Data mining

Jordi Gironès Roig

PID_00203552

UOC
Universitat Oberta de Catalunya

Índice

1. Qué es *data mining*

Bajo el concepto de minería de datos (*data mining*) se engloban un conjunto de metodologías, procesos de modelización y técnicas matemáticas, cuyo objetivo es la extracción de información previamente desconocida para soportar la toma de decisiones de negocio. Por información previamente desconocida entendemos relaciones, patrones de comportamiento y tendencias que subyacen en las estructuras de datos, pero que no son detectables mediante técnicas de consulta tradicionales.

Una definición precisa nos ayudará a fijar conceptos clave:

Entendemos por *data mining* el proceso de analizar datos provenientes de distintas fuentes informacionales con el objetivo de extraer información y conocimiento útil.

Nota

Aquí utilizaremos indistintamente los términos *minería de datos* y *data mining* para referirnos a este proceso.

- Por **útil** entendemos que debe haber una alineación con unos objetivos previamente establecidos, tanto desde el punto de vista del negocio, como desde el punto de vista más concreto del *data mining*.

- Por **información** entendemos asociaciones, relaciones y estadísticas básicas capaces de responder a preguntas como qué productos se están vendiendo, y cuándo y dónde se está produciendo esta venta.

- Por **conocimiento** (*knowledge*) entendemos patrones históricos, basados en la observación del pasado y entendemos también tendencias futuras, basadas en técnicas predictivas.

Dos aspectos diferencian *data mining* de los métodos estadísticos, del análisis multivariante o de la inteligencia artificial:

- El primero es su clara orientación al negocio.

- El segundo es la existencia en el mercado de soluciones específicas de *data mining* que combinan una relación de herramientas y técnicas especialmente apropiadas para ser utilizadas en tareas de análisis de negocio y para ser integradas en procesos operativos de negocio.

La adopción de soluciones *data mining* por parte de las empresas se sitúa, normalmente, en las últimas etapas de desarrollo de la infraestructura informacional/decisional. Su implementación, siempre que haya sido exitosa, proporciona unos niveles de conocimiento del desarrollo del negocio cualitativamente

muy superior al de otras tecnologías tradicionales. Además, facilita el cierre analítico, al permitir aplicar el conocimiento adquirido dentro de la operativa diaria.

2. *Web mining*

Internet se ha convertido *de facto* en la fuente de datos públicos más grande del mundo y la tecnología web es, a su vez, el estándar para acceder e interpretar esta enorme fuente de información.

Internet tiene una serie de características que lo llevan a ser especial desde el punto de vista de la minería de datos, convirtiendo las tareas de acceso a la información y búsqueda del conocimiento en Internet en una disciplina específica dentro de la minería de datos.

Veamos algunas de las características que hacen que la minería de datos adquiera personalidad propia cuando se aplica sobre Internet.

- La **cantidad de información** en Internet y su ritmo de crecimiento son enormes; además, el ámbito de conocimiento que cubre esta información es tremendamente diverso. Podemos encontrar información sobre prácticamente cualquier tema.

- Internet trabaja con **todo tipo de datos**: tablas estructuradas, datos semiestructurados, textos no estructurados y ficheros multimedia como imágenes, vídeos y audios.

- La información de la web es, por definición, **heterogénea**. Debido a las distintas autorías de las páginas web, varias páginas pueden presentar la misma información o información similar utilizando formatos y palabras completamente distintas.

- Una cantidad significativa de información en la web está relacionada mediante **enlaces**[1], que pueden apuntar a zonas del mismo sitio web o de otros sitios. De este modo, aquellas páginas que son apuntadas por enlaces de otras páginas son páginas "autoritativas", es decir que gozan de un prestigio, puesto que mucha gente confía en ellas.

(1)En inglés, *hyperlinks*.

- La información en la web contiene ruidos. En este sentido, podemos distinguir dos tipos de ruido:
  - Una página web, aparte del contenido principal y enlaces a otras páginas, también puede contener publicidad, notas legales, etc. Desde el punto de vista del *data mining*, el reto consiste en detectar y aislar lo que es accesorio al contenido principal, es decir, la publicidad y las notas legales entre otros.

  - Debido al hecho de que Internet no tiene una gestión centralizada de la calidad de la información, puede suceder que la información que

contenga sea de baja calidad, repetida, errónea, contradictoria o incluso deliberadamente incorrecta.

- Internet es también utilizada por parte de empresas que ven en ella una forma única de acceder a su público objetivo. Ejemplos de esta interacción empresa-público objetivo pueden ser transacciones como comprar productos, pagar facturas y rellenar formularios, entre otras. Para soportar toda esta actividad se hace necesaria la **automatización de servicios**. Un ejemplo pueden ser los servicios de recomendación en el propio proceso de compra.

- Internet es un **entorno tremendamente dinámico**, los cambios se suceden constantemente. Identificar y gestionar estos cambios es una parte importante de los objetivos que persigue afrontar la minería de datos.

- Internet también es una **sociedad virtual**. No son solo datos, información y servicios, también es interacción entre agentes: personas, organizaciones y servicios automatizados. Es posible comunicarse con agentes de cualquier lugar del mundo de una forma instantánea; también es posible emitir una opinión sobre cualquier tema en los foros de Internet, blogs, redes sociales y páginas especializadas.

Todas estas interacciones generan nuevos retos para el *data mining*, que se plasman en campos de aplicación como son la minería de opiniones y el análisis de redes sociales.

Adicionalmente, cabe destacar que la minería web no debe ser vista meramente como una de las posibles aplicaciones de *data mining*. La especificidad e importancia de Internet son tales que la minería web, en ocasiones, ha tenido que desarrollar sus propios algoritmos y utilizar sus propios procedimientos.

¿Qué es *web mining*?

> *Web mining* tiene por objetivo descubrir información y conocimiento útil tomando como fuentes informacionales los enlaces, los contenidos de las páginas web y los logs de uso de los recursos de Internet.

Es cierto que el *web mining* utiliza muchas técnicas habituales en *data mining*. No obstante, debido a la heterogeneidad de la web, a las estructuras de enlaces y a los datos no estructurados, también ha tenido que desarrollar sus propias técnicas.

Estas técnicas distintivas de *web mining* aparecen por la necesidad de dar solución a una lista de tareas específicas, que podrían agruparse en las siguientes tres categorías:

- ***Web structure mining***. La minería de estructuras de relaciones de enlaces trata de conocer la estructura de la web. Por ejemplo, a través del estudio de los enlaces, podemos estimar la relevancia de las páginas web, y también podemos descubrir comunidades de usuarios que potencialmente comparten ámbitos de interés.

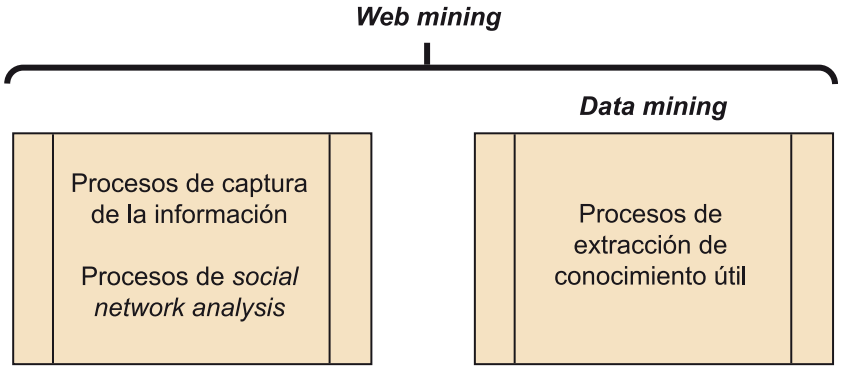
- ***Web content mining***. La minería de contenidos trata de extraer patrones a partir del contenido de las páginas web. Por ejemplo, podríamos automatizar un proceso para clasificar páginas web en función de su contenido. También podríamos extraer, de un sitio web, opiniones de los comentarios de los usuarios y descripciones de los productos que contiene.

- ***Web usage mining***. La minería del uso de Internet tiene por objetivo descubrir patrones de uso de los recursos de la red a partir de los logs que registran la actividad del usuario.

En resumen, la principal diferencia entre *data mining* y *web mining* radica en los procesos de captura de la información. Mientras que para la minería de datos tradicional las tareas empiezan a partir de que los datos ya están almacenados en un repositorio de datos, en la minería de datos web, el propio proceso de capturar los datos es una de las tareas más importantes y relevantes, puesto que supone rastrear un gran número de sitios web.

En la figura siguiente vemos esta idea, expresada de una forma gráfica.

Figura 1. *Web mining* frente a *data mining*



Fuente: Bing Liu en Web data mining

Los procesos de *social network analysis* se refieren al estudio de la relevancia de los contenidos, la relevancia de los actores participantes en las redes y el descubrimiento de comunidades en torno a un ámbito de interés.

3. *Data mining* en el entorno informacional

La función primordial de un entorno informacional[2] es conformar un área donde los datos susceptibles de aportar valor para el negocio son recopilados y almacenados con el propósito de ser analizados.

(2)El entorno informacional también se llama *entorno decisional*.

Esta colección de información corporativa deriva básicamente de los sistemas operacionales y de algunos orígenes de datos externos, tiene como misión soportar la toma de decisiones del negocio, no las operaciones del mismo, constituyéndose en el pilar sobre el que se sustentan las aplicaciones de inteligencia de negocio BI o *business intelligence*.

1) Entorno operacional frente a entorno informacional

Lo que marca la diferencia entre los entornos operacionales y los informacionales es el contexto del dato. En un entorno operacional el dato está contextualizado en torno a las aplicaciones transaccionales, mientras que en el entorno informacional el contexto es un área, o un conjunto de áreas de negocio.

Tabla 1. Entorno operacional frente a entorno informacional

Datos operacionales	Datos informacionales
Orientados a la aplicación	Orientados a un área o tema
Integración limitada	Integración total
Actualización constante	Datos no volátiles
Solo valores actuales	Valores a lo largo del tiempo
Soporte de las operaciones diarias	Soporte de las decisiones estratégicas

Este contexto marca inevitablemente como están estructurados y organizados los datos. En el entorno operacional hablamos de modelos normalizados y optimizados para garantizar una alta agilidad a la hora de procesar transacciones (inserciones, actualizaciones y borrados).

En el entorno informacional, sin embargo, la información se encuentra agregada y su principal razón de ser es agilizar las operaciones de lectura, propiciar entornos de análisis y facilitar la generación de entornos con versiones para posibilitar tareas de optimización, comparación y seguimiento.

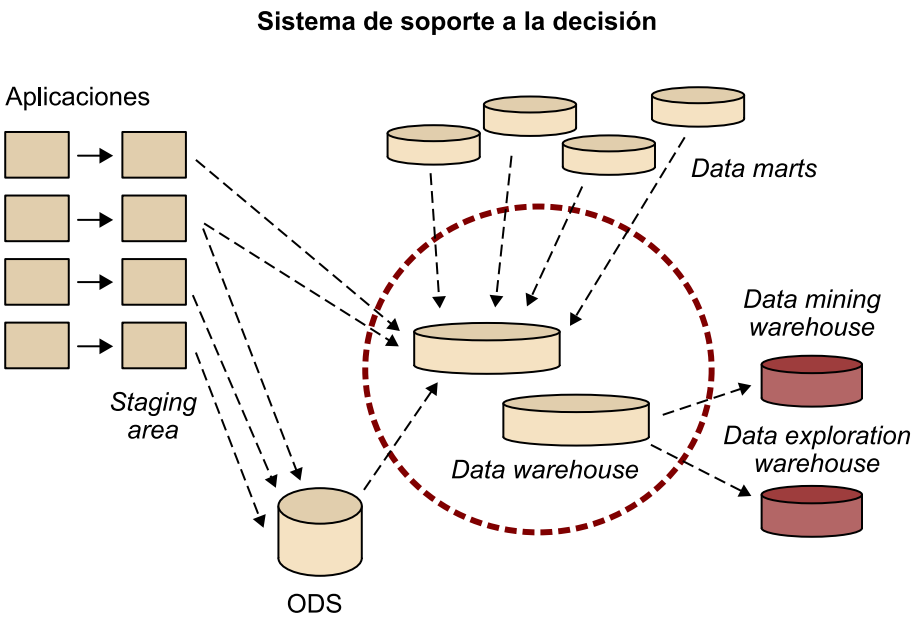
En definitiva, el entorno informacional debe proporcionar a la organización un conjunto de herramientas que den soporte a la toma de decisiones estratégicas, lo que se conoce como *decision support system* (DSS).

2) Sistemas de soporte a la decisión (DSS)

Data mining, al igual que otras tecnologías analíticas, necesita la información contextualizada alrededor de un área de negocio. Podemos hablar, por ejemplo, de un repositorio de clientes, sobre el cual realizar modelos de segmentación, propensión de compra o selección de público objetivo o bien de uno de ventas, con el suficiente nivel de granularidad como para diseñar promociones en función de la afinidad de unos productos con otros.

En la figura siguiente podemos ver cómo interactúan los distintos componentes que constituyen un sistema DSS de soporte a la decisión.

Figura 2. Arquitectura de un entorno DSS



Fuente: Bill Inmon en www.inmoncif.com

Las **aplicaciones** forman parte de la estructura operativa y son las encargadas de recoger la información que se genera en el día a día de la organización.

Ejemplos de aplicaciones pueden ser:

- *Enterprise resource planning* (ERP) para la gestión administrativa de los distintos departamentos de una organización.
- CRM para dar soporte a tareas de *help desk*, marketing, venta con equipos de movilidad o atención al cliente.
- SRM para gestionar las interacciones con los proveedores y la planificación y ejecución del aprovisionamiento.
- Gestión documental para digitalizar la comunicación en papel e integrarla con transacciones existentes en el ERP, CRM y SRM.
- Otras como el correo electrónico y aplicaciones a medida para resolver necesidades específicas de la organización.

La ***staging area*** o área intermedia es un repositorio de datos puente cuyo objetivo es facilitar las tareas técnicas de integración de datos en el *data warehouse*. Las tareas de transformación propias de los sistemas ETL (*extract, transform and load*) suelen realizarse bien justo antes de almacenar el dato en el área intermedia o justo después.

El **ODS** (*operational data sote*) es un repositorio de datos operacionales, está estructurado para ser consultado vía *query*. Reside en el *data warehouse* y es la solución para que desde este entorno se puedan consultar datos generados en un plazo de tiempo inferior o igual a un día, normalmente.

No hay que confundir *staging area* y ODS: mientras el primero está pensado como paso intermedio en las tareas de transformación o conversión de datos y, en consecuencia, no está estructurado para ser consultado vía *queries*, el segundo, está pensado y estructurado para ser consultado vía *query* y pretende dar respuesta a necesidades de consulta de datos operacionales de muy reciente creación.

Desde el punto de vista funcional, la pieza básica de la arquitectura en el entorno informacional es el *data warehouse*, que se constituye como el repositorio central de información corporativa. Esta arquitectura puede expandirse a muchos niveles con el fin de habilitar múltiples vistas de negocio construidas sobre la base proporcionada por el *data warehouse*. En este sentido, los *data marts* constituyen dichas vistas de negocio y, por lo general, vienen a satisfacer necesidades concretas de información.

De este modo, las tareas de exploración, modelización y validación que comprende la minería de datos deben realizarse en un *data mart* dependiente y a la vez integrado dentro del *data warehouse* corporativo. Esta es la forma de asegurar la coherencia y la unificación de la información que se procesa.

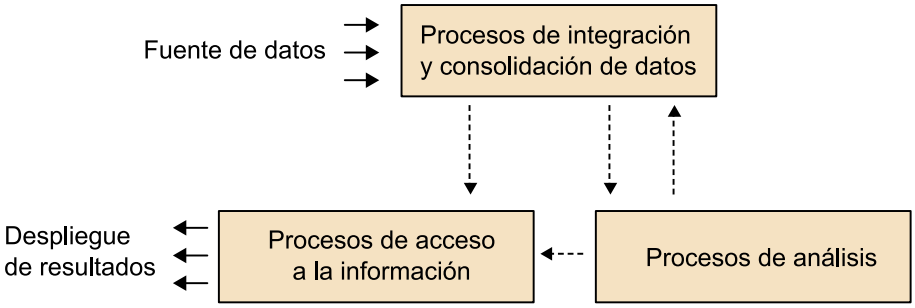
Ejemplos de necesidades corporativas que pueden ser cubiertas en un entorno *data warehouse* pueden ser:

- **Tareas financieras**, como la consolidación del balance y de la cuenta de resultados tanto contable como analítica, la presupuestación contable, la gestión de una tesorería avanzada, la gestión del riesgo con los clientes y el estudio de escenarios de viabilidad económica de nuevos proyectos.
- **Tareas comerciales y de marketing**, como la presupuestación de las ventas, los estudios sobre la actividad comercial como resultados y tendencias, el seguimiento de campañas y la gestión de la fidelización de clientes.
- **Tareas logísticas**, como la optimización de rutas, estudios de costes logísticos y seguimiento de niveles de servicio de proveedores logísticos.

3) Procesos del entorno informacional

Como parte sustancial del entorno informacional, la minería de datos no debería plantearse nunca como un área aislada. Es más, los procesos y servicios que soportan las tareas de modelización deben estar alineados con el resto de los mecanismos que regulan el entorno.

Figura 3. Procesos del entorno informacional



Básicamente, los procesos y los servicios del entorno informacional pueden clasificarse en tres categorías:

- **Procesos de integración y consolidación de datos**. Son los encargados de la transferencia de datos entre repositorios. Su función es asegurar la correcta circulación de la información entre los mismos, debiendo implementar los mecanismos de transformación de datos requeridos.

- **Procesos analíticos**. Una vez que los datos han sido consolidados en el repositorio de destino, es necesario implementar motores analíticos que permitan soportar la toma de decisiones por parte de los usuarios finales.

- **Procesos de acceso a la información**. Son los encargados de presentar y hacer llegar el conocimiento del entorno informacional a los usuarios finales.

Los servicios de gestión del entorno informacional deben considerarse dentro de la infraestructura global corporativa. Esto se hace necesario desde el momento en que su función no puede plantearse de forma aislada, y los requisitos de negocio imponen un acceso conjunto a todos los activos de información corporativa, como los repositorios de contenidos.

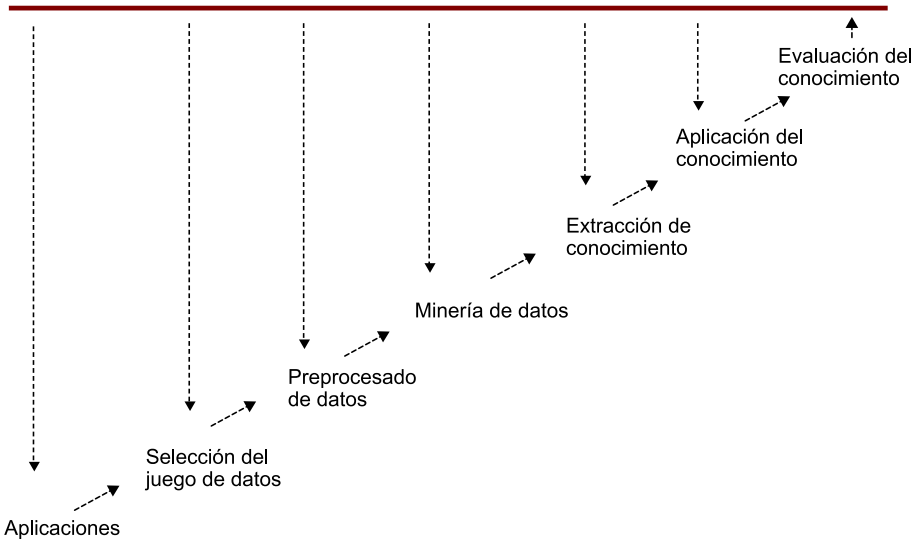
Como tal, los procesos de minería de datos se engloban dentro de los procesos analíticos, si bien requieren de los procesos de integración y consolidación tal y como se han planteado. A su vez, deben integrarse en los procesos de acceso, con el fin de facilitar el despliegue de los resultados a otras aplicaciones analíticas.

4) Proceso de extracción del conocimiento

Bajo esta concepción, los servicios de minería de datos deben proporcionar los mecanismos necesarios para la implementación y despliegue de modelos analíticos desarrollados mediante la explotación de grandes volúmenes de datos.

Tal y como muestra la figura 4, el proceso de extracción de conocimiento se subdivide en una cadena de subprocesos que comprenden la captura y transformación inicial de los datos, la aplicación de modelos *data mining* sobre los mismos, con el objetivo de obtener y extraer información de valor, es decir, conocimiento para la organización y, finalmente, la aplicación y evaluación de este conocimiento, que a su vez nos servirá para replantear, optimizar, mejorar y ampliar las tareas anteriores.

Figura 4. Extracción de conocimiento



La clave para la correcta implantación de este tipo de servicios reside en su correcta integración y automatización dentro de los procesos de negocio, sean estos informacionales u operacionales. En este sentido, uno de los principales beneficios derivados de su implantación reside, por ejemplo, en la capacidad de poder aplicar los modelos generados cerca del punto de contacto con clientes y proveedores. Esta aplicación debe poder realizarse tanto en tiempo real como según demanda, sin perjudicar en ningún caso los procesos transaccionales.

Hay que resaltar la importancia del cierre del ciclo analítico en este punto.

Nota

De todas las tecnologías analíticas empleadas en el entorno informacional, probablemente la minería de datos es la única que genera nueva información a partir de la ya existente.

Por generación de nuevos datos entendemos la información derivada por la aplicación de los modelos contra las estructuras de datos.

Algunos ejemplos de generación de nuevos datos son:

- El identificador de segmento asignado a un cliente por un modelo de segmentación. Este identificador puede ir acompañado por un indicador de calidad, que mide la bondad de la asignación, o incluso por un segundo indicador de asignación a otro segmento.
- El nivel de propensión (*scoring*) de un cliente a responder a una campaña promocional, dentro de un modelo de selección de público objetivo.
- Los identificadores de los productos relacionados con otro dentro de un análisis de asociaciones en cestas de la compra.

Además, cuando un cliente cambia de segmento de un periodo a otro, puede ser interesante desencadenar una alarma que lleve aparejada unas determinadas acciones, como por ejemplo, la inclusión del cliente dentro de una lista de contactos.

5) *Data mining* sobre entornos operacionales

Los modelos que se generan en la minería de datos comprenden una serie de parámetros que, al ser utilizados dentro de un algoritmo específico, permiten su aplicación. Precisamente, por cierre de ciclo analítico entendemos la transferencia de modelos del entorno informacional al operacional.

Con los componentes adecuados, el modelo puede ser invocado por procesos y aplicaciones según demanda, aplicándose a datos recientes que acaban de entrar en las aplicaciones.

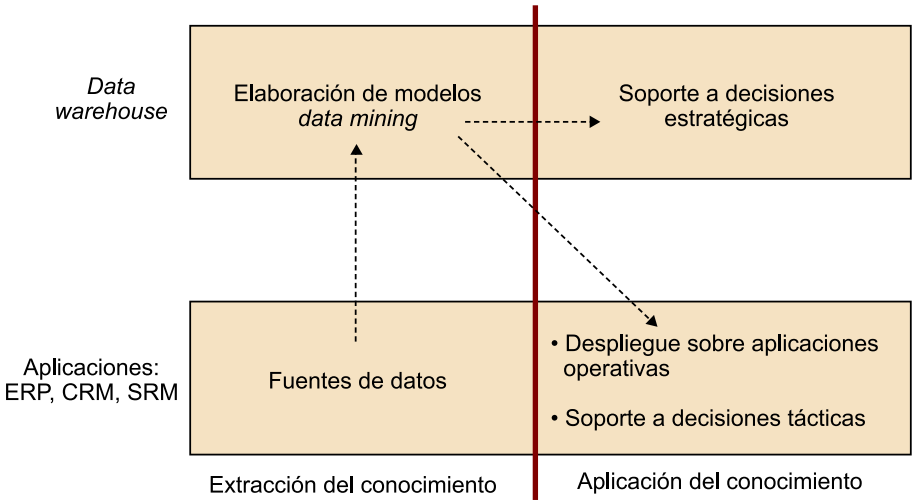
Como puede deducirse, la aplicación de modelos de minería de datos en entornos operacionales habilita las analíticas en tiempo real, es decir, el procesamiento de los datos a medida que estos se producen.

Algunos ejemplos de aplicaciones de modelos de minería de datos en entornos operacionales pueden ser:

- Personalización de ofertas a clientes en el mismo punto de venta.
- Emisión de alertas en un proceso de prestación de servicio, con el objetivo de prevenir el fraude.
- Asistencia en el proceso de grupaje de paquetería con el objetivo de optimización de rutas y maximización servicios entregados.

En la figura 5 vemos, de forma esquemática, cómo el entorno operacional y el entorno informacional interactúan constituyendo un ciclo analítico retroalimentado.

Figura 5. Cierre del ciclo analítico



Distinguimos una fase de extracción del conocimiento, que se inicia en el entorno operacional que actúa como fuente de datos para el entorno informacional, donde se elaboran los modelos de minería de datos.

Una segunda fase de aplicación del conocimiento o despliegue de modelos de minería de datos se inicia en el entorno informacional e impacta del siguiente modo:

- Por un lado, en el propio entorno informacional con la generación de conocimiento útil para la toma de decisiones estratégicas.

- Por otro lado, en el entorno operativo con el despliegue de modelos DM sobre aplicaciones operativas y también con la generación de conocimiento útil para la toma de decisiones tácticas.

Finalmente, una tercera fase de evaluación del conocimiento nos obligará a recalibrar modelos, a replantear objetivos, a ajustar e incluso a cambiar el despliegue de modelos. Todo ello con el objetivo de repercutir el conocimiento adquirido en beneficio de la organización.

3.1. *Scoring* de un modelo *data mining*

Se denomina *scoring* de un modelo a la aplicación del mismo a un conjunto de datos y la consecuente obtención de los resultados producidos.

Dependiendo del tipo de modelo, este producirá unos resultados u otros. La tabla 2 resume las métricas más usuales que se pueden obtener dependiendo del tipo de modelo.

Tabla 2. Métricas obtenidas al aplicar ciertos tipos de modelos

Tipo de modelo	Algoritmos más comunes	Resultados obtenibles – Métricas
Segmentación	Basados en distribuidores (Condorcet)	Identificador de segmento
	Basados en controides (Kohonen)	Confianza de asignación al segmento
		Calidad de asignación al segmento
Clasificación	Árboles de decisión	Clase predicha
	Redes neuronales	Confianza de asignación a la clase
	Regresión logística	Nodo de asignación
Predicción	Redes neuronales	Valor predicho
	Funciones de base radial (RBF)	Límites de confianza
		Desviación estándar de la predicción
Asociaciones	A priori	Listado de ítems relacionados (antecedentes y consecuentes)
	EstMerge	

Los modelos de asociaciones han sido vistos, tradicionalmente, como modelos puramente descriptivos. Es decir, a partir de su análisis se podrían obtener relaciones de productos o clases que eran adquiridos de forma conjunta por los clientes (por citar una de las aplicaciones más frecuentes).

Sin embargo, dada una entrada consistente en un artículo o conjunto de artículos, el modelo puede llegar a producir un resultado consistente en una relación de otros artículos que son adquiridos de forma conjunta. En este caso, los parámetros del modelo, como el soporte, la confianza o la elevación (*lift*), son empleados para la generación de esta selección.

Nota

Los modelos de asociaciones, muchas veces, son capaces de generar un gran número de reglas, de modo que se hace necesario saber cuáles de ellas son realmente significativas y tienen algún interés. En este sentido se hacen necesarias medidas como el soporte, la confianza y la elevación de una regla.

El **soporte** de un conjunto de valores podemos entenderlo como la proporción de transacciones de un juego de datos que contiene dicho conjunto de valores.

La **confianza** de dos conjuntos de valores puede entenderse como un estimador de la probabilidad de encontrar transacciones que contengan el primer conjunto de valores dentro de las transacciones que también contienen el segundo conjunto de valores.

Un conjunto de valores con **elevación** distinta de 1 nos indica que este conjunto de valores aparece en el juego de datos una cantidad de veces superior o inferior a lo esperado, entendiendo siempre bajo condiciones de independencia. Este hecho nos indicará que subyace una relación "no normal o no habitual" en el conjunto de valores.

Es evidente que para hacer *scoring* de un modelo, este debe contener la información necesaria para la generación de los resultados.

En definitiva, para aplicar un modelo contra nuevas estructuras de datos necesitamos:

- Un formato para poder definir los metadatos o métricas, de forma compacta y, sobre todo, altamente transferible entre aplicaciones. Los metadatos podrían ser:
  - Número, nombre, tipo y función de los campos de entrada al modelo. Es lo que se conoce como la firma del modelo (*signature*).
  - Información del tipo de modelo y algoritmo empleado.
  - Parámetros de ejecución del modelo (coeficientes, umbrales, estadísticas, etc.).
  - Número de segmentos, calidad, bondad y precisión del ajuste del modelo (según la tipología).
- Un motor que, dados estos metadatos, pueda interpretarlos y aplicarlos a nuevos datos, obteniendo los resultados esperados.

En función del punto de nuestra infraestructura tecnológica en el que ejecutamos el *scoring* de modelos, podemos distinguir diversas estrategias:

- **Uso aislado**. Es una estrategia típicamente utilizada por parte de organizaciones que se inician en el mundo de la minería de datos. En este caso, la minería de datos es un fin en sí misma. Sirve para extraer conocimiento, pero este no está integrado con el resto del entorno ni analítico ni operacional.
- **Integración con el modelo analítico**. En este caso, la información generada por los modelos es revertida al entorno informacional. Es evidente que este escenario es mucho más óptimo, en el sentido que el resto de analistas del negocio pueden reaprovechar los resultados generados por los modelos para otro tipo de investigaciones.

  Un ejemplo puede ser la segmentación de clientes. Al generar nuevas categorizaciones, estas pueden emplearse en nuevos modelos de análisis.

- **Cierre del ciclo analítico**. En este caso, los modelos no solo se aplican en el entorno informacional, sino que son exportados a los entornos operacionales, donde se explotan según demanda y/o en tiempo real.

Un ejemplo puede ser el caso del cupón descuento. Al negocio le interesa otorgar cupones solo a aquellos clientes que tengan mayor propensión de usarlos. Con este fin, es posible construir un modelo que, basándose en patrones de consumo históricos y el contenido de las transacciones más recientes, sea capaz de estimar la probabilidad de que se utilice un cupón descuento en su próxima visita. Una vez validado, el modelo puede ponerse en producción dentro del entorno transaccional. En el momento en que una transacción pase por el terminal, el modelo es aplicado. Para su aplicación se tendrá en cuenta la información histórica del cliente, así como el contenido de la propia transacción. Finalmente, si el modelo devuelve una propensión alta, el propio terminal imprimirá el cupón para el cliente.

- **Automatización del ciclo analítico**. El entorno operacional no es solamente un punto de aplicación, sino incluso de modelización. La idea es que, de forma planificada y, en principio, no supervisada, puedan construirse modelos a medida que se van realizando transacciones. Hoy en día solo algunos tipos de modelos pueden generarse bajo estas condiciones. Este es el caso de los modelos de asociaciones de compra.

Un ejemplo de este caso pueden ser las asociaciones de compra. Sobre la base de las transacciones realizadas en las tres primeras horas del día, se quiere generar un modelo capaz de listar una serie de promociones con el fin de dar salida a productos perecederos. En el caso de tiendas en línea, este proceso podría automatizarse por completo.

3.2. Servicios de minería de datos dentro de la infraestructura tecnológica

Desde un punto de vista funcional, es conveniente distinguir entre cuatro tipos de servicios:

- **Servicios de preparación y acondicionamiento de datos**. Se trata de todas aquellas tareas necesarias para garantizar que los datos lleguen a la fase de modelado con la calidad suficiente, es decir, formatos adecuados, variables significativas dentro del juego de datos, volumen de datos asumible... Esta funcionalidad debe estar cubierta e integrada dentro de los servicios genéricos de extracción, transformación y carga[3], así como los de integración y consolidación de la información.
  Con el fin de agilizar y automatizar al máximo la etapa posterior de modelización, es recomendable consolidar un repositorio de datos unificado y con los niveles de agregación necesarios, que permita una generación rápida y fiable de modelos.

[(3)]La sigla con la que se conocen estos servicios es ETL, de *extraction, transformation and loading*.

- **Servicios de modelización**. Se trata de la aplicación de un algoritmo de minería de datos sobre un juego de datos con el objetivo de extraer conocimiento del mismo. Dependiendo del volumen de datos que deba ser tratado, así como de su complejidad, la fase de modelización puede llegar a ser computacionalmente muy intensiva. Por este motivo, y con el fin de evitar la modelización sobre muestras por el riesgo que esto conlleva, es necesario disponer de un motor de modelización que trabaje dentro del mismo gestor que contiene los datos. De esta forma se evitarán procesos de extracción intermedios que inevitablemente redundan en una pérdida de escalabilidad.

Estos servicios deben ofrecer una alta flexibilidad a la hora de diseñar, parametrizar y planificar la ejecución de los modelos.

- **Servicios de visualización**. La validación de modelos es un aspecto fundamental de cara a asegurar su calidad y aplicabilidad dentro del negocio. Por este motivo, se hace necesario disponer de herramientas gráficas de visualización que permitan a los analistas de negocio validar los modelos obtenidos y analizar sus resultados.
  Los subservicios de visualización deben poder integrarse y ser invocados desde las mismas herramientas de usuario final empleadas en los servicios de análisis de la información. También deben ser compatibles con cualquier tipo de modelo con independencia de la tecnología con la que ha sido generado.

- **Servicios de aplicación (*scoring*)**. La puesta en producción de los modelos y su aplicación automatizada dentro de los procesos de negocio es la etapa final y más importante. En este sentido, el entorno de producción puede ser el informacional, donde la aplicación del modelo se llevará a cabo básicamente por lotes, o el operacional, donde la invocación y respuesta de los modelos deberá realizarse en tiempo real en la mayoría de los casos.

Incluso puede darse la circunstancia de que dentro de una misma infraestructura tecnológica, tengamos distintos servicios *data mining* implementados con soluciones software de distintos fabricantes. En este sentido, el lenguaje estándar PMML nos deberá ayudar a comunicar distintas soluciones *data mining*.

Nota

Predictive modeling markup language, basado en XML y desarrollado por Data Mining Group. Su objetivo es proporcionar un estándar para la construcción y distribución de modelos de minería de datos, facilitando así la intercomunicación entre aplicaciones de minería de datos elaboradas por distintos fabricantes.

PMML juega un papel importante en estos escenarios, puesto que facilitará enormemente las tareas de comunicación e integración entre aplicaciones.

La función del PMML es proporcionar un mecanismo estandarizado para la definición de modelos, con independencia de la herramienta específica con la que fueron generados. Respecto del soporte del PMML, pueden distinguirse entre dos grupos de aplicaciones:

- **Productoras de PMML**. Son aquellas con capacidad de generar modelos de minería de datos en formato PMML o con un formato propietario que después puede exportarse al estándar.

- **Consumidoras de PMML**. Una aplicación consumidora no tiene capacidad para construir modelos en PMML, pero sí para procesarlos. Este procesamiento puede consistir en su aplicación o *scoring*, o bien simplemente en su visualización y representación.

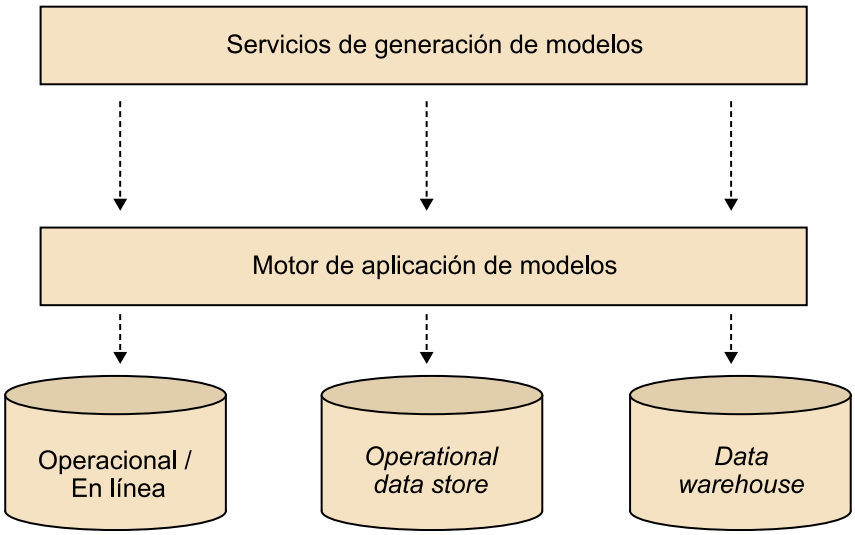
Es evidente que pueden existir aplicaciones que sean conformes al estándar PMML en los dos sentidos.

En general el mecanismo de funcionamiento sería el siguiente. La aplicación generadora de PMML puede exportar el modelo, un fichero que posteriormente es insertado dentro de una tabla de catálogo de la base de datos, por parte de la aplicación consumidora de PMML. Una vez allí, los servicios de *scoring* del gestor, una vez invocados, se encargan de procesar el modelo y ejecutarlo contra los datos de aplicación previamente especificados. El resultado de esta ejecución es almacenado dentro de la base de datos o transferido a otro proceso.

3.3. Escenarios para la puesta en producción de modelos

La figura 6 esquematiza distintos escenarios para el desarrollo y puesta en producción de modelos de minería de datos.

Figura 6. Escenarios de puesta en producción de modelos de minería de datos



Una vez superadas las fases de generación y evaluación de los modelos *data mining*, estos deben transferirse del entorno de desarrollo al de aplicación o *scoring*, donde se llevará a cabo su puesta en producción. En este entorno necesitaremos un motor de aplicación que sea capaz de procesar estos modelos y aplicarlos de forma planificada (por lotes), bajo demanda o tiempo real.

Básicamente podemos hablar de tres repositorios de datos donde se pueden aplicar los modelos:

- ***Data warehouse* (DW)**. La aplicación de modelos en este repositorio habilita la diseminación de los resultados de la aplicación del modelo dentro del entorno informacional. Normalmente, y debido a la llegada planificada de nueva información al *data warehouse*, la ejecución de modelos se lleva a cabo por lotes, de forma sincronizada con la carga.
  El tipo de aplicaciones que se llevan a cabo en este entorno son de carácter analítico, encaminadas a la investigación, por ejemplo, del comportamiento de clientes, selección de público objetivo para campañas..., es decir, no requieren una inmediatez desde el punto de vista de aplicación del modelo y acción directa de sus resultados.

- **Almacén de datos operacional (ODS)**. Por su naturaleza, un ODS es un repositorio a medio camino entre el entorno operacional y el informacional. Por ello, debe soportar una carga de transacciones y al mismo tiempo atender a peticiones de consulta sobre los datos. La aplicación de modelos de minería de datos en este entorno debe realizarse sobre información con una cierta profundidad histórica (dos o tres meses, dependiendo de la naturaleza del repositorio) y sobre datos recientes que acaban de ser replicados del entorno operacional.

- **Entorno operacional (OLTP)**. Respecto a la minería de datos, el entorno transaccional es de tiempo real. Es importante resaltar que una aplicación operacional debe centrar su rendimiento en el procesamiento de transacciones, las cuales, dependiendo del tipo de negocio, pueden ser muy voluminosas y constantes. De modo que hay que tratar de evitar perjudicar el rendimiento de este entorno. Una solución es que sea el propio gestor de la base de datos el encargado de aplicar los modelos.

Procesos de negocio generados automáticamente

La no inmediatez de la aplicación de los resultados no quiere decir que la ejecución del modelo no desencadene otros procesos de negocio de forma automática. Por ejemplo, es posible que la detección de un cambio de segmento de un cliente respecto del mes pasado desencadene una alerta, la cual puede ser enviada a un sistema de gestión de eventos.

Ejecución por lotes o en tiempo real

Las ejecuciones de modelos pueden ser por lotes o en tiempo real. Un ejemplo del primer caso se da en aplicaciones de atención al cliente (*call center*), en las que se desea ofrecer promociones, descuentos o gestionar créditos financieros. Respecto al segundo, un entorno de análisis de fraude o de blanqueo de capitales requerirá una aplicación de modelos más planificada.

Cuando hablamos de la aplicación de modelos en tiempo real sobre transacciones, hay que distinguir dos posibilidades:

- **Transacciones persistentes**. En este caso, la transacción ha sido previamente insertada en la base de datos. Sería el caso de la aplicación de un modelo de generación de cupones. Cuando aplicamos un modelo sobre una transacción persistente, el motor de *scoring* debe leer el registro de la base de datos, ejecutar el modelo y transmitir los resultados al proceso adecuado.

- **Transacciones no persistentes**. Esta situación se da cuando se quiere aplicar un modelo contra una transacción que no es registrada en la base de datos. Podemos encontrar un buen ejemplo en la personalización en Internet, donde el historial de navegación de los usuarios no tiene por qué estar registrado en la base de datos.

Es importante remarcar que la persistencia o no de las transacciones, se refieren al momento de aplicar el modelo (*scoring*), es decir, una transacción persistente será aquella que debe acceder a la base de datos en el momento de aplicar el modelo, y no posteriormente.

Dicho de otro modo, transacción persistente es aquella en la que el modelo debe acceder a la base de datos para obtener los datos de entrada, y transacción no persistente es aquella en la que el modelo no necesita acceder a la base de datos para obtener los datos de entrada. Sin embargo, en ambos casos, una vez aplicado el modelo, podríamos almacenar en la base de datos el resultado del *scoring*.

3.4. Soluciones tecnológicas DM

El conjunto de herramientas y aplicaciones de minería de datos puede clasificarse en dos categorías:

1) Entornos de modelización integrados (*workbench*). Este tipo de aplicaciones ofrecen a los usuarios y analistas de minería de datos un entorno integrado en el cual realizar las tareas de acondicionamiento de datos, modelización, validación y aplicación de modelos. La arquitectura acostumbra a ser a tres niveles:

- Servidor de datos (el repositorio o *datamart*).
- Servidor de modelización y aplicación, donde se lleva a cabo la construcción y validación de modelos.
- Entorno de cliente, dotado de una interfaz gráfica de usuario.

Estas aplicaciones suelen incorporar una serie de bibliotecas estadísticas que expanden las capacidades de análisis del entorno. Las características que mejor definen los entornos *workbench* son las siguientes:

- Los modelos son creados por un analista experto, normalmente a través de un entorno de modelización integrado. El conocimiento de este entorno suele tener una curva de aprendizaje considerable.
- La creación y refinamiento del modelo requiere una importante parametrización de los algoritmos empleados.
- Las conclusiones derivadas del modelo deben transmitirse al resto de usuarios de negocio involucrados.
- La puesta en producción del modelo suele hacerse, ya sea a través de la generación de código fuente, o bien a través de un servicio de *scoring*.

2) Servicios ligados al gestor de la base de datos (*in-database-data mining*). En este caso, es el propio motor relacional el encargado de implementar los servicios de minería de datos, lo cual redunda en una mayor escalabilidad y rendimiento. La arquitectura de estas aplicaciones suele ser a dos niveles:

- Servidor de datos con servicios de minería de datos integrados de forma nativa mediante tecnologías como PL/SQL.

- Entorno de usuario con herramientas cada vez más similares a las que podríamos encontrar en un entorno *workbench*, con la posibilidad de crear modelos, probarlos, aplicarlos y hacer el seguimiento de todo el proceso. Todo ello con una visión metodológica como la que podríamos encontrar en CRISP-DM.

Entre las limitaciones de este planteamiento cabe destacar que no acepta todas las tipologías de modelos. Entre las que sí acepta podemos destacar, a título de ejemplo, los modelos de segmentación de clientes, los modelos de categorización de puntos de venta, los análisis de cestas de la compra y los modelos para selección de público objetivo para promociones.

Las características principales de los entornos ligados al gestor de la base de datos son los siguientes.

- La capacidad de generar y aplicar modelos se hace transparente, habilitándose dentro del actual entorno de acceso a la información por parte de los usuarios.

- Se emplean herramientas comunes para parametrizar y ejecutar los modelos: generación de informes, OLAP, hojas de cálculo...

- El resultado de la aplicación del modelo se integra totalmente con los informes existentes o aplicaciones de cuadros de mando.

- Al reaprovechar los mecanismos de acceso existente, la curva de aprendizaje es mínima.

- Los analistas de negocio pueden realizar los procesos de modelización y aplicación.

- La minería de datos incrustada necesita una implementación basada en servicios, ya que es necesario integrarla con las herramientas de acceso a la información.

Como resumen, podemos decir que cada aproximación tiene sus ventajas y sus inconvenientes, de modo que en ningún caso se trata de sustituir las soluciones basadas en entornos de modelización integrados, ya que estos aportan funcionalidades insustituibles. Incluso, dependiendo del entorno, ambas aproximaciones tecnológicas pueden convivir perfectamente.

4. Presentación del lenguaje R

Para seguir este apartado es necesario que el estudiante ejecute en una pantalla de comandos R el código que se va exponiendo a lo largo del apartado. Consideramos que seguir el material didáctico y al mismo tiempo poner en práctica el código R que se propone es la mejor forma de asimilar la materia trabajada.

4.1. Proyecto R

R es un software orientado a proporcionar funcionalidades de manipulación de datos, cálculo y visualización de gráficos. Se trata de un software libre que se distribuye bajo licencia GNU. Tiene sus orígenes en otro lenguaje 100% orientado a la estadística llamado S.

Actualmente R es un estándar muy utilizado por la comunidad científica y cada vez más utilizado por parte de empresas que requieren un análisis intensivo de datos. Los principales aspectos que destacamos de R son los siguientes:

- Contiene multitud de paquetes adicionales (*packages*) fácilmente instalables y que permiten incorporar funcionalidades nuevas y organizadas por ámbitos de aplicación como, por ejemplo, algoritmos de regresión, árboles de decisión, reglas de asociación, algoritmos de segmentación, etc.

- El código de R es de libre acceso, de modo que es posible interpretar el mecanismo de funcionamiento de las implementaciones de algoritmos, modificarlas y de este modo adaptarlas a distintas necesidades.

- Es posible comunicarse con bases de datos como PostgreSQL, Oracle y MySQL, importar y exportar datos de Microsoft Excel, generar gráficos de distintos tipos y guardarlos en formatos pdf, jpg, bmp, etc.

- Dispone de un entorno de trabajo básico y orientado a comandos, pero también de entornos más visuales como Rattle GUI totalmente orientado a procesos de minería de datos.

- Dispone de paquetes específicos para acceder a algunos de los grandes generadores de datos de Internet como son Twitter, Google y Facebook.

En definitiva, R reúne las condiciones para ser considerado un software ideal para fines formativos en el ámbito de la minería de datos.

Veamos cómo instalar R y cómo ampliar sus funcionalidades mediante la carga de nuevos paquetes.

4.1.1. Descargar e instalar R

1) Convenciones

En adelante, enmarcaremos en una tabla de color gris todo el texto relativo a la consola de comandos R. Dentro de este entorno se utilizarán los siguientes símbolos:

- Al símbolo # le seguirán explicaciones sobre los comandos R que se invocan en la línea siguiente.
- Al símbolo > le seguirán comandos R.
- Las líneas con letras en azul representan el resultado mostrado por la consola R después de haber ejecutado un comando R.
- El lenguaje R es *case-sensitive*, es decir, distingue entre mayúsculas y minúsculas. Será importante tenerlo en cuenta cuando utilicemos la línea de comandos R.

2) Descargar e instalar R

Para descargar e instalar R deberemos acceder a la página web oficial del proyecto R en http://www.r-project.org/.

En la sección de descargas se nos redirigirá a la página CRAN (*Comprehensive R Archive Network*) para escoger el servidor de descargas R más cercano. CRAN es una red de servidores replicados (*mirror servers*) ideada para facilitar los procesos de descarga del propio software R y de los paquetes que lo complementan.

El proceso de instalación sobre el sistema operativo Microsoft Windows es tan sencillo como descargarse el fichero .exe y ejecutarlo en local.

En el proceso de instalación, R nos ofrece la posibilidad de indicar el directorio donde querremos almacenar los paquetes que se vayan instalando posteriormente.

Debemos comentar también que el código R presente en este material didáctico funciona a partir de la versión 2.15.3 de R.

3) Instalar un paquete nuevo

Se trata de una tarea que se deberá hacer con cierta frecuencia, puesto que los algoritmos, gráficos y otras funcionalidades que utilizaremos en este material didáctico se encuentran almacenadas en distintos paquetes.

Como ejemplo instalaremos el paquete Rcmdr, que nos dotará de un entorno GUI pensado para realizar tareas estadísticas.

```
# Ejecutamos el proceso de instalación del paquete Rcmdr
 > install.packages("Rcmdr")
# R nos pedirá que escojamos un Servidor CRAN para iniciar la descarga
```

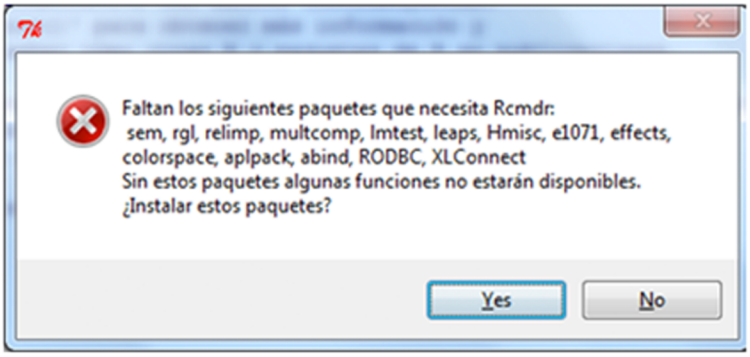
R se encarga de conectarse al servidor CRAN, de descargarse el paquete en formato comprimido, de descomprimirlo y de ubicarlo en el directorio que hayamos escogido para almacenar todos los paquetes que se vayan instalando.

```
# Para poder utilizar los objetos del paquete Rcmdr necesitamos cargarlo en memoria
 > library(Rcmdr)
```

Cuando cargamos el paquete, R verifica si hay dependencias con otros paquetes, es decir, puede ocurrir que un paquete utilice objetos definidos en otros paquetes no instalados todavía en nuestro ordenador.

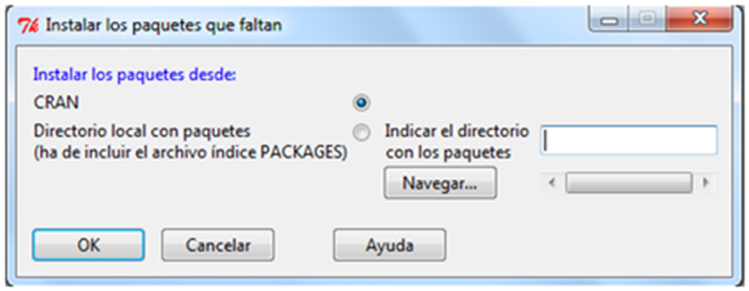
Si se diera el caso, R nos avisará y nos dará la opción de instalar los paquetes dependientes. Podéis ver el mensaje de aviso en la figura 7.

Figura 7. Notificación de paquetes requeridos



Si aceptamos, se iniciará el proceso de descarga e instalación de los paquetes dependientes. Simplemente aceptaremos el mensaje de la figura 8.

Figura 8. Instalación de paquetes dependientes



Con esto habremos conseguido instalar el paquete principal y todos sus paquetes dependientes.

Los siguientes comandos nos darán información sobre nuestra instalación.

```
# Lista de paquetes instalados
> library()
# Consultar los directorios donde R irá a buscar los paquetes instalados
> .libPaths()
[1] "C:/Users/user/Documents/R/win-library/2.15"
[2] "C:/Program Files/R/R-2.15.2/library"
```

En la tabla 3 tenéis la lista de los paquetes que necesitaremos tener instalados para seguir este material didáctico.

Tabla 3

Paquete	Ámbito de aplicación
party	Algoritmos de clasificación como los árboles de decisión
RPostgreSQL	Interacción con una base de datos PostgreSQL
foreach	Implementación de funcionalidades iterativas como bucles
arules	Algoritmos generadores de reglas de asociación
arulesViz	Gráficos para algoritmos de asociaciones
pmml	Genera el modelo en formato PMML
tm	Funciones para *text mining*
e1071	Algoritmo *support vector machines*
nnet	Implementa algoritmos de redes neuronales
FSelector	Sobre la ganancia de la información

4) Opciones de ayuda

En las direcciones que tenéis a continuación encontraréis la documentación oficial, en formato web o pdf, para las siguientes cuestiones:

- Procesos de instalación y administración:
  http://cran.es.r-project.org/doc/manuals/R-admin.html
  http://cran.es.r-project.org/doc/manuals/R-admin.pdf

- Introducción a R:
  http://cran.es.r-project.org/doc/manuals/R-intro.html
  http://cran.es.r-project.org/doc/manuals/R-intro.pdf

Adicionalmente y desde la misma línea de comandos podremos invocar pantallas de ayuda sobre objetos concretos.

Tabla 4

Función R	Descripción
help.start()	Mostrará un tutorial de R.
library(help=party)	Información sobre un paquete R. Versión, autor, lista de objetos, etc.
help(package=party)	Información en formato web sobre un paquete.
vignette('party')	Una viñeta es una documentación extra y específica para un tema en concreto.
RSiteSearch("function ctree")	Lanza una búsqueda en la web, por ejemplo, sobre la función *ctree*.
example('kmeans')	Muestra ejemplos sobre una función (en este caso, *kmeans*)

Finalmente queremos apuntar el esfuerzo que desde la UOC se está haciendo para promocionar el uso del lenguaje R como herramienta de aprendizaje de la minería de datos.

Enlace recomendado

http://data-mining.business-intelligence.uoc.edu/

4.1.2. Instrucciones básicas

A continuación detallaremos lo que consideramos que podría ser una lista de comandos básicos para poder realizar tareas relacionadas con la manipulación de juegos de datos.

1) Funciones para gestionar el entorno de trabajo

Las siguientes funciones nos permitirán interactuar con el entorno de trabajo, realizando acciones como listar los objetos R utilizados, valores por defecto o guardar texto y gráficos.

Tabla 5

Función R	Descripción
ls()	Lista los objetos cargados en memoria. La función *objects()* hace lo mismo.
vignette()	Devuelve las viñetas disponibles en los paquetes instalados.
apropos("zoo", mode="function")	Lista de funciones disponibles en un paquete
data()	Devuelve todos los juegos de datos presentes en los paquetes instalados.
getwd()	Muestra el directorio de trabajo por defecto.
setwd("mi.directorio")	Fija un directorio de trabajo por defecto.
history()	Devuelve las últimas 25 instrucciones R.
save.image("mi.fichero")	Guarda el entorno de trabajo.
save(objectlist, file="mi.fichero")	Guarda la lista de objetos concretos.
load("mi.fichero")	Carga el entorno de trabajo en la sesión actual.

2) Entradas y salidas de R

Por defecto, cuando cargamos una sesión R, se establece por defecto que las salidas se dirigen a la pantalla y que las entradas se recibirán por el teclado. A veces nos interesará alterar este comportamiento. Veamos cuándo nos interesará y cómo podemos hacerlo.

- La función *source("RUOC_01.R")* carga y ejecuta las instrucciones contenidas en el fichero, en la sesión actual. El sistema asume que el fichero se encuentra en el directorio de trabajo por defecto.

- La función *sink("mi.fichero")* redirige la salida de texto al fichero especificado. Por defecto, si el fichero existe, el contenido se sobrescribe. La opción *split=TRUE* enviará la salida de texto tanto al fichero como a la pantalla. La función *sink()* sin argumentos devuelve la salida de texto a la pantalla.

- Las funciones siguientes envían los gráficos a fichero en lugar de enviarlos a pantalla. Para volver a enviar gráficos a pantalla utilizaremos la función *dev.off()*.
  - *pdf("mi.fichero")* para formato pdf.
  - *png("mi.fichero")* para formato png.
  - *jpeg("mi.fichero")* para formato jpeg.
  - *bmp("mi.fichero")* para formato bmp.
  - *win.metafile("mi.fichero")* para formato Windows metafile.
  - *postscript("mi.fichero")* para formato PostScript.

Nota

En el aula de la asignatura se proporcionarán los *scripts* que se irán introduciendo en formato fichero de texto. El estudiante podrá ejecutarlos en su sesión de trabajo R mediante la función *source("RUOC_nn.R")*. Previamente nos habremos descargado el fichero *RUOC_nn.R* en el directorio de trabajo R establecido por defecto mediante la función *setwd()*.

3) Trabajando con datos

La forma más directa de disponer de datos en R es mediante las asignaciones de valores a una variable. Veamos algunos ejemplos.

```
# Asignamos valores numéricos a la variable vnum
> vnum <- c(7,3,5,8)
# Visualizamos el contenido de la variable vnum
 > vnum
[1] 7 3 5 8
# Visualizamos el tipo de datos que contiene nuestra variable
> vnum[0]
numeric(0)
# Visualizamos el valor 3 de nuestra lista
> vnum[3]
[1] 5
```

Otra forma de obtener datos es por la vía de la importación de datos procedentes de un fichero en formato csv.

Cuando se importan datos con la función *read.csv*, R utiliza un tipo de variable para almacenar los datos, llamada *data.frame*.

```
# Verificamos cuál es nuestro directorio de trabajo por defecto
> getwd()
# Opcionalmente podemos fijar nuestro directorio de trabajo
> setwd("C:/Users/A/Documents")
# Generamos un fichero en formato csv y lo almacenamos en el directorio de trabajo
# Guardamos en la variable vcsv el contenido del fichero ejemplo.csv
> vcsv <- read.csv(file="ejemplo.csv",head=TRUE,sep=",")
# Visualizamos el contenido del data.frame vcsv
> vcsv
  Cliente unidades valor
1      C1     12.0   120
2      C2     10.0   140
3      C3      6.0    80
4      C2      7.0   100
5      C1      9.5   130
6      C3      4.0   110
```

Sobre una lista de valores podemos hacer referencia a distintas partes de la misma.

```
# Visualizar los valores de la columna Cliente
> vcsv$Cliente
[1] C1 C2 C3 C2 C1 C3
Levels: C1 C2 C3
 # Visualizar los nombres de las columnas
> names(vcsv)
[1] "Cliente"  "unidades" "valor"
# Visualizar los nombres de las filas
> row.names(vcsv)
# Visualizamos todas las filas de una columna
 > vcsv[,2]
[1] 12.0 10.0  6.0  7.0  9.5  4.0
# Visualizamos todas las columnas de una fila
> vcsv[1,]
  Cliente unidades valor
1      C1       12   120
```

4) Tipos de datos

Antes de ampliar conocimientos en R, merecerá la pena dedicar un apartado a los tipos de datos que distingue R. La correcta comprensión de este apartado proporcionará al estudiante un buen fundamento para progresar en el manejo del lenguaje.

En R existen los siguientes tipos de datos básicos:

- **Enteros**: 1; 2; 3...

- **Numéricos**: 0.1; 0.2; 0.3...
- **Caracteres**: "a"; "b"; "c"...
- **Complejos**: 1+i; 2+i; 3+i...
- **Lógicos**: TRUE; FALSE

Asimismo, en R existen los siguientes tipos de datos compuestos:

- **Vector**: es una colección de datos del mismo tipo básico. Podríamos imaginarlos como matrices de una sola dimensión.

- **Factor**: es un vector que además almacena información sobre el nivel de agrupación de los valores que lo forman.

- **Matriz**: es un objeto de datos de 2 dimensiones (filas y columnas), todos ellos de un mismo tipo básico de datos.

- ***Array***: es un objeto de datos de más de 2 dimensiones. Puede pensarse como capas de matrices. Al igual que las matrices, solo puede contener datos de un mismo tipo básico.

- ***Data Frame***: es una matriz, es decir, un objeto de datos de 2 dimensiones, con la diferencia de que acepta datos de distinto tipo básico, es decir, un *data frame* puede mezclar por ejemplo números y caracteres.

- **Tabla**: es una colección de pares de valores.

Veamos un ejemplo de cada tipo:

```
# Ejemplos de tipos de datos compuestos
> mi.vector <- c(1,2,3,4)
> mi.factor <- factor(c(1,2,3,4))
> mi.matriz <- matrix(1:20, 2, 10, byrow = F)
> mi.array <- array(1:20, c(2,5,2))
> mi.data.frame <- data.frame(product = c("P1","P2","P3"), ventas = c(10,20,30))
```

Los siguientes dos comandos nos darán información acerca de la composición de las variables creadas en R.

```
# Podemos también verificar el tipo de cada variable
> attributes(mi.data.frame)
$names
[1] "product" "ventas"
$row.names
[1] 1 2 3
$class
[1] "data.frame"
```

```
# Visualizamos un estudio resumen del contenido del data.frame vcsv.
  Observar que el resumen lo da vector a vector.
> summary(vcsv)
 Cliente    unidades            valor
 C1:2    Min.    : 4.000    Min.   : 80.0
 C2:2    1st Qu. : 6.250    1st Qu.: 102.5
 C3:2    Median  : 8.250    Median : 115.0
         Mean    : 8.083    Mean   : 113.3
         3rd Qu. : 9.875    3rd Qu.: 127.5
         Max.    : 12.000   Max.   : 140.0
```

5. Juegos de datos

En este apartado presentaremos por orden de aparición los juegos de datos con los que trabajaremos en los apartados posteriores.

1) Contrato de suministro de consumibles

El juego de datos *ContratoSuministro.csv* proporcionado en el material didáctico, consta de 150 filas y de 5 columnas. Nos ofrece datos sobre una empresa ficticia cuya actividad económica consiste en suministrar consumibles de maquinaria.

Para ello la empresa Suministros y Componentes firma contratos de aprovisionamiento de suministros con sus clientes. En la tabla 6 se recoge su actividad comercial del último ejercicio.

Tabla 6. Estructura de datos para *ContratoSuministro.csv*

Cliente	ConsAnual	DevolAnual	Litigios	ConsUltTrim	Decision
	Consumo de suministro anual	Devoluciones de clientes	Devoluciones en litigio, es decir, no aceptadas por la empresa	Consumo de suministro del último trimestre	Decisión de renovación, ampliación o cancelación de contrato por parte del cliente

2) Segmentación de clientes

El juego de datos *Cust_Master*, que se construye sobre una base de datos PostgreSQL en el apartado "Ganancia de la información" de este material didáctico, consta de 152 filas y de 5 columnas. Nos ofrece datos sobre una empresa ficticia cuya actividad económica consiste en la venta y distribución de productos alimentarios.

La empresa Ventydis Alimentaria dispone de un portafolio de clientes a los que sirve un catálogo de productos alimentarios de forma regular. Recientemente, la empresa ha decidido ampliar su catálogo de productos y ha encargado a sus departamentos de ventas y marketing un estudio de mercado para determinar el grado de receptividad potencial que los clientes actuales podrían tener sobre los nuevos productos.

Para ello se ha alimentado el maestro de clientes con un atributo nuevo, "SENSI", que contiene valores del 0 al 100 donde 0 indica sensibilidad nula a los nuevos productos y 100 indica máxima sensibilidad a los nuevos productos.

Se utilizarán algoritmos de segmentación para determinar a qué segmento pertenece cada cliente con el objetivo de establecer estrategias comerciales específicas por segmento.

A continuación podéis ver la tabla maestro de clientes.

Tabla 7. Estructura de datos para *Cust_Master* en PostgreSQL

CUSNR	NAME	SENSI	SALES	CLUST
Código de cliente	Nombre del cliente	Grado de sensibilidad hacia los nuevos productos	Ventas anuales del ejercicio anterior	Segmento asignado al cliente

3) Estudio de aceptación de producto

Una empresa de productos alimentarios ha encargado a su departamento de marketing que realice un estudio de mercado sobre uno de los productos que está desarrollando.

El estudio consistirá en valorar la aceptación del producto en sus presentaciones de unidades individuales, packs de 2 unidades, packs de 3 unidades y packs de 4 unidades. Además, el estudio de mercado se hará en dos zonas geográficas y con dos estrategias de promoción distintas, la degustación y el obsequio.

Una vez establecido el concepto de rotación de un producto como el número de unidades vendidas en un mes, la dirección de la compañía fija una rotación mínima para considerar que el producto supera el estudio.

Con este criterio, por debajo de la rotación mínima consideraremos que la rotación es baja y por encima de la rotación mínima consideraremos que la rotación es alta.

La tabla 8 muestra la estructura del fichero *marketing.RData* que recoge los resultados del mencionado estudio.

Tabla 8. Estructura de datos para *marketing.RData* en formato R

Formato	Zona	Promocion	Rotacion
Formato de presentación del producto	Zona geográfica en la que se realiza el estudio de mercado	Promoción vía degustación o vía obsequio	Rotación del producto. Clasificada como alta y baja

Cada instancia refleja la rotación del producto en un punto de venta en un período determinado.

4) Clasificación de clientes

Una empresa fabrica y comercializa una nueva línea de productos industriales que tienen dos componentes: pinturas líquidas y dosificadores específicamente diseñados para estas pinturas.

Mientras que la rentabilidad del negocio se encuentra en la comercialización de las pinturas, se da la circunstancia de que la inversión inicial de ceder los dosificadores es considerable, hasta tal punto que la empresa se plantea para cada cliente la conveniencia de la cesión del dosificador o la venta del mismo.

Como resultado de un estudio interno se ha establecido que en función de nueve indicadores relativos a la actividad de cada cliente y en función de la experiencia del último año, se pretende que, cuando llega un cliente nuevo, seamos capaces de predecir si la mejor opción para nosotros será la cesión o la venta del dosificador.

Tabla 9. Estructura de datos para *Clasificacion.csv*

Id	**Actividad**	**Tipo**	**Categoría**
Identificador único de cada cliente	Adecuación de la actividad que desempeña el cliente	Tipología de clientes al que se dirige	Categoría crediticia del cliente
Familia	**País**	**Área**	**Ingresos**
Familia de productos que tienen más rotación	País en el que desempeña su actividad el cliente	Adecuación del área de influencia del cliente.	Nivel de ingresos del cliente
Establecimiento	**Especialización**	**Clase**	
Tipología de establecimientos (súper, híper, tienda...)	Grado de especialización en el mercado.	Cesión o venta	

Todos los atributos excepto la clase y el identificador de cliente son puntuaciones del 1 al 10 de la adecuación a nuestros intereses comerciales. De modo que un 10 en el atributo "Familia" indicará que los productos que tienen más rotación del cliente se corresponden con el tipo de pinturas que fabricamos nosotros.

Siguiendo con este ejemplo, para puntuaciones altas en un atributo nos interesará ceder el dosificador porque la rentabilidad de la operación justificará la cesión del mismo. Por el contrario, para puntuaciones bajas en un atributo nos interesará vender el dosificador por el riesgo de que la venta de pinturas no sea capaz de soportar el coste del dosificador.

6. Árboles de decisión

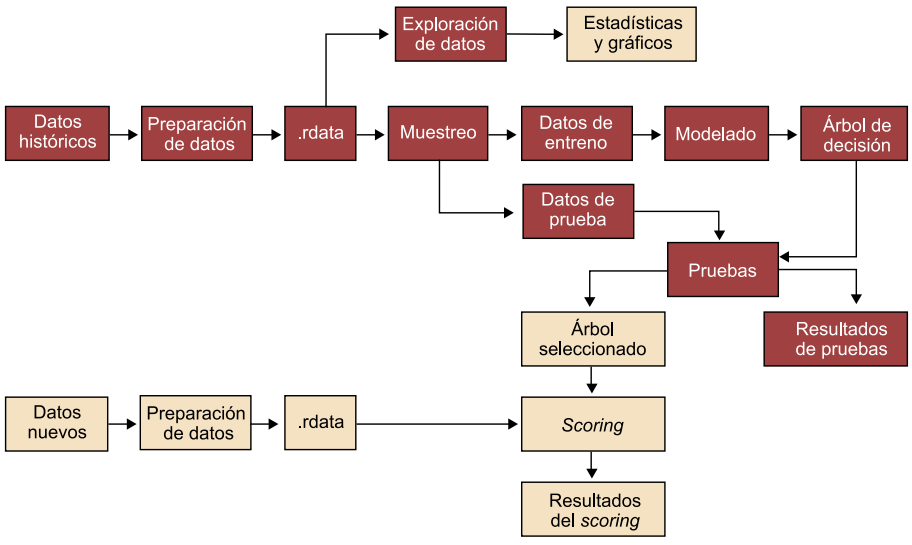
En este apartado utilizaremos la función *ctree()* del paquete *party* (Hothorn 2010). Mediante esta función crearemos un árbol de decisión a partir de un juego de datos de entrenamiento y mediante la función *predict()* realizaremos una predicción del atributo objetivo, "Decision", a partir de un juego de datos distinto.

Nuestro juego de datos es *ContratoSuministro.csv*, presentado anteriormente y que nos dará una lista de clientes que consumen suministros. De cada cliente se han recogido 5 características que nos ayudarán a predecir si renovarán o no el contrato de suministro.

Nuestro objetivo será predecir el atributo "Decision".

En la figura 9 vemos un esquema de procesos de minería de datos para generar un árbol de decisión. Vemos destacados los procesos que se cubren en este apartado.

Figura 9. Procesos de la minería de datos



1) Adaptación del juego de datos

La función *ctree()* requiere que el atributo a predecir sea la última columna del juego de datos. En el juego de datos *ContratoSuministro.csv*, el atributo "Decision" ocupa la columna 3 de 5. En la siguiente instrucción moveremos el atributo "Decision" al final del juego de datos.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_01.txt")* en vuestra sesión R.

```
# Cargamos el paquete party
> library(party)
```

```
# Cargamos el fichero csv
> Contratos <- read.csv(file="ContratoSuministro.csv",head=TRUE,sep=",")
# Pasamos la columna 3 a la última posición en el juego de datos cu.summary
> Contratos <- Contratos[,c(1,2,4,5,3)]
```

2) Obtención de los datos de entrenamiento

Nos disponemos a separar el juego de datos en dos partes con el objetivo de utilizar una parte como datos de entrenamiento y la otra parte como datos test para probar la capacidad de predicción del algoritmo.

Para conseguir separar el juego de datos en dos partes y hacerlo de una forma lo más aleatoria posible utilizaremos dos funciones R.

- La función *sample()* nos dará valores aleatorios siguiendo las probabilidades que se le pasan por parámetro.

- La función *set.seed()* fija la aleatoriedad, de modo que si posteriormente llamamos a una función que genera valores aleatorios como *rnorm()* o *sample()*, será posible posteriormente volver a llamar a estas funciones obteniendo los mismos resultados. Veamos un ejemplo.

```
# Obtenemos 5 números aleatorios
> rnorm(5)
[1] -0.7412560 -1.0076463  0.2890008 -0.8708841  0.5027605
# El problema que tenemos ahora es que no es posible ejecutar de nuevo la función
  rnorm(5) y obtener exactamente los mismos números. De modo que será complicado
  repetir el mismo experimento.
# Mediante la función set.seed() fijaremos un punto a partir del cual se repetirá
  el patrón de obtención de valores aleatorios.
> set.seed(123)
> rnorm(5)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
> rnorm(5)
[1]   1.7150650   0.4609162  -1.2650612  -0.6868529  -0.4456620

# Si volvemos a ejecutar la función set.seed con los valores 123, conseguiremos que la
  función rnorm(5) genere valores aleatorios siguiendo el mismo patrón
> set.seed(123)
> rnorm(5)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
> rnorm(5)
[1]   1.7150650   0.4609162  -1.2650612  -0.6868529  -0.4456620
```

Ahora nos encontramos en disposición de poder dividir el juego de datos *lista.coches* en dos juegos de datos distintos. El de entrenamiento contendrá el 70% de las entradas, mientras que el de test contendrá el 30% de entradas.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_02.txt")* en vuestra sesión R.

```
# Fijamos el puntero o semilla del patrón aleatorio
> set.seed(123)
# Generamos un vector con un 70% de valores 1 y un 30% de valores 2. Sumando un total
  de 117 valores que son el número de filas del juego de datos Contratos
> separa <- sample(2, nrow(Contratos), replace=TRUE, prob=c(0.7, 0.3))
> datos.entrena <- Contratos[separa==1,]
> datos.prueba <- Contratos[separa==2,]
```

3) Generación del árbol de decisión

Generaremos el árbol de decisión a partir de los datos de entrenamiento y utilizando el atributo "Decision" como objetivo de predicción.

```
# Guardamos el esquema que utilizará el árbol de decisión,
  donde indicamos que el atributo objetivo es Decision.
> mi.esquema <- Decision ~ ConsAnual + DevolAnual + ConsUltTrim + Litigios
# Generamos el árbol de decisión sobre nuestros datos de entrenamiento
> Contratos_ctree <- ctree(mi.esquema, data=datos.entrena)
```

En este momento la variable Contratos_ctree contiene las reglas que gobiernan la lógica de nuestro árbol. Es decir, la variable Contratos_ctree contiene una versión escrita del árbol de decisión.

```
# Obtenemos la versión escrita del árbol de decisión.
> print(Contratos_ctree)
         Conditional inference tree with 4 terminal nodes
Response:  Decision
Inputs:  ConsAnual, DevolAnual, ConsUltTrim, Litigios
Number of observations:  106

1) Litigios <= 4.05; criterion = 1, statistic = 99.501
  2)*  weights = 35
1) Litigios > 4.05
  3) ConsUltTrim <= 3.85; criterion = 1, statistic = 50.567
    4) Litigios <= 6.75; criterion = 0.999, statistic = 14.246
      5)*  weights = 30
    4) Litigios > 6.75
      6)*  weights = 9
  3) ConsUltTrim > 3.85
    7)*  weights = 32
```

Una vez generado el árbol, podemos obtener una medida de la capacidad de predicción sobre el juego de datos de entrenamiento generando una tabla de comparación entre los dos juegos de datos.

```
# Generamos una tabla que nos comparará los valores clasificados por el árbol
  de decisión sobre el juego de datos de entrenamiento, contra los valores
  que realmente tiene el juego de datos de entrenamiento.
> table (predict(Contratos_ctree), datos.entrena$Decision)
          amplia cancela renueva
  amplia      35       0       0
  cancela      0      36       3
  renueva      0       0      32
```

Vemos como solo ha cometido tres errores, es decir, el árbol de decisión ha clasificado tres contratos *cancela* cuando en realidad son *renueva*.

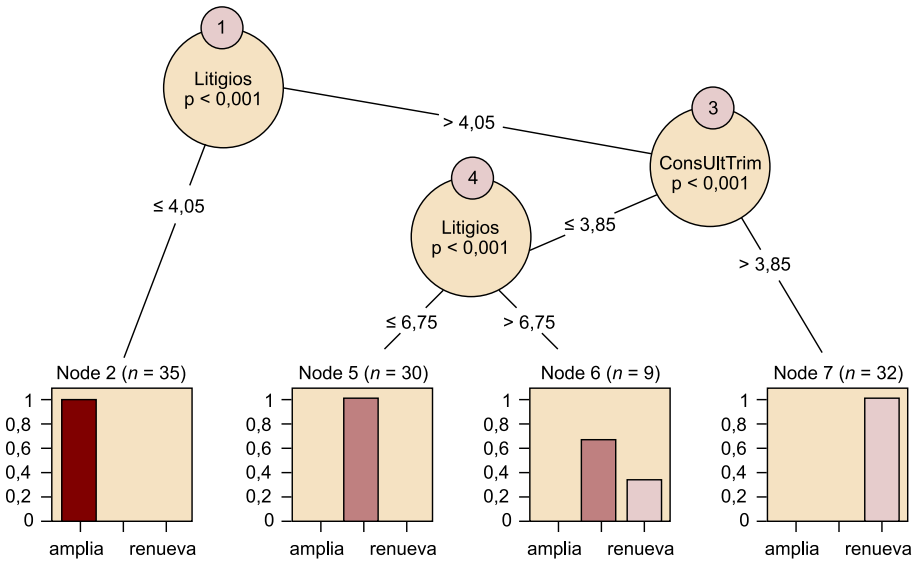
Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_03.txt")* en vuestra sesión R.

Visualizaremos de forma gráfica el árbol generado.

```
# Visualización gráfica del árbol de decisión generado. Formato simple
> plot(Contratos_ctree, type="simple")
# Visualización gráfica del árbol de decisión generado. Formato completo
> plot(Contratos_ctree)
```

Figura 10. Árbol de decisión sobre ContratoSuministro.csv



4) Evaluación del árbol de decisión

El último paso es la evaluación de la capacidad predictiva de nuestro árbol de decisión utilizando el segundo juego de datos de que disponemos. Los datos de prueba que anteriormente hemos reservado.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_04.txt")* en vuestra sesión R.

```
# Utilizamos el árbol generado para realizar una prueba de predicción sobre el juego
  de datos de prueba.
```

```
> Prueba.prediccion <- predict(Contratos_ctree,  newdata=datos.prueba)
# Generamos una tabla que nos comparará los valores clasificados por el árbol
  de decisión sobre el juego de datos de prueba, contra los valores que realmente
  tiene el juego de datos de prueba.
> table (Prueba.prediccion, datos.prueba$Decision)
Prueba.prediccion amplia cancela renueva
          amplia      15       0       0
          cancela      0      13       2
          renueva      0       1      13
```

Vemos cómo solo ha vuelto a cometer tres errores, es decir:

- El árbol de decisión ha clasificado dos contratos *cancela* cuando en realidad son *renueva*.

- El árbol de decisión ha clasificado un contrato *renueva* cuando en realidad es *cancela*.

5) Consideraciones

Una de las limitaciones de la función *ctree()* es que no es capaz de trabajar con datos NA (*not available*). Para suplir esta limitación, en R tenemos la función *na.omit()* que nos eliminará las entradas con datos no disponibles.

Finalmente se debe comentar que R dispone de otros paquetes que implementan funciones capaces de generar árboles de decisión. Mencionamos dos de los que consideramos que son más representativos:

- Función *rpart()* del paquete *rpart* (librería implementada por Therneau, 2010).

- Función *randomForest()* del paquete *randomForest* (librería implementada por Liaw and Wiener, 2002).

7. Ganancia de la información

En la asignatura *Business analytics* ya estudiamos que la ganancia de la información era el concepto clave para poder construir árboles de decisión. En este apartado veremos cómo el paquete *FSelector* nos ofrece dos funciones para calcular tanto la ganancia de la información como la ratio de la información en un juego de datos.

Recordemos las fórmulas de estas dos medidas:

$$IG(Ex, a) = H(Ex) - H(Ex, a)$$

La ganancia de información de un atributo es la diferencia entre la entropía del juego de datos y la entropía del atributo.

$$GR(Ex, a) = \frac{H(Ex) - H(Ex, a)}{H(Ex, a)}$$

La ratio de ganancia de un atributo es su ganancia de la información normalizada.

Los siguientes *scripts* parten del juego de datos utilizado para construir el árbol de decisión, de este modo corroboraremos que efectivamente el atributo "Litigios" es el que tiene mayor ganancia.

```
# Cargamos la biblioteca FSelector
> library(FSelector)
# Calculamos la ganancia de la información del juego de datos
# tomando como atributo objetivo 'Decision'
> ig <- information.gain(Decision~., datos.entrena)
> print(ig)
             attr_importance
ConsAnual          0.7875745
DevolAnual         0.3072767
Litigios           1.4789395
ConsUltTrim        1.4408864
# Mediante la function cutoff.k() obtenemos los k mejores índices
> selec_ind <- cutoff.k(ig, 1)
> selec_data <- as.simple.formula(selec_ind, "Decision")
> print(selec_data)
Decision ~ Litigios
```

Veremos que la ratio ganancia coincide con la ganancia de la información, en que el atributo "Litigio" es el que contiene mayor información. Aprovechamos para recordar que la ratio ganancia corrige la tendencia que tiene la ganancia de la información a favorecer los atributos que tienen más valores posibles.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_05.txt")* en vuestra sesión R.

```
# Calculamos el ratio ganancia
> gr <- gain.ratio(Decision~., datos.entrena)
> print(gr)
            attr_importance
ConsAnual         0.5007185
DevolAnual        0.3899109
Litigios          0.9336342
ConsUltTrim       0.9118229
# Mediante la function cutoff.k() obtenemos los k mejores índices
> selec_ind <- cutoff.k(gr, 1)
> selec_data <- as.simple.formula(selec_ind, "Decision")
> print(selec_data)
Decision ~ Litigios
```

Finalmente se debe comentar que las tres funciones (*information.gain()*, *gain.ratio()* y *cutoff.k()*) pertenecen al paquete FSelector.

Enlace recomendado

Para más información sobre las posibilidades de este paquete podéis consultar el siguiente enlace:

http://cran.r-project.org/web/packages/FSelector/FSelector.pdf.

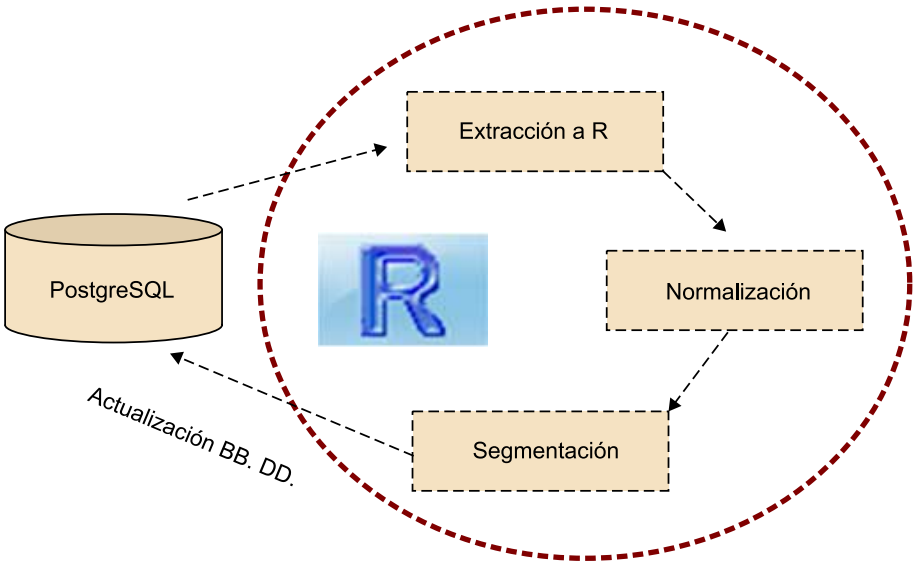
8. Segmentación

8.1. Algoritmo *Kmeans*

A partir del juego de datos *Segmentación de clientes* presentado antes, realizaremos un proceso de segmentación de clientes a partir de la implementación en R del algoritmo *Kmeans*.

En la figura 11 se visualiza el esquema de las actividades que llevaremos a cabo en el proceso de segmentación de clientes.

Figura 11. Actividades del proceso de segmentación



Para que la función *kmeans()* pueda segmentar correctamente, le pasaremos dos parámetros.

- Por un lado, el número de segmentos que deseamos generar. Animamos al estudiante a seguir la práctica y a jugar con este parámetro para ver el efecto en el resultado de la segmentación.

- Por otro lado, le pasaremos la matriz de coordenadas sobre la que se realizarán los cálculos de distancias o semejanzas entre puntos. En nuestro caso seleccionaremos dos coordenadas. Las ventas anuales de nuestros clientes y la medida de sensibilidad hacia los nuevos productos.

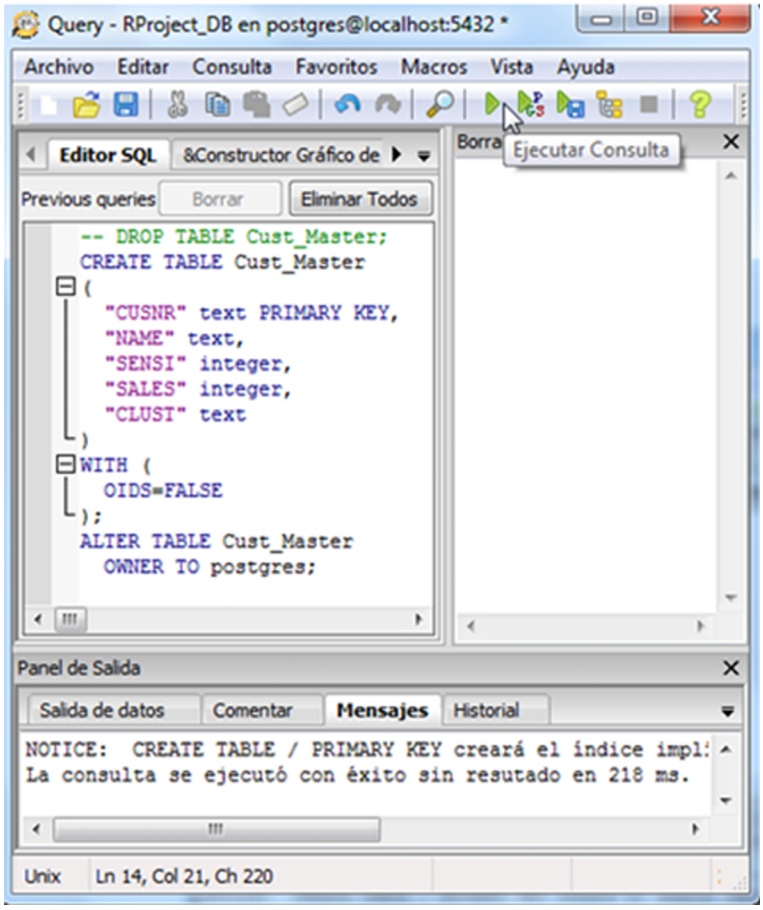
1) Carga de datos en PostgreSQL

Presuponemos que en PostgreSQL se ha creado una base de datos con nombre *RProject_DB*, usuario *postrgres* y contraseña *root*. Los *scripts* R que se presentan en este apartado están escritos bajo esta suposición.

Cualquier cambio en el nombre de la base de datos, usuario o contraseña deberán ser tenidos en cuenta por parte del usuario, que deberá proceder a adaptar los mencionados *scripts* R.

Como primer paso, en la figura 12 tenemos el *script* con el que procederemos a crear la tabla Cust_Master en nuestra base de datos PostgreSQL.

Figura 12. Creación de tabla en PostgreSQL

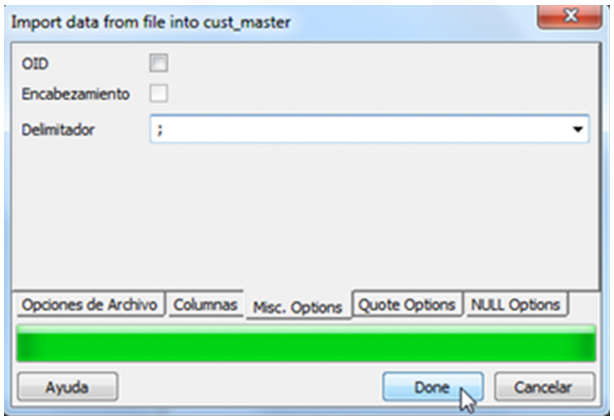


Nota

Adjunto al material didáctico disponéis del fichero *Cust_Master.csv*, que mediante la interfaz *pgAdmin* de PostgreSQL podréis importar a la tabla Cust_Master.

Simplemente hay que posicionarse sobre la tabla, con el botón derecho, seleccionar la opción importar, seleccionar el fichero *.csv* y completar las pantallas del proceso de importación especificando el símbolo punto y coma como delimitador, tal y como se muestra en la figura 13.

Figura 13. Importación de datos a PostgreSQL



2) Conexión R frente a PostgreSQL

Con el siguiente *script* crearemos una conexión desde R con la base de datos RProject_DB en PostgreSQL.

```
# Cargamos el paquete RPostgreSQL
> library(RPostgreSQL)
# Creamos la conexión con la base de datos RProject_DB
# Informamos de usuario y contraseña
> drv <- dbDriver("PostgreSQL")
> con <- dbConnect(drv, user="postgres", password="root",      dbname="RProject_DB")
```

Mediante esta conexión ya podemos recuperar datos y ejecutar procesos SQL sobre la base de datos RProject_DB.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_06.txt")* en vuestra sesión R.

```
# Recuperar los datos de la tabla Cust_Master de nuestra base de datos
> RCust_Master <- dbGetQuery(con, "select * from Cust_Master")
# Podemos visualizar la disposición de puntos (ventas, sensibilidad por nuevos productos)
> plot(RCust_Master[,3:4])
# Con la siguiente instrucción obtenemos el mismo resultado
> plot(RCust_Master[,c("SENSI","SALES")])
```

3) Algoritmo *Kmeans* en R

Para segmentar, solo consideraremos las columnas 3 y 4 de la tabla RCust_Master, es decir, las columnas SALES (ventas) y SENSI (sensibilidad por los nuevos productos).

En primer lugar normalizaremos los atributos “SALES” y “SENSI”.

```
# Copiamos la tabla original, antes de proceder a la normalización
> RCust_Segment <- RCust_Master
# Normalizamos el atributo SALES con la fórmula:
# y = ( x - media(x) ) / desviación(x)
> RCust_Segment[c("SALES")] <-- (RCust_Segment$SALES-mean(RCust_Segment$SALES,na.rm=T))
```

```
  /sd(RCust_Segment$SALES,na.rm=T)
# Normalizamos el atributo SENSI: y = ( x - media(x) ) / desviación(x)
> RCust_Segment[c("SENSI")] <-- (RCust_Segment$SENSI-mean(RCust_Segment$SENSI,na.rm=T))
  /sd(RCust_Segment$SENSI,na.rm=T)
```

Ejecutaremos el algoritmo de segmentación *Kmeans* sobre los atributos ya normalizados.

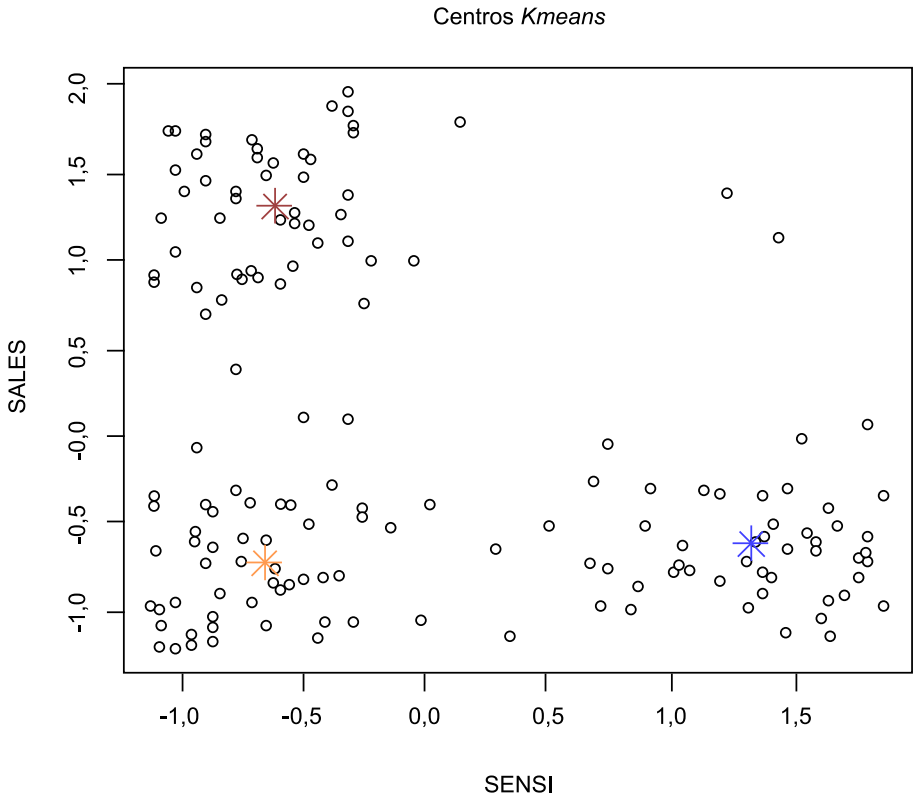
```
# Aplicamos la segmentación sobre los atributos SALES y SENSI de la tabla RCust_Segment,
  estableciendo como parámetro inicial que genere 3 segmentos.
> (kmeans.result <- kmeans(RCust_Segment[,3:4],3))
# Generamos un gráfico con los atributos normalizados y sus centros
> plot(RCust_Segment[c("SENSI","SALES")])
> points(kmeans.result$centers[,c("SENSI","SALES")], col=1:3, pch=8, cex=2)
```

Actividad

Se propone como ejercicio realizar esta misma segmentación sin el paso de la normalización de atributos. El estudiante observará la pérdida de precisión del algoritmo.

La figura 14 nos muestra el gráfico de los atributos "Ventas" y "Sensibilidad" normalizados, así como los centros que se han determinado mediante el algoritmo de segmentación *Kmeans*.

Figura 14. Centros de segmentación *Kmeans*



Volcamos los segmentos obtenidos a la tabla original. De este modo tenemos nuestro maestro de clientes original con el atributo "CLUST" (segmento) rellenado.

```
# Copia de los segmentos generados, a la tabla original
> RCust_Master$CLUST <- kmeans.result$cluster
```

Con los segmentos ya en la tabla original, podremos obtener un gráfico de los pares (ventas, sensibilidad) separados según el segmento asignado. Para facilitar la visualización estableceremos un color para cada segmento.

Nota

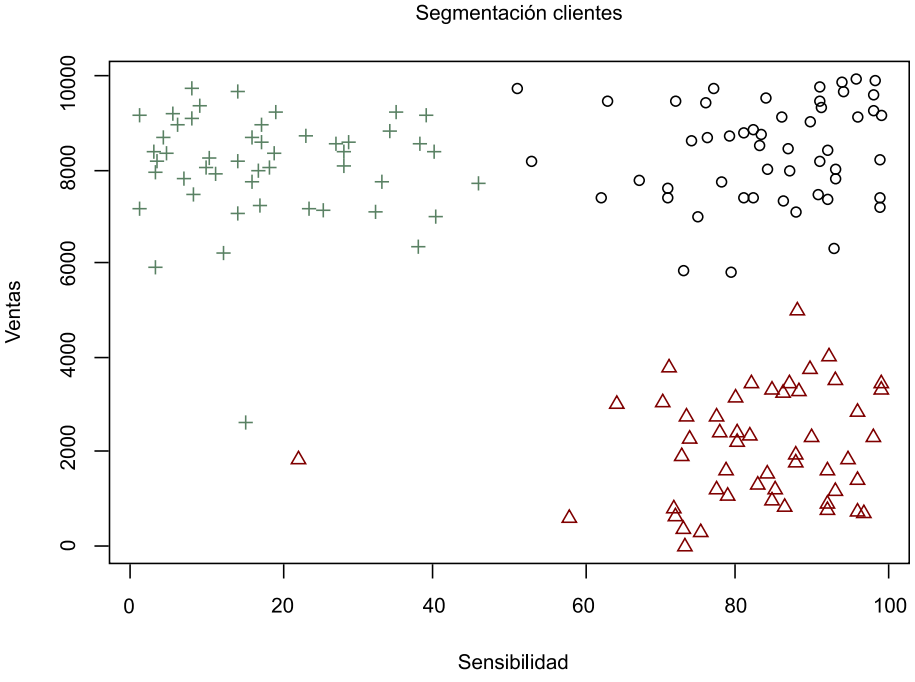
Podéis ejecutar este *script* escribiendo *source("RUOC_07.txt")* en vuestra sesión R.

En esta ocasión, en lugar de obtener un gráfico por pantalla, lo que haremos es exportarlo a pdf.

```
# Creamos el objeto pdf informando el nombre de fichero que se generará.
  Podemos especificar el directorio, pero tomaremos el establecido por defecto.
> pdf("Segmentacion.pdf")
# Construimos el gráfico de pares (ventas, sensibilidad) segmentados por color.
> plot(RCust_Master[,c("SENSI","SALES")],col=RCust_Master$CLUST, pch=RCust_Master$CLUST,
  main="Segmentación Clientes",ylab="Ventas",xlab="Sensibilidad")
# Cerramos el objeto pdf
> dev.off()
```

En la figura 15 vemos el gráfico que encontraremos en nuestro directorio de trabajo, con el nombre de fichero *Segmentacion.pdf*.

Figura 15. Segmentos de clientes, en función de ventas y sensibilidad



En el gráfico vemos cómo se distinguen claramente tres grupos de clientes:

- **En cruz verde**: clientes con mucho volumen de ventas, pero poca sensibilidad hacia los nuevos productos.

- **En círculo negro**: clientes con mucho volumen de ventas y con mucha sensibilidad hacia los nuevos productos.

- **En triángulo rojo**: clientes con poco volumen de ventas y con mucha sensibilidad hacia los nuevos productos.

En este caso vemos cómo un trabajo de segmentación puede ayudar a la empresa en la toma de decisiones estratégicas:

- Claramente el segmento *círculo negro* es el más apropiado para un lanzamiento inicial de los nuevos productos.

- Introducir los nuevos productos en el segmento *triángulo rojo* significará seguramente fácil penetración, pero poca rotación.

- Introducir los nuevos productos en el segmento *cruz verde* requerirá de un esfuerzo en promoción y marketing para convencer al cliente.

4) Actualización de datos de R a PostgreSQL

En este apartado actualizaremos la tabla Cust_Master de PostgreSQL, fijando el campo CLUST con el valor obtenido en el proceso de segmentación en el entorno R.

```
# Cargamos el paquete foreach
>  library(foreach)
# Creamos la función update() que realizará las siguientes tareas:
# 1. Guarda en la variable 'con' la conexión a la base de datos RProject_DB
# 2. Guarda en la variable 'upd' la sentencia UPDATE que correrá en PostgreSQL
# 3. Lanza la sentencia UPDATE
# 4. Cierra la conexión con la base de datos
> update <- function(i) {
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="RProject_DB", host="localhost", port="5432",
       user="postgres", password="root")
upd <- paste("UPDATE Cust_Master SET \"CLUST\"=","'",RCust_Master$CLUST[i],"'","
       where \"CUSNR\"=","'",RCust_Master$CUSNR[i],"'", sep="")
dbGetQuery(con, upd)
dbDisconnect(con)
}
```

Finalmente, ejecutaremos un bucle *foreach* para que se lance una actualización de la columna CLUST de la tabla Cust_Master en la base de datos RProject_DB para cada cliente.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_08.txt")* en vuestra sesión R.

```
# Generamos un bucle que lanzará la función update() para cada fila
  de la tabla RCust_Master
> foreach(i = 1:length(RCust_Master$CLUST))%do%{
update(i)
}
```

En este punto, hemos completado el proceso de volcado del resultado de una segmentación sobre nuestra base de datos transaccional.

8.2. Segmentación jerárquica

En este apartado haremos una práctica de segmentación jerárquica mediante la función R *hclust()* sobre el juego de datos *RCust_Master* utilizado en el apartado anterior.

Para ello tomaremos una muestra de 15 filas con el fin de que la visualización gráfica no esté saturada de entradas.

Es importante observar que en esta ocasión no normalizaremos los atributos, de modo que a la hora de calcular similitudes entre clientes, el atributo "SALES" (ventas) será mucho más relevante que el atributo "SENSI" (sensibilidad a los nuevos productos).

```
# Mediante la función set.seed() fijaremos un punto a partir del cual se repetirá
  el patrón de obtención de valores aleatorios.
> set.seed(123)
# Seleccionamos de forma aleatoria, 14 filas de la tabla RCust_Master
> idx <- sample(1:dim(RCust_Master)[1], 15)
> RCust_Master_Sample <- RCust_Master[idx,]
# Antes de calcular las distancias, eliminamos las columnas no numéricas
> RCust_Master_Sample$CUSNR <- NULL
> RCust_Master_Sample$NAME <- NULL
```

Una vez realizadas las tareas de preparación sobre la tabla RCust_Master, procederemos a ejecutar la función de segmentación jerárquica *hclust()*.

Es importante observar que uno de los parámetros que recibe la función *hclust()* es el cálculo de distancia entre clientes. Esto hace que la función de segmentación jerárquica pueda ser independiente del concepto de distancia.

Nota

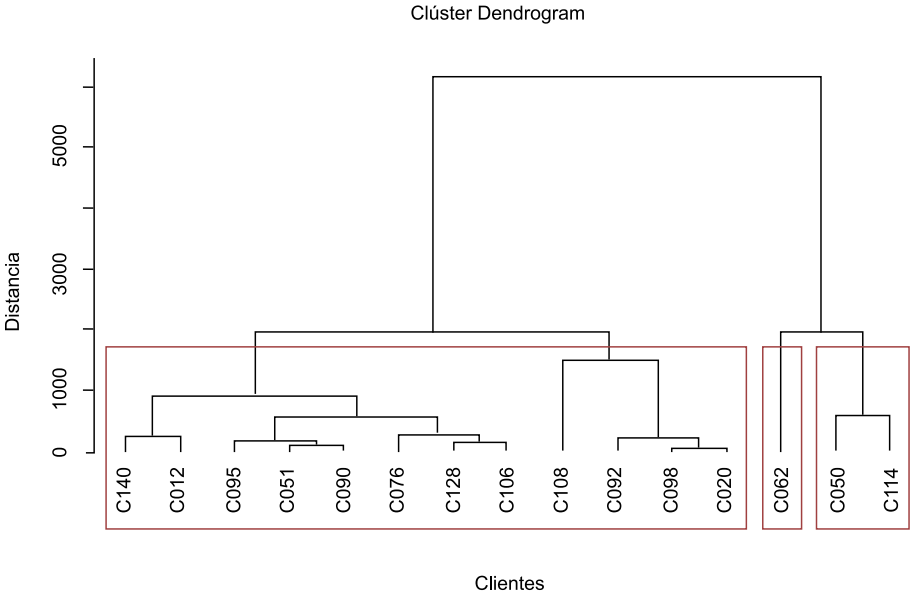
Podéis ejecutar este *script* escribiendo *source("RUOC_09.txt")* en vuestra sesión R.

```
# Lanzamos el proceso de segmentación jerárquica en base a la función
# distancia euclidiana dist()
> hc <- hclust(dist(RCust_Master_Sample), method="ave")
```

```
# Visualizamos en pantalla un gráfico del resultado de la segmentación
> plot(hc, hang = -1, labels=RCust_Master$CUSNR[idx])
# Marcamos con un rectángulo los tres segmentos principales
> rect.hclust(hc, k=3)
```

La figura 16 nos muestra la representación gráfica en forma de dendograma de la segmentación jerárquica que acabamos de procesar.

Figura 16. Segmentos jerárquicos de clientes, en función de ventas y sensibilidad



8.3. Detección de valores *outliers* por segmentación

En este apartado utilizaremos la función de segmentación *kmeans()* para detectar valores *outliers* en un juego de datos.

Inversamente, podríamos pensar en la tarea de detección y eliminación de valores *outliers* como un paso previo a un proceso de segmentación.

Como actividades previas copiaremos en una tabla de trabajo solo los atributos "SALES" (ventas) y "SENSI" (sensibilidad a los nuevos productos), para posteriormente proceder a su normalización.

```
# Creamos la tabla de trabajo RCust_Master2 solo con las columnas numéricas
> RCust_Master2 <- RCust_Master[,3:4]
# Normalizamos el atributo SALES: y = ( x - media(x) ) / desviación(x)
> RCust_Master2[c("SALES")] <-- (RCust_Master2$SALES-mean(RCust_Master2$SALES,na.rm=T))
  /sd(RCust_Master2$SALES,na.rm=T)
# Normalizamos el atributo SENSI: y = ( x - media(x) ) / desviación(x)
> RCust_Master2[c("SENSI")] <-- (RCust_Master2$SENSI-mean(RCust_Master2$SENSI,na.rm=T))
  /sd(RCust_Master2$SENSI,na.rm=T)
```

Segmentamos con el algoritmo *Kmeans*, buscando 3 segmentos. Posteriormente utilizaremos los centros determinados por *kmeans()* para calcular la distancia de cada instancia a su centro correspondiente.

El criterio que va a determinar si una fila es valor *outlier* o no es la distancia de cada fila a su centro.

```
# Lanzamos el proceso de segmentación con la función kmeans()
> kmeans.result <- kmeans(RCust_Master2, centers=3)
# Para cada fila de RCust_Master guardamos su centro
> centros <- kmeans.result$centers[kmeans.result$cluster,]
# Calculamos la distancia de cada fila a su centro asignado por kmeans
> distancias <- sqrt(rowSums((RCust_Master2 - centros)^2))
# Establecemos como valores outliers la 5 distancias mayores
> outliers <- order(distancias, decreasing=T)[1:5]
# Visualizamos las filas que hemos clasificado como outliers
> RCust_Master[outliers,]
```

El siguiente *script* construirá un gráfico donde veremos identificados los clientes como puntos (ventas, sensibilidad), los centros de segmento calculados por *kmeans()* y finalmente los puntos identificados como *outliers*.

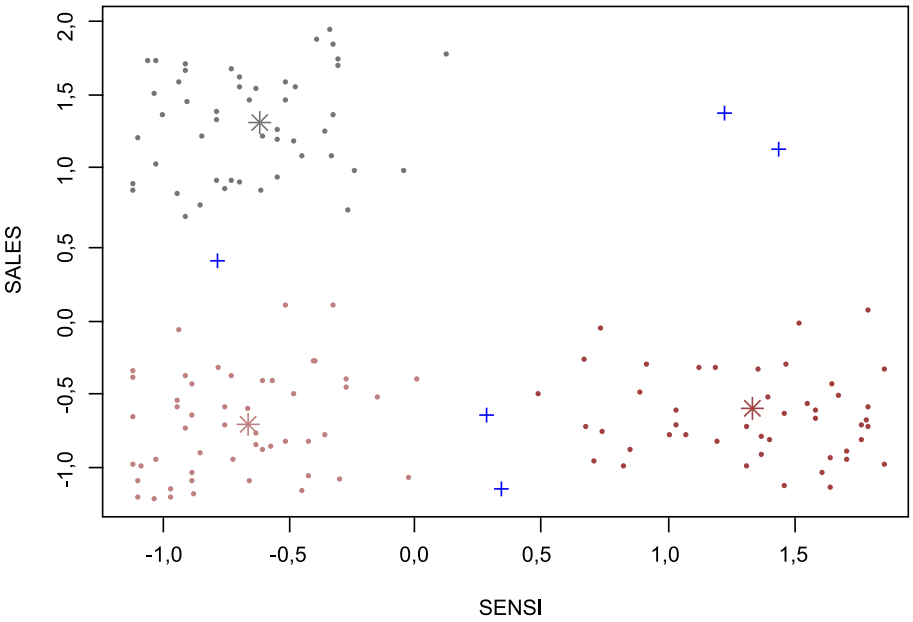
Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_10.txt")* en vuestra sesión R.

```
# Generamos un gráfico con los pares (ventas, sensibilidad por los nuevos productos)
> plot(RCust_Master2[,c("SENSI","SALES")], main="Detección de outliers", pch="o",
  col=kmeans.result$cluster, cex=0.3)
# Dibujamos con asterisco, los centros de segmentos
> points(kmeans.result$centers[,c("SENSI","SALES")], col=1:3, pch=8, cex=1.5)
# Dibujamos con signo + los puntos outliers
> points(RCust_Master2[outliers,c("SENSI","SALES")], col=4, pch="+", cex=1.5)
```

La figura 17 nos muestra los valores *outliers* con símbolo "+" detectados mediante el proceso de segmentación *kmeans*.

Figura 17. Detección de valores *outliers* por medio de segmentación



9. Asociaciones

En este apartado trabajaremos con el algoritmo APRIORI, planteado por los autores Agrawal y Srikant en 1994. Su implementación en R se realiza mediante el paquete *arules* (Hahsler, 2011) con la función *apriori()*.

En términos generales, podemos decir que el algoritmo APRIORI construye reglas a partir de la relevancia que sus componentes tienen en todo el juego de datos. Para entenderlo un poco mejor deberemos repasar las tres medidas de relevancia más importantes que soporta este algoritmo.

1) El **soporte** o *support*. El soporte de (A= =>B) = P(A U B) es una medida de la probabilidad de que se dé A o B en el juego de datos.

2) La **esperanza** o *confidence*. La esperanza de (A= =>B) = P(A | B) = $\frac{P(A \cup B)}{P(A)}$ es una medida de la probabilidad de que se dé A y B en el juego de datos.

3) La **elevación** o *lift*. La elevación de (A==>B) = $\frac{\mathrm{P(A \mid B)}}{P(B)} = \frac{P(A \cup B)}{P(B) \cdot P(A)}$ es una medida de la probabilidad de que se dé A y B en términos relativos respecto de las probabilidades respectivas de A y B.

Una elevación igual a 1 nos dirá que las dos probabilidades, P(A) y P(B), son independientes y en consecuencia las reglas que involucran A y B a la vez serán poco relevantes.

La función *apriori()* es capaz de trabajar con más de 20 medidas de relevancia distintas, entre las cuales encontramos *chi-square*, *conviction*, *gini* y *leverage*. Sin embargo, por defecto la función solo establece mínimos para las medidas de soporte, esperanza y longitud o número de componentes máximos de las reglas.

De hecho, la función establece los siguientes valores mínimos por defecto:

- soporte = 0,1;
- esperanza = 0,8;
- longitud máxima de las reglas = 10

1) Caso de estudio

Para poner en práctica lo aprendido del algoritmo APRIORI como generador de reglas de implicación, utilizaremos el juego de datos *marketing.RData* que tal y como se explica en el apartado “Juegos de datos”, describe el resultado

de un estudio de mercado para un producto presentado con cuatro formatos distintos en dos zonas geográficas de venta y ofertado bajo dos estrategias distintas de promoción.

Empezaremos por cargar los datos en nuestra sesión de trabajo. Merece la pena observar que en esta ocasión el formato del fichero es *.RData*, nativo de R.

Simplemente colocando el fichero en el directorio de trabajo de R y mediante la función *load()* ya dispondremos del *data.frame marketing* en nuestra sesión R.

```
# Cargamos el juego de datos marketing.RData
> load("marketing.RData")
# Inspeccionamos las primeras 6 entradas del juego de datos
> head(marketing)
    Formato    Zona   Promocion   Rotacion
1 Pack de 3u  Norte  Degustacion    Baja
2 Pack de 3u  Norte  Degustacion    Baja
3 Pack de 3u  Norte  Degustacion    Baja
4 Pack de 3u  Norte  Degustacion    Baja
5 Pack de 3u  Norte  Degustacion    Baja
6 Pack de 3u  Norte  Degustacion    Baja
# Vemos la estructura del juego de datos
> str(marketing)
'data.frame':   2201 obs. of  4 variables:
 $ Formato  : Factor w/ 4 levels "Pack de 1u","Pack de 2u",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Zona     : Factor w/ 2 levels "Norte","Sur": 1 1 1 1 1 1 1 1 1 1 ...
 $ Promocion: Factor w/ 2 levels "Degustacion",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Rotacion : Factor w/ 2 levels "Alta","Baja": 2 2 2 2 2 2 2 2 2 2 ...
# Visualizamos un estudio de frecuencias
> summary(marketing)
    Formato          Zona          Promocion        Rotacion
 Pack de 1u:325   Norte: 1731   Degustacion: 109    Alta: 711
 Pack de 2u:285   Sur  : 470    Obsequio   : 2092   Baja: 1490
 Pack de 3u:706
 Pack de 4u:885
```

2) Generación de reglas

Generaremos las primeras reglas utilizando los parámetros por defecto de la función *apriori()*.

```
# Cargamos el paquete arules que implemente la función apriori()
> library(arules)
# Generamos las reglas con parámetros por defecto
# El algoritmo nos devuelve información tipo log sobre los parámetros por defecto,
# tiempos empleados, número de reglas generadas, ....
> reglas1 <- apriori(marketing)
```

```
parameter specification:
 confidence minval smax arem  aval originalSupport support minlen maxlen target   ext
    0.8       0.1    1  none  FALSE    TRUE          0.1      1     10   rules   FALSE

algorithmic control:
 filter tree heap memopt load sort verbose
  0.1   TRUE TRUE FALSE  TRUE  2   TRUE

apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

3) Primeros inconvenientes

Inspeccionaremos las primeras reglas generadas para verificar qué conclusiones podemos establecer. Se debe aclarar que *lhs* significa *left hand side* o lado izquierdo de la regla, y que *rhs* significa *right hand side* o lado derecho de la regla.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_11.txt")* en vuestra sesión R.

```
# Obtenemos una lista de las reglas generadas (solo mostraremos las 6 primeras)
> inspect(reglas1)
lhs                                  rhs           support    confidence         lift
1  {}                   => {Promocion=Obsequio}  0.9504771     0.9504771    1.0000000
2  {Formato=Pack de 2u} => {Promocion=Obsequio}  0.1185825     0.9157895    0.9635051
3  {Formato=Pack de 1u} => {Promocion=Obsequio}  0.1449341     0.9815385    1.0326798
4  {Zona=Sur}           => {Promocion=Obsequio}  0.1930940     0.9042553    0.9513700
5  {Formato=Pack de 3u} => {Promocion=Obsequio}  0.2848705     0.8881020    0.9343750
6  {Rotacion=Alta}      => {Promocion=Obsequio}  0.2971377     0.9198312    0.9677574
```

En esta primera ejecución se han generado 27 reglas, pero encontramos los siguientes inconvenientes:

- Para nuestro estudio solo tiene sentido que el atributo "Rotación" aparezca en el lado derecho de las reglas. Para ello fijaremos el parámetro *rhs=c("Rotacion=Alta", "Rotacion=Baja")*.

- El resto de atributos podrán aparecer en el lado izquierdo de las reglas. Para ello fijaremos el parámetro *default="lhs"*.

- Observamos que la primera regla generada tiene la parte izquierda vacía. Eliminaremos estos casos, fijando el parámetro *minlen=2*.

- Finalmente, si queremos evitar que vuelva a salir por pantalla el log del proceso de generación de reglas, fijaremos el parámetro *verbose=F*.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_12.txt")* en vuestra sesión R.

Volvamos a generar reglas, teniendo en cuenta todos estos conceptos.

```
# Obtenemos una nueva lista de reglas
> reglas2 <- apriori(marketing, control = list(verbose=F), parameter = list(minlen=2,
  supp=0.005, conf=0.8), appearance = list(rhs=c("Rotacion=Alta", "Rotacion=Baja"),
  default="lhs"))
# Establecemos que las medidas tengan solo tres decimales
> quality(reglas2) <- round(quality(reglas2), digits=3)
# Ordenamos las reglas en función de la medida de elevación
> reglas2.ordenadas <- sort(reglas2, by="lift")
# Visualizamos las reglas obtenidas
 > inspect(reglas2.ordenadas)
lhs                                                    rhs      support confidence  lift
1  {Formato=Pack de 2u, Promocion=Degustacion}         =>{Rotacion=Alta} 0.011    1.000    3.096
2  {Formato=Pack de 2u, Zona=Sur Promocion=Degustacion}=>{Rotacion=Alta} 0.006    1.000    3.096
3  {Formato=Pack de 1u, Zona=Sur}                      =>{Rotacion=Alta} 0.064    0.972    3.010
4  {Formato=Pack de 1u, Zona=Sur, Promocion=Obsequio}  =>{Rotacion=Alta} 0.064    0.972    3.010
5  {Formato=Pack de 2u, Zona=Sur}                      =>{Rotacion=Alta} 0.042    0.877    2.716
6  {Formato=Pack de 4u, Zona=Sur}                      =>{Rotacion=Alta} 0.009    0.870    2.692
7  {Formato=Pack de 4u, Zona=Sur, Promocion=Obsequio}  =>{Rotacion=Alta} 0.009    0.870    2.692
8  {Formato=Pack de 2u, Zona=Sur, Promocion=Obsequio}  =>{Rotacion=Alta} 0.036    0.860    2.663
9  {Formato=Pack de 2u, Zona=Norte, Promocion=Obsequio}=>{Rotacion=Baja} 0.070    0.917    1.354
10 {Formato=Pack de 2u, Zona=Norte}                    =>{Rotacion=Baja} 0.070    0.860    1.271
11 {Formato=Pack de 3u, Zona=Norte, Promocion=Obsequio}=>{Rotacion=Baja} 0.176    0.838    1.237
12 {Formato=Pack de 3u, Zona=Norte}                    =>{Rotacion=Baja} 0.192    0.827    1.222
```

En esta ocasión hemos obtenido solo 12 reglas, como fruto del nivel de exigencia que hemos establecido. Un soporte de 0,005 sobre 2.201 casos significa que cada regla es válida al menos para 12 (= redondeo al alza de 0,005 × 2.201) casos, lo que parece razonable.

A pesar de este nivel de exigencia, todavía hay margen para mejorar las reglas.

9.1. Eliminación de la redundancia

Algunas de las reglas generadas en el paso anterior no aportan información nueva por el hecho de ser redundantes. Veámoslo.

- La regla 1 nos dice que para el formato pack de 2 unidades y con la promoción tipo degustación hemos obtenido una rotación alta con un soporte de 0,011 y una esperanza de 1.

- La regla 2 nos da la misma información que la regla 1, pero además añade que es para la zona comercial sur. Este apunte es del todo irrelevante puesto que con la regla 1 se entiende que ya se aplican a las dos zonas, la norte y la sur.

Este hecho no es aislado, puesto que los pares de reglas (4,3), (7,6) y (8,5) también presentan redundancia. ¿Cómo podemos eliminar este efecto?

Antes deberemos entender cómo trabaja la función *is.subset()*, también específica del paquete *arules*.

1) Función *is.subset()*

La documentación disponible en *vignette('arules')* describe perfectamente el sentido de esta función, pero para comprenderlo mejor introduciremos un poco de vocabulario:

- Entenderemos por **transacciones** los registros, instancias o filas de nuestro juego de datos de reglas. En nuestro caso, *reglas2.ordenadas* tiene 12 transacciones.

- Entenderemos por ***itemset*** los componentes de las reglas generadas. Por ejemplo, las reglas almacenadas en *reglas2.ordenadas* tienen un *set* formado por 10 *itemsets*. Estos son *Pack de 1u*, *Pack de 2u*, *Pack de 3u*, *Pack de 4u*, *Norte*, *Sur*, *Degustacion*, *Obsequio*, *Alta*, *Baja*.

- Entenderemos por ***subset*** un subconjunto de un set. Es decir, un subconjunto de un juego de datos. Para nuestro caso un *subset* de *itemsets* sería un subconjunto de componentes de una regla.

Con estos conceptos en mente ya podemos entender el significado de la siguiente expresión:

is.subset (X, Y, proper = TRUE)

El objetivo de la expresión es encontrar los subconjuntos de componentes (subsets de itemsets) de las reglas de X en los componentes de las reglas de Y.

El resultado será una matriz lógica, es decir, con valores TRUE o FALSE, de dimensión 12 × 12 en nuestro caso. De modo que, por ejemplo, la celda procedente de la columna 7, fila 6 tiene valor TRUE porque la regla (7) está incluida en la regla (6).

Con esta introducción ya estamos en disposición de ejecutar los pasos para eliminar las reglas redundantes.

2) Eliminación de reglas redundantes

Procederemos a eliminar las reglas redundantes del juego de datos de reglas *reglas2.ordenadas*.

```
# Construimos la matriz lógica que nos marcará las combinaciones redundantes
matriz.reglas <- is.subset(reglas2.ordenadas, reglas2.ordenadas)
# Eliminamos el triángulo inferior de la matriz lógica. El estudiante puede
# comprobar que los casos redundantes se encuentran en el triángulo superior
> matriz.reglas[lower.tri(matriz.reglas, diag=T)] <- NA
# Mostramos la matriz de reglas redundantes. Las columnas con valor TRUE
# marcan las reglas redundantes
> matriz.reglas
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]   [,10]  [,11]  [,12]
 [1,]   NA    TRUE   FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
 [2,]   NA    NA     FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
 [3,]   NA    NA     NA     TRUE   FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
 [4,]   NA    NA     NA     NA     FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE  FALSE
 [5,]   NA    NA     NA     NA     NA     FALSE  FALSE  TRUE   FALSE  FALSE  FALSE  FALSE
 [6,]   NA    NA     NA     NA     NA     NA     TRUE   FALSE  FALSE  FALSE  FALSE  FALSE
 [7,]   NA    NA     NA     NA     NA     NA     NA     FALSE  FALSE  FALSE  FALSE  FALSE
 [8,]   NA    NA     NA     NA     NA     NA     NA     NA     FALSE  FALSE  FALSE  FALSE
 [9,]   NA    NA     NA     NA     NA     NA     NA     NA     NA     FALSE  FALSE  FALSE
[10,]   NA    NA     NA     NA     NA     NA     NA     NA     NA     NA     FALSE  FALSE
[11,]   NA    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     FALSE
[12,]   NA    NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
# Las reglas redundantes serán aquellas columnas con algún valor TRUE
> redundantes <- colSums(matriz.reglas, na.rm=T) >= 1
# Mostramos las reglas redundantes
> which(redundantes)
[1] 2 4 7 8
```

Procedemos a eliminar las reglas redundantes, de modo que en todo el proceso habremos pasado de las 27 reglas iniciales a las 12 reglas más relevantes y finalmente a 8 reglas libres de redundancia.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_13.txt")* en vuestra sesión R.

```
# Eliminación de las reglas redundantes
> reglas3 <- reglas2.ordenadas[¡redundantes]
# Finalmente obtenemos 8 reglas libres de redundancia
> inspect(reglas3)
                            lhs                                  rhs        support confidence lift
1 {Formato=Pack de 2u, Promocion=Degustacion}          => {Rotacion=Alta} 0.011    1.000   3.096
2 {Formato=Pack de 1u, Zona=Sur}                      => {Rotacion=Alta} 0.064    0.972   3.010
3 {Formato=Pack de 2u, Zona=Sur}                      => {Rotacion=Alta} 0.042    0.877   2.716
```

```
4 {Formato=Pack de 4u, Zona=Sur}                        => {Rotacion=Alta} 0.009    0.870   2.692
5 {Formato=Pack de 2u, Zona=Norte, Promocion=Obsequio} => {Rotacion=Baja} 0.070    0.917   1.354
6 {Formato=Pack de 2u, Zona=Norte}                      => {Rotacion=Baja} 0.070    0.860   1.271
7 {Formato=Pack de 3u, Zona=Norte, Promocion=Obsequio} => {Rotacion=Baja} 0.176    0.838   1.237
8 {Formato=Pack de 3u, Zona=Norte}                      => {Rotacion=Baja} 0.192    0.827   1.222
```

9.2. Interpretación de las reglas

Llegados a este punto, el peligro al que se enfrenta el analista es el riesgo de interpretar erróneamente las reglas obtenidas. ¿Qué queremos decir con esto? Veámoslo en nuestro ejemplo.

La primera de las 8 reglas obtenidas nos dice que para el formato pack de 2 unidades y la promoción tipo degustación, se ha producido una rotación alta, con esperanza y elevación significativas.

Sin embargo, para una promoción tipo degustación no tenemos reglas generadas sobre formatos packs de 1 unidad o 3 unidades.

¿Podemos deducir que el formato pack de 2 unidades es el que mejor rotación obtiene con promociones tipo degustación?

¡Obviamente, no! La regla 1 solo nos habla de los formatos pack de 2 unidades y no podemos sacar conclusiones sobre ningún otro formato que no sea ese. Veámoslo.

Si realmente queremos comparar qué sucede para los distintos formatos de presentación, lo que deberemos hacer es extraer las reglas con la siguiente selección:

- rhs = (Rotación Alta).
- lhs = (Pack de 1u, Pack de 2u, Pack de 3u, Promoción Degustación y Promoción Obsequio).
- Reglas de al menos 3 componentes, con soporte 0,002 y esperanza 0,2 como mínimo.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_14.txt")* en vuestra sesión R.

```
# Generamos nuevas reglas con la selección establecida anteriormente
> reglas4 <- apriori(marketing, parameter = list(minlen=3, supp=0.002, conf=0.02),
  appearance = list(rhs=c("Rotacion=Alta"), lhs=c("Formato=Pack de 1u",
  "Formato=Pack de 2u", "Formato=Pack de 3u", "Promocion=Degustacion",
  "Promocion=Obsequio"), default="none"), control = list(verbose=F))
# Establecemos que las medidas tengan solo tres decimales
> quality(reglas4) <- round(quality(reglas4), digits=3)
# Ordenamos las reglas por medida de esperanza
> reglas4.ordenadas <- sort(reglas4, by="confidence")
```

```
# Visualizamos las 6 reglas obtenidas
> inspect(reglas4.ordenadas)
                          lhs                          rhs       support confidence lift
1 {Formato=Pack de 2u, Promocion=Degustacion}=>{Rotacion=Alta} 0.011   1.000    3.096
2 {Formato=Pack de 1u, Promocion=Degustacion}=>{Rotacion=Alta} 0.003   1.000    3.096
3 {Formato=Pack de 1u, Promocion=Obsequio}   =>{Rotacion=Alta} 0.090   0.618    1.912
4 {Formato=Pack de 2u, Promocion=Obsequio}   =>{Rotacion=Alta} 0.043   0.360    1.115
5 {Formato=Pack de 3u, Promocion=Degustacion}=>{Rotacion=Alta} 0.012   0.342    1.058
6 {Formato=Pack de 3u, Promocion=Obsequio}   =>{Rotacion=Alta} 0.069   0.241    0.746
```

Del resultado obtenido se desprende que no solo el formato pack de 1 unidad tiene rotación alta. El formato pack de 2 unidades también tiene rotación alta, lo que sucede es que en procesos anteriores esta regla se descartó porque no llegaba al mínimo establecido para las medidas de soporte y esperanza.

10. Máquinas de vectores de soporte

Si en el mundo de la inteligencia artificial existieran modas, podríamos decir que las *support vector machines* están de moda como algoritmos capaces de resolver problemas de clasificación y regresión. Actualmente está considerado el algoritmo más potente en reconocimiento de patrones.

Su eficiencia y los buenos resultados obtenidos en comparación con otros algoritmos han convertido esta técnica en la más utilizada en campos como reconocimiento de letra escrita y habla, predicción de series temporales y estudios sobre bases de datos de marketing. Pero también en otros campos como la secuenciación de proteínas y el diagnóstico de varios tipos de cáncer.

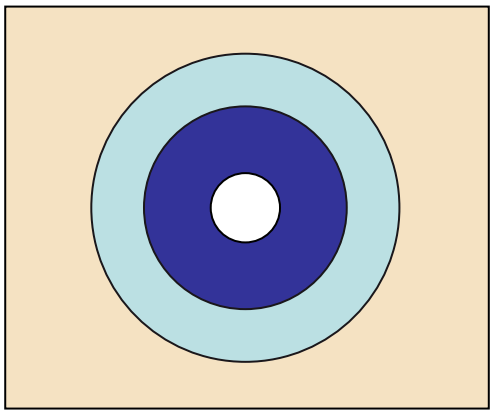
Support vector machine (SVM) es un algoritmo de aprendizaje supervisado capaz de resolver problemas de clasificación tanto lineales como no lineales.

Estrictamente hablando, SVM se ocupa tan solo de resolver problemas de clasificación lineales y se apoya en las funciones *kernel* para transformar un problema no lineal en el espacio original en un problema lineal en el espacio transformado.

1) ¿Cómo pueden ayudarnos las funciones *kernel*?

La figura 18 nos muestra un problema de clasificación de tres zonas: azul celeste, azul marino y blanco. Claramente se trata de un problema de clasificación no lineal puesto que visualmente observamos cómo solo dos circunferencias concéntricas son capaces de realizar una correcta clasificación.

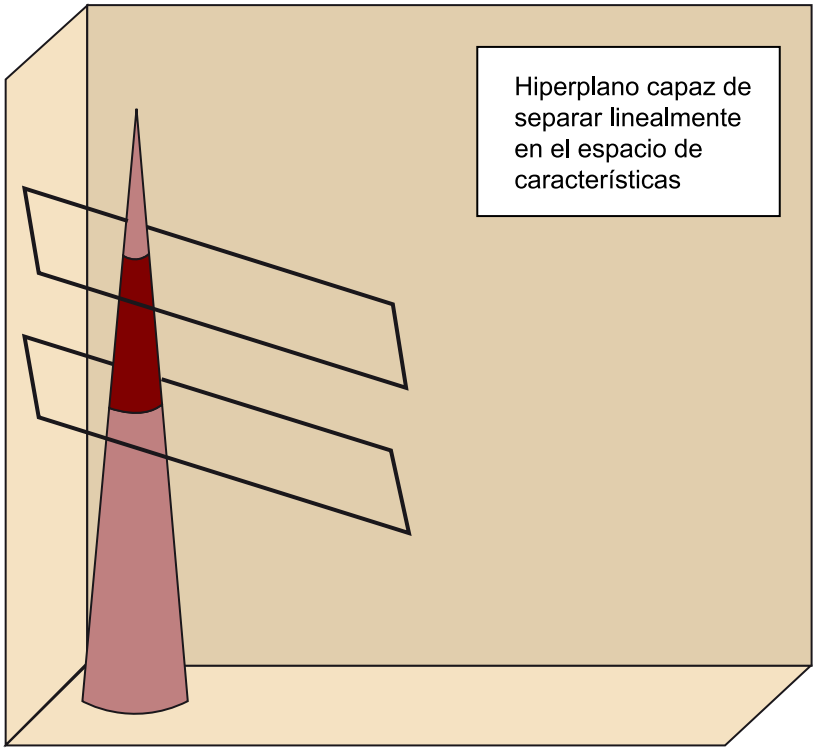
Figura 18. Clasificación no lineal con *support vector machines*



Supongamos que disponemos de una función *kernel* capaz de transportar la figura anterior a un espacio de tres dimensiones tal y como podemos ver en la figura 19.

De hecho podemos pensar la figura 18 como la vista aérea del cono dibujado en la figura 19.

Figura 19. Función *kernel* radial para *support vector machine*



En el espacio transformado por la función *kernel* (también espacio de características) sí que podemos separar los tres colores por planos, es decir, tenemos un problema de clasificación lineal. En este nuevo espacio podríamos aplicar el algoritmo SVM para clasificar las tres zonas de colores.

2) El concepto del margen

Uno de los factores diferenciadores de este algoritmo respecto de otros clasificadores es su gestión del concepto Margen.

Las SVM buscan maximizar el margen entre los puntos pertenecientes a los distintos grupos a clasificar. Maximizar el margen quiere decir que la curva de decisión o separación esté diseñada de tal forma que el máximo número de futuros puntos queden bien clasificados.

Por este motivo, las SVM utilizan las técnicas de optimización cuadrática propuestas por el matemático italiano Giuseppe Luigi Lagrange.

Ahora estamos en disposición de poner en práctica todo lo aprendido.

3) Caso de estudio

Utilizaremos el cuarto juego de datos, basado en el fichero *Clasificacion.csv*.

Recordemos que somos un fabricante de productos industriales que suministra pinturas junto a unos dosificadores diseñados específicamente para estas pinturas. Ante la disyuntiva de ceder o vender el dosificador, hemos construido un juego de datos que puntúa a nuestros clientes en nueve aspectos distintos.

Utilizaremos las SVM para construir un modelo capaz de predecir si para un nuevo cliente nos conviene ceder o vender el dosificador de pinturas. De este modo nos encontramos ante un problema de clasificación binaria no lineal.

Veamos cómo resolvemos este caso de estudio mediante R.

Empezaremos por cargar la biblioteca e1071, que contiene una implementación de SVM documentada en http://cran.r-project.org/web/packages/e1071/e1071.pdf.

Cargaremos también los datos en la tabla datos_ini, de la que mostraremos las seis primeras entradas.

```
# Cargamos la biblioteca e1071 que contiene la implementación svm
> library(e1071)
# Cargamos el juego de datos con las puntuaciones de clientes
> datos_ini <- read.csv('Clasificacion.csv',head=TRUE)
> head(datos_ini)
     Id      Activ  Tipo  Categ  Familia  Pais  Area  Ingresos Establec  Espec  Clase
1 1000025     5     1      1       1      2      1      3        1        1    venta
2 1002945     5     4      4       5      7      10     3        2        1    venta
3 1015425     3     1      1       1      2      2      3        1        1    venta
4 1016277     6     8      8       1      3      4      3        7        1    venta
5 1017023     4     1      1       3      2      1      3        1        1    venta
6 1017122     8     10    10       8      7      10     9        7        1    cesion
```

El identificador de cliente claramente es un atributo que no nos aporta nada, de modo que lo eliminaremos. Así mismo, procederemos a trabajar con dos tablas, una de datos y otra de clases.

```
# Separamos en dos tablas, los atributos y las clases
> datos_todo <- subset(datos_ini,select=c(-Id,-Clase))
> clases_todo <- subset(datos_ini,select=Clase)
# Mostramos los 6 primeros registros de la tabla de clases
> head(clases_todo)
   Clase
1  venta
2  venta
```

```
3  venta
4  venta
5  venta
6 cesion
```

Construimos dos juegos de datos por separado:

- Los datos de entrenamiento que nos servirán para construir el modelo.
- Los datos de test que nos servirán para validar el modelo.

```
# Separamos una muestra para datos de entrenamiento
> datos_entrena <- datos_todo[1:400,]
> clases_entrena <- clases_todo[1:400,]
# Separamos otra muestra para los datos de test
> datos_test <- datos_todo[401:699,]
> clases_test <- clases_todo[401:699,]
```

Construiremos el modelo a partir de datos y clases de entrenamiento.

La función *svm()* por defecto genera un modelo de clasificación, pero también puede generar modelos de regresión.

Al detectar que no se trata de un problema de clasificación lineal, la función ha optado por utilizar una función *kernel* radial como la descrita en la figura 19.

La función *svm()* trabaja con dos parámetros asociados a las funciones de Lagrange, Coste y gamma. Más adelante veremos cómo estos parámetros pueden ser ajustados para optimizar los resultados obtenidos.

```
# Generamos el modelo mediante la función svm()
> modelo1 <- svm(datos_entrena, clases_entrena)
> print(modelo1)
Call:  svm.default(x = datos_entrena, y = clases_entrena)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1111111
Number of Support Vectors:  82
```

Probaremos la fiabilidad del modelo obtenido, ejecutando una predicción sobre el juego de datos de test. Una matriz de confusión nos indicará el grado de aciertos del modelo.

```
# Utilizamos los datos de test para validar el modelo
> pred1 <- predict(modelo1, datos_test)
# Matriz de confusión
```

```
> table(pred1,t(clases_test))
pred1    cesion venta
  cesion     69     4
  venta       1   225
```

En la tabla de confusión, la lectura en vertical se corresponde con lo que tenemos en el juego de datos de test, mientras que la lectura en horizontal se corresponde con lo que tenemos en el juego de datos de predicción.

Podemos apreciar cómo el modelo generado solo ha cometido 5 errores sobre 299 casos de prueba:

- Ha predicho en cuatro ocasiones clase cesión cuando en realidad era clase venta.

- Ha predicho en una ocasión clase venta cuando en realidad era clase cesión.

Tal y como hemos comentado anteriormente, los parámetros coste y gamma pueden ser ajustados con el fin de tratar de obtener mejores resultados en el proceso de predicción.

La función *tune()* nos propondrá valores distintos de los parámetros en función de los datos de entrenamiento y los datos de test. Se trata de una función que permite jugar con los datos hasta encontrar un modelo eficiente y estable.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_20.txt")* en vuestra sesión R.

```
# Determinacion de cost y gamma óptimos (Lagrange)
> tune(svm, train.x=datos_entrena, train.y=clases_entrena, validation.x=datos_test,
  validation.y=clases_test, ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
  control = tune.control(sampling = "fix"))
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
  gamma cost
    1    4
- best performance: 0.06
```

Uno de los inconvenientes que presentan las SVM es su difícil interpretación por parte de personas no especialistas en el algoritmo. La capacidad de convertir los resultados de los modelos generados en información inteligible por parte del personal de negocio se convierte en un reto.

La representación gráfica del modelo es una herramienta con la que contamos para conseguir el objetivo de transmisión del significado del modelo.

4) Gráfico de clasificación

Trataremos de generar un gráfico donde podamos apreciar de una forma visual el modelo de clasificación generado por la función *svm()*.

Empezaremos por aclarar que la función *svm()* tiene dos implementaciones:

- La que hemos utilizado en el caso anterior y basada en juego de datos.
- Y una segunda implementación basada en fórmula y que utilizaremos para generar el gráfico de clasificación.

Para conseguir un gráfico fácil de entender, seleccionaremos de nuestro juego de datos dos atributos: los que tengan mayor ganancia de información.

```
# Cargamos la biblioteca que implementa la función ig()
> library(FSelector)
# Buscamos los atributos que contienen mas información respecto del atributo clase
> ig <- information.gain(Clase~., datos_ini)
> print(ig)
                attr_importance
Id                    0.0000000
Actividad             0.4363081
Tipo                  0.6658232
Categoria             0.6426183
Familia               0.4297682
Pais                  0.5048526
Area                  0.5846536
Ingresos              0.5286070
Establecimiento       0.4658462
Especializacion       0.1978519
```

Los atributos con mayor ganancia son “Tipo” y “Categoría”, de modo que generaremos el gráfico con estos dos atributos por ser los más representativos.

Empezaremos por crear un juego de datos con todos los atributos excepto el identificador de cliente.

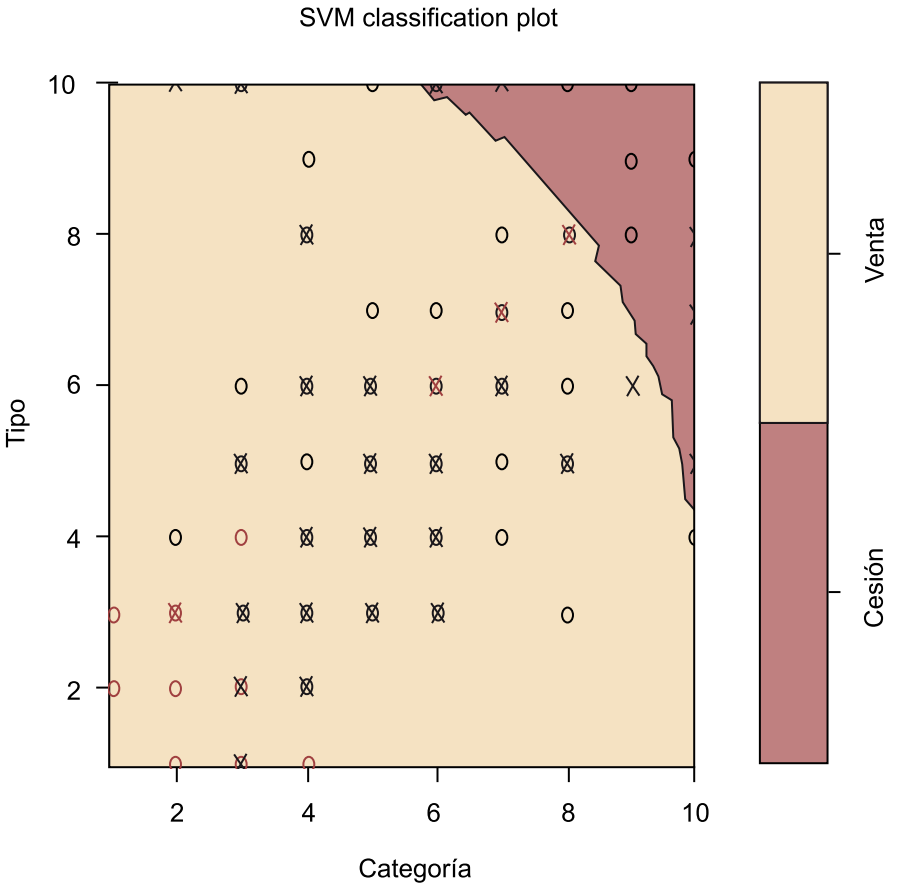
Para el juego de datos de entrenamiento, seleccionaremos las 400 primeras entradas.

```
# Construimos el juego de datos de trabajo
> datos2_todo <- subset(datos_ini,select=c(-Id))
# Reservamos 400 registros como datos de entrenamiento
> datos2_entrena <- datos2_todo[1:400,]
# Generamos el modelo a partir de los datos de entrenamiento
> modelo2 <- svm(Clase~., data = datos2_entrena)
# Generamos el gráfico que nos permitirá visualizar el modelo sobre 2 atributos
```

```
> plot(modelo2, datos2_entrena, Tipo ~ Categoria)
```

La figura 20 nos muestra el modelo obtenido en el que sobre dos atributos se distingue la curva de decisión entre las dos clases: venta o cesión.

Figura 20. Gráfico de visualización de un modelo SVM



En el gráfico se muestran dos tipos de puntos. Los que provienen del juego de datos de entrenamiento y los que provienen del espacio de transformación que se ha utilizado mediante la función *kernel*.

Se debe comentar también que la función *plot()* para la implementación SVM contiene parámetros que permiten adaptar el gráfico en cuanto a símbolos, colores y etiquetas.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_21.txt")* en vuestra sesión R.

```
# Función plot() con personalización de parámetros
> plot(modelo2, datos2_entrena, Tipo ~ Categoria, svSymbol = 1, dataSymbol = 2,
  symbolPalette = rainbow(4), color.palette = terrain.colors)
```

11. Redes neuronales

Utilizaremos el algoritmo de redes neuronales implementado en el paquete R *nnet* con el objetivo de clasificar el juego de datos *Clasificacion.csv*. De modo que nos situaremos en el mismo caso de estudio que hemos utilizado para el estudio del algoritmo de máquinas de vectores de soporte. Esto nos permitirá comparar las capacidades de ambos algoritmos.

Empezaremos recordando que una red neuronal está formada por los siguientes componentes:

- Datos de entrada.
- Neuronas organizadas por capas. Las neuronas a su vez se componen de una función de entrada, un vector umbral y una función de activación.
- Matrices de pesos que realizan la función de conectores entre neuronas.
- Datos de salida.

El proceso de aprendizaje consistirá en ajustar las matrices de pesos conectores de neuronas y de los vectores umbrales internos de las neuronas, de tal forma que quede minimizado el error de la red, entendiendo como error de la red la diferencia entre las salidas producidas y las salidas esperadas.

1) Aplicación en R

Recordemos que nuestro caso de estudio se basa en el juego de datos *Clasificación.csv* y que tiene por objetivo clasificar los nuevos clientes en función de si conviene ceder o vender los dosificadores de pinturas.

Inicialmente cargaremos la biblioteca R *nnet* e incorporaremos los datos del fichero csv.

```
# Cargamos la biblioteca nnet
> library("nnet")
# Incorporamos el juego de datos de trabajo
> datos_ini <- read.csv('Clasificacion.csv',head=TRUE)
# Eliminamos el atributo Id por no ser representativo
> datos3_todo <- subset(datos_ini,select=c(-Id))
# Seleccionamos 400 registros que formarán el juego de datos de entrenamiento
> samp <- c(sample(1:400,200), sample(450:699,200))
```

La función *nnet()* construirá un modelo capaz de predecir el atributo Clase en base al resto de atributos. Mediante el parámetro size = 2 establecemos que solo trabaje con 2 neuronas y mediante el parámetro maxit = 200 establecemos que solo el proceso de minimización del error como máximo realice 200 iteraciones.

```
# Fijamos un escenario de pruebas para poder repetir el experimento
> set.seed(2)
# Construimos el modelo mediante la función nnet()
> modelo_nnet <- nnet(Clase ~ ., data = datos3_todo, subset = samp, size = 2,
  rang = 0.2, decay = 5e-4, maxit = 200)
# Listamos los objetos que contiene el modelo generado
> ls(modelo_nnet)
[1]  "call"         "censored"      "coefnames"     "conn"
[5]  "convergence"  "decay"         "entropy"       "fitted.values"
[9]  "lev"          "n"             "nconn"         "nsunits"
[13] "nunits"       "residuals"     "softmax"       "terms"
[17] "value"        "wts"           "xlevels"
```

En el siguiente *script* generaremos una tabla de confusión donde la lectura en vertical se corresponde con lo que tenemos en el juego de datos de predicción, mientras que la lectura en horizontal se corresponde con lo que tenemos en el juego de datos de test.

Podemos apreciar cómo el modelo generado ha cometido 11 errores sobre 299 casos de prueba:

- Ha predicho en ocho ocasiones clase cesión cuando en realidad era clase venta.
- Ha predicho en tres ocasiones clase venta cuando en realidad era clase cesión.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_22.txt")* en vuestra sesión R.

```
# Generamos la matriz de confusión para los datos test
> table(datos3_todo$Clase[-samp], predict(modelo_nnet, datos3_todo[-samp,],
  type = "class"))
           cesion   venta
  cesion       104       3
  venta          8     184
```

Acabaremos con la generación de un gráfico donde veremos el estilo de red neuronal que hemos generado. Cabe notar que la función *plot.nnet()* se implementa en el *script RUOC_23.txt* de forma que será necesario ejecutarlo para poder obtener el gráfico.

Nota

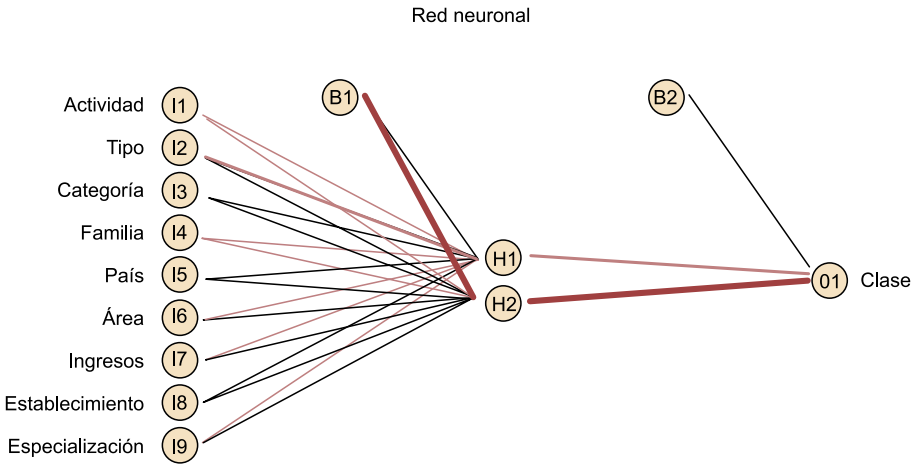
Podéis ejecutar este *script* escribiendo *source("RUOC_23.txt")* en vuestra sesión R.

```
# Gráfico de la red neuronal con parámetros por defecto
> plot.nnet(modelo_nnet,main='Red Neuronal')
```

```
# Gráfico de la red neuronal con personalización de parámetros
> plot.nnet(modelo_nnet,pos.col='darkgreen',neg.col='darkblue',alpha.val=0.7,
  rel.rsc=15,circle.cex=5,cex=1.4,circle.col='yellow',main='Red Neuronal')
```

La figura 21 nos muestra la salida de la función *plot.nnet()*, donde vemos que el modelo de red neuronal generado contiene dos neuronas H1 y H2 y dos matrices de pesos conectores B1 y B2.

Figura 21. Clasificación de clientes con una red neuronal



12. Visualización de datos

La facilidad de R para generar gráficos es uno de sus puntos fuertes. Tanto es así que grandes corporaciones como Google y Facebook, que realizan un uso intensivo del análisis de datos, han utilizado en muchas ocasiones el motor de gráficos R para explicar estudios de tráfico, uso y tendencias.

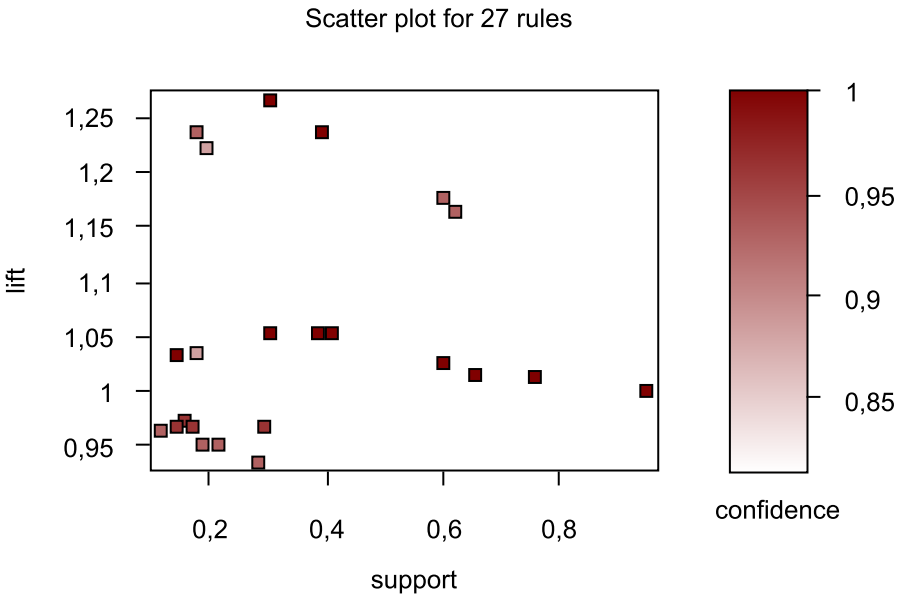
Las matemáticas, la estadística y los algoritmos propios de la minería de datos han demostrado a lo largo del tiempo que tienen un enorme potencial para encontrar patrones, tendencias y relaciones aparentemente imposibles.

Sin embargo, muchas veces fallan en la tarea de ofrecer los resultados de una forma fácil de entender y de asimilar. En este sentido, la comunidad de desarrolladores que ha hecho crecer R ha entendido perfectamente y desde el principio que los gráficos juegan un papel crucial en la fase de comunicación de resultados en todo proceso de análisis de datos.

En este sentido animamos al estudiante a investigar y experimentar las posibilidades de R en este campo. Utilizaremos las reglas generadas en el estudio de asociaciones para presentarlas mediante diferentes tipos de gráficos R.

```
# Cargamos las bibliotecas necesarias
> library(arulesViz)
# Gráfico de dispersión en base a las medidas soporte, elevación y confianza
plot(reglas1,measure=c("support","lift"),shading="confidence")
```

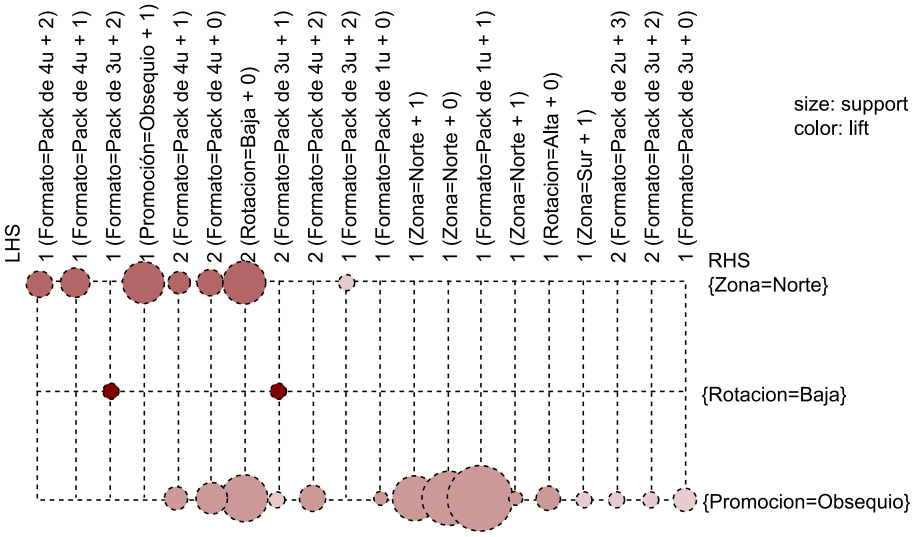
Figura 22. Gráfico de dispersión de medidas de reglas



Este gráfico es especialmente útil para representar un gran número de puntos. Observad que mediante una imagen 2D se gestionan tres variables.

```
# Matriz de reglas agrupadas. Grouped matrix-based
plot(reglas1, method='grouped')
```

Figura 23. Matriz de agrupaciones de reglas



Utilizaremos este tipo de gráfico para explicar un universo de pocos casos.

Para entender cómo hay que leerla tomaremos, por ejemplo, la quinta entrada empezando por la izquierda. Esta nos dice lo siguiente:

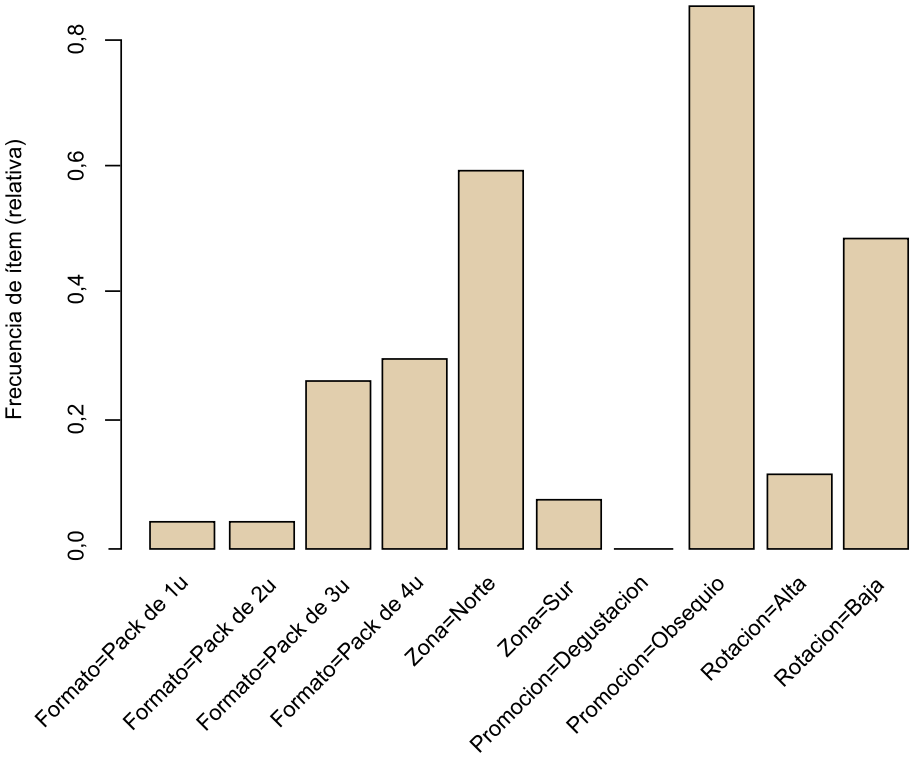
2 (Formato = Pack de 4u + 1)

Esto significa que con el soporte y elevación indicados en el gráfico, hay 2 reglas que tienen dos componentes antecedentes *lhs*, uno de los cuales es *Pack de 4u*.

En el gráfico se utilizan el tamaño y color de las bolas para representar medidas numéricas.

```
# Histograma de frecuencias relativas de items
itemFrequencyPlot(items(reglas1), main="Histograma de componentes")
```

Figura 24. Histograma de frecuencias relativas de componentes



En este caso vemos que el formato Pack de 3u aparece 7 veces en el juego de datos *reglas1*, de modo que tiene una frecuencia relativa de 0,26 = 7 / 27.

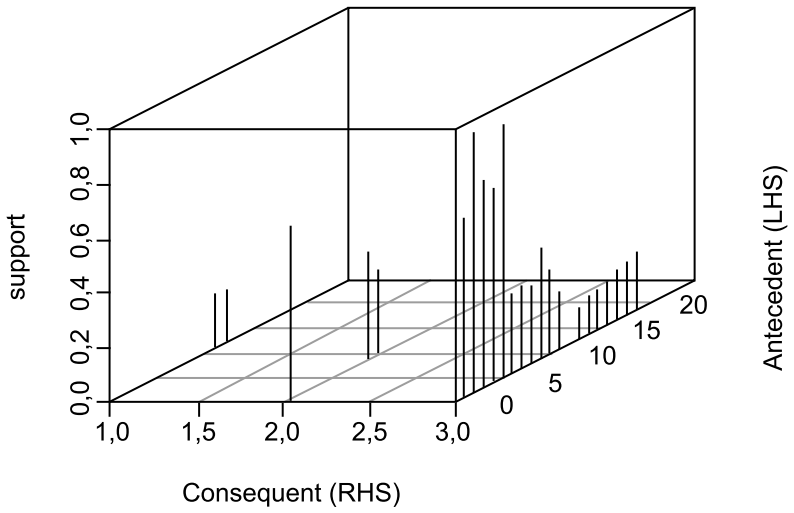
El siguiente gráfico que presentamos es un gráfico de dispersión en 3D. Cada varilla vertical representa una regla, su altura indica su soporte, mientras que su localización en el suelo, depende de sus componentes *lhs* y *rhs*.

Puesto que su interpretación puede llevar a confusión, se acompaña el gráfico de una salida de texto en el que se detallan la organización de las reglas representadas.

```
# Gráfico de dispersión en base a medidas: soporte, elevación y confianza
>plot(reglas1,method="matrix3D",measure="support",control=list(reorder=TRUE))
Itemsets in Antecedent (LHS)
 [1] "{Rotacion=Baja}"                                "{}"
 [3] "{Zona=Norte}"                                   "{Zona=Norte,Rotacion=Baja}"
 [5] "{Formato=Pack de 4u,Zona=Norte}                 "{Formato=Pack de 4u,
                                                       Zona=Norte,Rotacion=Baja}"
 [7] "{Rotacion=Alta}"                                "{Formato=Pack de 3u}"
 [9] "{Formato=Pack de 4u}"                           "{Formato=Pack de 4u,
                                                       Rotacion=Baja}"
[11] "{Formato=Pack de 3u,Zona=Norte}"                "{Formato=Pack de 3u,
                                                       Zona=Norte,
                                                       Promocion=Obsequio}"
[13] "{Formato=Pack de 2u}"                           "{Zona=Sur,Rotacion=Alta}"
[15] "{Formato=Pack de 1u}"                           "{Zona=Norte,Rotacion=Alta}"
```

```
[17] "{Formato=Pack de 3u,Zona=Norte,Rotacion=Baja}"          "{Zona=Sur}"
[19] "{Formato=Pack de 3u,Rotacion=Baja}"                     "{Formato=Pack de 3u,
                                                               Promocion=Obsequio,
                                                               Rotacion=Baja}"
[21] "{Formato=Pack de 4u,Promocion=Obsequio,Rotacion=Baja}" "{Formato=Pack de 4u,
                                                               Promocion=Obsequio}"
[23] "{Promocion=Obsequio,Rotacion=Baja}"
Itemsets in Consequent (RHS)
[1]  "{Rotacion=Baja}"      "{Zona=Norte}"         "{Promocion=Obsequio}"
```

Figura 25. Cubo 3D de reglas



Finalmente incluimos una salida no a gráfico sino a formato PMML. Recordamos que PMML es un estándar basado en XML para almacenar el resultado del *scoring* de modelos de minería de datos.

En el siguiente *script* puede observarse lo fácil que es para R generar un fichero xml con el resultado de nuestro análisis de asociaciones.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_15.txt")* en vuestra sesión R.

Tras ejecutar el *script*, encontraréis en el directorio de trabajo R un fichero con nombre *reglas.xml*.

```
# Cargamos las bibliotecas necesarias
> library(pmml)
> write.PMML(reglas2.ordenadas, 'reglas.xml')
```

En la siguiente tabla se da una lista de los paquetes especializados en funciones para generar gráficos:

Tabla 10

Paquete	Descripción
ggplot2	Mejora las funciones básicas de R.
tabplot	Table plot. Para visualización de juegos de datos con muchos atributos.

Paquete	Descripción
plotrix	Gráficos radiales, circulares, formato reloj, etc.
wordcloud	Nube de palabras.
quantmod	Gráficos de evolución de valores financieros.
waterfall	Gráficos en formato cascada.
sp	Gráficos basados en posicionamiento geográfico.
VennDiagram	Visualización de intersecciones de conjuntos.
playwith	Produce gráficos interactivos. Va acompañado de su propia GUI.
igraph	Gráficos interactivos.
rggobi	Gráficos interactivos pensados para la exploración de datos.
latticist	Gráficos interactivos pensados para la exploración de datos.

13. *Text mining*

En este apartado utilizaremos R para ejecutar tareas propias de la minería de textos. Hemos recuperado quince opiniones extraídas de la página web que Nokia puso a disposición para su modelo de móvil Lumia 800: http://www.nokia.com/es-es/productos/moviles/lumia800/reviews/.

Estas opiniones se han guardado en quince ficheros de texto con nombre *opinion_nn.txt*. De esta forma, el proceso empieza por la carga de esta información en una variable tipo 'Corpus', específica para el tratamiento de textos.

Sobre esta variable realizaremos tareas de preprocesado del texto: conversión de mayúsculas, eliminación de números, eliminación de direcciones URL, eliminación de preposiciones y conjunciones (*stopwords*) y agrupación de palabras en función de su raíz (*stemming*).

Sobre esta base, se construirá una **matriz de palabras**, en la que cada columna corresponderá a un documento, cada fila a una palabra y cada celda contendrá el número de veces que aparece la palabra en el documento.

La matriz de palabras es el objeto central sobre el que se ejecutarán algoritmos propios de la minería de datos. Estudiaremos la frecuencia de aparición de palabras y generaremos una nube de palabras.

1) Tareas de preprocesado de texto

El siguiente *script* requiere que el estudiante en el directorio de trabajo R haya creado una carpeta con nombre *txt*, donde se habrán guardado los 15 ficheros *opinion_nn.txt*.

Se procede a cargar los 15 ficheros y a realizar eliminaciones básicas disponibles en el paquete *tm*. Podemos verificar todas las posibilidades de eliminación, ejecutando la función *getTransformations()* en la sesión R.

```
library(tm)
# Guardamos en la variable dir la ruta del directorio txt
> dir <- paste(getwd(),'/txt', sep="")
# Generamos la variable Corpus con todos los documentos
> (nokia <- Corpus(DirSource(dir, encoding = "UTF-8"),
  readerControl = list(language = "es")))
# Eliminamos las mayúsculas
> nokia <- tm_map(nokia, tolower)
# Eliminamos los signos de puntuación
> nokia <- tm_map(nokia, removePunctuation)
```

```
# Eliminamos los números
> nokia <- tm_map(nokia, removeNumbers)
```

En el proceso de eliminación de direcciones URL utilizamos la expresión propia de R *alnum*, que significa 'que contenga valores alfanuméricos y numéricos'.

Nota

Se pueden consultar todas las expresiones propias, con el comando R *?regex*.

La función *gsub()* nos servirá para suprimir cadenas de texto.

```
# Eliminamos URLs
> quitarURL <- function(x) gsub("http[[:alnum:]]*", "", x)
> nokia <- tm_map(nokia, quitarURL)
# Visualizamos el contenido del Corpus
> inspect(nokia)
```

Continuaremos con el procedimiento para eliminar las *stopwords*.

R dispone de un diccionario básico de *stopwords* por idioma. En esta ocasión lo ampliaremos con palabras adicionales que nos interesará eliminar por carecer de significado propio para nuestro caso.

```
# Guardamos las conjunciones, preposiciones y demás palabras
  que querremos eliminar
> quitarStopwords <- c(stopwords('spanish'), "mas", "dado", "hecho", "tan", "deben",
  "decir")
# Mantenemos algunas palabras incluidas en el diccionario stopwords('spanish')
> quitarStopwords <- setdiff(quitarStopwords, c("trabajo"))
# Prodcedemos a la eliminación de las Stopwords
> nokia <- tm_map(nokia, removeWords, quitarStopwords)
```

Seguiremos con el proceso de agrupación de palabras según su raíz[4]. Este proceso se lleva a cabo en tres pasos:

(4)En inglés, *stemming*.

- inicialmente, realizaremos una copia del corpus,
- posteriormente, reduciremos las palabras a sus raíces, y
- finalmente, a partir de la copia del corpus, pasaremos de la palabra raíz a la palabra original.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_16.txt")* en vuestra sesión R.

```
# Agrupación de las palabras derivadas. Stemming
# Guardamos una copia
> nokiaCopia <- nokia
> nokia <- tm_map(nokia, stemDocument)
# Substituimos los radicales por los originales
> nokia <- tm_map(nokia, stemCompletion, dictionary=nokiaCopia)
```

```
> inspect(nokia)
```

2) Matriz de palabras

Para construir la matriz de palabras, consideraremos palabras a partir de 2 caracteres. Para ello, utilizaremos la función *TermDocumentMatrix()*, que además de generar la matriz, también nos dará unas estadísticas básicas.

Por ejemplo, podremos observar que hay un alto grado de dispersión puesto que el 91% de las celdas de la matriz de palabras tienen el valor cero.

```
# Construimos la Matrix que relaciona palabras con documentos
> nokiaTdm <- TermDocumentMatrix(nokia, control=list(wordLengths=c(2,Inf)))
> nokiaTdm
A term-document matrix (271 terms, 15 documents)
Non-/sparse entries: 362/3703
Sparsity           : 91%
Maximal term length: 16
Weighting          : term frequency (tf)
```

A partir de la matriz de palabras podemos comprobar en qué documentos aparece, por ejemplo, la palabra *batería*. También podemos realizar un estudio de frecuencia de presencia de palabras en los documentos.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_17.txt")* en vuestra sesión R.

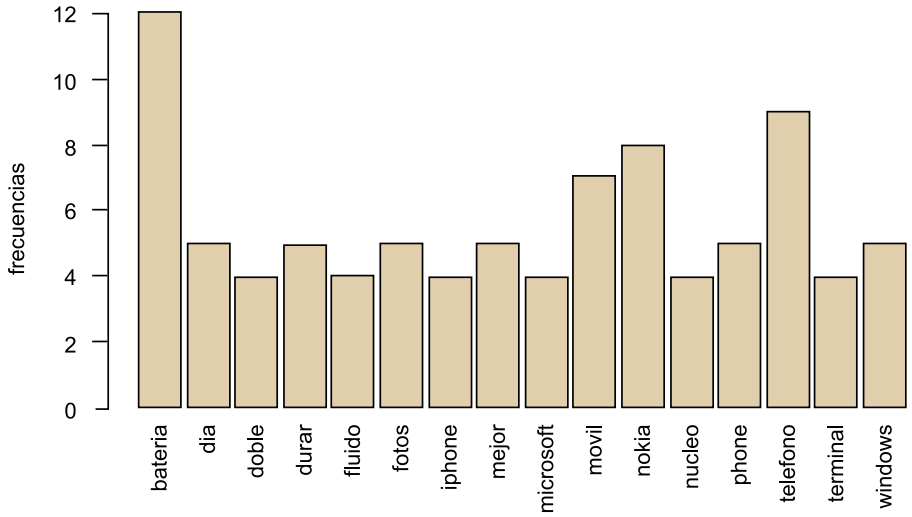
Un gráfico tipo histograma es una herramienta excelente para visualizar las palabras con frecuencia superior a cuatro veces.

```
# Cuantos hablan sobre la bateria
> inspect(nokiaTdm["bateria",])
# Estudio de frecuencias
> print(findFreqTerms(nokiaTdm, lowfreq=4))

# Gráfico de frecuencias de palabras
> frecPalabras1 <- rowSums(as.matrix(nokiaTdm))
# Gráfico de frecuencias superiores a 4
> frecPalabras2 <- subset(frecPalabras1, frecPalabras1>=4)
> barplot(frecPalabras2, las=2, ylab="Frecuencias", main="Histograma de frecuencias")
```

La figura 26 nos muestra claramente que el cliente usuario del modelo Nokia Lumia tiene una clara percepción de que la batería es un punto a mejorar en las prestaciones del móvil.

Figura 26. Frecuencia de palabras en documentos



3) Nube de palabras

Para visualizar la importancia de las palabras en un grupo de documentos, el gráfico tipo nube de palabras es una buena herramienta y R la implementa de una forma muy sencilla a través del paquete *wordcloud*.

A partir de la matriz de palabras, generaremos un vector de frecuencias que utilizaremos primero para asignar a cada palabra distintas intensidades de color gris en función de su frecuencia de aparición.

```
> library(wordcloud)
# Convertimos la matriz de frecuencia de palabras
> matriz <- as.matrix(nokiaTdm)
# Generamos el vector de frecuencia de las palabras
# y las ordenados de forma descendiente
> frecPalabras3 <- sort(rowSums(matriz), decreasing=TRUE)
```

Utilizaremos también el vector de frecuencias en la propia función *wordcloud()* donde fijaremos el parámetro *random.order=F* para conseguir que primero imprima en la parte central las palabras más relevantes.

Nota

Podéis ejecutar este *script* escribiendo *source("RUOC_18.txt")* en vuestra sesión R.

```
# Fijamos una semilla para que el ejemplo se pueda repetir
> set.seed(123)
# Calculamos los niveles de grises en función de la relevancia de las palabras
> nivelGrises <- gray( (frecPalabras3+10) / (max(frecPalabras3) + 10))
# Generamos la nube de palabras
> wordcloud(words=names(frecPalabras3), freq=frecPalabras3, min.freq=3, random.order=F,
  colors=nivelGrises)
```

La figura 27 nos muestra la nube de palabras asociadas a las quince opiniones sobre el móvil Nokia Lumia. Observamos cómo la batería ocupa un espacio central y cómo se relaciona mucho el aparato con su sistema operativo Microsoft y con su competidor iPhone.

Figura 27. Nube de palabras para el modelo Nokia Lumia

4) Segmentación de palabras

También podemos encontrar clústeres de palabras a partir de la función *hclust()*.

Para ello procederemos a eliminar las palabras que tienen poca frecuencia de aparición en los documentos.

```
# Reduciremos la dispersión de palabras en los documentos
> nokiaTdm2 <- removeSparseTerms(nokiaTdm, sparse=0.82)
# Guardamos las palabras que están en algún documento
> a <- rowSums(as.matrix(nokiaTdm2))
# Creamos una matriz con las palabras seleccionadas
> matriz2 <- as.matrix(nokiaTdm2)[names(a),]
# Calculamos las distancias entre las palabras
> matrizDistancias <- dist(scale(matriz2))
```

Seguiremos el proceso ejecutando la función de *clustering hclust()* con el método "Ward", que persigue minimizar la variancia y es especialmente apropiado para encontrar segmentos muy compactos.

Nota

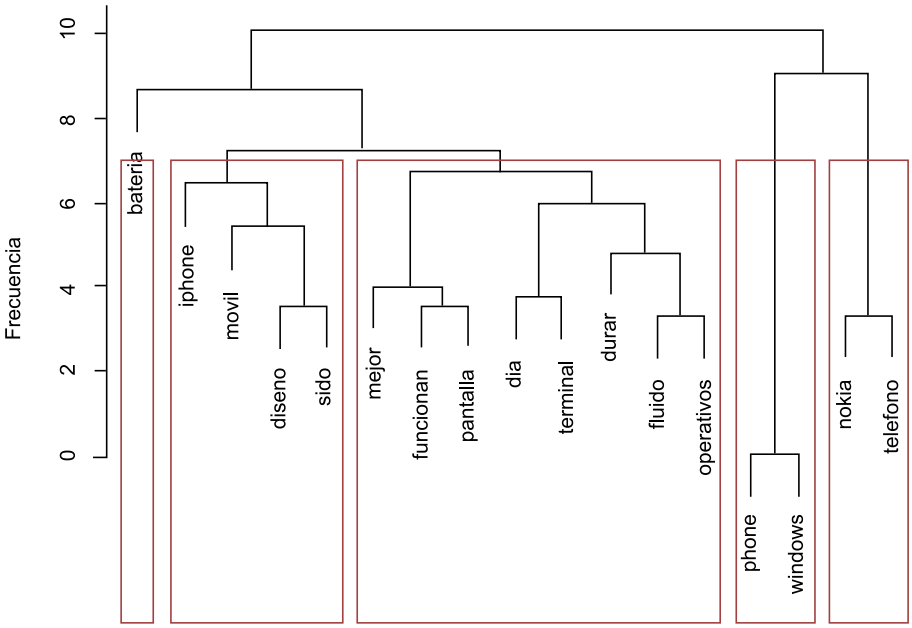
Podéis ejecutar este *script* escribiendo *source("RUOC_19.txt")* en vuestra sesión R.

```
# Lanzamos la segmentación
> segmentacion <- hclust(matrizDistancias, method="ward")
# Generamos el gráfico dendograma
> plot(segmentacion, ylab="Frecuencia", main="Segmentación de palabras")
# Dibujamos 5 clústers
```

```
> rect.hclust(segmentacion, k=5)
```

En la figura 28 observamos cómo se distingue un gran clúster central que aglutina atributos y adjetivos de un móvil (pantalla, terminal, mejor, durar) y un clúster específico para la batería, dada su relevancia en las quince opiniones seleccionadas.

Figura 28. *Clustering* de palabras



5) Paquete twitteR

Por último, hemos creído adecuado presentar el paquete *TwitteR*, ya que recoge una funcionalidad ideal para llevar a cabo tareas de minería de textos. Las funciones *userTimeline()* y *searchTwitter()* permiten descargar al entorno R mensajes con una facilidad impresionante.

En este material didáctico no exploraremos estas posibilidades puesto que *TwitteR* exige disponer de licencia usuario desarrollador que, pese a no ser complicado obtenerla, excedería el ámbito de lo que se le pide al estudiante.

Resumen

Para presentar las conclusiones del material didáctico usaremos el caso práctico de 7 Eleven que os exponemos a continuación.

7 Eleven es una de las mayores franquicias a escala mundial de tiendas de conveniencia. Su historia es realmente dilatada, hasta tal punto que su centro de decisión ha basculado varias veces entre Estados Unidos y Japón.

James Keyes, presidente de la compañía, en la década de los noventa se dio cuenta de que no tenían el control de lo que se vendía en sus tiendas, los proveedores habituales sabían mejor que ellos mismos lo que necesitaba cada tienda. Para mejorar las deficiencias que esto provocaba inició un proceso de informatización de toda la compañía.

Fue una visita a Japón lo que le hizo abrir los ojos, de tal forma que consiguió, en términos de gestión de la información, ir mucho más allá de lo que por aquel entonces hacía el resto de la competencia.

Las tiendas de Japón habían desarrollado un sistema tecnológicamente muy rudimentario pero con una excelente visión estratégica de gestión de la información. Cada tienda reportaba tres veces al día a un repositorio de datos central el detalle de sus operaciones. Sobre estos datos aplicaban de forma continua técnicas de minería de datos con el fin de saber para cada tienda individual no solo qué necesitaba el cliente de la zona sino también cuando y en qué presentación era más conveniente.

Las franquicias de Japón funcionaban tan bien porque consiguieron que la información fluyera de una forma inteligente desde el punto de venta hasta los centros suministradores y distribuidores. Rudimentario, pero eficiente... Keyes insistía en que la clave para exportar este modelo al resto de franquicias del globo era conseguir formar un equipo IT capaz de pensar como los comerciantes.

Todos los algoritmos de minería de datos aprendidos, tanto desde un punto de vista teórico como desde una aproximación más práctica con R, requerirán siempre en todas sus fases, desde el planteamiento inicial hasta el despliegue del modelo, una visión de negocio. Este será siempre el factor clave que distingue la estadística y la inteligencia artificial del *business analytics* y del *data mining*.

Conseguir que nuestros departamentos o nuestras empresas puedan ser consideradas *data driven organizations* será cada vez más el reto del ser o no ser en el futuro.

Con este material didáctico nos hemos acercado de una forma paulatina y consistente al entorno de programación R como una herramienta capaz de desarrollar capacidades propias de la minería de datos.

Hemos explorado algoritmos de clasificación como los árboles de decisión, las máquinas de vectores de soporte y las redes neuronales. También hemos estudiado las posibilidades de algoritmos de segmentación como *Kmeans*.

R también nos ha permitido realizar una práctica detallada de minería de textos, unas técnicas absolutamente necesarias para entender las posibilidades de Internet.

Finalmente, hemos explorado las posibilidades de R en cuanto a la visualización de datos. Muchos expertos consideran este el aspecto más fuerte de R como entorno de desarrollo de algoritmos.

Los fundamentos adquiridos con este material didáctico permitirán al estudiante continuar su camino de crecimiento profesional con una nueva visión de la minería de datos y, por ende, una nueva visión del mundo empresarial moderno.

Bibliografía

Chang, Winston (2012). *R Graphics Cookbook*. O'Reilly.

Kabacoff, Robert I. (2011). *R in Action. Data Analysis and graphics with R*. Manning.

Kuhn, Max; Johnson, Kjell (2013). *Applied Predictive Modeling*. Springer.

Mining at UOC (prácticas de minería de datos con R): http://data-mining.business-intelligence.uoc.edu/

Ohri, A. (2012). *R for Business Analytics*. Springer.

Planet R (sobre nuevos paquetes y funcionalidades R): http://planetr.stderr.org/

R-Bloggers (*hub* de blogs sobre temática R): http://www.r-bloggers.com/

R-Statistics (compilación de funcionalidades R): http://www.r-statistics.com/

R-Studio (promotores del entorno de desarrollo IDE): http://www.rstudio.com/

Rseek (buscador web vía Google, de temática R): http://rseek.org/

Styleguide (guía de estilos Google para la programación R): http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html#attach

Theodoridis, Sergios; Koutroumbas, Konstantinos (2009). *Pattern Recognition* (4.ª ed.). Academic Press.

Zhao, Yanchang (2013). *R and DATA MINING. Examples and Case Studies*. Academic Press.