Software Comparison Dealing with Bayesian Networks. In Advances in Neural Networks—ISNN 2011; Liu, D., Zhang, H., Polycarpou, M., Alippi, C., He, H., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, pp. 168–177.
- Margaritis, D. (2003). Learning Bayesian Network Model Structure from Data. Ph.D. thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Margolin, A., Nemenman, I., Basso, K., Wiggins, C., Stolovitzky, G., Dalla-Favera, R., y Califano, A. (2006). ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context. *BMC Bioinformatics, 7.*
- Martínez, D., Albín, J., Cabaleiro, J., Pena, T., Rivera, F., y Blanco, V. (2009). El criterio de información de Akaike en la obtención de modelos estadísticos de rendimiento. *XX Jornadas de Paralelismo*, A Coruña (España), pp. 439-444.
- Mihaljevic, B. (2013). Bayeclass: an R package for learning Bayesian Network Classifiers. Applications to neuroscience (Master thesis). Universidad Politécnica de Madrid. Madrid, España.
- Mihaljevic, B., Bielza, C., y Larrañaga, P. (2018). bnclassify: Learning Bayesian Network Classifiers. *The R Journal*, *10*(2), 455–468.
- Mihaljevic, B. (2018). Bayesian Networks with R. Tomado de: http://gauss.inf.um.es/umur/xjurponencias/talleres/J3.pdf
- Nagarajan, R., Scutari, M., y Lèbre, S. (2013). Bayesian Networks in R with Applications in Systems Biology. Use R! series. Springer.
- Paniego, S. (2019). Visualization and Interpretation in Large Bayesian Networks (Master thesis). Universidad Politécnica de Madrid. Madrid, España.
- Pazzani, M. (1996). Constructive induction of Cartesian product attributes. *In Proceedings of the Information, Statistics, and Induction in Science Conference*, pages 66–77.
- Pearl, J. (1993). Comment: Graphical models, causality and intervention. *Statistical Science, 8*, 266–269.
- Peng, R. D. (2019). R Programming for Data Science. Recuperado 23 de abril de 2020, de https://bookdown.org/rdpeng/rprogdatascience/

- Peña, J. (2008). Learning Gaussian Graphical Models of Gene Networks with False Discovery Rate Control. I*n Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, pp. 165–176.*
- Russell, J., y Norvig, P. (2009). Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall.
- Sambo, F., y Franzini, A. (2019). bnstruct: an R package for Bayesian Network structure learning with missing data. Recuperado 25 de abril de 2020, de https://cran.r-project.org/web/packages/bnstruct/index.html
- Scanagatta, M., Salmerón, A., y Stella, F. (2019). A survey on Bayesian network structure learning from data. *Progress in Artificial Intelligence*, *8*, 425-439.
- Scheines, R., Spirtes, P., Glymour, C., Meek, C., y Richardson, T. (1998). The TETRAD Project: Constraint based aids to casual model specification. *Multivariate Behavioral Research, 33*, 65-117.
- Scutari, M. (Sin fecha). bnlearn - an R package for Bayesian network learning and inference. Recuperado 4 de mayo de 2020, de https://www.bnlearn.com/
- Scutari, M. (2010). Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software, 35*(3), 1-22.
- Scutari, M., y Denis, J. (2014). Bayesian Networks with Examples in R. CRC Press, Taylor & Francis Group. Boca Raton.
- Scutari, M. (2016). An Empirical-Bayes Score for Discrete Bayesian Networks. *Journal of Machine Learning Research, 52,* 438–448.
- Scutari, M. (2017). Understanding Bayesian Networks with Examples in R [Slides]. Recuperado 10 de abril de 2020, de https://www.bnlearn.com/about/teaching.html
- Scutari, M., Graafland, C., y Gutiérrez, J. (2019). Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning*, *115*, 235-253.
- Shachter, R., y Peot, M. (1990). Simulation Approaches to General Probabilistic Inference on Belief Networks. In Henrion, M., Shachter, R. D., Kanal, L. N., and Lemmer, J. F., *Uncertainty in Artificial Intelligence 5*. North Holland, Amsterdam, 221–231.
- Spirtes, P., Glymour, C.N., y Scheines, R. (2000): Causation, Prediction, and Search. MIT Press.
- Sucar, L. E. (Sin fecha). Redes Bayesianas. Capítulo 1. Recuperado 10 de abril de 2020, de https://ccc.inaoep.mx/~esucar/Clases-mgp/caprb.pdf
- Suzuki, J. (2016). A Theoretical Analysis of the BDeu Scores in Bayesian Network Structure Learning. *Behaviormetrika*, *44*(1), 97–116.
- The R Foundation. (Sin fecha). The R Project for Statistical Computing. Recuperado 23 de abril de 2020, de https://www.r-project.org/
- Tsamardinos, I., Aliferis, C., y Statnikov, A. (2003)., Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations. *In the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 673–678.
- Tsamardinos, I., Aliferis, C., y Statnikov, A. (2003). Algorithms for Large Scale Markov Blanket Discovery." *In Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference*, pp. 376-381. AAAI Press.

- Tsamardinos, I., Brown, L., y Constantin, A. (2006). The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *Machine Learning, 65*, 31-78.
- Webb, G., Boughton, J., y Wang, Z. (2005). Not so Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning, 58*(1), 5–24.
- Yaramakala, S., y Margaritis, D. (2005). Speculative Markov Blanket Discovery for Optimal Feature Selection. *In ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 809–812. IEEE Computer Society, Washington, DC, USA.
- Yu, H., Moharil, J., y Blair, R. (2019). BayesNetBP: An R package for probabilistic reasoning in Bayesian Networks. Recuperado 27 de mayo de 2020, de https://www.researchgate.net/publication/336859662_BayesNetBP_An_R_package_for_probabilistic_reasoning_in_Bayesian_Networks.

7 Anexos

Tabla A.1. Paquetes objeto de estudio, descripción, tabla completa

Paquete	Versión	Año publicación	Año versión actual	Disponibilidad de manual de uso	Disponible en el task-view de gRaphical Models - gR	Disponible en CRAN	Descargas históricas*	Descarga semestral*
catnet	1.15.7	2010	2020	Sí	No	Sí	49.603	6.858
sparsebn	0.1.0	2016	2019	Sí	Sí	Sí	12.116	3.179
pcalg	2.6-10	2006	2020	Sí	Sí	Sí	89.004	13.598
deal	1.2-39	2002	2018	Sí	No	Sí	53.857	6.358
bnclassify	0.4.5	2015	2020	Sí	Sí	Sí	27.325	5.528
bnstruct	1.0.6	2016	2019	Sí	Sí	Sí	27.226	4.717
bnlearn	4.5	2007	2019	Sí	Sí	Sí	259.676	40.783
gRain	1.3-4	2008	2020	Sí	Sí	Sí	114.351	22.196
BayesNetBP	1.4.0	2017	2018	Sí	No	Sí	13.833	3.106

Las descargadas fueron obtenidas usando el paquete **cranlogs** de R, el cual se configura como una API al repositorio "CRAN" específicamente a la base de datos de descargas de paquetes desde RStudio (IDE de R).
*desde octubre de 2012 hasta marzo de 2020
**desde octubre de 2019 hasta marzo de 2020

Tabla A.1. Continuación - Paquetes objeto de estudio, ubicación en línea

Paquete	GitHub	Dirección URL	Dirección URL CRAN
catnet			https://cran.r-project.org/web/packages/catnet/index.html
sparsebn	https://github.com/itsrainingdata/sparsebn		https://cran.r-project.org/web/packages/sparsebn/index.html
pcalg		http://pcalg.r-forge.r-project.org/	https://cran.r-project.org/web/packages/pcalg/index.html
deal			https://cran.r-project.org/web/packages/deal/index.html
bnclassify	http://github.com/bmihaljevic/bnclassify		https://cran.r-project.org/web/packages/bnclassify/index.html
bnstruct	http://github.com/sambofra/bnstruct		https://cran.r-project.org/web/packages/bnstruct/index.html
bnlearn		https://www.bnlearn.com/	https://cran.r-project.org/web/packages/bnlearn/index.html
gRain		http://people.math.aau.dk/~sorenh/software/gR/	https://cran.r-project.org/web/packages/gRain/index.html
BayesNetBP	https://github.com/hyu-ub/BayesNetBP		https://cran.r-project.org/web/packages/BayesNetBP/index.html

Tabla A.2. Funcionalidades específicas de los paquetes objeto de análisis, tabla completa

Paquete	Visualización	Inclusión conocimiento previo	Conexiones a	Restricciones en nodos padres	Importar - Exportar gráficos
bnstruct	Incluida por medio de RgraphViz - plot()	layering parámetro - vector que contiene las capas a las que cada nodo pertenece mandatory.edges parámetro - indica la presencia obligatoria de una arista entre dos nodos	NRI	max.fanin parámetro (no especifica valor) max.parents parámetro (no especifica valor) max.parents.layers parámetro (no especifica valor) layer.struct parámetro (no especifica valor)	archivos BIF archivos DSC archivos NET
catnet	No incluida. El usuario debe usar Graphviz por fuera de R	parentsPool parámetro - especifica el conjunto de posibles padres de cada nodo fixedParents parámetro - especifica reglas de inclusión de aristas edgeProb parámetro - especifica la probabilidad de que una arista exista entre dos nodos dirProb parámetro - especifica las probabilidades a prior	Graphviz	maxParentSet parámetro (no especifica valor)	archivos SIF archivos BIF
sparsebn	Incluida, la estructura puede ser representada por cualquiera de los siguientes objetos: graphNEL, igraph, network - plotDAG ()	whitelist parámetro - especifica las aristas que deben estar presentes en el gráfico blacklist parámetro - especifica las aristas que deben estar ausentes en el gráfico specify.prior() - función	bnlearn pcalg	NRI	Conversión a objeto igraph Conversión a objeto graphNEL Conversión a objeto network Conversión a objeto bn
pcalg	Incluida por medio de RgraphViz o igraph	fixedGaps parámetro - especifica las aristas que deben estar ausentes en el gráfico fixedEdges parámetro - especifica las aristas que deben estar presentes en el gráfico addBgKnowledge () - función, agrega conocimiento previo sobre el CPDAG o del PDAG	dagitty	NRI	NRI

bnlearn	Incluida por medio de RgraphViz o graph	whitelist parámetro - especifica las aristas que deben estar presentes en el gráfico blacklist parámetro - especifica las aristas que deben estar ausentes en el gráfico	gRain graph igraph pcalg deal parallel	maxp parámetro (no especifica valor)	archivo BIF archivo DSC archivo NET archivo DOT Conversión a objeto gRain Conversión/Lectura a objeto pcalg
deal	Incluida por medio de RgraphViz	specifygraph parámetro -permite al usuario ingresar o eliminar aristas drawnetwork() - función, posibilita la prohibición o marcación de aristas de forma gráfica banlist parámetro - listado de las aristas prohibidos	Hugin	Nodos discretos sólo pueden otros nodos discretos como padres. Nodos continuos pueden tener nodos discretos o continuos como padres	NRI
bnclassify	Incluida por medio de RgraphViz	NRI	caret mlr gRain graph bnlearn	NRI	Conversión a objeto gRain Conversión a objeto graphNEL
gRain	Incluida por medio de RgraphViz o graph	-	Hugin	-	archivo NET Lectura objeto bn Lectura objeto graphNEL
BayesNetBP	Incluida por medio de RgraphViz o graph	-	bnlearn pcalg catnet deal RHugin	-	Lectura objeto graphNEL

Tabla A.3. Resultados aprendizaje red ASIA, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu	BDs	MBDe	BDla	K2
catnet	SearchOrder	Listado de redes potenciales	20	3.51 seg		7	10						
	SA	Listado de redes potenciales	20	4.11 seg		9	10						
sparsebn	R. MLE	Listado de redes potenciales	20	109.1 seg				5					
deal	Greedy search	DAG	1	7.69 seg									
	GSRR	DAG	1	22.4 seg									
pcalg	PC	DAG	1	0.42 seg	5								
	FCI	PAG	1	0.71 seg	4								
	RFCI	PAG	1	0.58 seg	5								
	FCIPLUS	PAG	1	0.89 seg	5								
	GES	-	-	No tiene score para datos discretos									
	GIES	-	-										
	SM	-	-										
bnstruct	MMPC	WPDAG	1	0.06 seg	7								
	SM	DAG	1	0.19 seg		14	9		7				
	HC	DAG	1	0.27 seg		13	7						
	MMHC	DAG	1	0.12 seg		7	7		7				
bnlearn	PC	CPDAG	1	0.06 seg	5								
	MMPC	CPDAG	1	0.07 seg	6								
	GS	DAG	1	0.05 seg	4								
	IAMB	DAG	1	0.05 seg	4								
	Fast-IAMB	DAG	1	0.11 seg	4								
	Inter-IAMB	DAG	1	0.04 seg	4								
	IAMB-FDR	DAG	1	0.07 seg	4								
	SI-HITON-PC	CPDAG	1	0.03 seg	5								
	HPC	CPDAG	1	0.11 seg	4								
	HC	DAG	1	0.15 seg		7	11	28	7	7	7	9	11
	Tabu	DAG	1	0.19 seg		7	11	24	7	7	7	9	10
	MMHC	DAG	1	0.05 seg	5								
	H2PC	DAG	1	0.09 seg	4								
	RSMAX2	DAG	1	0.05 seg	5								

Nota: red ASIA verdadera está compuesta por 8 nodos y 8 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.

Tabla A.4. Resultados aprendizaje red ALARM, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu	BDs	MBDe	BDla	K2
catnet	SearchOrder	Listado de redes potenciales	169	58.4 seg		26	33						
	SA	Listado de redes potenciales	194	203.7 seg		32	34						
sparsebn	R. MLE	-	-	> 1 hora									
deal	Greedy search	-	-	Error									
	GSRR	-	-	Error									
pcalg	PC	-	-	Error - sesión R se cae									
	FCI	-	-										
	RFCI	-	-										
	FCIPLUS	-	-										
	GES	-	-	No tiene score datos discretos									
	GIES	-	-										
	SM	-	-										
bnstruct	MMPC	-	-	Error - sesión R se cae									
	SM	-	-										
	HC	DAG	1	42.5 seg		94	56		54				
	MMHC	DAG	1	73.1 seg		50	46		47				
bnlearn	PC	CPDAG	1	2.4 seg	42								
	MMPC	CPDAG	1	2.0 seg	32								
	GS	CPDAG	1	4.2 seg	27								
	IAMB	CPDAG	1	2.9 seg	35								
	Fast-IAMB	CPDAG	1	1.8 seg	34								
	Inter-IAMB	CPDAG	1	1.7 seg	38								
	IAMB-FDR	CPDAG	1	5.9 seg	40								
	SI-HITON-PC	CPDAG	1	4.3 seg	30								
	HPC	CPDAG	1	5.9 seg	44								
	HC	DAG	1	5.1 seg		53	71	No determinado	56	55	56	55	52
	Tabu	DAG	1	4.5 seg		51	71		53	54	53	54	54
	MMHC	DAG	1	2.1 seg	32								
	H2PC	DAG	1	6.9 seg	42								
	RSMAX2	DAG	1	4.3 seg	28								

Nota: red ALARM verdadera está compuesta por 37 nodos y 46 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.
No determinado = no se pudo contar el número de aristas

Tabla A.5. Resultados aprendizaje red HAILFINDER, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu	BDs	MBDe	BDla	K2
catnet	SearchOrder	Listado de redes potenciales	1,149	102.9 seg		39	44						
	SA	Listado de redes potenciales	1,091	1,799 seg		52	52						
sparsebn	R. MLE	-	-	> 1 hora									
deal	Greedy search	-	-	Error									
	GSRR	-	-	Error									
pcalg	PC	-	-	Error - sesión R se cae									
	FCI	-	-										
	RFCI	-	-										
	FCIPLUS	-	-										
	GES	-	-	No tiene score para datos discretos									
	GIES	-	-										
	SM	-	-										
bnstruct	MMPC	WPDAG	1	132.1 seg	No determinado								
	SM	DAG	1	Error - sesión R se cae									
	HC	DAG	1	161.1 seg		151	74		85				
	MMHC	DAG	1	172.7 seg		78	58		56				
bnlearn	PC	CPDAG	1	4.5 seg	35								
	MMPC	CPDAG	1	2.2 seg	35								
	GS	CPDAG	1	27.1 seg	21								
	IAMB	CPDAG	1	3.4 seg	34								
	Fast-IAMB	CPDAG	1	3.2 seg	31								
	Inter-IAMB	CPDAG	1	3.9 seg	34								
	IAMB-FDR	CPDAG	1	5.81 seg	35								
	SI-HITON-PC	CPDAG	1	1.9 seg	35								
	HPC	CPDAG	1	6.6 seg	37								
	HC	DAG	1	7.8 seg		64	80		70	64	70	78	71
	Tabu	DAG	1	8.9 seg		65	81		70	64	70	76	71
	MMHC	DAG	1	2.3 seg	34								
	H2PC	DAG	1	5.8 seg	34								
	RSMAX2	DAG	1	1.8 seg	34								

Nota: red HAILFINDER verdadera está compuesta por 56 nodos y 66 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.
No determinado = no se pudo contar el número de aristas

Tabla A.6. Resultados aprendizaje red ASIA (*Missing*), comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu
catnet	SearchOrder	Listado de redes potenciales	20	3.72 seg		7	11		
	SA	Listado de redes potenciales	20	6.20 seg		10	12		
sparsebn	R. MLE	-	-						
deal	Greedy search	-	-						
	GSRR	-	-						
pcalg	PC	-	-						
	FCI	-	-						
	RFCI	-	-						
	FCIPLUS	-	-						
	GES	-	-						
	GIES	-	-						
	SM	-	-						
bnstruct	MMPC	WPDAG (imputación con knn)	1	0.08 seg	6				
	SM	DAG (imputación con knn)	1	0.20 seg		14	9		9
	HC	DAG	1	0.25 seg		14	9		7
	MMHC	DAG	1	0.16 seg		7	7		7
	SEM	DAG	1	197.3 seg	8				
bnlearn	PC	Imputación requiere especificación de la red							
	MMPC		-						
	GS		-						
	IAMB		-						
	Fast-IAMB		-						
	Inter-IAMB		-						
	IAMB-FDR		-						
	SI-HITON-PC		-						
	HPC		-						
	HC		-						
	Tabu		-						
	MMHC		-						
	H2PC		-						
	RSMAX2		-						
	SEM	DAG	1	0.47	7				

Nota: red ASIA verdadera está compuesta por 8 nodos y 8 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.

Tabla A.7. Resultados aprendizaje red MARKS, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu	BGe
catnet	SearchOrder	-	-	-						
	SA	-	-	-						
sparsebn	R. MLE	Listado de redes potenciales	20	0.31 seg				4		
deal	Greedy search	-	-	No soporta datos solo continuos						
	GSRR	-	-							
pcalg	PC	DAG	1	0.15 seg	6					
	FCI	PAG	1	0.08 seg	6					
	RFCI	PAG	1	0.04 seg	6					
	FCIPLUS	PAG	1	0.05 seg	6					
	GES	Essential - Parametric graph	2	0.03 seg		6	5			
	GIES	Essential - Parametric graph	2	0.03 seg		6	5			
	SM	Essential - Parametric graph	2	0.04 seg		6	5			
bnstruct	MMPC	WPDAG	1	0.05 seg	10					
	SM	DAG	1	0.03 seg		3	15		4	
	HC	DAG	1	0.04 seg		3	15		4	
	MMHC	DAG	1	0.07 seg		3	15		4	
bnlearn	PC	CPDAG	1	0.02 seg	6					
	MMPC	CPDAG	1	0.02 seg	6					
	GS	CPDAG	1	0.02 seg	6					
	IAMB	CPDAG	1	0.02 seg	6					
	Fast-IAMB	CPDAG	1	0.02 seg	6					
	Inter-IAMB	CPDAG	1	0.03 seg	6					
	IAMB-FDR	DAG	1	0.03 seg	4					
	SI-HITON-PC	CPDAG	1	0.02 seg	6					
	HPC	CPDAG	1	0.04 seg	6					
	HC	DAG	1	0.03 seg		6	6	8		4
	Tabu	DAG	1	0.03 seg		6	6	8		4
	MMHC	DAG	1	0.03 seg	6					
	H2PC	DAG	1	0.03 seg	6					
	RSMAX2	DAG	1	0.01 seg	6					

Nota: red MARKS verdadera está compuesta por 5 nodos y 6 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.

Tabla A.8. Resultados aprendizaje red NOVARTIS, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	Log-likelihood	BDeu	BGe
catnet	SearchOrder	-	-	-						
	SA	-	-	-						
sparsebn	R. MLE	Listado de redes potenciales	12	7.57 seg				235		
deal	Greedy search	-	-	Error						
	GSRR	-	-	Error						
pcalg	PC	DAG	1	4.09 seg	184					
	FCI	PAG	1	105. seg		No determinado	No determinado			
	RFCI	PAG	1	56.3 seg						
	FCIPLUS	PAG	1	70.3 seg						
	GES	Essential - Parametric graph	2	111.5 seg						
	GIES	Essential - Parametric graph	2	136.5 seg						
	SM	Essential - Parametric graph	-	No soporta redes grandes						
bnstruct	MMPC	Requiere discretización (muy grande)	-	-						
	SM		-	-						
	HC		-	-						
	MMHC		-	-						
bnlearn	PC	CPDAG	1	69.8 seg	184					
	MMPC	CPDAG	1	228.9 seg	157					
	GS	-	-	Demarado - > 1 hora						
	IAMB	CPDAG	1	40.7 seg	280					
	Fast-IAMB	CPDAG	1	191.4 seg	272					
	Inter-IAMB	CPDAG	1	48.1 seg	283					
	IAMB-FDR	CPDAG	1	140.2 seg	185					
	SI-HITON-PC	CPDAG	1	30.5 seg	162					
	HPC	-	-	Demarado - > 1 hora						
	HC	DAG	1	223.4 seg		1,503	2,828			378
	Tabu	DAG	1	196.6 seg		1,505	2,821			378
	MMHC	DAG	1	7.25 seg	159					
	H2PC	-	-	Demarado - > 1 hora						
	RSMAX2	DAG	1		161					

Nota: no se tiene el tamaño real de la red
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.
No determinado = no se pudo contar el número de aristas

Tabla A.9. Resultados aprendizaje red KSL, comparativo de resultados – tiempo y número de aristas

Paquete	Método	Resultado	Número de resultados	Tiempo de usuario	Test dependencia	BIC	AIC	BDeu
catnet	SearchOrder	-	-	-				
	SA	-	-	-				
sparsebn	Regularized MLE	-	-	-				
deal	Greedy search	DAG	1	4.80 seg				17
	GSRR	DAG	1	5.30 seg				17
pcalg	PC	-	-	-				
	FCI	-	-	-				
	RFCI	-	-	-				
	FCIPLUS	-	-	-				
	GES	-	-	-				
	GIES	-	-	-				
	SM	-	-	-				
bnstruct	MMPC	-	-	-				
	SM	-	-	-				
	Greedy search - HC	-	-	-				
	MMHC	-	-	-				
bnlearn	PC	CPDAG	1	0.36 seg	13			
	MMPC	CPDAG	1	0.39 seg	13			
	GS	CPDAG	1	0.19 seg	7			
	IAMB	CPDAG	1	0.20 seg	10			
	Fast-IAMB	CPDAG	1	0.31 seg	11			
	Inter-IAMB	CPDAG	1	0.26 seg	10			
	IAMB-FDR	CPDAG	1	0.31 seg	10			
	SI-HITON-PC	CPDAG	1	0.42 seg	13			
	HPC	CPDAG	1	0.71 seg	13			
	HC	DAG	1	0.09 seg		11	21	
	Tabu	DAG	1	0.11 seg		11	21	
	MMHC	DAG	1	0.33 seg	10			
	H2PC	DAG	1	0.47 seg	10			
	RSMAX2	DAG	1	0.37 seg	10			

Nota: red KSL verdadera está compuesta por 9 nodos y 17 aristas.
Las pruebas fueron realizadas en un computador portátil con procesador Inter® Core ™ i-5-8250 U y memoria RAM instalada de 8 GB.

Tabla A.10. Resultados inferencia probabilística, comparativo de resultados – tiempo de ejecución

Paquete	Conjunto de datos (red)			
	ASIA	ALARM	HAILFINDER	MARKS
gRain	0.06 seg	0.05 seg	0.05 seg	-
BayesNetBP	0.05 seg	0.03 seg	0.08 seg	0.04 seg
bnlearn	0.02 seg	0.02 seg	0.06 seg	0.04 seg
bnstruct	-	-	-	-

Nota: no se muestran resultados para bnstruct debido a que por la falta de una función propia para observar los mismos no se pudo establecer el tiempo de ejecución.