Selección de Instancias y Atributos en Conjuntos de Datos mediante Algoritmos sobre Grafos

Carlos A. García Vallejo, 31217773-B

vallejo@us.es

Dirigida por el Prof. Dr. José Antonio Troyano Jiménez

Memoria de Tesis Doctoral

Índice general

Índice de figuras

Índice de tablas

Índice de Algoritmos

Agradecimientos

Si el agradecimiento se midiera por el número de líneas de texto, esta memoria de tesis debería tener dos volúmenes, uno de ellos dedicado a todo lo que le debo a José Antonio Troyano. Por todo: por sus brillantes ideas, por su aguda inteligencia, por su bonhomía, por su optimismo a prueba de bomba, por saber levantarte cuando piensas que todo está mal, por tantas cosas... pero sobre todo, por creer en mí e incluso, más difícil todavía, conseguir que yo mismo creyera en mí.

Una buena proporción del otro volumen tendría que dedicárselo a mi buen amigo Paco, por su conocimiento enciclopédico, por descubrirme que Perron Frobenius, pese a su nombre, no muerde, por las apasionantes charlas sobre ciencia, por su interés en mi trabajo, por escucharme con atención, por enseñarme lo que es la buena estadística.

A los compañeros del grupo de investigación, Fermín, Fernando, Javi, por todo lo que saben, por todo lo que hacen y porque me han mostrado que una reunión de investigación no tiene que ser aburrida. Al resto de los itálicos, por su acogida y por hacerme más sencilla la vida.

A Jose Ra. por su amistad y su impagable servicio LaTeX 24 horas.

A Miguel porque hace ya ¡22 años! creyó que yo podría ser un buen profesor de universidad. A Pepe, por la revisión que le hizo a mi primer artículo y sin la cual quizás aún estaría sin publicar. A R.C. por sus retos sobre los plazos de esta tesis. A Manolo, porque me hizo muy sencillo el aterrizaje en la asignatura cuando llegué al Departamento.

A los añorados compañeros de ADAEDA y a los de FP, y, en general, a todos los buenos compañeros, con los que da gusto encontrarse y tomar un café. Quiero mencionar especialmente a Mayte, por esas largas charlas que tanto bien nos hicieron, a Toñi, y su incansable capacidad de trabajo, a María José por su amistad. No puedo olvidar también los ensayos con Irene y Diana el día antes de mi presentación en CAEPIA; sus comentarios

mejoraron mi exposición y me dieron la tranquilidad que necesitaba y no tenía.

A Lupe, por el interés con el que escucha mis explicaciones sobre instancias, atributos, aristas, convergencias, test estadísticos…, hasta que se aburre y se rasca la oreja con la pata trasera.

A Victoria, con la que compartí el mes más productivo de mi vida en Lisboa.

A los chicos y chicas del Bulebar y de El Pimiento, con los que siempre encontraba compañía al final de los días de intenso trabajo.

A María y Paco, por los cafés y las tostadas matutinas sin las que no soy persona.

A la familia de Irene, por hacerme sentir en familia.

A Nuria y Diego, por darme un espacio propio y compartido en el mundo.

A la panda del Puma, por la que los jueves son esperados con más ganas.

A las amigas y los amigos (muchos los cuales figuran también más arriba), por eso, porque están ahí (y porque la amistad es lo más bonito que hay ;-)

A Leonardo y Don Paco (y a Lola, claro), por su granito de arena, sin el que la máquina no se hubiera puesto a funcionar.

A los alumnos internos, José Manuel Camacho Sosa y Gabriel Muñoz Ríos, por su ayuda en las implementaciones.

Al clúster de supercomputación de CICA, sin el que todavía estaría esperando varios años a terminar el trabajo experimental.

Voy parando ya, que "yo no cumplo años, yo cumplo con to el mundo". Si se me olvida alguien (que seguro que sí, ¡ay, la edad!), que silbe, que aparecerá en la versión 2.0.

Dedicatoria

A
Irene,
los amigos,
mi tita Rosi,
los deliciosos almuerzos que prepara mi tita Rosi:
todo aquello por lo que la vida merece la pena.

Y a la memoria de
mi tío abuelo Luis, que me contagió la fascinación por la técnica,
mi padre, que me transmitió la pasión por los números,
mi madre, por su cariño incondicional.

Menos es más

Ludwig Mies van der Rohe, arquitecto y diseñador

Menos es más

(Liu y Motoda, 1998a).

Menos no necesariamente es más.

Como hijo del modernismo he oído este mantra toda mi vida. Menos es más. Una mañana al despertar me di cuenta de que era una solemne tontería, una proposición absurda y bastante sin sentido. Pero suena muy bien, porque contiene dentro de sí una paradoja que se resiste a la compresión. Pero queda sencillamente anulada cuando piensas en la historia visual del mundo. Si miras una alfombra persa, no puedes decir que menos es más porque te das cuenta de que cada parte de esa alfombra, cada cambio de color, cada cambio en la forma es absolutamente esencial para su éxito estético. No puedes convencerme de que una alfombra lisa de color azul es de ninguna manera superior. Esto también sirve para el trabajo de Gaudí, las miniaturas persas, el art nouveau y todo lo demás. Sin embargo, tengo una alternativa para esa frase que creo que es más apropiada. "**Justo lo necesario es más**".

Milton Glaser, diseñador del logo I ♥ NY.

Resumen

Los grafos son de utilidad en un sinfín de problemas. Con la aparición de las redes de comunicaciones, internet, las redes sociales, etcétera, que no son sino grafos, su utilidad y aplicabilidad se ha visto incrementada. La Minería de Datos se ha utilizado ampliamente para analizar los grafos; sin embargo en pocas ocasiones se han utilizado los grafos para desarrollar algoritmos de Minería de Datos.

Muchos problemas están descritos naturalmente como grafos, como todos aquellos que tengan que ver con rutas, flujos, redes, etcétera; este tipo de problemas se resuelve evidentemente con algoritmos sobre grafos. Otros problemas no tienen en principio nada que ver con un grafo, pero haciendo las transformaciones oportunas, pueden convertirse en un grafo, pudiéndose ya resolver con los algoritmos correspondientes.

La posibilidad que ofrecen los grafos de analizar informaciones en un contexto global nos animó a abordar problemas clásicos de la Minería de Datos sobre esta estructura, en concreto la selección de subconjuntos de atributos y de subconjuntos de instancias para ser aplicados a la clasificación.

Hemos desarrollado sendos algoritmos: uno para la selección de un subconjunto instancias, en el que consideramos las instancias como nodos de un grafo sobre el que aplicamos un algoritmo similar al PageRank que nos permite establecer un rango entre las instancias. A partir de este ranking hemos desarrollado un algoritmo de selección de instancias que resulta tan bueno en precisión como los mejores algoritmos existentes, mejorándolos en reducción.

El otro problema que hemos abordado es el de la selección de un subconjunto de atributos. Para esto hemos representado los atributos como nodos de un grafo, usando correlaciones como aristas y a este grafo le hemos aplicado un algoritmo clásico, el del max-flow/min-cut, que nos devuelve un subconjunto de atributos. El algoritmo desarrollado obtiene unos resultados muy buenos.

Siendo conscientes de que en el mundo de la Minería de Datos el avan-

ce de una décima puede ser poco, para dotar de fortaleza nuestras afirmaciones hemos realizado un riguroso análisis estadístico de los resultados que nos permite afirmar que sí que hemos conseguido buenos algoritmos.

Parte I

Préambulo

Capítulo 1

Introducción

1.1. Motivación

Dentro del Aprendizaje Supervisado en la Minería de Datos, el problema de la Clasificación consiste en predecir la clase de un nuevo elemento de un conjunto de datos a partir de datos etiquetados previamente conocidos. Una de las técnicas más utilizadas y que mejores resultados proporciona es la del vecino o los vecinos más cercanos. Sin embargo, esta técnica adolece de dos problemas: por una parte, al no construir un modelo, hay que mantener almacenados todos los datos en memoria, con el consiguiente coste de almacenamiento; por otra parte, y este es el principal problema, cuando clasificamos un nuevo elemento (en la llamada fase de generalización), este tiene que ser comparado con todos los que conocemos (el conjunto de entrenamiento), con el consiguiente coste computacional. Si llamamos n al número de elementos del conjunto de datos, la clasificación de un elemento usando esta sencilla técnica es $O(n)$.

Por otra parte, comparar dos elementos tiene también un coste. Si llamamos m al número de atributos del conjunto de datos la comparación de dos elementos tendrá un coste $O(m)$. Por tanto, la técnica de los vecinos más cercanos tiene un coste $O(nm)$.

La forma clásica de mejorar este rendimiento se reduce a disminuir el valor de n o el de m, intentando no perder precisión en el proceso. Las técnicas de Selección de Instancias buscan escoger un subconjunto de las

instancias originales con las que se clasifique casi tan bien (si no mejor) que con el conjunto completo: se disminuye n. Las técnicas de Selección de Atributos buscan escoger un subconjunto de los atributos originales con los que se clasifique casi tan bien (si no mejor) que con todos los atributos: se disminuye m.

En el presente trabajo vamos a abordar ambos problemas, y lo vamos a hacer de una forma novedosa: en el primer caso (selección de instancias) modelando las instancias y las relaciones entre ellas como un grafo y aplicando un algoritmo sobre él. En el segundo caso (selección de características o atributos), modelando los atributos y las relaciones entre ellos como un grafo y aplicando sobre él un algoritmo.

1.2. Hipótesis

La hipótesis fundamental de nuestro trabajo es la siguiente:

Hipótesis 1: *Los grafos tienen una potencia expresiva que permite modelar y resolver sobre ellos problemas aparentemente lejanos.*

Esta hipótesis se fundamenta en la riqueza de los grafos, una estructura de datos muy genérica que es capaz de reflejar relaciones complejas entre elementos.

Hipótesis 2: *Las relaciones entre los atributos de un conjunto de datos pueden modelarse mediante un grafo; utilizaremos las correlaciones entre los atributos y entre estos y las clases como pesos de las aristas correspondientes.*

Este grafo nos dará información sobre atributos relevantes y redundantes.

Hipótesis 3: *A partir del grafo anterior desarrollaremos un algoritmo de selección de atributos.*

El corte mínimo en un flujo establece una manera de cortar en dos los vértices de un grafo y ha sido aplicado con éxito en problemas de optimización.

Hipótesis 4: *Las instancias de un conjunto de datos se pueden modelar como un grafo haciendo que cada una de ellas sea un nodo del grafo. Las aristas serán las relaciones de vecindad entre ellas. Este modelo permitirá establecer la importancia de los nodos dentro del grafo, o lo que es lo mismo, de las instancias en el conjunto de datos.*

Para ello utilizaremos un algoritmo similar al de PageRank.

Hipótesis 5: *Los nodos que sean más relevantes según esta medida serán más representativos del conjunto de nodos de su clase.*

Basándonos en esta hipótesis desarrollaremos un algoritmo de selección de instancias que seleccione prioritariamente nodos relevantes.

1.3. Contribuciones

Las contribuciones del presente trabajo son, muy sintéticamente, las siguientes:

- Realizamos una visión general de la etapa del Preprocesamiento de Datos dentro de la Minería de Datos.
- Vemos algunas definiciones y conceptos sobre grafos haciendo especial hincapié en el estudio de las medidas de centralidad y de ranking de nodos.
- Definimos el Problema de la Selección de Atributos haciendo un estudio del estado del arte.
- Presentamos nuestra primera contribución: el algoritmo de selección de atributos *MCBF*, cuya principal novedad es que se sustenta sobre la representación de los atributos y las relaciones entre ellos como un grafo. Planteamos un marco experimental y realizamos un estudio estadístico de los resultados obtenidos para mostrar la bondad de estos.
- Definimos el Problema de la Selección de Instancias haciendo un estudio del estado del arte.
- Presentamos nuestra segunda contribución: *InstanceRank*, que es un algoritmo que permite establecer un ranking entre las instancias de un conjunto de datos.
- Presentamos nuestra tercera contribución, el algoritmo de selección de instancias *ISR*, que utiliza el rango anterior. Establecemos un marco experimental, y realizamos los correspondientes análisis estadísticos para mostrar la bondad de los resultados.

1.4. Estructura del documento

El resto de esta memoria de tesis está organizada en cinco partes:

- **Parte II: Fundamentos.**
  - **Capítulo 2: Minería de Datos: Preprocesamiento de Datos y Clasificación.** En este capítulo estudiamos las etapas en las que se divide la Minería de Datos, deteniéndonos en la del Preprocesamiento de Datos, en la que se encuadran nuestras aportaciones. Presentamos la Clasificación, centrándonos en la técnica de los vecinos más cercanos. Vemos algunos criterios de evaluación de la calidad de los clasificadores, y, por último estudiamos las medidas de distancia entre instancias de los conjuntos de datos.
  - **Capítulo 3: Grafos: Algoritmos, Minería de Datos, Métricas y Ranking.** Comenzamos con un repaso de las definiciones sobre los grafos, sus implementaciones y algunos algoritmos. Vemos algunos algoritmos de Minería de Datos aplicados a grafos, especialmente a los de más moderna aplicabilidad, como los de bases de datos de compuestos químicos, redes sociales, etcétera. Terminamos viendo algunas medidas de centralidad y de ranking en grafos.
- **Parte III: Selección de Atributos.**
  - **Capítulo 4: Introducción a la Selección de Atributos.** Vemos la definición del problema de la selección de atributos, presentando los conceptos de relevancia y redundancia. A continuación vemos una descripción general de estos algoritmos, junto con sus características y las medidas de evaluación de los subconjuntos de atributos. Vemos una taxonomía de estos algoritmos y terminamos mostrando algunos especialmente relevantes.
  - **Capítulo 5: Nuestra aportación: la técnica MCBF.** En este capítulo presentamos nuestra primera propuesta. Comenzamos estudiando los flujos en grafos, estudiando el problema del Flujo Máximo - Corte Mínimo, en el que se basa nuestra propuesta, viendo algunas de las soluciones al problema. A continuación estudiamos las correlaciones entre atributos y entre estos y las clases (en los que también se basa nuestra propuesta). Presentamos el algoritmo, el marco de experimentación, los criterios de

comparación y los resultados. Terminamos viendo un análisis estadístico de los resultados obtenidos.

- **Parte IV: Selección de Instancias.**
  - **Capítulo 6: Introducción a la Selección de Instancias.** Comenzamos este capítulo definiendo el problema de la Selección de Instancias en un conjunto de datos. A continuación vemos una taxonomía que nos permite además estudiar los distintos enfoques que se han dado a este problema. Analizamos las estrategias de evaluación de estos algoritmos y vemos algunas de las técnicas más relevantes, bien por razones históricas, bien por su calidad.
  - **Capítulo 7: Nuestras aportaciones: InstanceRank e ISR.** En este capítulo vemos nuestras otras dos aportaciones. En primer lugar vemos nuestro algoritmo de ranking sobre elementos dentro de un conjunto de datos, InstanceRank, justificándolo algebraicamente y demostrando la convergencia del algoritmo. A continuación presentamos nuestra tercera aportación, el algoritmo de selección de instancias ISR. Presentamos el marco experimental y los algoritmos con los que vamos a establecer la comparación. Mostramos los resultados obtenidos y realizamos un análisis estadístico de estos. Finalmente estudiamos las características de las instancias seleccionadas por el algoritmo.
- **Parte V: Consideraciones finales.**
  - **Capítulo 8: Conclusiones y Oportunidades.** Aquí presentamos las conclusiones extraídas de los trabajos realizados y los resultados obtenidos por nuestras aportaciones, así como algunas líneas de trabajo que podrían plantearse a partir de nuestras propuestas.
- **Parte VI: Apéndice.**
  - **Apéndice A: PolarityRank: Justificación algebraica y Convergencia.** En este apéndice presentamos la justificación algebraica y la justificación de la convergencia de PolarityRank.

Parte II

Fundamentos

Capítulo 2

Minería de Datos: Preprocesamiento de Datos y Clasificación

2.1. Introducción

En este capítulo vamos a contextualizar los problemas que vamos a abordar y resolver en este trabajo. Para ello, vamos a presentar algunos conceptos relacionados con la minería de datos, centrándonos en el aprendizaje supervisado (dentro del que se encuadran nuestras aportaciones). Abordaremos el problema de la clasificación, presentando algunas medidas de calidad de los clasificadores y, finalmente, mostraremos las métricas más comúnmente utilizadas.

2.2. Introducción a la Minería de Datos

Se dice popularmente que vivimos en la "Era de la Información". Esto es cierto, pero también es cierto que estamos inmersos en la "Era de los Datos" (Han y otros, 2012). Cada día se generan ingentes cantidades de datos en el mundo empresarial, en las noticias, en la medicina, en la ciencia en general, etcétera. La aparición de Internet ha producido un aumento

exponencial de la cantidad de datos y, sobre todo, su accesibilidad. Por ejemplo, Google responde diariamente cientos de millones de búsquedas. Esto tiene unas enormes exigencias tecnológicas, pero Google también utiliza las consultas que se realizan a su buscador para afinar en los intereses de los usuarios. Por ejemplo, el número de búsquedas que se realizan en Google sobre la gripe y los lugares desde las que se realizan permiten predecir la evolución de la epidemia anual de gripe dos semanas antes que los métodos tradicionales (Ginsberg y otros, 2009).

Witten y Frank (2005) definen la Minería de Datos como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos. Dicho en otras palabras, la Minería de Datos consiste en encontrar patrones a partir de datos en bruto de una manera automática o semiautomática. Estos patrones son *estructurales* porque capturan la estructura de decisión de una manera explícita: nos ayudan a explicar cosas sobre los datos.

Por ejemplo, si tenemos en una base de datos información sobre las elecciones de compra de clientes de una empresa, junto con los perfiles de esos clientes, aplicando Minería de Datos sobre ella podremos encontrar patrones de comportamiento que nos permitan identificar características de aquellos que son fieles a la marca y de los que no lo son. Una vez que se encuentran esas características se pueden identificar qué clientes puede que estén pensando en cambiar de proveedor y aplicar sobre ellos un tratamiento especial para fidelizarlos, tratamiento que sería demasiado costoso emplear sobre todos los clientes. También podrían identificarse clientes a los que le interesarán nuevos servicios que la empresa puede ofertar y que no gozan actualmente de su interés, para aplicarles una campaña especial de promoción de estos servicios.

Analicemos las posibilidades de la Minería de Datos con un ejemplo manejable, contact-lenses, que es uno de los conjuntos de datos clásico del repositorio de datos de la Universidad de California en Irvine (UCI) (Frank y Asuncion, 2010). Describe el tipo de lentes de contacto prescrita a 24 pacientes de los que se conocen cuatro atributos: el grupo de edad, si son miopes o hipermétropes, si tienen astigmatismo y su producción natural de lágrima (hemos escogido este conjunto de datos por su reducido tamaño). A partir de esta descripción se indica si es adecuado o no el uso de lentes de contacto y, en caso afirmativo, de qué tipo; esta es la *clase* del conjunto de datos. Podemos ver el contenido de este conjunto de datos en la Tabla 2.1.

Esta tabla tiene 24 filas (instancias) porque el primer atributo, la edad, puede tener tres valores, Joven, Pre-présbita, y Présbita (se considera que la presbicia aparece en torno a los 45 años). El resto de los atributos toman

Tabla 2.1 – Conjunto de datos de UCI contact-lens. Describe las características de una serie de personas y si se les prescribió o no lentes de contacto y, en caso afirmativo, de qué tipo.

Age	Spectacle Prescription	Astigmatism	Tear Production Rate	Recommended Lenses
Young	Myope	None	Reduced	None
Young	Myope	None	Normal	Soft
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	None	Reduced	None
Young	Hypermetrope	None	Normal	Soft
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	None	Reduced	None
Pre-presbyopic	Myope	None	Normal	Soft
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	None	Reduced	None
Pre-presbyopic	Hypermetrope	None	Normal	Soft
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	None	Reduced	None
Presbyopic	Myope	None	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	None	Reduced	None
Presbyopic	Hypermetrope	None	Normal	Soft
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

valores binarios, por lo que el número total de combinaciones, cada una de ellas correspondiéndose con una instancia, es $3 \times 2 \times 2 \times 2 = 24$. Esto es un caso poco realista: en la mayoría de las situaciones el conjunto de datos no está completo, tiene ruido (valores mal medidos o mal etiquetados), pueden faltar las medidas de algún o algunos atributos e incluso de la clase, etcétera.

Parte de la información estructural (como se mencionó previamente) que se puede extraer de este ejemplo puede expresarse (según la salida suministrada por Weka 3.7) en la forma que se puede observar en la Figura 2.1. Este mismo árbol de decisión puede verse expresado gráficamente en la Figura 2.2.

Esta descripción es la obtenida en Weka mediante el clasificador J48,

```
tear-prod-rate = reduced: none (12.0)
tear-prod-rate = normal
|   astigmatism = no: soft (6.0/1.0)
|   astigmatism = yes
|   |   spectacle-prescrip = myope: hard (3.0)
|   |   spectacle-prescrip = hypermetrope: none (3.0/1.0)
```

Figura 2.1 – Salida del algoritmo J48 aplicado al conjunto de datos contact-lenses.

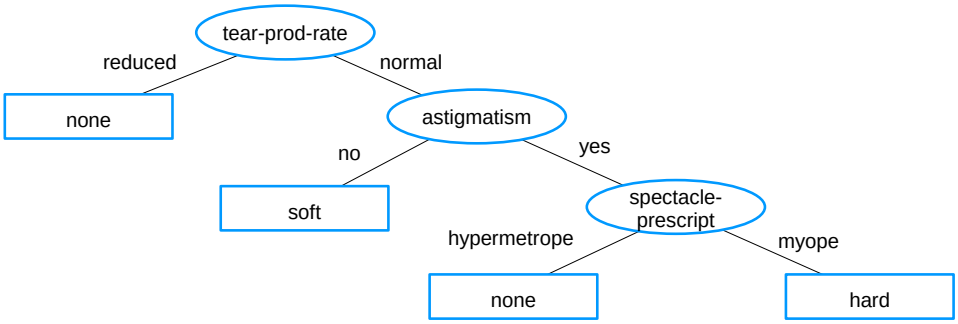

Figura 2.2 – Árbol de decisión obtenido con J48 sobre el conjunto de datos contact-lens con los parámetros por omisión, expresado gráficamente.

que es un árbol de decisión, equivalente a C4.5 (Quinlan, 1990, 1993). Las cifras que aparecen al final de las líneas (junto a las *hojas del árbol*) en la Figura 2.1 representa el número de casos que cumplen la condición y, en caso de aparecer una barra, el número de ellos que han sido incorrectamente clasificados por la regla. Como vemos, hay dos casos que están mal clasificados por este árbol. Se podría plantear generar un árbol con más ramas que clasificase correctamente todos los casos; sin embargo esto no siempre es beneficioso, por el fenómeno conocido como sobreajuste (en inglés *overfitting*). En el mundo real los fenómenos tienden a cumplir el principio de la parsimonia; el sobreajuste provoca que el clasificador intente amoldarse (modelar) al posible ruido, haciéndonos creer que este forma parte del modelo. Por otra parte, los modelos obtenidos serán más confusos y, por tanto, de más difícil explicación para los expertos.

Las descripciones estructurales no tienen que tener forzosamente la forma de reglas, como las que se han presentado anteriormente.

El proceso de Minería de Datos se encuadra dentro de un esquema más general denominado *Descubrimiento de Conocimiento en Bases de Datos*

(en inglés *Knowledge Discovery in Databases* o, abreviadamente, *KDD*). El esquema general del *KDD* se representa como se puede ver en la Figura 2.3, adaptada de (Pal y Pal, 2001).

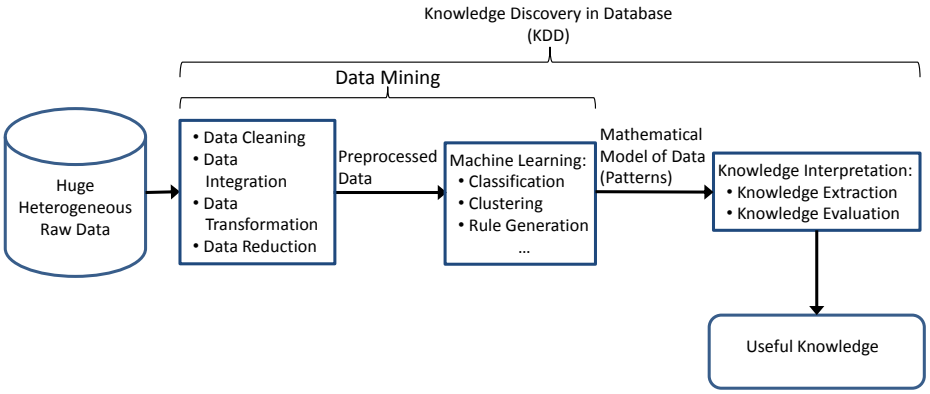

Figura 2.3 – Esquema general del proceso de adquisición de conocimientos en conjuntos de datos (*KDD*).

Los problemas abordados en el presente trabajo se encuadran, dentro del esquema general del *KDD*, en la etapa de Preprocesamiento de Datos.

2.2.1. Definiciones

Antes de seguir adelante vamos a exponer algunas definiciones que serán de utilidad en el resto de esta memoria, tomadas de (Ruiz, 2006).

Definición 2.1 (Dominio)
Un Dominio *es un conjunto de valores del mismo tipo. Distinguiremos entre dos tipos: continuo (tiene un conjunto infinito de valores reales) y nominal (tiene un conjunto finito de valores discretos).* □

Definición 2.2 (Universo de discurso)
El Universo de discurso *es el entorno donde se define un determinado problema. Viene representado por el producto cartesiano de un conjunto finito de dominios.* □

Definición 2.3 (Atributo)
Un atributo *es la descripción de alguna medida existente en el universo de discurso que toma valores en un determinado dominio. El atributo i-ésimo*

se representa F_i, su valor f_i y su dominio $Dom(F_i)$. Según la clasificación de la Definición 2.1, puede ser de dos tipos, continuo o discreto. Si es continuo existe un rango de valores $[a, b]$ con $a, b \in \mathbb{R}$, de valores posibles; si es discreto, existe un conjunto finito de valores posibles. Se denomina vector atributos $\overline{f} = \{f_1, \ldots, f_m\}$ *al conjunto de valores correspondientes a cada uno de los atributos, y $\mathcal{F}$ al espacio formado por el conjunto de atributos, $\mathcal{F} = Dom(F_1) \times \ldots Dom(F_m)$, siendo m el número de atributos.* □

En el conjunto de datos de ejemplo, contact-lens, los atributos son todos nominales.

Definición 2.4 (Clase)
En el caso particular de un tipo de aprendizaje que denominados supervisado, se dispone de un atributo especial de salida denominado clase, *que indica la pertenencia a un determinado grupo de casos. Se denomina* etiquetas *de clase al conjunto o dominio de valores que la clase puede tomar. La clase es el atributo sobre el que se realiza la predicción, por lo que es también denominada* atributo de decisión, *en contraposición al resto de atributos, denominados* de condición. *El atributo de clase se representa Y y su dominio $dom(Y)$, teniendo k posibles valores $y_1, \ldots, y_k$. Donde resulte más conveniente se denotará la clase de la instancia x como x_c.* □

Si el conjunto de datos no dispone de clase, se hablará de *aprendizaje no supervisado*. El ejemplo citado, contact-lens, se trataría de aprendizaje supervisado, puesto que sí hay una clase,"Recommended Lenses".

Definición 2.5 (Instancia)
Una instancia, ejemplo, muestra o caso *es una tupla del universo del discruso representada por un conjunto de valores de atributos y una etiqueta de clase que lo clasifica. Se representa x. Si estamos hablando de un entorno de aprendizaje no supervisado la instancia carece de clase.* □

Definición 2.6 (Conjunto de Datos)
Se define un conjunto de datos *(en inglés dataset) como un subconjunto finito de instancias x_i, donde $i = 1, \ldots, n$. Un conjunto de datos se caracteriza por el número de instancias n que tiene y por el número m de atributos y su tipo.* □

2.2.2. La Etapa de Preprocesamiento de Datos

Datos de entrada de baja calidad conducen a resultados de baja calidad (*Garbage in, garbage out*). Por otra parte, en aplicaciones del mundo real los datos provienen de numerosas fuentes, lo que motiva que estos puedan

estar mal tomados, ser incompletos, etc. Por ello es imprescindible llevar a cabo un preprocesamiento de los datos.

Unos datos de calidad deben cumplir, como mínimo, tres requisitos: deben ser *precisos*, *completos* y *consistentes*.

- Los datos son **precisos** si no contienen instancias ruidosas, esto es, instancias que contienen atributos mal medidos o mal transcritos o se corresponden con valores atípicos (en inglés, *outliers*): valores que se desvían de la norma. Pueden aparecer datos imprecisos por muy distintas razones: los instrumentos utilizados para la medida pueden ser imprecisos; pueden haber ocurrido errores, bien humanos, bien informáticos, en la entrada de datos; los usuarios pueden haber dado datos erróneos (como la fecha de nacimiento o el código postal) para preservar la privacidad o por otras razones; puede haber inconsistencias en el formato de los datos, etcétera.

- Los datos son **completos** si están presentes todos los atributos necesarios y no hay en ellos o en las clases valores perdidos: atributos sin ningún valor asignado. Los datos pueden ser incompletos por diversas razones: algún atributo puede no haber estado disponible en el momento de la recogida; puede no haberse considerado de interés en el momento en que se introdujeron; puede haber habido una mala comprensión de las instrucciones de recogida; pueden haber ocurrido errores de funcionamiento en algún dispositivo, etcétera.

- Los datos son **consistentes** si no contienen valores incompatibles con otros datos. Si hay datos inconsistentes hay que eliminarlos, determinando cuál es el válido.

Existen diversas descripciones de las tareas de las que consta la etapa de preprocesamiento de datos, pero fundamentalmente estas son:

Limpieza de datos (*Data cleaning*) Esta tarea se ocupa de la limpieza de los datos: se rellenan los valores perdidos con valores "razonables", como la media, la moda, la mediana, etcétera; se suavizan los elementos ruidosos, se identifican y eliminan los outliers y se resuelven las inconsistencias.

Integración de datos (*Data integration*) Esta tarea consiste en recopilar las distintas fuentes de las que pueden provenir los datos, dándoles un formato homogéneo y unos identificadores únicos y eliminando los datos redundantes.

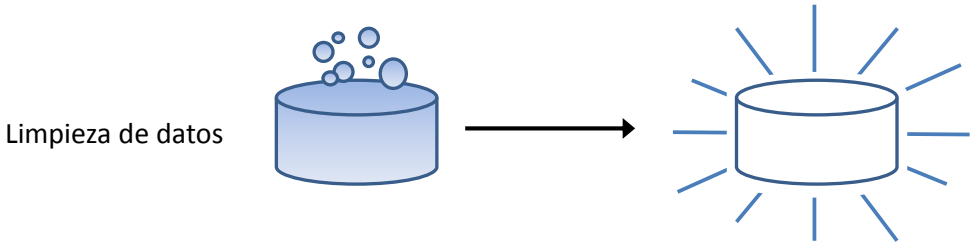

Figura 2.4 – Preprocesamiento de datos: limpieza de datos.

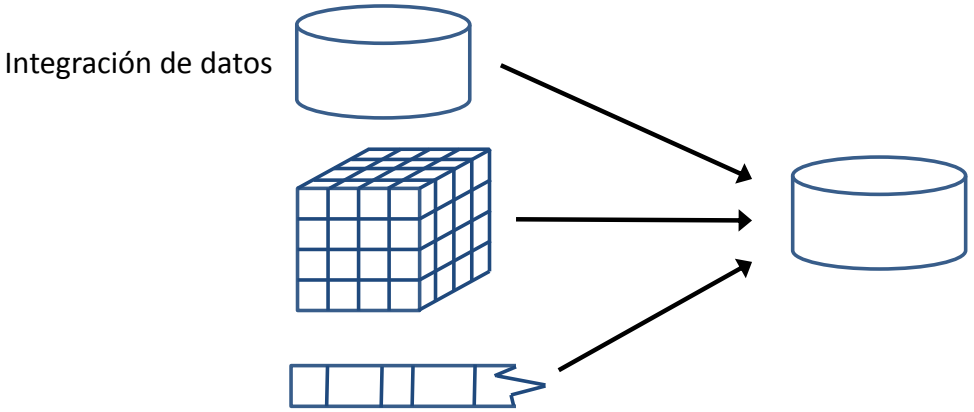

Figura 2.5 – Preprocesamiento de datos: integración de datos.

Tranformación de datos (*Data tranformation*) Esta tarea se encarga de la normalización de los datos, la discretización, la generación de jerarquías de conceptos, etcétera.

Reducción de datos (*Data reduction*) En esta tarea se reduce el tamaño del conjunto de datos, intentando que esto se haga sin pérdida en la calidad de los resultados. Esto se consigue con distintas estrategias: agregación de datos (generando cubos de datos), selección de atributos (eliminando los atributos que no aportan información), selección de instancias (dejando solo un subconjunto de instancias que reflejen el comportamiento del conjunto completo), usar alguna forma de codificación alternativa que reduzca el tamaño de los datos, reemplazar los datos originales por otras representaciones, como clústeres o modelos paramétricos.

Estas tareas quedan reflejadas en las Figuras 2.4, 2.5, 2.6 y 2.7.

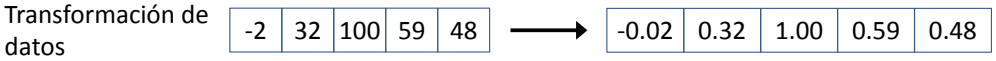

Figura 2.6 – Preprocesamiento de datos: transformación de datos. En este ejemplo, los datos son normalizados en el intervalo $[-1, 1]$.

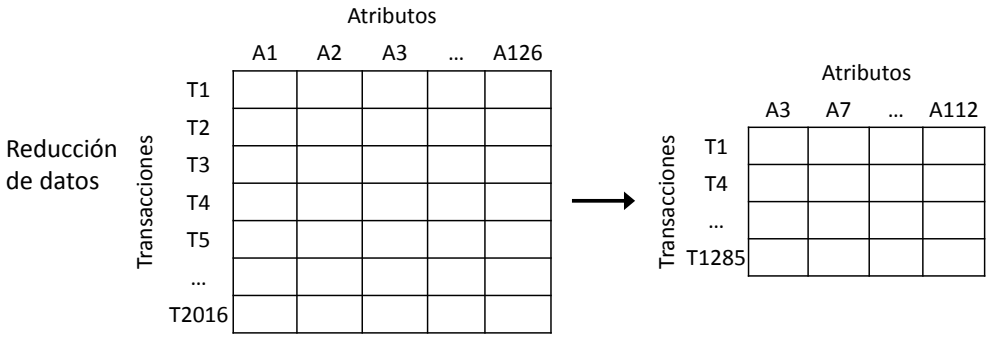

Figura 2.7 – Preprocesamiento de datos: reducción de datos. En esta figura reflejamos las dos técnicas que se desarrollan en el presente trabajo: la selección (y por lo tanto reducción) de atributos y de instancias.

Nuestro trabajo se encuadra dentro del último elemento de esta etapa, concretamente en la selección de atributos y de instancias.

2.3. Clasificación

Dentro de la Minería de Datos, la fase posterior a la de preprocesamiento es la de *Aprendizaje Automático* (en inglés, *Machine Leaning*). Estas son las técnicas que nos permiten aprender a inferir patrones, predecir comportamientos, ayudarnos a entender el modelo subyacente al fenómeno en estudio, etcétera, a partir de los datos.

El aprendizaje automático tiene una fuerte analogía con el aprendizaje humano, en el que se utilizan las experiencias pasadas para obtener nuevos conocimientos que nos permitan mejorar nuestra habilidad para realizar tareas futuras (por ejemplo, sobrevivir) (Liu, 2007).

Existen muchas tareas dentro del Aprendizaje Automático: Generación de Reglas, Clustering, etc. Pero nos vamos a centrar en la que es de interés para este trabajo, que es la *Clasificación*.

La Clasificación, también llamada *Aprendizaje inductivo*, consiste en, dado un conjunto de datos de entrenamiento T, cada uno con una clase asig-

Figura 2.8 – Esquema del proceso de generación/evaluación de un clasificador.

nada, tratar de inferir las clases de otro conjunto de datos no visto previamente, denominado conjunto de test. Por tanto, el objetivo es producir una función de clasificación o predicción, que relacione los valores de los atributos (que es lo que conocemos de los datos), con las clases. Esa función se denomina *modelos de clasificación*, *modelo predictivo* o, más sencillamente *clasificador*. En la figura 2.8 podemos ver un esquema del proceso de generación y evaluación de un clasificador.

Como se ha mencionado previamente, se deben utilizar dos conjuntos de datos: uno, denominado *conjunto de entrenamiento* con el que entrenamos el clasificador, y otro, denominado *conjunto de test* con el que comprobamos la calidad del clasificador obtenido. En la subsección 2.3.3 veremos cómo elegir el conjunto de entrenamiento y el de test, y en la subsección 2.3.4 veremos diversos modos de evaluar la calidad del clasificador obtenido.

2.3.1. Técnicas de Clasificación

Existen muchísimas técnicas de clasificación. Entre ellas podemos destacar las máquinas de vectores de soporte o SVM (Vapnik, 1995), los árboles de decisión (Hunt y otros, 1966; Quinlan, 1990, 1993) del que hemos visto un ejemplo en contact-lens, y Naïve Bayes (Bayes, 1763) (publicado dos años después de su fallecimiento), que ha mostrado, pese a su sencillez, una sorprendente precisión (Domingos y Pazzani, 1997; Kohavi y otros, 1997) y, por último la técnica de los vecinos más cercanos, que es la que empleamos en nuestro trabajo, por lo que la estudiaremos con un poco más de profundidad.

2.3.2. La técnica de los vecinos más cercanos

La técnica de los vecinos más cercanos (*Nearest Neighbor* o, en abreviatura, *NN*) se basa en la idea de que un nuevo elemento pertenecerá a de aquel o aquellos que sean más parecidos a él. Tiene dos variantes: el ve-

cino más cercano o los k vecinos más cercanos (kNN). En el primer caso al nuevo elemento se le asigna la clase del vecino más cercano. En el segundo, se encuentran los k vecinos más cercanos y se le asigna al nuevo elemento la clase mayoritaria entre estos; k suele tomarse impar (para evitar empates en conjuntos de datos con dos clases) y con valores pequeños, generalmente 3, 5 o 7. El vecino más cercano es un caso particular de los k vecinos más cercanos, con $k = 1$. En adelante hablaremos de los k vecinos más cercanos.

Obsérvese que, como hemos dicho, cuando se pretende clasificar un elemento nuevo, se buscan los k más cercanos y se le asigna la clase mayoritaria. Esto quiere decir que el nuevo elemento debe compararse con todos los n elementos de conjunto de datos de entrenamiento. Por tanto, clasificar este elemento tiene un coste $O(n)$. Obsérvese también que, a diferencia de los árboles de decisión o las SVM, no se genera ningún modelo; por esta razón, se dice que la técnica de los k vecinos más cercano es un método *lazy* (perezoso), en contraposición a los otros, que son *eager*. No hay ningún coste de generación del modelo, pero todo el coste se traslada al momento en que se clasifica el nuevo elemento, denominado en ocasiones *fase de generalización*.

Esta técnica data de los años cincuenta del siglo pasado (Fix y Hodges, 1951, 1952) y fue desarrollada en su formulación actual en los sesenta (Cover y Hart, 1967; Cover, 1968). Este método pese a la sencillez de su formulación y de su programación obtiene una gran precisión en la clasificación.

En este método es de especial trascendencia la métrica (medida de similitud entre instancias) utilizada para determinar cuál es el vecino más cercano. Distintas métricas pueden dar lugar a resultados distintos. Estas métricas se estudiarán con más detalle en la sección 2.4.

2.3.3. Métodos de Evaluación

Una vez construido un clasificador, antes de aplicarlo en problemas del mundo real debemos medir su calidad. La principal medida de calidad es la *precisión* o *exactitud* (en inglés *accuracy*). Esta se obtiene clasificando los elementos del conjunto de entrenamiento y se define como la razón entre el número de instancias del conjunto de entrenamiento correctamente clasificadas y el número total de elementos del conjunto de entrenamiento, T.

$$precisión = \frac{\text{número de instancias correctamente clasificadas de } T}{|T|}$$

Otra medida, menos utilizada y absolutamente ligada a la anterior, es la *tasa de error*, que es $1 - precisión$. Veremos más medidas en la subsección 2.3.4.

En cualquier caso, ¿cómo escoger el conjunto de entrenamiento y el de test a partir de los datos disponibles? El propio conjunto de entrenamiento no debe ser empleado para la evaluación porque el clasificador estará sobreajustado respecto al conjunto de entrenamiento y, por tanto, la precisión resultante estará sesgada respecto a este conjunto, con lo que los resultados no resultarían fiables. Por tanto debemos tener algún método para escoger el conjunto de entrenamiento y el de test. Los métodos más comunes son tres:

Holdout Set Se utiliza cuando se dispone de un conjunto grande de datos. Llamemos D al conjunto completo de datos. Este se divide en dos subconjuntos, que llamaremos D_{train} y D_{test}, de modo que $D_{train} \cup D_{test} = D$ y $D_{train} \cap D_{test} = \varnothing$. Normalmente se toman $50\,\% - 50\,\%$ o dos tercios para el entrenamiento y un tercio para el test. En cuanto a cómo repartir los elementos se pueden hacer tres cosas:

- Escoger al azar los elementos.
- Escoger los elementos de los dos conjuntos de manera que haya una proporción similar de elementos de cada clase. Esto se denomina *estratificación*.
- Escoger los elementos más antiguos para el conjunto de entrenamiento y los más nuevos para la clasificación (esto exige que conozcamos el orden en el que se recogieron los elementos). Este modo de escoger los conjunto refleja muy bien la situación del clasificador en el mundo real, puesto que si se utiliza en él se entrenará con los datos existentes y se utilizará con los que vayan llegando en el futuro.

Multiple Random Sampling Si el conjunto de datos disponibles es muy pequeño, el método anterior no sería fiable debido al tamaño del conjunto de test. En este caso se pueden tomar al azar los dos conjuntos y repetir el proceso un determinado número de veces. La precisión final será la media de las precisiones obtenidas.

Validación cruzada (*Cross-Validation*) Es la más utilizada, especialmente si el tamaño de los datos es pequeño. En la validación cruzada de n

pliegues (n-fold cross-validation) el conjunto de datos se parte en n subconjuntos disjuntos de igual tamaño. A continuación cada subconjunto se utiliza como test y los $n-1$ restantes se usan para entrenamiento. Esto se realiza n veces, con cada uno de los n subconjuntos, dando como resultado n precisiones. La precisión final se toma como la media de las n precisiones. Los valores típicos de n son 5 o 10, siendo este el más comúnmente utilizado en la literatura.

La validación cruzada puede combinarse con la estratificación: se toman los n subconjuntos de modo que la proporción de elementos de cada clase en cada uno de los n subconjuntos sea similar.

Un caso especial de validación cruzada es la conocida en inglés como *leave-one-out cross-validation*. En este método cada pliegue de la validación cruzada tiene solo un elemento y el resto se usa como entrenamiento. Se utiliza solo si el conjunto de datos es extraordinariamente pequeño, puesto que obliga a construir tantos clasificadores como elementos tenga el conjunto.

2.3.4. Calidad de la clasificación

La precisión (*accuracy*), como hemos dicho, es la medida más utilizada en clasificación. Sin embargo, hay ocasiones en las que esta medida no es adecuada: cuando la clase que nos interesa especialmente es muy pequeña. Esto se da, por ejemplo, en la detección de intrusos en una red o en en la detección de fraudes financieros. Si utilizamos la precisión puede que obtengamos unos valores altísimos, por encima del $99\,\%$, pero que estos no nos sirvan para nada, porque serán precisamente los de las clases que no nos interesan (los casos "normales": no hay intruso o no hay fraude). Vamos a llamar a la clase que nos interesa clase *positiva* (intruso o fraude), y vamos a llamar *clase negativa* a la que no nos interesa (las clases negativas se pueden combinar en una sola).

En estos casos se usan dos medidas, *Precision* [1] y *Recall* (cobertura). Para llegar a ellas debemos introducir previamente el concepto, muy útil también, de matriz de confusión. La matriz de confusión (Tabla 2.2) refleja la información sobre los resultados reales y los dados por el clasificador. En ella TP (verdaderos positivos) es el número de clasificaciones correctas de instancias positivas. FN (falsos negativos) es el número de clasificaciones

[1] Puede haber una cierta confusión con la traducción al castellano de *precision* y *accuracy*, porque la traducción al castellano de ambas es precisión. Como en nuestras aportaciones utilizamos como medida la *accuracy*, hemos optado por traducir esta como precisión en todo el trabajo, haciendo esta observación solo en este punto y utilizando aquí el término en inglés (sin tilde).

Tabla 2.2 – Matriz de confusión de un clasificador. TP son los verdaderos positivos, FN los falsos negativos, FP los falsos positivos y TN los verdaderos negativos.

	Clasificado positivo	Clasificado negativo
Realmente positivo	TP	FN
Realmente negativo	FP	TN

incorrectas de instancias positivas. FP (falsos positivos) es el número de clasificaciones incorrectas de instancias negativas. TN (verdaderos negativos) es el número de clasificaciones correctas de instancias negativas.

A partir de la matriz de confusión, se definen *precision* y *recall* como

$$p = \frac{TP}{TP + FP}$$

$$r = \frac{TP}{TP + FN}$$

Es decir, p es el número de instancias positivas clasificadas correctamente dividido entre el número total de instancias que son clasificadas como positivas, lo sean o no. La cobertura r es el número de instancias positivas clasificadas correctamente dividido entre el número total de instancias realmente positivas.

Una medida como esta que tiene dos componentes es difícil de manejar. Por ello se utiliza otra derivada de las anteriores, denominada **F-score** y que se define como

$$F = \frac{2pr}{p + r}$$

2.4. Medidas de distancia entre instancias

2.4.1. Atributos continuos

Como hemos dicho anteriormente, es de crucial importancia la medida de distancia entre instancias, o métrica, que se utilice. Vamos a estudiar las más comunes. En lo que sigue vamos a denotar como $\boldsymbol{x}$ y $\boldsymbol{y}$ los vectores que representan los atributos de dos instancias x e y (se excluyen las clases). Llamaremos x_i al componente i-ésimo del vector $\boldsymbol{x}$ (análogamente para $\boldsymbol{y}$), con $i = 1, \ldots, m$, siendo m el número de atributos del conjunto de datos.

Una métrica muy utilizada es la de Minkowski, que se define como

$$d = \sqrt[p]{\sum_{j=1}^{m} d_j^p} \, .$$

donde d_j se define para los atributos continuos como

$$d_j = |x_j - y_j| \, ,$$

Algunos valores particulares de p, la distancia de Minkowski se corresponde con una métrica especial, como queda reflejado en la tabla 2.3.

Tabla 2.3 – Métricas obtenidas para distintos valores de p de la métrica de Minkowski.

Distancia de Manhattan	$p = 1$	$d = \sum_{j=1}^{m} d_j$
Distancia euclidea	$p = 2$	$d = \sqrt{\sum_{j=1}^{m} d_j^2}$
Distancia de Chebychev	$p = \infty$	$d = \max_{j=1}^{m} \lvert d_j \rvert$

En muchas ocasiones es conveniente normalizar los valores de los atributos continuos, para evitar que aquellos que tengan valores mayores influyan más que los que tengan valores pequeños. Esto se puede conseguir de distintas maneras; en general se hace dividiendo entre algún valor que dependa del rango del atributo, como puede ser la media, la mediana, la desviación típica, etc.

Existen muchas otras métricas, como la de Mahalanobis, Camberra, del coseno, etc., pero vamos a centrar el resto del estudio en HEOM y HVDM, que serán de utilidad en la parte de nuestra aportación relativa a la selección de instancias.

2.4.2. Atributos nominales y continuos

HEOM

Para el caso de los atributos discretos, una solución muy sencilla, utilizada, por ejemplo, en (Aha y otros, 1991; Aha, 1992), es utilizar una métrica denominada HEOM (*Heterogeneous Euclidean-Overlap Metric*) que permite mezclar atributos continuos y discretos. Para los atributos discretos se usa la *distancia de solapamiento* (*overlap*): si los valores de los atributos coinciden se les da una distancia 1; en caso contrario, 0. También se incluyen los valores perdidos, para los que se toma distancia 1.

$$overlap(x_j, y_j) = \begin{cases} 0 & x_j = y_j \\ 1 & \text{en otro caso} \end{cases}$$

Ahora se pueden mezclar los valores discretos, los continuos y la normalización, de modo que para el atributo f_j, sea nominal o continuo, es

$$d_j = \begin{cases} 1 & \text{si } x \text{ o } y \text{ son perdidos; en otro caso} \\ overlap(x_j, y_j) & \text{si } f_j \text{ es nominal; en otro caso} \\ rn\_dif_{f_j}(x_j, y_j) & \end{cases}$$

donde $rn\_dif_{f_j}$ es la diferencia normalizada en rango (*range-normalized difference*), definida como

$$rn\_dif_{f_j}(x_j, y_j) = \frac{|x_j - y_j|}{rango_{f_j}}$$

donde $rango_{f_j}$ se define como

$$rango_{f_j} = max_{f_j} - min_{f_j}$$

La definición anterior devuelve un valor en el rango $[0, 1]$, sea el atributo continuo o discreto. Finalmente, se define la distancia entre dos vectores (atributos de instancias) $\boldsymbol{x}$ e $\boldsymbol{y}$ con atributos posiblemente heterogéneos como

$$HEOM(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{j=1}^{m} d_j^2}$$

VDM

Esta métrica fue introducida por Stanfill y Waltz (1986) para suministrar una medida de distancia mejorada para los atributos nominales. En una versión simplificada (sin ponderación de atributos), se define la distancia entre los valores x, y del atributo f como

$$vdm_f(x,y) = \sum_{c=1}^{C} \left| \frac{N_{f,x,c}}{N_{f,x}} - \frac{N_{f,y,c}}{N_{f,y}} \right|^q = \sum_{c=1}^{C} |P_{f,x,c} - P_{f,y,c}|^q,$$

donde

- C es el número de clases;
- $N_{f,x}$ es el número de instancias en el conjunto de entrenamiento que toman el valor x para el atributo f;
- $N_{f,x,c}$ es el número de instancias en el conjunto de entrenamiento que toman el valor x para el atributo f y tienen clase c;
- q es una constante; normalmente 1 o 2;
- $P_{f,x,c}$ es la probabilidad condicional de que la clase sea c supuesto que el atributo f tiene el valor x, es decir

$$P_{f,x,c} = \frac{N_{f,x,c}}{N_{f,x}} \tag{2.1}$$

Se cumple que

$$N_{f,x} = \sum_{c=1}^{C} N_{f,x,c}$$

HVDM (Heterogeneous Value Difference Metric)

Wilson y Martinez (1997) presentan HVDM, que combina las dos mejoras anteriores.

Dados dos vectores $\boldsymbol{x}$ e $\boldsymbol{y}$ se define $HVDM(\boldsymbol{x}, \boldsymbol{y})$ como

$$HVDM(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{j=1}^{m} d_{f_j}(x_j, y_j)^2}$$

El valor de la función d para el atributo f y los valores x e y se define como

$$d_f(x,y) = \begin{cases} 1, & \text{si } x \text{ o } y \text{ son perdidos; en otro caso} \\ normalized\_vdm_f(x,y), & \text{si } f \text{ es nominal} \\ normalized\_dif_f(x,y), & \text{si } f \text{ es continuo} \end{cases}$$

El problema que presenta la normalización con alguno de los estadísticos habituales es que estos se toman sobre los elementos conocidos, lo que no garantiza en absoluto que en otros valores el rango normalizado se mantenga en el intervalo $[0,1]$. Para solucionar esto se toma 4 veces la desviación estándar con lo que el $95\,\%$ de los valores estarán dentro de dos desviaciones estándar de la media. De aquí que se tome

$$normalized\_dif_f(x,y) = \frac{\|x-y\|}{4\sigma_f},$$

siendo σ_f la desviación estándar del atributo f.

Para el otro elemento los autores probaron tres posibilidades concluyendo que los valores óptimos se daban para

$$normalized\_vdm_f(x,y) = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{f,x,c}}{N_{f,x}} - \frac{N_{f,y,c}}{N_{f,y}} \right|}$$

2.5. Resumen

En este capítulo hemos presentado algunos conceptos básicos del descubrimiento de conocimiento en bases de datos (*KDD*), centrándonos en primer lugar en la etapa de preprocesamiento de datos, que es en la que se encuadra el trabajo de investigación que hemos desarrollado y que presentaremos a lo largo de esta memoria. Hemos dado algunas definiciones que se utilizarán en los capítulos posteriores.

Posteriormente hemos analizado el problema de la clasificación, centrándonos en la de los vecinos más cercanos, que es la técnica con la que trabajaremos en nuestras aportaciones. Hemos visto algunos métodos para elegir los conjuntos de entrenamiento y de datos, y mostrado algunas medidas de calidad alternativas a la precisión.

En la última parte del capítulo hemos estudiado las medidas de distancia entre instancias, que serán de utilidad en la segunda de nuestras aportaciones, la selección de instancias.

Capítulo 3

Grafos: Algoritmos, Minería de Datos, Métricas y Ranking

3.1. Introducción

Nuestras dos aportaciones se basan en la representación de los atributos, en un caso, y las instancias, en el otro, como grafos, y en la aplicación sobre estos grafos de sendos algoritmos. En este capítulo vamos a estudiar los grafos, viendo algunas definiciones que resultarán de utilidad en capítulos posteriores, los algoritmos clásicos más relevantes sobre grafos, algunas medidas de ranking y centralidad en los grafos y la relación entre estos y la Minería de Datos.

3.2. Grafos: definiciones, implementaciones y algoritmos

3.2.1. Introducción histórica

La primera referencia a los grafos de la que se tiene conocimiento data de una comunicación realizada ante la Academia de San Petersburgo en 1735 por Leonard Euler, que fue publicada seis años más tarde en (Euler, 1741). Es la formulación y solución del famoso problema de los puentes

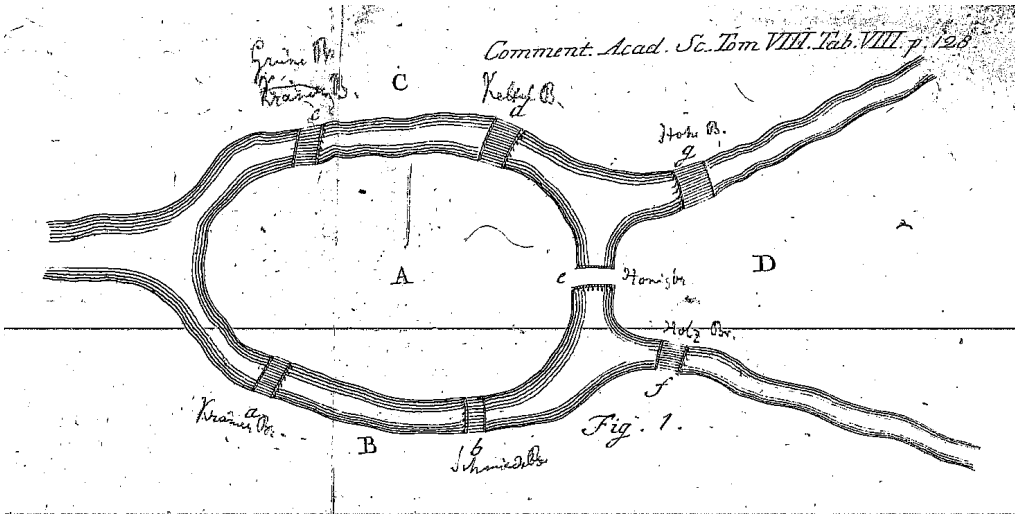

Figura 3.1 – Imagen original del plano de la ciudad de Königsberg, tal como aparece en (Euler, 1741).

de Königsberg (la actual Kaliningrado). En la Figura 3.1 podemos ver la Figura 1 de la publicación original, que es una representación esquemática de la ciudad de Königsberg. Esta ciudad está cruzada por dos brazos del río Pregel, que la divide por la mitad, dejando el lado *B* de la figura a la izquierda y el *C* a la derecha. Las islas son *A* y *D*. En aquella época la ciudad tenía siete puentes: *a* y *b*, que unían la isla *A* con el margen izquierdo *B*, *c* y *d*, que unían la isla *A* con el margen derecho *C*, *e* que unía las dos islas, *f* que unía la isla *D* con el margen izquierdo *B* y el puente *g* que unía la isla *D* con el margen derecho *C*. Se le planteó a Euler el problema de si era posible cruzar todos los puentes pasando por cada uno de ellos solo una vez.

Euler consideró el problema trivial, pero, como se lee en la correspondencia que mantuvo con el matemático italiano Giovanni Marinori, " Este problema es trivial, pero me llamó profundamente la atención que ni la geometría, ni el álgebra, ni siquiera el arte de contar fuera suficiente para resolverlo ". Efectivamente, para resolverlo había inventado la teoría de grafos y la topología.

En la publicación de Euler se puede ver que el matemático abstrae el problema en lo que en una representación moderna sería el grafo de la Figura 3.2, en la que se las islas y las orillas son los vértices y los puentes son aristas entre los vértices.

El razonamiento de Euler fue que si se cruza un puente una vez, este

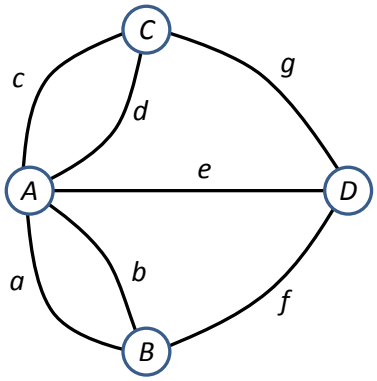

Figura 3.2 – Grafo equivalente al problema de los puentes de Königsberg.

tiene que tener grado par (el número de aristas que incluyen a ese vértice es par). El vértice en el que se comience el paseo puede tener grado impar; lo mismo ocurre con el vértice en el que se termina el paseo. Sin embargo, todos los vértices intermedios tienen que tener grado par. En el grafo que representa Königsberg, los cuatro vértices tienen grado impar, por lo que el problema no tiene solución. En honor a Euler, este tipo de recorrido de un grafo se denomina camino euleriano.

Una vez vista esta introducción histórica, vamos e ver algunas definiciones que nos resultarán de utilidad en secciones posteriores.

3.2.2. Definiciones

Grafos no dirigidos

Un grafo $G = (V, E)$ está formado por un conjunto de *vértices* V, también llamados *nodos*, y un conjunto de *aristas* E que son pares de elementos de V. Si estos pares son no ordenados, el grafo se denomina *no dirigido*; en la Figura 3.3 podemos ver un ejemplo de grafo dirigido. En los grafos no dirigidos, $\forall e \in E$, e está identificado con un par $[u, v]$, con $u, v \in V$, denotando la arista como $e = [u, v]$. Si tenemos una arista $e = [u, v]$ decimos que u y v son *extermos* de e y que u y v son nodos *adyacentes* o *vecinos*.

El *grado* de un vértice en un grafo no dirigido es el número de aristas que lo contienen. Un vértice de grado 0 se dice que es un vértice o nodo *aislado*. En el grafo de la Figura 3.3 el nodo G es un nodo aislado.

Se define el *camino* en un grafo de un nodo u a un nodo v a una sucesión de vértices $v_0, v_1, \ldots, v_n$ donde $\forall i = 1, \ldots, n$ v_{i-1} es adyacente a v_i (existe una arista $[v_{i-1}, v_i]$), $u = v_0$ y $v = v_n$. Se define la *longitud* del camino

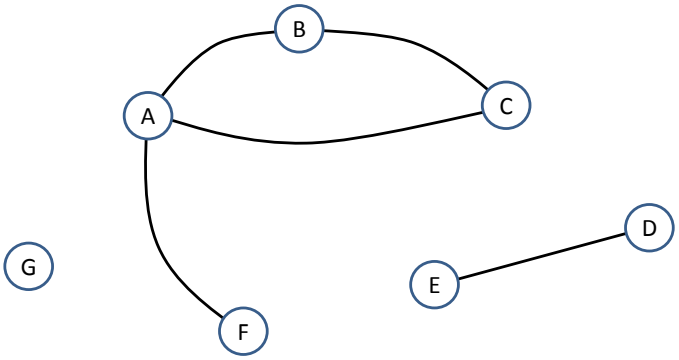

Figura 3.3 – Ejemplo de grafo no dirigido.

anterior como n, es decir, el número de nodos del camino menos 1. Existe camino de todo nodo a sí mismo, de longitud 0.

Un camino se dice que es *cerrado* si $v_0 = v_n$, es decir, empieza y termina en el mismo vértice.

Un camino se dice que *simple* si todos los nodos que lo componen son distintos, a excepción del último, que puede ser igual al primero (en un camino cerrado).

Un ciclo es un camino simple cerrado de longitud 3 o más. En el ejemplo de la Figura 3.3 hay un ciclo compuesto por los vértices A, B y C. Un grafo es *acíclico* si no contiene ningún ciclo.

Un grafo es *conexo* si existe camino entre todo par de vértices del grafo.

Un grafo es *completo* si existe arista entre todo par de vértices distintos del grafo. Se demuestra fácilmente que un grafo dirigido completo de n vértices tiene exactamente $n(n-1)/2$ aristas.

Un subgrafo de un grafo que sea completo se llama *clique*.

Dos aristas e y e' se llaman *aristas múltiples* si tienen los mismos extremos; es decir, si $e = [u, v]$ y $e' = [u, v]$. En este caso las aristas no forman un conjunto de pares, sino un multiconjunto.

Una arista se denomina *bucle* si los dos extremos son el mismo vértices, es decir, si $e = [u, u]$. Un bucle es un camino de longitud 1.

Un grafo es *bipartito* si el conjunto de vértices V se puede descomponer en dos subconjuntos V' y V'' ($V' \cup V'' = V$ y $V' \cap V'' = \varnothing$) y toda arista del grafo tiene un extremo en V' y el otro en V''.

Un grafo es *planar* si puede ser dibujado sin que sus aristas se corten.

Si un grafo admite aristas múltiples o bucles, se denomina un *multigrafo*. La definición clásica de grafo no admite aristas múltiples (las aristas

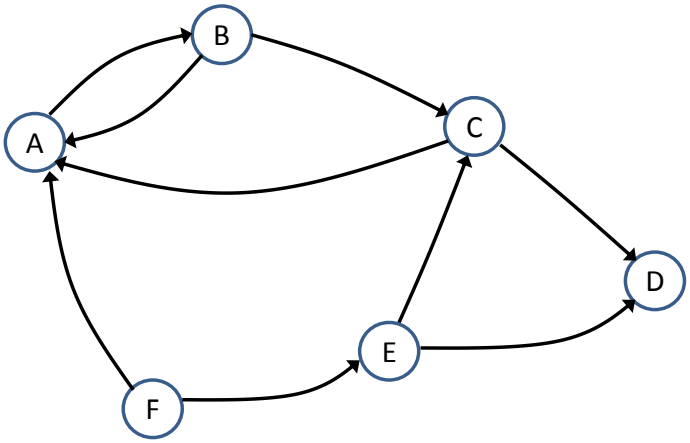

Figura 3.4 – Ejemplo de grafo dirigido.

forman un conjunto) ni bucles. A estos grafos les llamamos grafos *simples*. El grafo que modela el problema de los puentes de Königsberg (Figura 3.2) es un multigrafo, puesto que tiene aristas múltiples.

Los grafos no dirigidos se representan como una nube de puntos, los vértices, junto con unas líneas que los unen, que son las aristas.

Grafos dirigidos

Si los pares son ordenados, hablamos de grafo *dirigido*. Si el grafo es dirigido, tendremos que $\forall e \in E$, e está identificado por un par de vértices $u, v \in V$; la arista e se denota como $e = (u, v)$. En la Figura 3.4 podemos ver un ejemplo de grafo dirigido.

La mayoría de las definiciones vistas para los grafos no dirigidos son válidas, con la adaptación correspondiente al hecho de que las aristas tienen una orientación, a los grafos dirigidos:

Si en un grafo dirigido tenemos una arista (también llamada *arco*) $e = (u, v)$, se dice que

- e *empieza* en u y *termina* en v.
- u es *origen* o *punto inicial* de e.
- v es *destino* o *punto terminal* de e.
- u es *predecesor* de v o v es *sucesor* de u.
- u es *adyacente* a v (pero v no es adyacente a u).

En un grafo dirigido, se habla de grado de entrada y grado de salida: el *grado de entrada* de un vértice es el número de aristas que llegan a él, esto es, el número de aristas en las que el vértice aparece como el segundo elemento del par ordenado. El *grado de salida* es el número de aristas que parten de él, esto es, el número de aristas en las que el vértice aparece como el primer elemento del par.

En un grafo dirigido, un vértice con grado de entrada nulo y grado de salida positivo se denomina nodo *fuente*. Un vértice con grado de entrada positivo y grado de salida nulo se denomina nodo *sumidero*.

Se define el *camino dirigido* de un nodo u a un nodo v a una sucesión de vértices $v_0, v_1, \ldots, v_n$ donde $\forall i = 1, \ldots, n$ v_{i-1} es adyacente a v_i (existe una arista dirigida (v_{i-1}, v_i)), $u = v_0$ y $v = v_n$. Se define la *longitud* del camino anterior como n, es decir, el número de nodos del camino menos 1.

Un camino se dice que es *cerrado* si $v_0 = v_n$, es decir, empieza y termina en el mismo vértice.

Un camino se dice que *simple* si todos los nodos que lo componen son distintos, a excepción del último, que puede ser igual al primero (en un camino cerrado).

Un *ciclo* en un grafo dirigido es un camino dirigido, de longitud 1 o más, que empieza y termina en el mismo nodo. En el grafo de la Figura 3.4 se puede observar la existencia de un ciclo (A, B, C, A).

Un grafo dirigido se dice *conexo*, o *fuertemente conexo*, si para todo par de vértices u y v existe un camino de u a v y otro de v a u. El grafo se dice que es *unilateralmente conexo* si $\forall u, v \in V$ hay un camino de u a v o hay un camino de v a u.

Dos aristas son *paralelas* si tienen el mismo origen y el mismo destino. En este caso, al igual que en los no dirigidos, las aristas del grafo no forman un conjunto sino un multiconjunto.

Un grafo dirigido es *simple* si no tiene aristas paralelas. Un grafo simple puede tener bucles, pero no más de un bucle por nodo.

Un grafo dirigido es *completo* si tiene un arco desde cualquier nodo a cualquier otro, incluyéndose él mismo. Un grafo dirigido completo con n nodos tiene exactamente n^2 aristas.

Un nodo v es alcanzable desde u si existe un camino dirigido de u a v.

Los grafos dirigidos se representan como una nube de puntos, los vértices, junto con un conjunto de flechas que unen pares de vértices.

Un grafo no dirigido puede modelarse como un grafo dirigido con los mismos vértices y en el que hay una arista en cada sentido si hay una arista en el grafo no dirigido original.

Grafos etiquetados y ponderados

Los grafos, dirigidos o no dirigidos, pueden ser *etiquetados*: un grafo es *etiquetado* si cada arista e tiene una etiqueta e_l asignada. Si esta arista tiene un valor numérico no negativo se dice que el grafo es *ponderado* y a sus etiquetas se les llama *pesos*.

En los grafos ponderados se define el peso de un camino como la suma de los pesos de las aristas que lo componen.

Medidas globales sobre grafos

Lejanía de un nodo Es la suma de las distancias de un nodo al resto de nodos.

Cercanía de un nodo Es la inversa de la lejanía.

Autoridad de un nodo Son medidas de la importancia del nodo dentro del grafo. Se verán con más detalle en la Sección 3.4.

Intermediación de un nodo Es el número de caminos mínimos que pasan por él.

3.2.3. Implementaciones de grafos

Existen dos implementaciones clásicas de los grafos, mediante matrices y mediante lo que se denominan listas de adyacencias (aunque en la práctica sea mucho mejor usar *Maps*). Estas dos implementaciones mantienen el grafo en memoria. Existe una libreria de uso libre y gratuito implementada en Java denominada JGraphT (`http://jgrapht.org`) que tiene una completa implementación de los grafos en memoria junto con un conjunto de algoritmos de aplicación sobre estos. Viene acompañada de otra librería denominada JGraph (`http://www.jgraph.com/`) que se conecta con JGraphT y que permite visualizar y manipular en pantalla los grafos.

Además de las anteriores, hay otras implementaciones de grafos de gran tamaño sobre bases de datos. Podemos encontrar estudios sobre métodos de implementación sobre bases de datos y lenguajes de interrogración para estos en (He y Singh, 2010), sobre algoritmos de caminos y alcance en grafos de gran tamaño sobre bases de datos en (Yu y Cheng, 2010) y sobre búsquedas eficientes en este tipo de implementaciones en (Wang y Aggarwal, 2010).

El software de uso libre Neo4j (`http://neo4j.org/`) soporta los grafos en una base de datos y tiene operaciones específicas de grafos, como los recorridos, además de las propias de una base de datos.

Por supuesto, algunas aplicaciones trabajan directamente con el grafo que representa el problema si este es accesible, como puede ser el caso de las redes sociales o la web, que son estructuralmente grafos.

Implementación de grafos mediante matrices

Tenemos dos variantes de las representaciones de grafos mediante matrices, dependiendo de si los grafos son etiquetados (o ponderados) o no. En cualquier caso, si el grafo tiene n vértices, se representa mediante una matriz de $n \times n$ elementos. En lo que sigue suponemos que los vértices están numerados como $u_1, u_2, \ldots u_n$.

Implementación mediante matrices de grafos no etiquetados En este caso los elementos de la matriz son booleanos, denominada *matriz de adyacencias*. El elemento a_{ij} de la matriz de adyacencias A se define como

$$a_{ij} = \begin{cases} \textit{true} & \text{si } v_i \text{ es adyacente a } v_j \\ \textit{false} & \text{en caso contrario} \end{cases}$$

Obsérvese que de diferentes ordenaciones de los nodos del grafo resultarán distintas matrices de adyacencia, pero cada una se obtendrá de otra permutando filas y columnas, por lo que los resultados son equivalentes.

Implementación mediante matrices de grafos etiquetados En este caso el valor que se le asigna al elemento ij es la etiqueta de la arista que va de v_i a v_j si la hubiera o un valor que no puedan tomar las aristas (generalmente *null*) en caso contrario.

Implementación mediante matrices de grafos ponderados Aquí la matriz será del tipo numérico del que sean los pesos de las aristas, enteros o reales y la matriz se denomina *matriz de pesos* o de *costos*, C. El elemento c_{ij} se define como

$$c_{ij} = \begin{cases} \text{peso de la arista que va de } v_i \text{ a } v_j & \text{si } v_i \text{ es adyacente a } v_j \\ \infty & \text{en caso contrario} \end{cases}$$

El valor ∞ puede sustituirse por otro valor que no pueda ser tomado por ninguna arista, pero la marca de la inexistencia de arista mediante el valor infinito es conveniente para algunos algoritmos, como el de Dijkstra o el de Floyd.

Implementación de los grafos mediante listas de adyacencias

Aunque tradicionalmente se denomina así por razones históricas, esta implementación utiliza realmente *Maps* de adyacencias. Veamos en primer lugar como se representan los grafos dirigidos.

El conjunto de vértices no es ahora tal cosa, sino que será el conjunto de las claves de un Map; si el grafo es no etiquetado, el valor asociado a cada clave u del Map (cada vértice) es un conjunto con los vértices adyacentes con el vértice u.

Si el grafo es etiquetado, el valor asociado con un vértice u (la clave u) será un a su vez un *Map* que contendrá una entrada por cada vértice v con el que el vértice u sea adyacente, y cuyo valor asociado será la etiqueta (el peso si es un grafo ponderado) de la arista (u, v).

Si el grafo es no dirigido, se representará según la manera anterior, pero como si por cada arista no dirigida $[u, v]$ hubiera una arista (u, v) y otra (v, u).

3.2.4. Algoritmos clásicos sobre grafos

Hay una infinidad de problemas clásicos que se pueden plantear y resolver con los grafos. Exponerlos todos excede la introducción a los grafos que pretendemos hacer en este capítulo. Destacaremos algunos por su importancia, aunque no entraremos en detalles sobre ellos.

Aparte de los que exponemos a continuación hay muchos otros problemas sobre grafos, con sus correspondientes algoritmos, con aplicaciones prácticas, como la planaridad, el coloreado, la asignación de tareas, la localización de servicios, las rutas de flotas, etc.

Recorridos

Los **recorridos**, o **búsquedas**, (en inglés *traversal*) son maneras de recorrer sistemáticamente todos los vértices de un grafo. Existen dos maneras fundamentales: en profundidad y en anchura. En los dos casos se parte de un determinado vértice u.

Recorrido en profundidad En el recorrido en profundidad se recorre en primer lugar un vecino v del vértice u; a continuación un vecino w de v que todavía no haya sido visitado, etc.; cuando ya no se pueda seguir, bien porque ya no queden más vértices, bien porque los que nos encontramos ya han sido visitados, se retrocede hasta encontrar un vértice que tenga algún

vecino que aún no haya sido visitado y se sigue desde allí. Obsérvese que esto es precisamente el núcleo *bactracking* o vuelta atrás.

Recorrido en anchura En el recorrido en anchura se recorren en primer lugar todos los vecinos de u. A continuación los vecinos de los vecinos de u que aún no hayan sido visitados. Después, los vecinos de los vecinos de u que aún no hayan sido visitados, etc.

Ambos recorrido llegan hasta todos los vértices alcanzables desde el vértice de partida. Sirven para resolver múltiples problemas sobre los grafos; por ejemplo, nos permite averiguar si un nodo es alcanzable desde otro, es decir, si existe un camino del segundo al primero.

Algoritmos de caminos

Se denominan así genéricamente los algoritmos que permiten averiguar qué nodos son alcanzables desde qué nodos o cuáles son y qué coste tienen los caminos mínimos entre pares de vértices de un grafo. Vamos a ver tres por orden de antigüedad. Los tres tienen una implementaciones extraordinariamente simples y eficientes en su implementación mediante matrices.

Algoritmo de Dijkstra Este algoritmo es de un pionero holandés de la informática. Dado un grafo dirigido y ponderado, su algoritmo (Dijkstra, 1959) encuentra los caminos mínimos entre un vértice dado y todos los demás y la longitud de estos. La implementación original lo hacía en $O(|V|^2)$ Posteriores mejoras lo han dejado en $O(|E| + |V| \log |V|)$. Es un algoritmo de Programación Dinámica. Es de una sencillez impresionante pese a la potencia de su resultado.

Algoritmo de Warshall Dada la matriz de adyacencias de un grafo dirigido, el algoritmo de Warshall (Warshall, 1962) devuelve la que se denomina *matriz de caminos* o *matriz de alcance* P del grafo. En esta matriz el elemento p_{ij} será verdadero si existe camino entre v_i y v_j y falso en caso contrario. Lo resuelve en $O(|V|^3)$ con unos requisitos de memoria de $O(|V|^2)$, es decir, el tamaño de la matriz de adyacencia o la de alcance. Si imaginamos la matriz de alcance como la matriz de adyacencia de otro grafo, este se denomina el cierre transitivo del grafo original. Es, al igual que el anterior, un algoritmo de Programación Dinámica.

Algoritmo de Floyd Publicado solo cinco meses después del anterior (Floyd, 1962), es de una potencia extraordinaria, siendo, a la vez, de una sencillez y elegancia insuperables. Dada la matriz de pesos de un grafo dirigido y ponderado devuelve la matriz de caminos mínimos Q, donde el elemento q_{ij} es la longitud del camino más corto de v_i a v_j si existe tal camino o ∞ si no existe ningún camino. Devuelve también una matriz que permite reconstruir cuáles son los caminos más cortos. Lo hace, al igual del de Warshall, del que es una variante, con una complejidad de $O(|V|^3)$ en tiempo y de $O(|V|^2)$ en memoria. Es, como los dos anteriores, un algoritmo de Programación Dinámica.

Ordenación topológica

Si representamos en un grafo las tareas necesarias para resolver un proyecto como vértices y con una arista dirigida entre u y v la condición de que v no puede comenzar hasta que se haya completado u, el grafo resultante, por su propia naturaleza, es acíclico y representa lo que se conoce como una ordenación parcial sobre los vértices del grafo (una relación que cumple la propiedad irreflexiva, antisimétrica y transitiva). Sobre un grafo dirigido acíclico se puede aplicar lo que se denomina una *ordenación topológica*, que es una ordenación lineal de los nodos del grafo original que preserva la ordenación parcial del grafo original. Si en el grafo original u va antes que v, entonces u va antes que v en la ordenación topológica. El primero en aportar una solución fue Kahn (1962).

Árboles de extensión de coste mínimo

Dado un grafo no dirigido, conexo y ponderado $G = (V, E)$, un *árbol de extensión*(en inglés *spanning tree*) para dicho grafo es otro grafo $G' = (V, E')$ compuesto por los mismo vértices y donde las aristas son un subconjunto de $E' \subset E$ que cumple que conecta todos los vértices de V y que no tiene ciclos (este tipo de grafo se conoce también como *árbol libre*). Se puede demostrar que ese grafo tiene $|V| - 1$ aristas. Un grafo puede tener muchos árboles de extensión. De entre todos ellos interesa encontrar el que tiene coste mínimo, es decir, aquél que siendo un árbol de extensión, minimiza la suma de los costes de las aristas de las que está compuesto. Esto se resuelve mediante el algoritmo de Kruskal (Kruskal, 1956) en $O(|E| \log |V|)$ y el de Prim (Prim, 1957) en $O(|V|^2)$.

Figura 3.5 – Conexiones entre usuarios de la red social facebook.

3.3. Relación entre Grafos y Minería de Datos

Así como las aplicaciones de la Minería de Datos a datos estructurados en grafos son numerosísimas, como veremos a continuación, el camino inverso, es decir, la aplicación de los grafos para desarrollar algoritmos de Minería de Datos apenas si tiene aparición, más allá de que las instancias en un conjunto de datos pueden representarse como una nube de puntos (nodos) y las distancias entre cada par como la etiqueta de la arista que los una. De esta representación cabe derivar inmediatamente algoritmos de clustering, semisupervisado o no supervisado. Los algoritmos que forman el núcleo de nuestro trabajo hacen precisamente el camino indicado: representamos los datos o las relaciones entre ellos en forma de grafos y, trabajando sobre estos grafos obtenemos algoritmos de selección de instancias y de selección de atributos.

Vamos a ver someramente en esta sección algunas aportaciones que hace la Minería de Datos en el estudio de los datos estructurados en grafos. Antes de continuar, tenemos que tener en cuenta que muchos problemas están definidos mediante un grafo; por ejemplo el problema del viajante (Applegate y otros, 2007), los que tienen que ver con redes sociales (Figura 3.5), redes de distribución o comunicaciones, carreteras, callejeros, la Web, redes de coautorías o citas en publicaciones científicas o en películas, redes de transmisión de virus (informáticos o biológicos), relaciones entre enfermedades y genes (Figura 3.6), compuestos químicos, estructura de las proteínas, interconexiones entre empresas, archivos XML, y un largo etcétera en el que se incluye, por supuesto, cualquier problema que se modele mediante una nube de puntos.

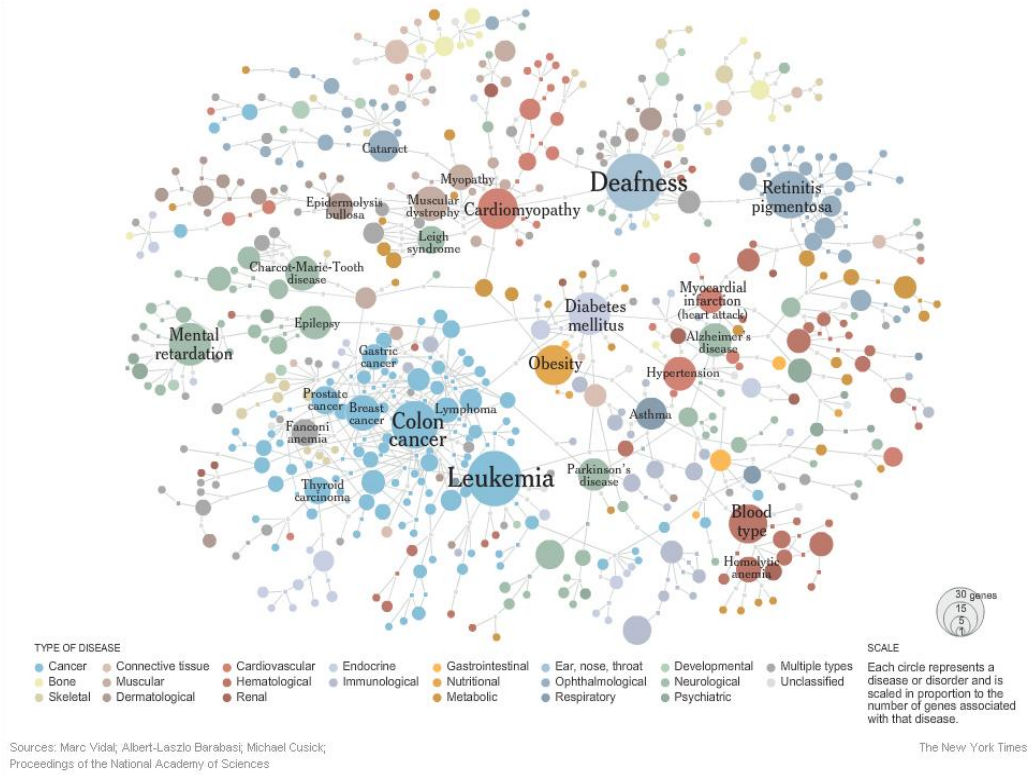

Figura 3.6 – Grafo que representa las relaciones entre enfermedades, representadas por círculos, y genes, representados por cuadrados. Fuente: (Pollack, 2008)

Hay otros problemas que no son obvia o directamente un grafo pero que se pueden modelar con ellos. El problema de los puentes de Königsberg, es un ejemplo; la planta de un museo o un supermercado se puede modelar como un grafo para determinar los puntos en los que hay que ubicar vigilantes o cámaras; la distribución óptima de tareas o el problema de los matrimonios estables (son equivalentes) se pueden modelar como un problema de emparejamiento en grafos bipartitos.

Hay muchas aplicaciones de la minería de datos a datos en grafos. Veamos a continuación algunas de ellas.

3.3.1. Clasificación de Datos en Grafos

Bajo este nombre podemos encuadrar dos tipos de tareas diferentes, pero relacionadas:

Propagación de etiquetas Supongamos que solo está etiquetado un sub-

conjunto de los nodos de un grafo. La tarea que se plantea es aprender un modelo a partir de los nodos etiquetados y utilizarlo para clasificar, esto es, asignar una etiqueta, el resto de los nodos no etiquetados.

Clasificación de grafos La tarea ahora consiste en lo siguiente: partimos de un conjunto de grafos, algunos de los cuales están etiquetados. La tarea ahora es aprender un modelo a partir de los datos etiquetados y usar ese modelo para etiquetar los grafos no etiquetados.

Vamos a ver con más detalle cada una de estas dos tareas.

Propagación de etiquetas

Esta tarea se plantea en numerosas situaciones, como se puede ver en (Neuhaus y Bunke, 2007). Por ejemplo, puede plantearse el uso de una red social para realizar una campaña de marketing orientada a usuarios concretos. Se tiene información sobre los clientes que han recibido a determinadas promociones. Los clientes que han respondido a las promociones son etiquetados como nodos positivos en la red social y aquellos que han recibido la promoción y no han respondido se etiquetan como nodos negativos. El objetivo es enviar las promociones a clientes que con más probabilidad vayan a responder positivamente. Hay que aprender el modelo a partir de los únicos datos de los que se dispone: aquellos clientes que han recibido las promociones y a partir de ellos hay que predecir los clientes potenciales de la red social. Se trata, por tanto, de ver cómo propagar a través de la red social los nodos positivos y negativos.

Se parte de la intuición de que nodos "similares" tendrán etiquetas similares. El problema fundamental es encontrar una función de distancia que mida la similitud entre los nodos del grafo. Una aproximación muy utilizada es contar el número de pasos que hay que dar para llegar de un nodo al otro a través de un paseo aleatorio (*random walk*) (Kondor y Lafferty, 2002). Esta aproximación tiene dos inconvenientes que la hacen inoperante en grafos de grandes dimensiones, como son los que cabe esperar en este tipo de problemas: calcular todas las distancias tiene un coste $O(n^3)$ en tiempo y $O(n^2)$ en almacenamiento. Aprovechando que en la práctica estos grafos suelen ser muy dispersos (los nodos tienen un grado muy bajo respecto al número total de nodos), se han desarrollado métodos para optimizar este cálculo. Por ejemplo, Zhou y otros (2003) presentan un método que es casi lineal con el número de aristas del grafo.

Clasificación de grafos

Métodos basados en *kernel* Los métodos basados en kernel emplean también los *random walks* para encontrar la similitud entre dos grafos. Por cada grafo, se calculan sus caminos y se calculan las probabilidades de que ocurran dichos caminos. La medida es la comparación entre los conjuntos de caminos y sus probabilidades.

Métodos basados en *boosting* En estos métodos se busca la subestructura de los grafos: se crea un vector binario basado en la presencia o la ausencia de las subestructuras y se aplica un clasificador clásico.

Las subestructuras más habitualmente utilizadas son las de patrones más frecuentes, aunque los patrones más frecuentes no tienen que ser necesariamente los más relevantes. Por ejemplo, en grafos que reflejen sustancias químicas (en química orgánica) los patrones `C-C` o `C-C-C` son los más frecuentes, pero no suelen tener la menor relevancia en la predicción de características importantes como la actividad, la toxicidad, etc.

Se usa boosting para seleccionar automáticamente un conjunto de subgrafos relevantes. Por ejemplo, LPBoost (*Linear Program Boost*) aprende una función discriminante lineal para seleccionar las características. Se muestra en (Saigo y otros, 2009) que esta técnica obtiene muy buenos resultados, alcanzando mejor exactitud que los métodos basados en kernels, con la ventaja adicional de que a la vez encuentra explícitamente las subestructuras relevantes.

3.3.2. Búsqueda de patrones en grafos

Con este nombre nos podemos referir a dos problemas distintos: la búsqueda de un determinado patrón (unos valores en los vértices y/o aristas junto con una determinada relación entre las aristas) en un grafo, o bien, la búsqueda de patrones comunes, en el mismo sentido anterior, en un conjunto de grafos. En cualquier caso, decimos que dos subgrafos son equivalentes si cumplen las tres condiciones:

1. Las etiquetas de los nodos en los dos grafos deben ser las mismas.

2. La existencia de aristas entre los nodos correspondientes de los dos grafos debe corresponderse una con otra.

3. Las etiquetas de las aristas que se corresponden según el punto anterior deben coincidir.

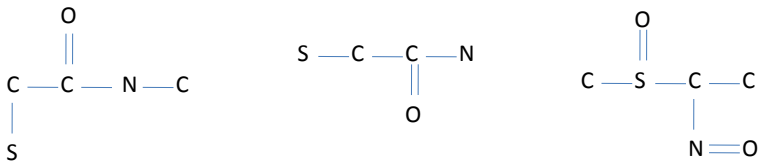

Figura 3.7 – Tres grafos representando la fórmula de tres compuestos químicos orgánicos; en ellos se quieren encontrar subgrafos frecuentes.

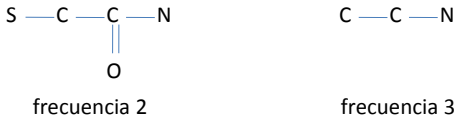

Figura 3.8 – Subgrafos frecuentes en los grafos de la Figura 3.7. Podemos ver que el de la izquierda aparece en dos de ellos, el primero y el segundo (realmente es igual que el segundo), mientras que el de la derecha aparece en los tres.

Las aplicaciones de esta tarea son múltiples. Dentro de la minería de grafos ha sido utilizado, por ejemplo, para la clasificación, el agrupamiento y la indexación de grafos. En aplicaciones soportadas sobre grafos ha sido utilizada para el descubrimiento de estructuras químicas activas, para la clasificación de compuestos químicos, para estudiar la relación entre proteínas, encontrar rutas en redes metabólicas, etcétera.

Búsqueda de patrones comunes en un conjunto de grafos

Podemos ver un ejemplo de este tipo de problema en la Figura 3.7. Vemos representadas las fórmulas de tres compuestos químicos. En ellos queremos encontrar patrones frecuentes. En la Figura 3.8 podemos ver dos subgrafos frecuentes en los grafos originales, junto con su frecuencia.

Tenemos un conjunto de grafos $D = \{G_1, \ldots, G_n\}$ y queremos encontrar patrones frecuentes dentro de ellos. En primer lugar definimos el *soporte* de un subgrafo como el número de veces que aparece el subgrafo. Los subgrafos frecuentes serán aquellos cuyo soporte supere un determinado umbral.

Este problema se puede abordar con técnicas propias de la búsqueda de patrones frecuentes en no-grafos. Por ejemplo, se pueden utilizar algoritmos del estilo del método de generación de reglas de asociación Apriori. Se parte de la premisa de que todos los subgrafos de un grafo frecuente

son también frecuentes. A continuación el algoritmo calcula el conjunto de subgrafos frecuentes de tamaño $k+1$ partiendo del conjunto de subgrafos de tamaño k generando candidatos de tamaño $k+1$ extendiendo los subgrafos y quedándose solo con los que sean frecuentes. Dos subgrafos de tamaño k se pueden unir si tienen una estructura de tamaño $k-1$ en común. Se pueden encontrar soluciones de estas características es el algoritmo AGM (Inokuchi y otros, 2000), en el que se define la estructura común entre dos grafos en términos del número de vértices en común, o en el algoritmo FSG (Kuramochi y Karypis, 2001) en el que se utilizan las aristas comunes.

Búsqueda de patrones comunes en un grafo

El problema ahora consiste en encontrar patrones frecuentes dentro de un grafo. En este caso es mucho más difícil definir el soporte de un subgrafo, porque pueden ocurrir solapamientos; en este caso no se cumple el principio de anti-monotonicidad, que es esencial en los algoritmos de búsqueda de patrones frecuentes en no-grafos. Kuramochi y Karypis (2005) proponen dos algoritmos eficientes, HSIGRAM y VSIGRAM, basados en búsquedas en anchura y en profundidad, respectivamente.

3.3.3. Algoritmos de clustering para datos en grafos

Bajo este nombre podemos encuadrar dos tipos de problemas:

Algoritmos de clustering de nodos Partimos de un solo grafo y la tarea consiste en encontrar agrupaciones de nodos basándonos en la distancia (o similitud) entre ellos reflejadas por los valores de las aristas, que deben tener valores numéricos. Un caso particular se da cuando esos valores solo pueden tomar los valores 1 y 0, reflejando la presencia o ausencia de aristas entre los vértices correspondientes.

Algoritmos de clustering de grafos En este caso partimos de un conjunto de grafos y queremos agruparlos según su estructura. En este caso el problema radica principalmente en la dificultad de definir la similitud entre estructuras.

Algoritmos de clustering de nodos

Este problema, conocido como partición de grafo (*graph partitioning*) o problema del corte mínimo multicamino (*minimum multi-way cut problem*), también se conoce como la búsqueda de *quasi-cliques*, es decir, conjuntos

de nodos (subgrafos) entre los que, con una probabilidad predeterminada, existen aristas: dado un valor $\gamma \in (0, 1]$, y un subgrafo de k nodos, el grado de cada nodo es, al menos, $\gamma \cdot k$.

Flake y otros (2003) estudian varios algoritmos para el clustering de nodos desde el punto de vista de los problemas del corte mínimo en un grafo ponderado y de la partición de este. En este caso se minimizan los costes de las aristas que unen cada partición. El caso más simple es el problema del corte mínimo en dos particiones (*2-way minimum cut problem*), que se puede resolver por la aplicación iterada del problema del flujo máximo corte mínimo, que se estudiará ampliamente en la subsección 5.2.1 del capítulo 5. Se eligen dos nodos como nodos fuente y sumidero (s y t, respectivamente) y se halla el $s - t$ cut; cambiando los nodos s y t es posible encontrar un corte mínimo global en tiempo polinomial respecto al número de nodos del grafo. Otra manera de abordar el problema es desde un punto de vista probabilístico: se van eligiendo aristas y colapsando los nodos en conjuntos cada vez más grandes. Se van eligiendo diferentes secuencias de aristas y guardando los valores óptimos (Tsay y otros, 1999). Esta solución también es de tiempo polinomial respecto al número de nodos.

Cuando el número de particiones es mayor que 2 es un problema NP. En este caso se trata de encontrar k particiones del grafo ($k > 2$) de modo que la suma de los pesos de las aristas cuyos extremos estén en distintas particiones sea mínimo. Una solución la aporta el algoritmo Kerninghan-Lin (Kernighan y Lin, 1970). Es un algoritmo de *hill-climbing*. Se parte de un corte aleatorio del grafo; en cada iteración se intercambian un par de vértices de dos particiones y se comprueba si con esto se reduce el valor total del corte (la suma de los costes de las aristas cuyos extremos estén en distintas particiones). Si es así, se intercambian los nodos; en otro caso, se elige al azar otro par de nodos. Se repite el proceso hasta que se converge a una solución óptima, que no tiene necesariamente que ser global.

Algoritmos de clustering de grafos

Como hemos dicho, el clustering de grafos, es decir, la búsqueda de agrupamientos de grafos según algún criterio de similitud entre ellos, tiene su principal dificultad en la determinación de la distancia o similitud entre dos grafos. Este es un problema muy difícil si se trata en general, pero tradicionalmente, se ha aplicado a un caso particular de grafo: los datos en formato XML. Con esta restricción existen algoritmos, como XClust (Lee y otros, 2002). Utiliza un método de clustering aglomerativo jerárquico.

3.4. Medidas de centralidad y de ranking en grafos

En análisis de redes sociales puede ser de gran interés el grado de un nodo; el grado de salida de un nodo (una persona) puede suministrar información sobre la capacidad de influencia de esa persona sobre otras; esta medida se llama *expansiveness*, que podríamos traducir por *expansividad*. Por otra parte, el grado de entrada es una medida de la *popularidad* de un nodo en la red. En una red, generalmente hay caminos muy cortos entre todos los pares de nodos; esto se conoce como el *efecto del mundo pequeño* (en inglés *"small-world effect"*) (Milgram, 1967). Por otra parte, habitualmente, en una red uno pocos nodos tienen muchas conexiones con muchos nodos mientras que la mayoría de los nodos tienen muy pocas conexiones con los demás; esto se conoce como el efecto de la "larga cola" (*"the long tail"*), que tiene su equivalencia en lingüística en la ley de Zipf, que dice que si llamamos n a la posición que ocupa una palabra en orden de frecuencia de uso y P_n a la probabilidad de que esa palabra sea usada, es $P_n \approx n^{-\gamma}$ con γ próxima a 1. Esta expresión también es conocida como la ley de potencias.

3.4.1. Relación entre grado y selectividad

Se habla de redes *selectivas* (en inglés *assortatives*) o *no selectivas* (*disassortatives*). Si definimos como $E\left[k_{nn}(k)\right]$ como el grado medio de de los vecinos de un vértice de grado k (Newman y Park, 2003), si este valor aumenta cuando aumenta k decimos que la red es selectiva; en caso contrario es no selectiva. Esto equivale a decir que si $E\left[k_{nn}(k)\right]$ sigue la ley de potencias, esto es $E\left[k_{nn}(k)\right] \approx k^{-\gamma}$, en las redes selectivas γ es negativa, mientras que en las no selectivas es positiva. Las redes sociales tienden a ser selectivas mientras que las biológicas o tecnológicas tienden a ser no selectivas.

3.4.2. Centralidad y prestigio

Centralidad

Una cuestión fundamental en las redes sociales es la determinación de la importancia de un nodo dentro de la red (Wasserman y Faust, 1994). La *centralidad* es un valor que puede determinar esta importancia. Hay diversas definiciones de centralidad.

- La *centralidad de cercanía* (*closeness centrality*); es el grado de un nodo, normalizado, esto es, dividido entre el total de nodos de la red.
- La *centralidad de distancia* (*distance centrality*) $\mathcal{D}_c$ de un nodo u como la distancia media de u a los demás nodo del grafo:

$$\mathcal{D}_c(u) = \frac{1}{|V| - 1} \sum_{v \neq u} d(u, v),$$

  donde $d(u, v)$ es la distancia del camino más corto de u a v.
- La *Centralidad de intermediación* (*Betweeness centrality*) $\mathcal{B}_c$ de un nodo u es el número medio de los caminos más cortos que pasan por él:

$$\mathcal{B}_c(u) = \sum_{s \neq u \neq t} \frac{\sigma_{st}(u)}{\sigma_{st}},$$

donde $\sigma_{st}(u)$ es el número de los caminos más cortos del nodo s al nodo t que pasan por u y σ_{st} es el número total de caminos más cortos que van de s a t.

Prestigio

Una idea distinta (pero relacionada) a la centralidad es la del *prestigio*, que es la capacidad de un nodo de atraer enlaces entrantes. La idea es que un enlace de un nodo u a un nodo v denota apoyo o respaldo (*endorsement*). En su forma más simple, el prestigio de un nodo se puede medir como su grado de entrada. En el contexto de la Web esto se conoce como *visibilidad* (Bray, 1996) y fue el criterio utilizado por los primeros buscadores.

3.5. Algoritmos de Ranking basados en enlaces

La aparición de la World Wide Web, y de los motores de búsqueda, dotó de mucha importancia a la priorización de los nodos dentro de la red. Además de las nociones apuntadas en la sección anterior, aparecieron otros conceptos de importancia basados en la relación global de todos los enlaces dentro de la red.

Como se ha indicado cuando se hablaba del prestigio, una primera aproximación fue el grado de entrada. Marchiori (1997) propone modelos basados en los enlaces entrantes y salientes y en la propagación del prestigio entre los nodos. Estos modelos están basados en el prestigio de

las publicaciones (el índice de impacto) y fueron estudiados originalmente por Eugene Garfield en el ámbito de la bibliometría, que en 1955 funda una empresa que se convertiría posteriormente en el *Institute for Scientific Information, ISI* y la *Web of Knowledge (WoK)*; es el creador del *Journal Citation Reports (JCR)*, que mide el prestigio de una revista en función del número de citas que los artículos publicados en ella reciben de otras revistas, pero teniendo en cuenta también el prestigio de estas.

Los dos principales algoritmos de este tipo son PageRank y HITS. Vamos a estudiarlos con más detenimiento.

3.5.1. PageRank

La principal debilidad del concepto de visibilidad, visto en la subsección 3.4.2, es que puede ser manipulada con mucha facilidad creando páginas de spam que apunten a una página determinada únicamente para aumentar su prestigio. PageRank (Page y otros, 1998) usa la información de los enlaces entrantes y de los enlaces salientes de modo iterativo para determinar el rango de un nodo. Se basa en el modelo del "navegador aleatorio" (*"random surfer"*); el navegador va visitando páginas web siguiendo hiperenlaces al azar, y ocasionalmente visita una página concreta que no está enlazada con la última que había estado visitando. Un poco más formalmente, el navegador comienza a navegar desde un nodo (una web) elegido al azar; a continuación, en cada paso el navegador hace una de estas dos cosas:

- Elige un hiperenlace de la página actual con probabilidad d y va al documento enlazado por él.
- Salta con una probabilidad $1 - d$ a una página elegida al azar, de acuerdo con determinada distribución, y que no está enlazada desde la página actual. La distribución se asume que es la uniforme.

El rango del nodo v, denominado su PageRank, y que denotaremos como $PR(v)$, es el tiempo que el navegador pasa en el nodo v o, lo que es lo mismo, la probabilidad de que en un instante determinado esté visitando ese nodo. Es, por tanto, una medida de la importancia de ese nodo.

Si denotamos como $In(v)$ como el conjunto de nodos que tienen una arista con destino en v y como $Out(v)$ como el conjunto de los nodos que tienen una arista con origen en v, se tiene la expresión

$$PR(v) = (1 - d) + d \sum_{w \in In(v)} \frac{PR(w)}{|Out(w)|},$$

donde d es un parámetro entre 0 y 1 (en el artículo original recomiendan $0{,}85$).

Esta expresión conduce a un algoritmo iterativo que permite calcular el PageRank de todos los nodos.

La formulación original del PageRank incluía otro parámetro que permitía dar mayor importancia a ciertos nodos del grafo. Con esta variante la expresión del PageRank queda

$$PR(v) = (1-d)e_v + d \sum_{w \in In(v)} \frac{PR(w)}{|Out(w)},$$

donde e_v refleja la mayor o menor importancia que se le asigna a priori al nodo v, y que debe cumplir que $\sum_{v \in V} e_v = 1$. Obsérvese que si se le asigna a todos los e_v el mismo valor $1/|V|$ se obtiene una expresión equivalente a la primera.

Con posterioridad se han desarrollado muchos trabajos en los que se optimiza el tiempo de computación (Kamvar y otros, 2003; Corso y otros, 2005).

3.5.2. HITS

PageRank parte de la premisa de que todas las páginas, en mayor o menor medida tienen una determinada autoridad. HITS (*Hypertext Induced Topic Selection*) (Kleinberg, 1999) distingue dos conceptos, *hub* y *authority* [1]:

- *Authority* refleja la capacidad de un nodo de dar información. Para ello, usa la valoración que los demás nodos hacen de él. En principio, un nodo tiene mayor *authority*, cuanto más nodos apunten hacia él (mayor sea su grado de entrada).
- *Hub* refleja la capacidad del nodo de decir donde hay información. En principio, un nodo tiene mayor *hub* si apunta a más nodos (cuanto mayor sea su grado de salida).

Cada nodo actúa como una *authority* para los nodos que lo enlazan y como un *hub* para aquellos a los que enlaza.

Hemos dicho en las definiciones "en principio" porque el valor final depende de un doble cálculo iterativo:

$$Authority(v) = \sum_{w \in In(v)} Hub(w)$$

[1] Hemos optado por mantener los términos en inglés porque creemos que las traducciones litelares, centro y autoridad, no reflejan correctamente los conceptos.

$$Hub(v) = \sum_{w \in Out(v)} Authority(w)$$

Tras cada iteración se normalizan los valores de modo que

$$\sum_{v \in V} Authority(v)^2 = 1, \qquad \sum_{v \in V} Hub(v)^2 = 1.$$

Valores elevados de *hub* y *authority* para un nodo significa que el nodo es relevante, aunque en este caso el nodo puede jugar mejor uno de los dos roles.

3.6. Resumen

En este capítulo hemos dado una serie de definiciones sobre grafos y sus implementaciones. Hemos visto también algunos algoritmos clásicos sobre grafos junto con sus posibles aplicaciones. Posteriormente hemos estudiado la relación entre grafos y minería de datos, centrándonos en algunas aplicaciones de la minería de datos sobre datos organizados como grafos. Finalmente hemos visto medidas de centralidad de nodos, de prestigio y algoritmos que nos permiten establecer rangos sobre los nodos.

Parte III

Selección de Atributos

Capítulo 4

Introducción a la Selección de Atributos

La Selección de Atributos es una parte muy importante del aprendizaje automático, transversal a muchas de las técnicas que este emplea. Hablando de una forma muy general, consiste en eliminar aquellos atributos que aportan poco o nada a los datos, de modo que los algoritmos de aprendizaje puedan emplear solo aquellos atributos que son útiles para la fase de aprendizaje y la posterior predicción. ¿Qué atributos hay que eliminar? Aquellos que no son relevantes (no aportan información útil) o son redundantes (la información que aportan ya es suministrada por otro u otros). Para sintetizar, lo que se consigue con selección de atributos es:

- Reducir el tiempo y el espacio de almacenamiento necesario para las subsiguientes operaciones a realizar con los datos.
- Obtener un conjunto de atributos que permita una comprensión más sencilla del modelo subyacente al problema que se está investigando.
- Eliminar los efectos perniciosos (el sobreajuste) de la "maldición de la dimensionalidad", termino acuñado por Richard E. Bellman (Bellman, 1957), que consiste en el crecimiento exponencial del número de instancias necesarias para describir los datos cuando crece linealmente el número de atributos.
- Disminuir el coste de la obtención de datos en futuras investigaciones.

La selección de atributos pertenece a la fase de preprocesamiento de datos en el esquema general de la minería de datos, que podemos ver en la figura 4.1 (Pal y Pal, 2001).

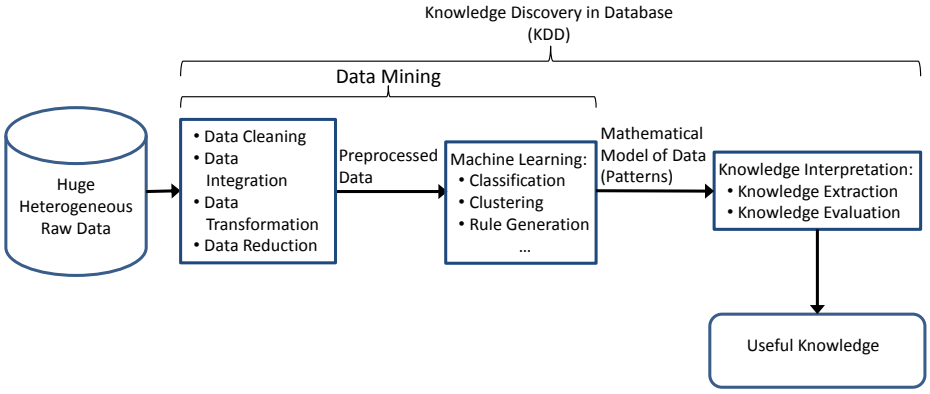

Figura 4.1 – Esquema general del proceso de adquisición de conocimientos en conjuntos de datos (*KDD*).

A continuación vamos a ver una introducción al problema de la selección de atributos, así como un estudio de los algoritmos existentes y de medidas de la bondad de estos.

4.1. Definición del problema

Antes de definir el problema, debemos tener en cuenta la distinción entre el planteamiento continuo y el binario. El problema *continuo* de la selección de atributos se refiere a la asignación de pesos a cada uno de los atributos de modo que se preserve el orden de importancia teórico de cada atributo. El problema *binario* de la selección de atributos, también llamado "Selección de un Subconjunto de Atributos" (en inglés *Feature Subset Selection*) se refiere a la asignación de pesos binarios: los atributos se usan o no, lo que equivale a que se seleccionan o no. En lo que sigue, cuando hablemos de Selección de Atributos (en inglés *Feature Selection* o, *FS*), estaremos refiriéndonos exclusivamente a este segundo problema.

El problema de la Selección de Atributos puede plantearse del siguiente modo: dado un conjunto de atributos, seleccionar un subconjunto de ellos que

- tenga un tamaño predeterminado y que optimice alguna medida de evaluación.
- tenga el tamaño menor posible y que satisfaga cierta restricción sobre la medida de evaluación.
- tenga el mejor compromiso entre el tamaño y el valor de la medida de evaluación.

De un modo general, el problema de encontrar un buen subconjunto de atributos puede visualizarse en el esquema que vemos en la Figura 4.2: se parte de un conjunto de entrenamiento; aplicamos sobre ellos un mecanismo de selección de atributos, que nos devuelve un subconjunto de estos. Aplicamos el conjunto de entrenamiento reducido a un algoritmo de clasificación (un árbol de decisión, el vecino más cercano, etc.), con el que evaluamos un conjunto de test, en el que se han dejado solo los atributos seleccionados por el algoritmo de selección de atributos. Este algoritmo de clasificación dará una precisión que será una medida del merito de la selección de atributos realizada.

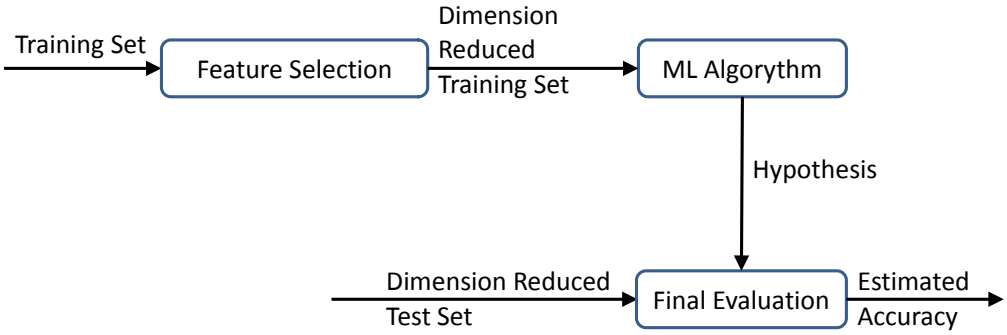

Figura 4.2 – Esquema general del problema de la búsqueda de un subconjunto de atributos.

La selección de atributos puede visualizarse como un problema de búsqueda, donde cada estado se corresponde con un subconjunto de atributos. Evidentemente, una búsqueda exhaustiva en el espacio de soluciones, que garantizaría el hallazgo del subconjunto óptimo, es exponencial respecto al número total de atributos. Esto resulta prohibitivo en cuanto tenemos un número mediano de atributos (tengamos en cuenta que la utilidad de la selección de atributos se pone de manifiesto cuando tenemos un número mediano o grande de atributos: si tenemos un número

pequeño puede que no sea necesario ni útil). Los algoritmos que se plantean buscan, por tanto, una solución subóptima, pero que se obtenga en tiempo polinomial. De ahí que las estrategias generalmente añaden (o eliminan) atributos secuencialmente, por lo que, en general, necesitaremos una medida de la relevancia de estos.

Vamos a dar algunas definiciones formales del problema (González Navarro, 2011):

Sea $\mathcal{X}$ el conjunto original de atributos, donde $|\mathcal{X}| = m > 0$. Sea la medida de evaluación $\mathcal{L} : \mathcal{P}(\mathcal{X}) \to \mathbb{R}^+ \cup \{0\}$; esta es la medida que tenemos que maximizar, donde $\mathcal{P}(\mathcal{X})$ denota el conjunto de las partes de $\mathcal{X}$, esto es, $\mathcal{P}(\mathcal{X}) = \{S|S \subset \mathcal{P}\}$. Denotaremos como $X_k \subseteq \mathcal{X}$ un subconjunto de atributos seleccionados, con $|X_k| = k$. Por tanto es $X_0 = \varnothing$ y $X_m = \mathcal{X}$.

Definición 4.1 (Selección de Atributos)
Sea $\mathcal{L}$ una medida de evaluación a optimizar (maximizar). La selección de un subconjunto de atributos puede hacerse bajo dos premisas:

- *Sea k tal que $0 < k < m$. Encontrar $X_k \subset \mathcal{X}$ tal que $\mathcal{L}(X_m)$ sea máximo.*
- *Sea un valor real $\mathcal{L}_{\text{mín}} > 0$, esto es, el valor mínimo de $\mathcal{L}$ que va a ser aceptado. Encontrar $X_k \subseteq \mathcal{X}$ con el menor k tal que $\mathcal{L}(X_k) \geq \mathcal{L}_{\text{mín}}$. Alternativamente, dado un $\epsilon > 0$, encontrar el $X_k \subseteq \mathcal{X}$ con el menor k, tal que $|\mathcal{L}(X_k) - \mathcal{L}(\mathcal{X})| < \epsilon\mathcal{L}(\mathcal{X})$.* □

En estas condiciones, siempre existirá un subconjunto óptimo de atributos, no necesariamente único. Si denotamos X^* una de las soluciones óptimas, puede ocurrir $\mathcal{L}(X^*) > \mathcal{L}(\mathcal{X})$, $\mathcal{L}(X^*) = \mathcal{L}(\mathcal{X})$, o $\mathcal{L}(X^*) < \mathcal{L}(\mathcal{X})$, esto es, puede mejorarse, igualarse o empeorarse la medida obtenida por el subconjunto seleccionado respecto de la obtenida con el conjunto completo de atributos.

4.2. Relevancia y redundancia

Dos conceptos claves en la selección de atributos son los de relevancia y redundancia. Vamos a analizar más detalladamente estos conceptos.

4.2.1. Relevancia

Intuitivamente, un atributo es relevante si aporta información sobre la clase. Obviamente, toda técnica debe seleccionar atributos relevantes. Sin

embargo, la afirmación inversa no es cierta: si un atributo es relevante no tiene por qué ser necesariamente seleccionado.

De acuerdo con Blum (Blum y Langley, 1997), no hay una única definición de relevancia de un atributo: depende de con qué se esté relacionando. La noción más simple es la de "relevancia con el concepto objetivo" (en las definiciones siguientes, dada una instancia A denotaremos $c(A)$ a la clase de la instancia A).

Definición 4.2 (Relevancia con el objetivo)
Un atributo f_i es relevante con el concepto objetivo c si existe un par de instancias A y B en el conjunto de datos tal que A y B únicamente difieren en el valor de f_i y $c(A) \neq c(B)$. □

Esto significa que A y B solo pueden ser clasificadas usando el atributo f_i. Esta definición, aunque es la más simple e inmediata tiene el inconveniente de que si ese atributo es redundante con otro, otros o una combinación de otros, no encontraremos ningún par de instancias que únicamente difieran en él. Para evitar este inconveniente John, Kohavi y Pfleger (John y otros, 1994) proponen dos definiciones que se corresponden con la noción de "relevancia respecto a una muestra", donde una muestra se entiende como un subconjunto de las instancias del conjunto de datos.

Definición 4.3 (Relevancia fuerte con la muestra)
Un atributo f_i es fuertemente relevante para la muestra S si existen instancias A y B en S que difieren únicamente en el valor de f_i y pertenecen a clases distintas. □

Esta definición es análoga a la definición 4.2, pero considerando ambas instancias en una muestra.

Definición 4.4 (Relevancia débil con la muestra)
Un atributo f_i es débilmente relevante respecto a la muestra S si es posible eliminar un subconjunto de los atributos de modo que f_i pase a ser fuertemente relevante. □

Definición 4.5 (Irrelevancia)
Atributos irrelevantes son aquellos que no entran en ninguna de las categorías anteriores. □

Entre los atributos irrelevantes se encuentran aquellos que son redundantes y aquellos que aportan ruido, que además degradan la clasificación. En cualquier caso todos los atributos irrelevantes degradan la velocidad del algoritmo al que se apliquen los datos.

La figura 4.3 expresa gráficamente la relación entre estas definiciones de atributos. Los atributos fuertemente relevantes deben ser seleccionados por los algoritmos de selección de atributos; los débilmente relevantes pueden ser seleccionados o no, mientras que los irrelevantes nunca deberían ser seleccionados. Existen otras definiciones de relevancia, que pue-

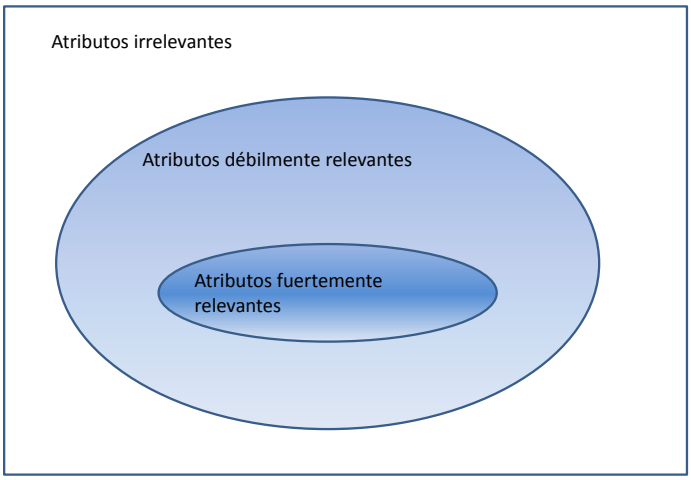

Figura 4.3 – Espacio de atributos. Atributos irrelevantes, débilmente relevantes y fuertemente relevantes.

den ser útiles en algunos esquemas de selección de atributos. Por ejemplo, en (Wang y otros, 1998) se da una definición basada en conceptos de Teoría de la Información:

Definición 4.6 (Relevancia variable o entrópica)
La relevancia variable del atributo f_i respecto de la clase C, $r(f_i, C)$, se define como la relación entre la información mutua entre el atributo y la clase y la entropía de la clase.

$$r(f_i, C) = \frac{I(f_i, C)}{Ent(C)}$$

□

Una definición que nos parece especialmente interesante por la estrecha relación que guarda con la búsqueda en el espacio de subconjuntos de atri-

butos es la *Utilidad incremental*, formulada por Caruana y Freitag (Caruana y Freitag, 1994).

Definición 4.7 (Utilidad incremental)
Dado un conjunto de instancias, un algoritmo de aprendizaje I y un subconjunto de atributos X ($X \subseteq \mathcal{X}$), un atributo f_i es incrementalmente útil *para I con respecto a X si la precisión del algoritmo I usando $X \cup \{f_i\}$ es mejor que la obtenida usando solo X.* □

4.2.2. Redundancia

Al igual que un buen algoritmo de selección de atributos debe quedarse con los atributos relevantes, también debe eliminar los reduntantes. Intuitivamente, un atributo es reduntante con otro si la información que aporta puede obtenerse del otro. Esto conduce a la idea de correlación, aunque no hemos de perder de vista que el tipo de correlación no tiene que ser lineal. Si buscamos la correlación lineal, la forma más sencilla es calcular el coeficiente de correlación de Pearson, ρ.

Definición 4.8 (Coeficiente de correlación de Pearson)
Si tenemos dos atributos f_i y f_j y denotamos como f_{ik} el valor del atributo f_i para la instancia k*-ésima y $\overline{f_i}$ como la media del atributo f_i, el coeficiente de correlación de Pearson, ρ se define como*

$$\rho = \frac{\sum_k \left(f_{ik} - \overline{f_i}\right)\left(f_{jk} - \overline{f_j}\right)}{\sqrt{\sum_k \left(f_{ik} - \overline{f_i}\right)^2}\sqrt{\sum_k \left(f_{jk} - \overline{f_j}\right)^2}}.$$

□

Si buscamos correlaciones no lineales se suelen utilizar medidas de información mutua; por ejemplo, la incertidumbre simétrica (en inglés *symmetric uncertainty*), que usa los conceptos de entropía y de ganancia de información (solo es válido para clases y atributos discretos).

Definición 4.9 (Entropía de un atributo)
Dado un conjunto de datos D, si denotamos $|f_i|$ el número de valores distintos que toma f_i, como f_{i_j} cada uno de esos valores y como $D(f_{i_j})$ el subconjunto de D en el que el atributo f_i toma el valor f_{i_j}, es $p(f_{i_j}) = \frac{|D(f_{i_j})|}{|D|}$ la probabilidad de que f_i tome el valor f_{i_j}. Se define la entropía del atributo f_i como

$$H(f_i) = -\sum_{j=1}^{|f_i|} p(f_{i_j}) \log_2 \left(p(f_{i_j})\right)$$

□

La entropía será mínima si el atributo toma sus valores con igual probabilidad (tienen exactamente la misma proporción en el conjunto de datos).

Según (Hall, 1999), si los valores observados de f_i en el conjunto de datos se particionan de acuerdo con los valores de un segundo atributo f_j, y la entropía de f_i respecto a las particiones inducidas por f_j es menor que la entropía de f_i antes de la partición, entonces hay una relación entre f_i y f_j.

Definición 4.10 (Entropía condicionada)
Se define la entropía condicionada del atributo f_i después de la observación del atributo f_j como

$$H(f_i|f_j) = -\sum_{k=1}^{|f_j|} p(f_{j_k}) \sum_{l=1}^{|f_i|} p(f_{i_l}|f_{j_k}) \log_2(p(f_{i_l}|f_{j_k})$$

□

El valor en el que se decrementa la entropía de f_i muestra la cantidad de información sobre f_i que suministra f_j. A partir de aquí se puede definir la ganancia de información (*information gain*) (Quinlan, 1990).

Definición 4.11 (Ganancia de información)
Se define la ganancia de información como

$$\begin{aligned} gain &= H(f_i) - H(f_i|f_j) \\ &= H(f_j) - H(f_j|f_i) \end{aligned}$$

□

La ganancia de información también se conoce como *información mutua* (Shannon y Weaver, 1949).

La ganancia de información es una medida simétrica (la información ganada por f_i tras observar f_j es la misma que la ganada por f_j tras observar f_i).

Tras estas consideraciones estamos en condiciones de definir la incertidumbre simétrica (*symmetric uncertainty*).

Definición 4.12 (Incertidumbre simétrica)

$$SU(f_i, f_j) = 2{,}0 \times \left[\frac{gain}{H(f_i) + H(f_j)} \right]$$

□

La incertidumbre simétrica compensa la desviación que sufre la ganancia de información hacia atributos que tienen más valores y, además, devuelve un valor normalizado en el rango $[0, 1]$.

Otra medida de redundancia ampliamente utilizada es la denominada *Markov blanket*, propuesta en (Koller y Sahami, 1996). Esta mide la posible redundancia con un conjunto de atributos, no solo entre pares de atributos. Para definir esta medida es preciso definir antes el concepto de *Independencia condicional*.

Definición 4.13 (Independencia condicional)
Dos atributos, o conjuntos de atributos X, Y se dice que son condicionalmente independientes de un tercer atributo o conjunto de atributos Z si dado Z se hacen X e Y independientes, esto es, la distribución de X, conocidas Y y Z, es igual que la distribución de X conocida solo Z, o lo que es lo mismo $P(X|Y, Z) = P(X|Z)$. Por tanto Y no tiene influencia sobre X. □

Definición 4.14 (Markov blanket)
Dado el conjunto de atributos F, un atributo f_i y la clase C, el subconjunto $M \subseteq F(f_i \notin M)$ es un Markov blanket *de f_i si, f_i es condicionalmente independiente de $F - M - \{f_i\}$ y la clase C.* □

Se demuestra que una vez que se encuentra una Markov blanket M para un atributo f_i en el conjunto de todos los atributos F, se puede eliminar f_i.

4.3. Esquema general de los algoritmos de selección de atributos

Los algoritmos de selección de atributos realizan una búsqueda en el espacio de subconjuntos de atributos. Por esta razón, de modo general se puede decir que los algoritmos deben tener (Blum y Langley, 1997):

Punto de comienzo: Se debe establecer un punto (o puntos) de comienzo del espacio de búsqueda. Esto establece de manera implícita una dirección: si la búsqueda comienza por un conjunto vacío, la única dirección posible es hacia adelante; este procedimiento se denomina *forward selection*. Como alternativa, se puede comenzar por el conjunto completo e ir eliminando atributos; este procedimiento de búsqueda se denomina *backward elimination*. También pueden plantearse variantes de estos dos modelos básicos. En la Figura 4.4 podemos ver

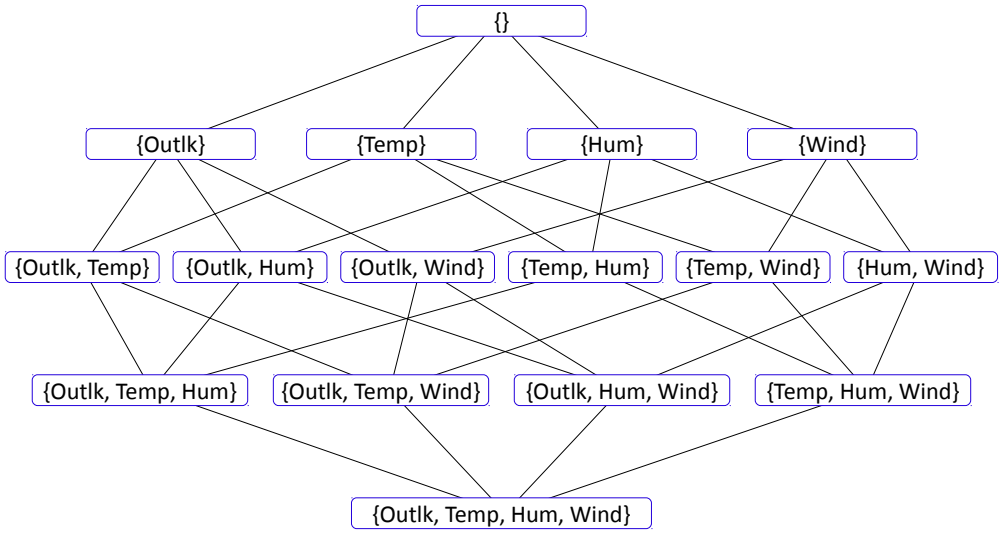

Figura 4.4 – Espacio de subconjuntos de atributos para el conjunto de datos Golf. En este conjunto de datos la clase es la posibilidad de jugar o no, dependiendo de cuatro factores: el pronóstico (Outlk), la temperatura (Temp), la humedad (Hum) y el viento (Wind).

el conjunto de datos "Golf"que se utiliza típicamente para ilustrar los conceptos de esta sección. La dirección hacia adelante (*forward selection*) buscaría de arriba abajo, mientras que la que va hacia atrás (*backward elimination*) haría el recorrido inverso.

Organización de la búsqueda: El espacio de búsqueda se puede recorrer siguiendo distintos criterios. Los tres tipos fundamentales de búsqueda son: exhaustiva, heurística y aleatoria. Se verá con mayor detalle en 4.3.1.

Medida de evaluación: Se debe disponer de algún criterio para medir la bondad del conjunto de atributos seleccionados por el algoritmo. Entre ellas podemos destacar la ganancia de información, la distancia, la dependencia, la consistencia y la precisión. Se verán en 4.3.3.

Criterio de parada: Debe establecerse algún criterio para dejar de buscar en el espacio de posibilidades. Por ejemplo, puede dejar de añadirse o eliminarse atributos cuando no se mejore el merito alcanzado, normalmente la precisión obtenida por algún algoritmo de clasificación; puede seguir buscando subconjuntos mientras la precisión no

se degrade; o, finalmente, puede seguir buscando posibilidades hasta que se encuentra el otro extremo del espacio de búsqueda (si se partió del conjunto vacío, cuando se llegue al conjunto completo; si se partió del completo, cuando se llegue al vacío) y entonces escoger el mejor. Una última posibilidad es que en un parámetro del algoritmo se determine un punto de parada.

4.3.1. Organización de la búsqueda

Evidentemente, la búsqueda exhaustiva es impracticable incluso para conjuntos de datos con relativamente pocos atributos: si m es el número de atributos, el número de posibles subconjuntos es 2^m, por lo que incluso para valores de m en torno a 20–30, unos valores muy realistas, se vuelve impracticable. Por ejemplo, la mediana de los conjuntos de datos con los que hemos realizado la parte experimental de nuestro trabajo es de $28{,}50$, la media, $493{,}76$, y el máximo de 10000. Vamos a centrarnos por tanto en las otras dos estrategias.

Búsqueda heurística

Las búsquedas heurísticas buscan evitar tener que explorar todo el espacio de búsqueda; se examinan muchos menos subconjuntos, por lo que disminuyen el orden de complejidad de la búsqueda exhaustiva, que recordemos que es $O(2^m)$. Las estrategias de búsqueda heurística generalmente conducen a algoritmos de orden $O(m^2)$. Por supuesto, tenemos que tener presente que cualquier búsqueda que no sea exhaustiva no tiene que encontrar necesariamente el subconjunto óptimo, pero generalmente merece la pena porque para conjuntos de datos con muchos atributos se gana mucho en velocidad y se pierde generalmente poco en precisión.

Como hemos mencionado, las búsquedas se producen hacia adelante (*forward selection*) o hacia atrás (*backward elimination*), que se ilustraban en la Figura 4.4, o una mezcla de ambas. En general, las búsquedas son *ávidas* (*greedy*): una vez que se añade o se elimina un atributo, este ya no es considerado nunca más. En cualquier caso, el algoritmo puede considerar todos los cambios locales posibles al conjunto de datos actual (todas las líneas hacia arriba o hacia abajo), o puede elegir el primer cambio que mejora el mérito del conjunto actual. En el algoritmo 4.1 podemos ver el algoritmo de *hill climbing* voraz. Dependiendo de si el conjunto inicial S es el vacío y los hijos de cada nodo son los que se obtienen añadiendo un atributo más al conjunto S o si el inicial son todos los atributos y los hijos

Algoritmo 4.1: Esquema de un algoritmo de *hill climbing* voraz

Input: D: Conjunto de Datos
Output: S: Subconjunto de Atributos
$S \leftarrow$Estado inicial
repeat
  Expandir S haciendo cada cambio local posible
  Evaluar cada hijo T de S
  $S' \leftarrow$ hijo de T con la más alta evaluación $e(T)$
  if $e(S') \geq e(S)$ **then** `/* Mejora al mejor hallado hasta el momento */`
    $S \leftarrow S'$
  end if
until $e(S') < e(S)$

Algoritmo 4.2: Esquema de un algoritmo de búsqueda del primero mejor

Input: D: Conjunto de Datos
Output: S: Subconjunto de Atributos
$S \leftarrow$Estado inicial
$O \leftarrow$ Lista con solo el Estado inicial
$C \leftarrow$ Lista vacía
repeat
  $v = \underset{w \in OPEN}{\text{argmax}}\ e(w)$ `/* Obtiene el estado de O con mejor evaluación */`
  $O.remove(v)$
  $C.add(v)$
  if $e(v) \geq e(S)$ **then** `/* Mejora al mejor hallado hasta el momento */`
    $S \leftarrow v$
  end if
  foreach *(w: hijo de* $v | v \notin O \cup C$*)* **do** O.add(w)
until *S no haya cambiado en las últimas k iteraciones*

son los que se obtienen eliminando un atributo, obtenemos la heurística de *forward selection* o de *backward elimination*, respectivamente.

La búsqueda del primero mejor (*Best First Search*) es una variante del anterior que permite hacer vuelta atrás si se está explorando una rama que resulta ser no prometedora. Como esta estrategia, al usar vuelta atrás, puede explorar todo el espacio de posibilidades, suele incluirse un criterio de parada. El Algoritmo 4.2 (Kohavi y John, 1997) refleja esta estrategia.

Búsqueda aleatoria

Las búsquedas aleatorias realizan un proceso de muestreo al azar o probabilístico. Intentan evitar caer en mínimos locales.

Algunos algoritmos que hacen uso de la búsqueda son LVF (Liu y Setiono, 1996) y Relief (Kira y Rendell, 1992a,b). LVF selecciona subconjuntos de atributos al azar y los valora según una medida de la inconsistencia que muestran con los datos completos.

Algoritmo 4.3: Esquema de un algoritmo genético

Input: D: Conjunto de Datos
Output: x: Individuo que representa un subconjunto de atributos
Genera aleatoriamente una población P
repeat
  foreach $(w \in P)$ **do** Calcula $e(x)$
  Define una distribución de probabilidad p sobre los miembros de P donde $p(x) \propto e(x)$
  repeat
    Selecciona dos miembros x e y según p
    Aplica el operador de cruce a x e y para producir x' e y'
    Aplica mutación a x' e y'
    Inserta x' e y' en P' `/* La siguiente generación */`
  until $|P'| \geq |P|$
  $P \leftarrow P'$
until *No hay más generaciones que procesar*
$x = \underset{x \in P}{\text{argmax}}\, e(x)$

Relief le asigna un peso a cada atributo dependiendo de su capacidad de clasificación. Para ello, selecciona instancias al azar del conjunto de entrenamiento y, para cada uno de ellos, encuentra la instancia más cercana de la misma clase y de la clase opuesta (este algoritmo solo funciona con conjuntos de datos de dos clases). El peso del atributo se determina a partir de su bondad para distinguir la instancia más cercana de su misma clase y de la otra: recibe el peso más alto si diferencia entre las diferentes clases. Después de asignar los pesos, el algoritmo selecciona los atributos que superan un determinado umbral. ReliefF (Kononenko, 1994) es una extensión de Relief que puede manejar conjuntos de datos con más de dos clases.

Dentro de esta heurísitica podemos encuadrar la familia de los algoritmos de selección de atributos basados en algoritmos genéticos. Los algoritmos genéticos usan los principios de la selección natural en la biología (Holland, 1975). Emplean una población de individuos (soluciones) que evolucionan competitivamente. La representación típica del individuo es un array binario (representando valores booleanos) del mismo número de elementos que atributos; cada elemento (bit) representará la ausencia o la presencia en el individuo (subconjunto de atributos) del atributo. En las sucesivas generaciones se aplican aleatoriamente los operadores de mutación (se cambian uno o varios bits del individuo) o cruce (se intercambian dos trozos de dos individuos). Podemos ver un esquema general de este tipo de algoritmo en el Algoritmo 4.3.

4.3.2. Métodos de Filtro (*Filter*), Envolventes (*Wrapper*) y Empotrados (*Embedded*)

Una importante y útil clasificación de los algoritmos de selección de atributos puede realizarse en función de cómo utilizan el algoritmo de clasificación con el que se evalúa la precisión. Según esta clasificación los algoritmos pueden ser de Filtro, Envolventes o Empotrados.

Los métodos denominados empotrados son aquellos en los que el algoritmo de aprendizaje tiene algún mecanismo interno de selección de atributos. El ejemplo más típico son los algoritmos que construyen árboles de decisión, por ejemplo C4.5 (Quinlan, 1990, 1993), que seleccionan los atributos que utilizan para generar el árbol. Por ejemplo, la implementación de Weka del algoritmos C4.5 (J48) aplicada al conjunto de datos Iris da como salida el árbol (en el formato Weka):

```
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   petalwidth > 1.7: Iris-virginica (46.0/1.0)
```

y cuya representación gráfica podemos ver en la figura 4.5 que, como podemos ver, solo utiliza los atributos petalwidth y petallength, que son los seleccionados por prácticamente todos los algoritmos de selección de atributos para este conjunto de datos.

En los métodos de Filtro la selección de los atributos se realiza de forma totalmente independiente del algoritmo de aprendizaje utilizado. En los métodos Envolventes, por el contrario, el algoritmo de selección de atributos utiliza el algoritmo de clasificación para evaluar los subconjuntos de atributos que va generando. Esta distinción aparece en primer lugar en (John y otros, 1994), que fueron los que introdujeron el método envolvente. Aparece a continuación en el exhaustivo estudio de (Langley, 1994). Un esquema de los métodos de filtro y envolventes se puede ver en las figuras 4.6 y 4.7, respectivamente. Los métodos Wrapper suelen ser mucho menos eficientes en tiempo que los basados en Filtro.

4.3.3. Medidas de evaluación de atributos

Para determinar la bondad de un subconjunto de atributos se utilizan diversas medidas. Las más utilizadas (Liu y Motoda, 1998b; Liu y Yu, 2005)

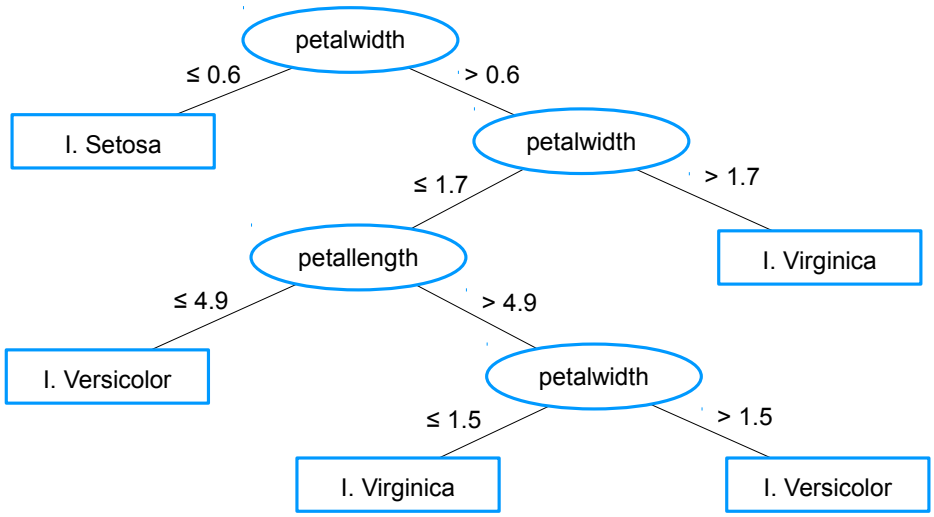

Figura 4.5 – Árbol de decisión para el conjunto de datos Iris.

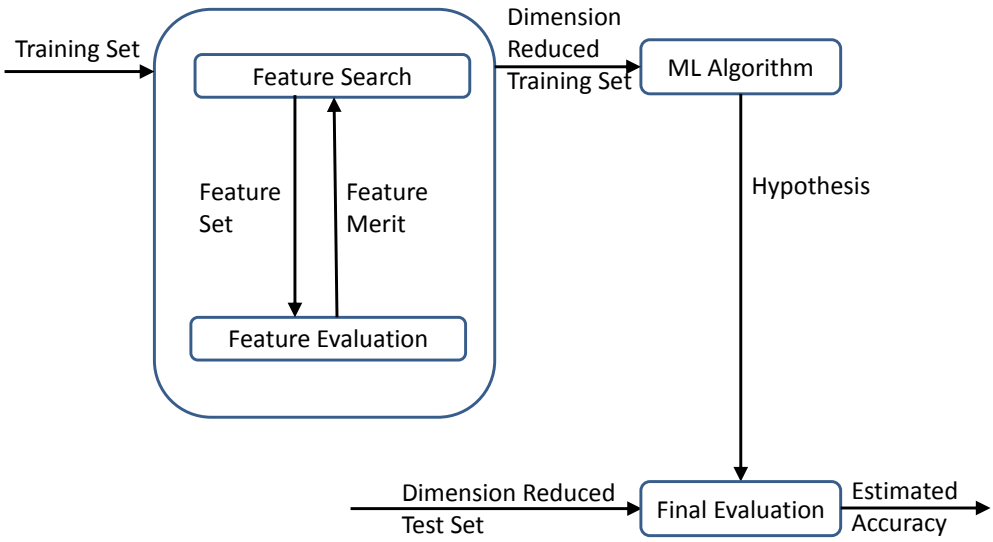

Figura 4.6 – Ilustración de los métodos de Filtro.

son las siguientes:

Medidas de distancia Son medidas de la capacidad de un subconjunto

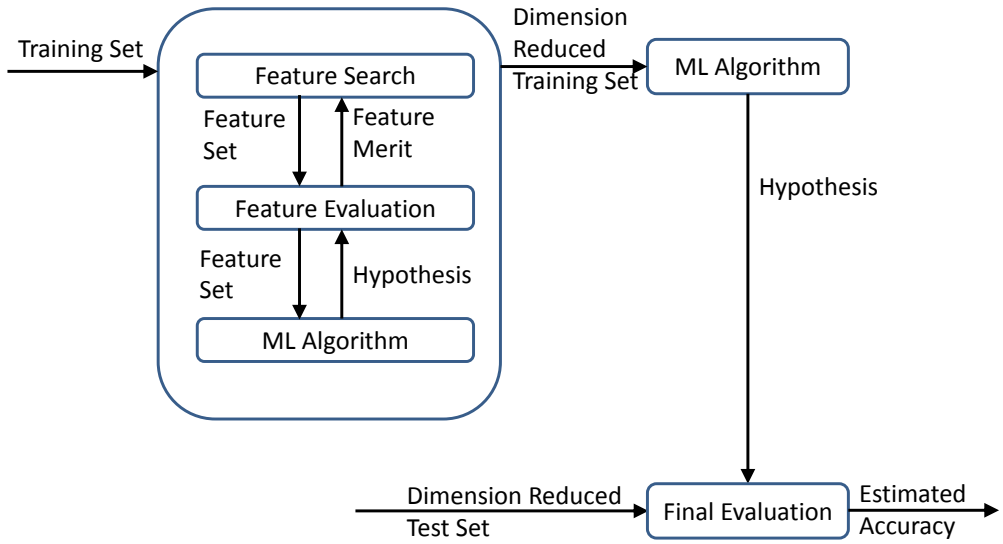

Figura 4.7 – Ilustración de los métodos de Envolvente.

de atributos para separar las clases. Asumiendo que las instancias forman una nube, un buen subconjunto de atributos será aquél que mantiene los elementos de la misma clase próximos entre sí y distantes de los de la otra u otras clases. Se denominan también medidas de separabilidad, de divergencia o de discriminación.

Medidas de información Son las que hemos visto en la subsección 4.2.1. Miden la ganancia de información que se obtiene al utilizar un atributo, esto es, la reducción de la entropía.

Medidas de dependencia También llamadas medidas de correlación o de similitud. Miden la capacidad de predecir el valor de una variable (atributo o clase) a partir de otra (atributo), que hemos visto cuando hablamos de redundancia y relevancia.

Medidas de consistencia Estas medidas buscan el mínimo conjunto de atributos que son capaces de separar las clases tan consistentemente como lo hace el conjunto completo de atributos. Hay inconsistencia si dos instancias tienen el mismo valor de los atributos pero pertenecen a distinta clase; hay que tener especial cuidado con esta medida cuando los datos tienen un identificador (clave) único, puesto que dos instancias distintas siempre diferirán en al menos ese atributo.

Medidas de precisión Es la medida más utilizada cuando el objetivo es la clasificación, puesto que la mejor virtud de un buen clasificador es la capacidad para predecir la clase, esto es, la precisión. Esta es la medida que utilizan especialmente las técnicas *Wrapper*.

4.3.4. Taxonomía de los algoritmos de selección de atributos

En (Liu y Yu, 2005) se presenta una taxonomía de los algoritmos de selección de atributos. En ella los algoritmos se caracterizan según varios criterios: de un lado si son Filtro, Envoltura o Híbridos; para los métodos de Filtro, según el criterio de evaluación: distancia, información, dependencia y consistencia. Según la estrategia de búsqueda: completa, secuencial o al azar. Una última distinción se hace para los métodos que evalúan la calidad según la limpieza de los *clusters* definidos o mediante un clasificador, lo que denominan Tarea de Minería de Datos. Todo esto conduce a una tabla (4.1) en la que se puede ubicar cada algoritmo de selección de atributos.

Tabla 4.1 – Esquema de una taxonomía de los algoritmos de selección de atributos.

			Estrategia de búsqueda					
			Completa		Secuencial		Aleatoria	
Criterio de evaluación	Filtro	Distancia						
		Información						
		Dependencia						
		Consistencia						
	Wrapper	Precisión predictiva o Bondad del cluster						
	Híbrido	Filtro + Wrapper						
			Clasif.	Clust.	Clasif.	Clust.	Clasif.	Clust.
			Tareas de Minería de Datos					

Un estudio muy completo, donde se ubica también cada algoritmo en un punto de una taxonomía, es el de (Ruiz, 2006).

4.4. Algunos algoritmos de selección de atributos

Los algoritmos de selección de atributos tienen una larga trayectoria, quizás desde el trabajo de (Hughes, 1968). Aunque en 1984, en la respuesta a la ponencia de Miller (Miller, 1984) R. L. Plackett dijo que *"If variable elimination has not been sorted out after two decades of work assisted by high-speed computing, then perhaps the time has come to move on to other problems"*, el crecimiento en cantidad y variedad de estos algoritmos ha continuado creciendo, como demuestra el título del trabajo de (Liu y otros, 2010): *"Feature Selection: An Ever Evolving Frontier in Data Mining"*. Por esta razón es muy difícil seleccionar los principales algoritmos de selección de atributos: es un campo fértil en continua evolución. Por ejemplo, en el trabajo de (Liu y Yu, 2005) aparecen 52 técnicas; en la tesis doctoral de Roberto Ruiz (Ruiz, 2006), donde se hace una completísima revisión de las técnicas habidas hasta el momento, quedan reflejadas en ella 64 propuestas, que son, por supuesto, propuestas punteras que han perdurado en el tiempo; hay muchas más en la literatura. Por otra parte, hay muchas más familias de algoritmos de las que hemos descrito en las secciones previas, como los algoritmos no supervisados o semi-supervisados, los híbridos (Filter-Wrapper), los basados en clustering, los basados en conjuntos rugosos, los basados en análisis espectral, etc. Por estas razones resulta muy difícil decidir cuáles elegir; vamos a describir algunos que nos han parecido relevantes por su importancia en la historia de estos algoritmos o por la perspectiva desde la que los abordan.

En nuestra aportación hemos utilizado las correlaciones que propone el algoritmo CFS (Hall y Smith, 1998; Hall, 1999, 2000), por lo que se verá con todo detalle en la subsección 5.2.2 del Capítulo 5.

4.4.1. Branch and Bound

Narendra y Fukunaga (Narendra y Fukunaga, 1977) proponen un método de selección de atributos basado en Ramificación y Poda. Exige que haya monotonicidad en los atributos: si un subconjunto de atributos S_1 esta incluido en otro subconjunto S_2, $S_1 \subset S_2$, se obtiene una mejor clasificación con S_2 que con S_1. Esta hipótesis es muy restrictiva: en muchas ocasiones se obtiene un mejor resultado con un subconjunto de atributos que con todos ellos.

Se da un valor p que es el cardinal del conjunto de atributos que se desea obtener. A continuación se van generando subconjuntos con m, $m-$

1, $m-2$, etc. atributos, hasta el valor p introducido por el usuario. Si una rama es peor que la mejor solución encontrada hasta el momento, usando el principio de monotonicidad, se rechaza, puesto que por ese camino no se encontrará una solución óptima. En la generación del árbol se van obteniendo los nodos en el orden $f_1, f_2, \ldots, f_m$ para evitar duplicidades.

En la Figura 4.8 podemos ver el árbol de posibilidades generado para un problema con seis atributos donde se desea obtener un subconjunto con cuatro. Si por ejemplo, se comprueba que con los nodos $\{f_1, f_2\}$ no se cumple el principio de suboptimalidad, se elimina todo el subárbol que esta debajo de esa secuencia (seis nodos), puesto que por ese camino no se encontrará la solución óptima.

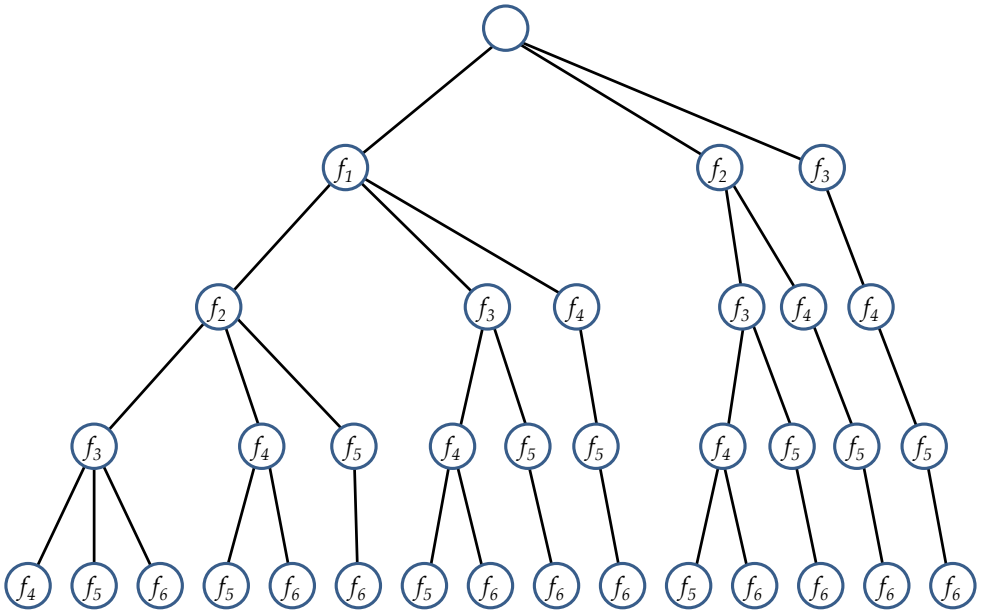

Figura 4.8 – Árbol de posibilidades generado por el algoritmo Branch and Bound para $m = 6, p = 4$.

Con posterioridad al trabajo original se han formulado múltiples variantes de este algoritmo.

4.4.2. Relief

Propuesto por Kira y Rendell (Kira y Rendell, 1992a,b). Está planteado originalmente para conjuntos de datos con dos clases; posteriormente fue extendido por Kononenko (Kononenko, 1994) a conjuntos con más

Algoritmo 4.4: Relief

Input: D: Conjunto de Datos, t: tamaño de muestreo, τ: umbral de relevancia
Output: S: Subconjunto de Atributos

Separa D según la clase de las instancias en dos subconjuntos D^+ y D^- llamados arbitrariamente positivos y negativos
$w = \{0, 0, \ldots, 0\}$ `// W es un array de pesos`
$R = \{0, 0, \ldots, 0\}$ `// R es un array de relevancias`
for $i = 1$ **to** t **do**
  Elige una instancia x al azar de D
  Elige al azar una instancia z^+ de D^+ más cercana a x
  Elige al azar una instancia z^- de D^- más cercana a x
  if $x \in D^+$ **then**
    $nearHit = z^+$
    $nearMiss = z^-$
  else
    $nearHit = z^-$
    $nearMiss = z^+$
  end if
  `// Se actualizan los pesos`
  for $j = 1$ **to** m **do**
    $W_j = W_j - d(x_j, nearHit_j)^2 + d(x_j, nearMiss_j)^2$
    $R_j = (1/t)W_j$
  end for
end for
$S = \emptyset$
for $j = 1$ **to** m **do**
  if $R_i \geq \tau$ **then** `// el atributo es relevante`
    $S = S \cup \{f_i\}$
  end if
end for

clases en el algoritmo ReliefF. Asigna un peso a cada algoritmo y selecciona aquellos que superan un determinado umbral que ha de darse como parámetro.

Supongamos que tenemos p instancias elegidas al azar del conjunto de datos, el criterio de evaluación que utiliza Relief es

$$\mathrm{SC}_R(f_i) = \frac{1}{2} \sum_{t=1}^{p} d(f_{t,i} - f_{NM(x_t),i}) - d(f_{t,i} - f_{NH(x_t),i}),$$

donde $f_{t,i}$ es el valor del atributo f_i en la instancia x_t, $f_{NH(x_t),i}$ y $f_{NM(x_t),i}$ denotan los valores del atributo f_i del punto más cercano a x_t de la misma clase (*near hit*) y de la otra clase (*near miss*) y d es una medida de distancia (utiliza VDM (Wilson y Martinez, 1997)).

El algoritmo Relief, tal como fue formulado en los trabajos originales, puede verse en el Algoritmo 4.4. La complejidad del algoritmo es $O(nm)$, siendo n el número de instancias y m el número de atributos.

Para el problema con varias clases (ReliefF), el criterio es:

$$\mathrm{SC}_R(f_i) = \frac{1}{p}\sum_{t=1}^{p}\left\{-\frac{1}{m_{x_t}}\sum_{x_j\in NH(x_t)} d(f_{t,i}-f_{j,i}) + \sum_{y\neq y_{x_t}}\frac{1}{m_{x_t,y}}\frac{P(y)}{1-P(y_{x_t})}\sum_{x_j\in NM(x_t,y}} d(f_{t,i}-f_{f,i})\right\},$$

donde y_{x_t} es la clase de la instancia x_t y $P(y)$ es la probabilidad de que una instancia pertenezca a la clase y (la frecuencia relativa de la clase). $NH(x)$ y $NM(x,y)$ son ahora el conjunto de los puntos más próximos a x de la misma clase que x o de otras clases, respectivamente; m_{x_t} y m_{x_ty} son los tamaños de los conjuntos $NH(x_t)$ y $NM(x_t,y)$, respectivamente. El tamaño de los conjuntos $NH(x)$ y $NM(x,y)$ es un valor k que se da como parámetro.

4.4.3. FCBF

FCBF (*Fast Correlation Based Filter*) (Yu y Liu, 2003) es considerado uno de los más rápidos y que mayor reducción consiguen. FCBF utiliza la incertidumbre simétrica (*SU*). En primer lugar selecciona un subconjunto de atributos S' que correlacionan con la clase, según la SU, con un valor por encima de un parámetro δ suministrado por el usuario; para ello se hace lo siguiente: se dice que un atributo f_i, con una incertidumbre simétrica con la clase, $SU_{i,c}$, es predominante si $SU_{i,c}\geq\delta$ y no hay ningún otro atributo f_j tal que $SU_{j,i}\geq SU_{i,c}\forall f_j\in S'(j\neq i)$. Si existe algún atributo f_j que sí cumpla la condición ($SU_{j,i}\geq SU_{i,c}$), se dice que f_j es redundante con f_i. El conjunto de atributos redundantes con f_i se denota S_{P_i}, que se descompone en dos subconjuntos $S^+_{P_i}$ y $S^-_{P_i}$ según que $SU_{j,c}>SU_{i,c}$ o $SU_{j,c}\leq SU_{i,c}$, respectivamente.

A continuación se aplican tres heurísticas sobre los conjuntos S_{P_i}, $S^+_{P_i}$ y $S^-_{P_i}$, que se basan en la hipótesis de que si dos instancias son redundantes y una debe eliminarse, hay que optar por eliminar aquella que es menos relevante respecto a la clase, puesto que mantiene más información para predecir esta.

Heurística 1 Se aplica si $S^+_{P_i}=\emptyset$. f_i es un atributo predominante; elimina todos los atributos que estén en $S^-_{P_i}$.

Heuristica 2 Se aplica si $S^+_{P_i}\neq\emptyset$. Se procesan todos los atributos de $S^+_{P_i}$ antes de tomar una decisión sobre f_i: si ninguno de ellos es predominante, se sigue la heurística 1; en caso contrario se elimina f_i y se

Algoritmo 4.5: FCBF

```
   Input: D: Conjunto de Datos, δ: umbral
   Output: S: Subconjunto de Atributos
 1 S' = lista vacía
 2 for i = 1 to m do
 3 |   Calcula SU_{i,c} para f_i
 4 |   if SU_{i,c} ≥ δ then
 5 |   |   Añade f_i al final de S'
 6 |   end if
 7 end for
 8 Ordena S' en orden descendente de valores SU_{i,c}
 9 f_p = getFirstElement(S')
10 repeat
11 |   f_q = getNextElement(S')
12 |   if (f_q ≠ NULL) then
13 |   |   repeat
14 |   |   |   f'_q = f_q
15 |   |   |   if SU_{p,q} ≥ SU_{q,c} then
16 |   |   |   |   elimina f_q de S'
17 |   |   |   |   f_q = getNextElement(S', f'_q)
18 |   |   |   else
19 |   |   |   |   f_q = getNextElement(S', f_q)
20 |   |   |   end if
21 |   |   until F_q == NULL
22 |   end if
23 |   f_q = getNextElement(S', f_q)
24 until f_p == NULL
25 S = S'
```

decide eliminar o no los atributos de $S^-_{P_i}$ basándose en el resto de los atributos de S'.

Heurística 3 Se comienza por aquí: se toma el atributo con el mayor $SU_{i,c}$, que siempre será predominante, por lo que se selecciona y a partir de él se van eliminando el resto de los atributos (o no), siguiendo las heurísticas 1 y 2.

Una descripción completa se da en el Algoritmo 4.5. Este algoritmo tiene una complejidad $O(nm \log m)$.

4.4.4. BIRS

El algoritmo BIRS (*Best Incremental Ranked Subset*) (Ruiz, 2006; Ruiz y otros, 2006, 2008) el un algoritmo de tipo Wrapper que hace una búsqueda muy rápida en el espacio de atributos y que permite el uso de cualquier clasificador. Utiliza como fundamentos teóricos la utilidad incremental (Definición 4.7) y el concepto de *Markov blanket* (Definición 4.14).

Podemos ver su mecánica en el Algoritmo 4.6. El significado de los elementos que aparecen es: U es una medida de la bondad de los atributos

Algoritmo 4.6: BIRS (*Best Incremental Ranked Subset).*

Input: D: Conjunto de Datos, U: medida de evaluación de atributos, L: clasificador
Output: S: Subconjunto de Atributos

```
1  R = Cola de prioridad vacía
2  foreach atributo f_i do
       /* asigna una puntuación al atributo según la medida U:      */
3      punt = calcula(f_i, U, D)
       /* introduce el atributo en la cola de prioridad según su
          puntuación:                                               */
4      añade f_i según la punt
5  end foreach
6  mejorClasif = 0
7  S = ∅
8  for i = 1 to m do
9      f = obtienePrimerElemento(R)  // lo devuelve y lo elimina de la cola
10     subconjuntoTemporal = S ∪ {f_i}
11     valorClasif = WrapperClasif(D, subconjuntoTemporal, L)      /* Evalúa el
       subconjunto */
12     if valorClasif ⩾ mejorClasif then
13         S = subconjuntoTemporal      /* Ese el mejor encontrado hasta el
           momento */
14         mejorClasif = valorClasif                          // Y ese su valor
15     end if
16 end for
```

que permita establecer un ranking. R es una cola de prioridad: en ella estarán los atributos de mayor a menor ranking. La operación de la línea 8 devuelve y elimina el primer elemento de la cola, es decir, el de máxima prioridad, que en este caso es el de mayor ranking. $WrapperClassif$ evalúa según el clasificador L el conjunto de datos utilizando solo los atributos de $subconjuntoTemporal$. El símbolo $\gtrdot$ denota que se produce una mejora significativa; los autores emplean un test de Student para ver si la mejora es estadísticamente significativa (a un nivel de $0{,}1$).

Este algoritmo, pese a ser una aproximación Wrapper consigue una notable velocidad al realizar relativamente pocas evaluaciones. Por ejemplo, en la Tabla 4.2 podemos ver una traza del algoritmo para nueve atributos.

Roberto Ruiz también es autor de una interesante técnica de ranking de atributos denominada SOAP (*Selection of Attributes by Projections* (Ruiz y otros, 2002). Esta técnica se basa en la evaluación de las proyecciones de las instancias de un conjunto de datos sobre cada atributo individual. La idea es la siguiente: para cada atributo de un conjunto de datos, se proyectan las clases de cada atributo sobre el eje que representa ese atributo, contando el número de veces que se cambia de clase, al que denomina *NLC* (*Number of Label Changes*). Por ejemplo, en la figura 4.9 podemos ver los elementos del conjunto de datos Iris representados usando como ejes de coordenadas $sepallength$ y $sepalwidth$. Si imaginamos los elementos por

Tabla 4.2 – Traza de ejecución del algoritmo BIRS. Suponemos que los rangos ordenan los atributos en el orden $f_5, f_7, f_4, f_3, f_1, f_8, f_6, f_2, f_9$.

Subconjunto evaluado	Prec.	p-val	$mejorClasif$	S
$\{f_5\}$	80		80	$\{f_5\}$
$\{f_5, f_7\}$	82			
$\{f_5, f_4\}$	81			
$\{f_5, f_3\}$	83			
$\{f_5, f_1\}$	84	$< 0{,}1$	84	$\{f_5, f_1\}$
$\{f_5, f_1, f_8\}$	84			
$\{f_5, f_1, f_6\}$	86			
$\{f_5, f_1, f_2\}$	89	$< 0{,}1$	89	$\{f_5, f_1, f_2\}$
$\{f_5, f_1, f_2, f_9\}$	87			

ejemplo sobre el eje horizontal ($sepallength$), podemos ver que es muy difícil distinguir los elementos; tan solo hay dos intervalos relativamente limpios, uno para *Iris Setosa* y otro para *Iris Virginica*; el resto está completamente mezclado. Lo mismo ocurre si la proyección se realiza sobre el eje vertical ($sepalwidth$): tan solo hay un intervalo limpio para *Iris Setosa*; el resto está mezclado. Formalmente esto equivaldría a que ambos atributos tienen un *NLC* alto. Sin embargo, si las proyecciones las hacemos sobre

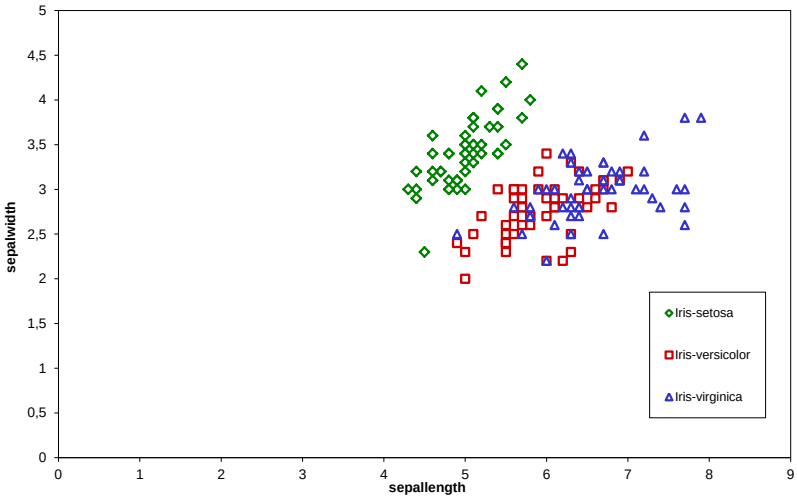

Figura 4.9 – Conjunto de datos Iris representado usando como ejes *sepallength* y *sepalwidth*.

los ejes $petallength$ o $petalwidth$ (figura 4.10) vemos que las proyecciones resultan mucho más limpias: para $petallength$ *Iris Setosa* está totalmente limpia, en *Iris Virginica* se producen algunos cambios de clase en la transición entre *Versicolor* y *Virginica* y luego vuelve a haber solo elementos de *Iris Virginica*. Lo mismo pasa si lo proyectamos sobre $petalwidth$. Esto se traduce en que ambos atributos tienen un *NLC* bajo. Efectivamente, sabemos que $petallength$ y $petalwidth$ son los atributos que se deben elegir para clasificar Iris.

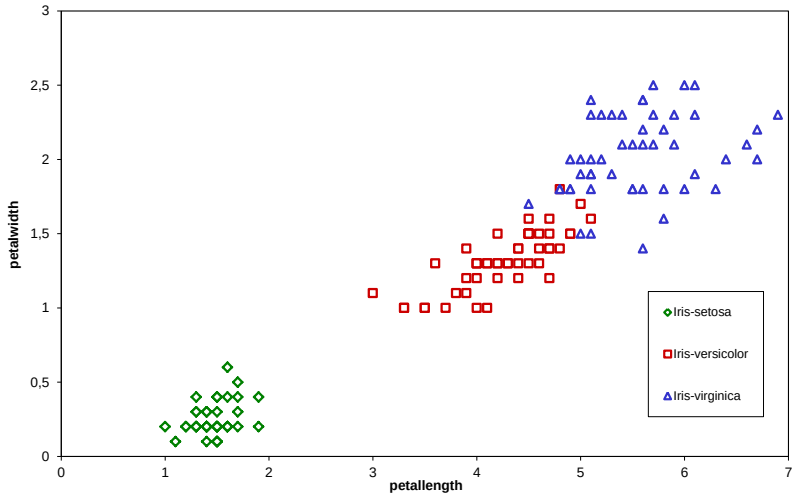

Figura 4.10 – Conjunto de datos Iris representado usando como ejes *petallength* y *petalwidth*.

4.5. Resumen

A lo largo de este capítulo hemos descrito el problema de la selección de atributos encuadrándolo dentro del esquema del proceso de adquisición de conocimientos en Minería de Datos. Hemos analizado los importantes conceptos de Relevancia y Redundancia, presentando algunas definiciones que resultarán de utilidad más adelante. Hemos visto una descripción general de los algoritmos de selección de atributos, deteniéndonos en los métodos de generación de subconjuntos. Hemos visto las distintas maneras de organizar la búsqueda de los subconjuntos de atributos y la distinción entre métodos de Filtro y Envolventes. Hemos visto algunas

medidas de bondad en la evaluación de atributos y hemos finalizado este estudio analizando en profundidad varios algoritmos de la literatura.

Capítulo 5

Nuestra aportación: la técnica MCBF

5.1. Introducción

En este apartado presentamos nuestra aportación en el ámbito de la selección de atributos: el algoritmo MCBF.

En general, las técnicas de selección de atributos intentan dos cosas:

- Quedarse con los atributos más *relevantes*: aquellos que más correlacionan con la clase –que mejor la predicen–, o bien, aquellos que en interacción con otros correlacionan con la clase.
- Eliminar los *redundantes*: aquellos atributos cuyo poder de predicción ya está provisto por otro u otros.

Todas las investigaciones coinciden en que los algoritmos de selección de atributos deben escoger los atributos más relevantes. Aunque muchas investigaciones enfatizan la eliminación de los atributos redundantes, otras advierten sobre el posible perjuicio que esta eliminación puede causar sobre la exclusión de atributos potencialmente relevantes (Liu y otros, 2010; Zhao y otros, 2010).

Elegir el mejor subconjunto de atributos supone explorar todos los posibles subconjuntos de atributos; esto es una tarea de coste 2^m, siendo m

el número de atributos del conjunto de datos, y por tanto nos encontramos ante un problema NP. Si encontramos alguna forma de transformar este problema en un problema de optimización tratable, lo habremos convertido en un problema de orden polinomial, aunque, naturalmente, la solución que encontremos no tiene que ser la óptima puesto que ésta sólo se puede encontrar mediante la búsqueda exhaustiva.

La idea fundamental que nos ha guiado en este trabajo ha sido la de aplicar un algoritmo clásico sobre grafos en el aparentemente lejano campo de la selección de atributos; esto ya lo hemos realizado con éxito en la selección de instancias (Vallejo y otros, 2010). La idea general de nuestro algoritmo, que desarrollaremos en detalle en la sección 5.3 es aplicar el algoritmo min-cut para definir dos conjuntos de atributos: los que seleccionaremos y los que descartaremos. Dado un conjunto de datos generamos un grafo en el que los vértices se corresponden con los atributos del conjunto de datos, al que añadiremos dos más, y que denominaremos s y t. Los pesos de las aristas que unen vértices que representan atributos serán las correlaciones entre estos. Los de las que unen s y t con los vértices anteriores serán unas transformaciones sobre las correlaciones entre los atributos y la clase. Para calcular estas correlaciones utilizamos las definiciones dadas por Hall para su algoritmo CFS (Correlation-based Feature Selection) (Hall, 1999) que veremos con más detalle en la subsección 5.2.2. Una vez construido el grafo, hemos convertido el problema de la selección de atributos en un problema de optimización sobre grafos, que resolvemos mediante el clásico algoritmo max-flow min-cut (Ford y Fulkerson, 1962). Como veremos, los vértices que quedan en la parte del min-cut donde está s formarán un buen subconjunto de los atributos originales. Nuestra técnica tiene además unos parámetros que provocarán que el algoritmo tome más o menos atributos.

Dos de las técnicas muy bien consideradas actualmente en la selección de atributos (Saeys y otros, 2007) son CFS, por la calidad del subconjunto de atributos seleccionados y que veremos con más detalle en la subsección 5.2.2, y FCBF (Fast Correlation-Based Filter Solution) (Yu y Liu, 2003), vista en el capítulo anterior, por su capacidad para trabajar con conjuntos de datos con muchos atributos y el reducido tamaño del subconjunto de atributos seleccionados. En la parte experimental compararemos el tamaño de los conjuntos de atributos seleccionados por nuestra técnica y por estas otras dos; también compararemos la precisión obtenida por nuestra técnica y la del vecino más cercano, así como con las dos técnicas mencionadas.

Veremos a continuación una serie de conceptos previos necesarios para la exposición del funcionamiento de nuestro algoritmo.

5.2. Conceptos previos

5.2.1. Problema del Flujo Máximo / Corte Mínimo

El problema que vamos a formular a continuación ha sido tratado ampliamente en numerosas referencias (Ford y Fulkerson, 1962; Even, 1979; van Leeuwen, 1990; Cormen y otros, 2001). Vamos a ver en primer lugar una descripción informal del problema,y posteriormente lo formalizaremos.

Supongamos que tenemos una red de ferrocarriles que conectan ciudades. A cada ciudad pueden llegar trenes desde más de un lugar y de cada ciudad pueden partir trenes hacia más de un lugar. Cada vía tiene asignada una cierta capacidad (las mercancías o los viajeros que puede transportar). El primer problema que puede plantearse es: dadas dos ciudades cualesquiera, encontrar la máxima cantidad de mercancía (o viajeros) que puede circular desde la primera hacia la segunda. Un segundo problema es averiguar cuál es el menor número de vías que deberían inutilizarse para evitar que fluya ninguna cantidad de mercancía (o viajeros) desde la primera ciudad hasta la segunda. Este segundo problema tiene evidentes tintes bélicos y, efectivamente, estuvo motivado por el análisis de la red de ferrocarriles soviéticos durante la guerra fría (Harris y Ross, 1955) buscando lo que llamaban *cuello de botella*; el gráfico que ilustraba aquel trabajo podemos verlo en la Figura 5.1. Aunque existen algunos precedentes en la literatura soviética de los años 30 y 40 sobre análisis de flujo orientado a la logística del transporte (Schrijver, 2002), y Ford y Fulkerson posteriormente citaron un trabajo de 1941 (Hitchcock, 1941), puede considerarse el escrito de Harris y Ross como el primer artículo en el que se plantea el problema del flujo máximo en redes en su forma actual.

Veamos a continuación una descripción formal. Partimos de un grafo dirigido $G = (V, E)$ formado por un conjunto de vértices V y un conjunto E de artistas, que son pares ordenados (el grafo es dirigido) de vértices. Sean s y t dos vértices distinguidos de V, que denominaremos, respectivamente, fuente y sumidero. Aunque en general no es necesario, como es suficiente para nuestros propósitos vamos a suponer que el grado en entrada de s y el grado de salida de t son cero.

Usamos $u \to v$ para denotar la arista dirigida que va del vértice u al vértice v. En estas condiciones, se define un (s,t)*−flujo* (o simplemente un *flujo* si el nodo fuente y el sumidero están claros por el contexto), como una función $f : E \to \mathbb{R}^+ \cup \{0\}$ que satisface la condición denominada

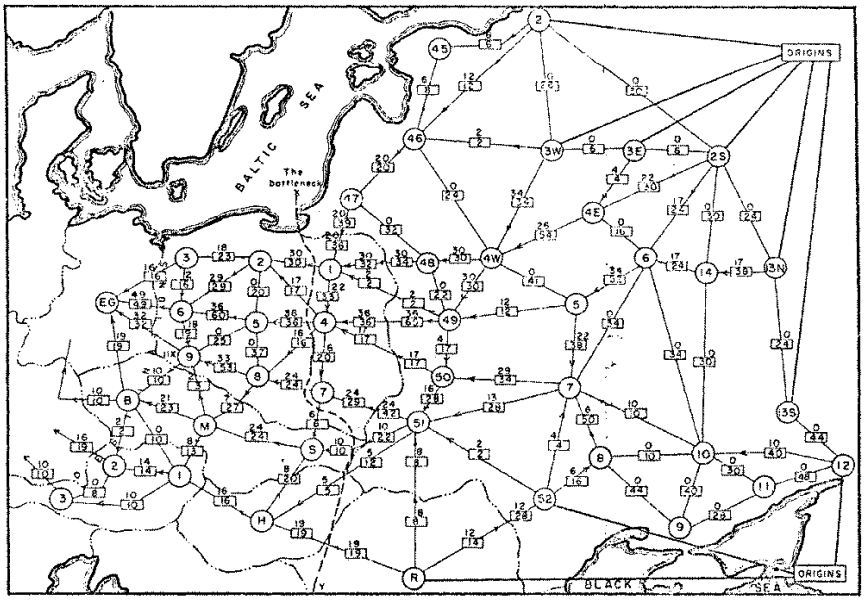

Figura 5.1 – Grafo de 1.955 que modela la red de ferrocarriles rusos. Tiene un flujo máximo (y por tanto un corte mínimo) de 163.000 toneladas. Podemos ver dibujado el corte mínimo indicado como "The bottleneck".

restricción de equilibrio para todos los vértices excepto s y t:

$$\sum_u f(u \to v) = \sum_w f(v \to w).$$

Esto quiere decir que el flujo total que llega a cualquier vértice es igual que el flujo total que sale de ese vértice (asumiendo por simplicidad y sin pérdida de generalidad que $f(u \to v) = 0$ si no hay arista $u \to v$). Supongamos también que puede salir de s una cantidad de flujo tan grande como se quiera y que t es igualmente capaz de absorber cualquier cantidad de flujo que le llegue. El *valor* del flujo f se define como el flujo que sale del nodo fuente, que, evidentemente, es igual que el flujo que llega al sumidero:

$$|f| = \sum_v f(s \to v) = \sum_u f(u \to t).$$

Supongamos que cada arista e tiene asociada una *capacidad* $c(e)$ que es un valor numérico no negativo. Se dice que un flujo f *está sujeto* a la función de capacidad (o es *viable* con la función de capacidad) si $\forall e \in E\ f(e) \leq c(e)$.

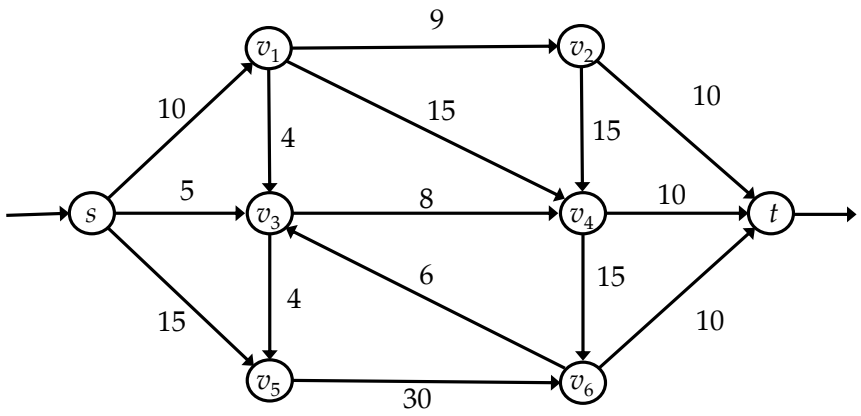

Figura 5.2 – Grafo de ocho vértices en el que las etiquetas indican las capacidades de las aristas.

Flujo máximo

El *problema del flujo máximo* (en inglés *maximum flow problem* o, abreviadamente, *max-flow*) es calcular un $(s,t) - \text{flujo}$ sujeto a la función de capacidad y cuyo valor sea el mayor posible. Por ejemplo, en la figura 5.2 podemos ver un grafo formado por seis vértices. Como hemos dicho, suponemos que a s puede llegar cualquier cantidad de flujo y que, análogamente, t puede absorber todo el flujo que le llegue.

En esta situación, un $(s,t) - \text{flujo}$ para el grafo anterior sería el que se refleja en la figura 5.3. Ahora las aristas reflejan el flujo que corre por ellas junto con la capacidad máxima. Podemos observar, por ejemplo, que al vértice v_3 llegan ocho unidades, tres de ellas procedentes de v_6 y cinco de s; de v_3 salen ocho unidades (se cumple el principio de restricción de equilibrio) en dirección a v_4. A v_6 llegan trece unidades procedentes de v_5 y salen otras trece: tres hacia v_3 y diez hacia t. El valor del flujo en este caso es 28, que es, naturalmente, tanto lo que sale del nodo fuente como lo que llega al nodo sumidero.

Corte Mínimo

Un $(s,t)-$*corte* (o simplemente un *corte* cuando por el contexto esté claro quiénes son los nodos fuente y sumidero) es una partición de los vértices del grafo en dos conjuntos disjuntos S y T ($S \cup T = V$ y $S \cap T = \varnothing$)

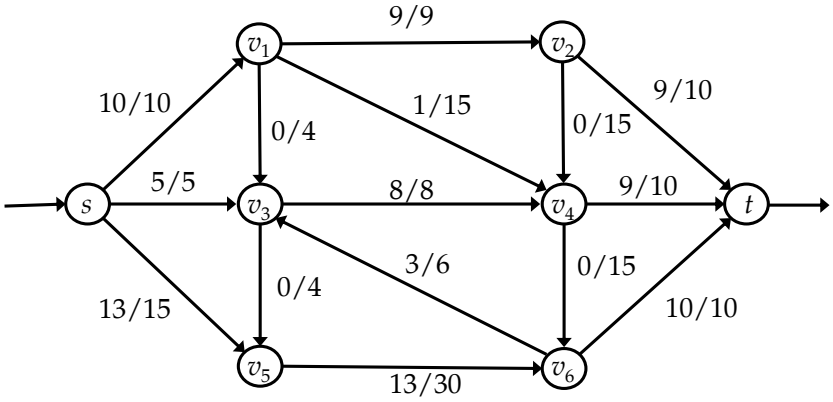

Figura 5.3 – $(s,t)-\text{flujo}$ para el grafo anterior. Las etiquetas de las aristas indican flujo/capacidad.

donde $s \in S$ y $t \in T$.

Si disponemos de una función de capacidad c que asocie a cada arista un coste, $c : E \to \mathbb{R}^+ \cup \{0\}$, el coste de un corte es la suma de las capacidades de las aristas que comienzan en S y terminan en T:

$$\|S,T\| = \sum_{v \in S} \sum_{w \in T} c(v \to w).$$

La definición es asimétrica: las aristas que parten del conjunto T y llegan al conjunto S no cuentan. Por ejemplo, en la figura 5.4 podemos ver un corte en el que $S = \{s\}$ y $T = V - S = \{v_1, v_2, v_3, v_4, v_5, v_6, t\}$ cuyo valor es 30: la suma de las capacidades de las aristas que parten de S y llegan a T; en este caso, $s \to v_1$, $s \to v_3$ y $s \to v_5$. En la figura 5.5 $S = \{s, v_1, v_3, v_5\}$ y $T = \{v_2, v_4, v_6, t\}$; en este caso el valor del corte es 62. Téngase en cuenta que sólo se tienen en cuenta las aristas que van de vértices de S a vértices de T ($v_1 \to v_2$ de capacidad 9, $v_1 \to v_4$ de capacidad 15, $v_3 \to v_4$ de capacidad 8 y $v_5 \to v_6$ de capacidad 30), pero no a la inversa ($v_6 \to v_3$).

El problema del *corte mínimo* (en inglés *minimum cut* o, abreviadamente *min-cut*) es encontrar un $(s,t)-\text{corte}$ cuyo coste sea tan pequeño como sea posible. En la figura 5.6 podemos ver el corte mínimo para el grafo de la figura 5.2. En este caso $S = \{s, v_3, v_5, v_6\}$ y $T = \{v_1, v_2, v_4, t\}$. El valor del corte es 28, la suma de las capacidades de las aristas $s \to v_1$ (10), $v_3 \to v_4$ (8) y $v_6 \to t$ (10). Si probamos todas las combinaciones, veremos que este corte es el menor posible.

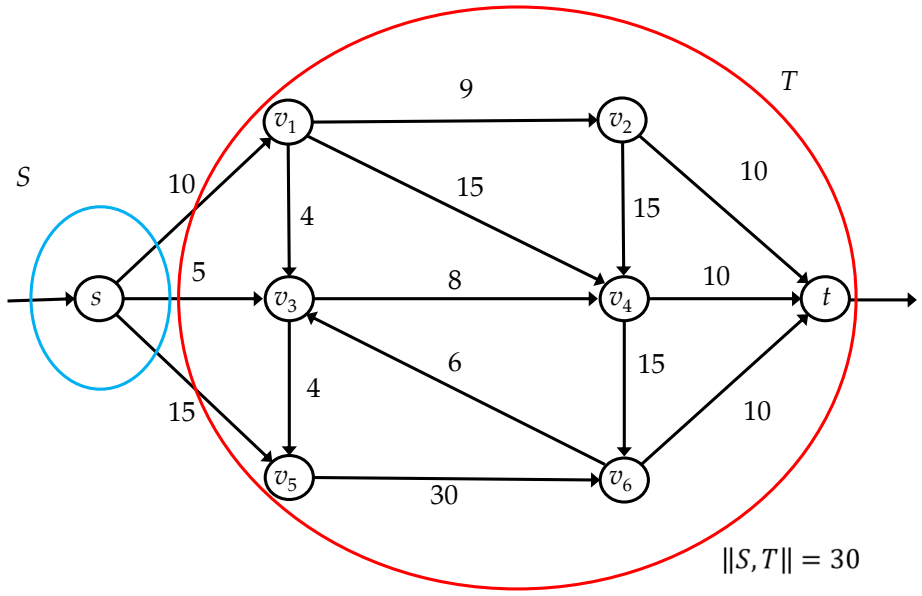

Figura 5.4 – $(s,t) - \text{corte}$ para el grafo anterior. $S = \{s\}$.

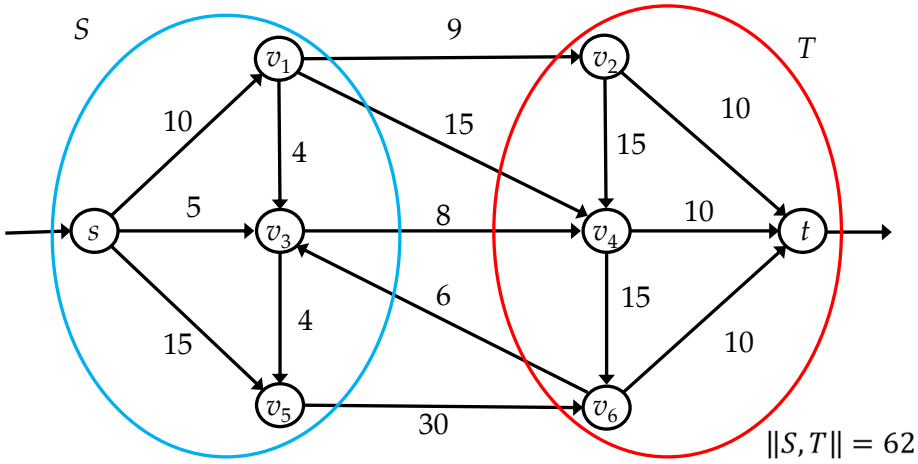

Figura 5.5 – $(s,t) - \text{corte}$ para el grafo anterior. $S = \{s, v_1, v_3, v_5\}$.

Podemos interpretar el corte mínimo como la forma más económica de interrumpir todo el flujo de s a t.

El teorema de Ford-Fulkerson (Ford y Fulkerson, 1956) enuncia que el valor del flujo máximo coincide precisamente con el coste del corte mínimo. En el mismo artículo se da una forma de calcular tanto el flujo máximo

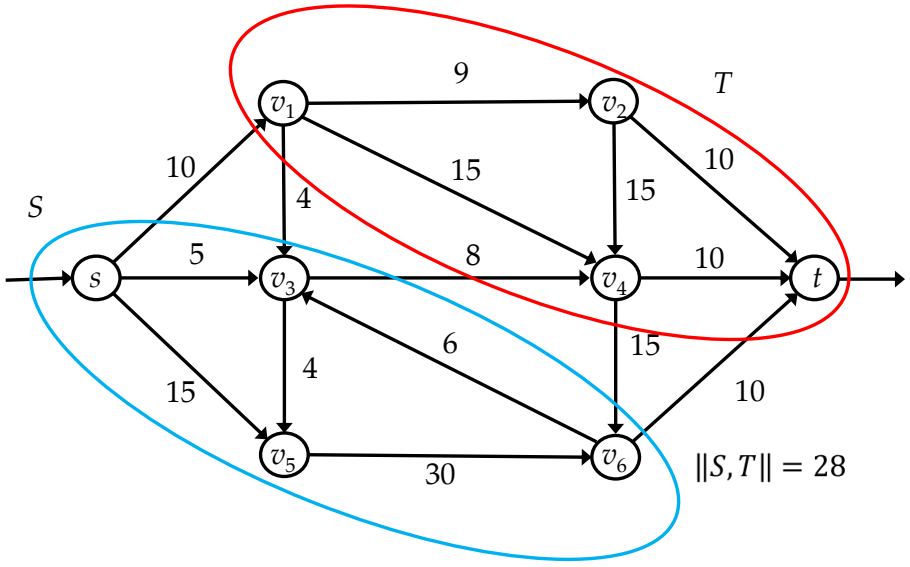

Figura 5.6 – Corte mínimo para el grafo anterior. $S = \{s, v_3, v_5, v_6\}$.

como el corte mínimo. En la figura 5.3 observamos que el flujo máximo en nuestro grafo ejemplo tenía valor 28 y en la figura 5.6 vimos que el corte mínimo tenía precisamente ese valor.

El problema del flujo máximo - corte mínimo tiene la belleza intrínseca inherente a muchos problemas matemáticos: la dualidad entre dos problemas aparentemente distintos. Por otra parte tiene numerosas aplicaciones; por supuesto, todas las relacionadas con encontrar flujos en redes: transportes, suministros, fluidos (gas, electricidad, aguas residuales), paquetes de datos en redes de ordenadores, internet, conectividad, fiabilidad en redes...; pero también tiene aplicaciones no tan obvias, como la computación distribuida (Johnson y otros, 2004), el procesamiento de imágenes (Greig y otros, 1989), el análisis de opiniones en textos subjetivos (Pang y Lee, 2004), la asignación de clases a conjuntos de datos incompletos (Blum y Chawla, 2001), etcétera.

Algoritmos para el cálculo del mincut

El problema del flujo máximo - corte mínimo en grafos ha sido ampliamente estudiado desde el trabajo original de Ford y Fulkerson y se han formulado numerosas soluciones. En principio, hay dos grandes familias de soluciones:

- Las que parten de la idea original de Ford y Fulkerson, basadas en lo

que se denomina el *camino aumentante* (en inglés *aumenting path*).

- Las que parten de una idea surgida poco después de la de Ford y Fulkerson, debida a Edmonds y Karp (Edmonds y Karp, 1972) y desarrolladas por Goldberg y Tarjan (Goldberg y Tarjan, 1988), basadas en lo que se denomina *impulso-reetiquetado* (en inglés *push-relabel*).

La técnica del *aumenting path* se basa en ir disminuyendo la capacidad de las aristas en un determinado valor dado por el flujo encontrado hasta el momento. Esto da lugar a un grave problema: depende de la capacidad máxima de las aristas. Esto se traduce en soluciones muy lentas incluso en grafos relativamente simples, como el reflejado en la Figura 5.7. Se trata de un problema muy sencillo, donde el flujo total es, evidentemente, de dos millones de unidades. Las soluciones del tipo *aumenting path* necesitarían dos millones de iteraciones para resolverlo.

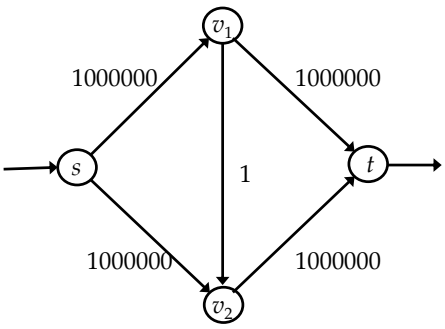

Figura 5.7 – Sencillo problema de flujo máximo / corte mínimo en el que las soluciones del tipo *aumenting path* necesitarían 2.000.000 de iteraciones.

Hay un tercer grupo de algoritmos, de aparición mucho más reciente, que se basan en el método de Monte-Carlo que consiguen un tiempo casi lineal en la mayoría de los casos (Karger y Stein, 1996; Karger, 2000; Brinkmeier, 2009).

En el trabajo de Boykov y Kolmogorov (Boykov y Kolmogorov, 2004) puede verse un completo estudio de casi todos los algoritmos de cálculo del flujo máximo - corte mínimo en redes. En (Chekuri y otros, 1996) se hace una exhaustiva comparativa de todos los algoritmos existentes hasta el momento de su publicación. Podemos comprobar que todos los algoritmos tienen coste polinomial, resultado que tendrá importancia más adelante.

Implementaciones Muchas de las soluciones publicadas para el problema del flujo máximo - corte mínimo han partido del mundo de las matemáticas: muchos trabajos se limitan a enunciar una posible solución, demostrar el orden de complejidad y ...nada más. Esto quiere decir que, aunque, como hemos dicho, hay muchos trabajos publicados con algoritmos que dan solución al problema, hay muy pocos programas implementados que realmente lo solucionen.

En nuestra aportación hemos usado una solución muy eficiente: el algoritmo hi-pr (Cherkassky y Goldberg, 1994), del tipo *push-relabel*. La complejidad de este algoritmo es $O(n^2\sqrt{m})$ siendo $n = |V|$ el número de vértices del grafo (incluidos s y t) y $m = |E|$ el número de aristas. Este algoritmo está escrito en C mientras que el resto del código que hemos desarrollado lo hemos escrito en Java, para hacerlo compatible e integrable con Weka y poder utilizar su API. El algoritmo original recibe el grafo desde un fichero de texto en un determinado formato y devuelve el resultado en la salida estándar; en las versiones preliminares de nuestro algoritmo se transforma el grafo que se genera internamente al formato que necesita hi-pr, captura su salida y la analiza para encontrar el flujo máximo y el corte mínimo. La generación del archivo de texto consumía mucho tiempo, por lo que en versiones posteriores optamos por realizar una traducción del algoritmo a Java, sin utilizar ningún fichero intermedio. Esta solución no es publicable en el momento actual porque el código fuente de hi-pr tiene copyright. Estamos en conversaciones con los autores para encontrar una fórmula que nos permita distribuir nuestro algoritmo con nuestra implementación de hi-pr que satisfaga a ambas partes, o bien, utilizar alguna implementación alternativa que sea abierta.

5.2.2. Correlaciones entre atributos y clases

El término correlación se usa en lo que sigue no en el sentido clásico de las correlaciones lineales, sino del grado de dependencia o predictibilidad de una variable respecto de otra, sean estas clases o atributos.

Las correlaciones que empleamos en nuestro algoritmo son las calculadas por la técnica CFS (Hall y Smith, 1998; Hall, 1999, 2000). El principio que rige la tesis de Mark A. Hall es *"A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other"*. Esta idea se deriva de ideas previas tomadas de la teoría de la medición en psicología (Ghiselli, 1964).

La técnica CFS en primer lugar discretiza los atributos continuos, usando el método de discretización supervisada de Fayyad e Irani (Fayyad y

Irani, 1993), que es una variante de la correlación lineal estándar (o de Pearson).

- Si los dos atributos son continuos es

$$r_{XY} = \frac{\sum xy}{m\sigma_x\sigma_y}, \tag{5.1}$$

  donde X e Y son dos variables continuas expresadas en términos de desviaciones.

- Cuando una variable es continua y la otra discreta se aplica correlación de Pearson ponderada. Si Y es la variable continua y X es la discreta con k valores

  - Se establecen k atributos binarios.
  - Cada uno de los $i = 1, \ldots, n$ atributos binarios toman el valor 1 cuando X toma el valor i-ésimo y 0 en caso contrario.
  - Cada una de las $i = 1, \ldots, n$ correlaciones se ponderan por la probabilidad a priori de que X tome el valor i-ésimo

$$r_{XY} = \sum_{i=1}^{k} p(X = x_i) r X_{bi} Y, \tag{5.2}$$

    donde X_{bi} es el atributo binario, que vale 1 si X vale x_i y 0 en caso contrario.

- Si los dos atributos son discretos se crean los atributos binarios anteriores para los dos y se calculan las correlaciones ponderadas para todas las combinaciones:

$$r_{XY} = \sum_{i=1}^{k} \sum_{j=1}^{l} p(X = x_i, Y = y_j) r X_{bi} Y_{bj}. \tag{5.3}$$

- Los valores perdidos se sustituyen por la media de los valores en los atributos continuos y por el valor mayoritario en los discretos.

Con todo lo anterior se calcula una "matriz de correlaciones", que es la que a nosotros nos interesa para nuestro trabajo. En cualquier caso, para completar la descripción de la técnica, a continuación se define el "mérito"de un subconjunto S de atributos (M_S) con cardinalidad k como

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \tag{5.4}$$

donde $\overline{r_{cf}}$ es la media de la correlación entre atributos y clases, con $f \in S$, y $\overline{r_{ff}}$ es la media de la correlación entre atributos. El numerador de esa expresión puede interpretarse como una indicación de la capacidad de predicción de la clase por un conjunto de atributos, mientras que el denominador es una medida de la redundancia entre los atributos seleccionados. Una vez establecido el ranking entre los atributos mediante la ecuación 5.4, se usa el método voraz del "primero mejor"(*Best First*) (Rich y Knight, 1991) para seleccionar el subconjunto de atributos.

Complejidad del cálculo de la matriz de correlaciones
El cálculo de la matriz de correlaciones, que es la única que nosotros utilizamos, precisa $n((m^2-m)/2$ operaciones, siendo n el número de instancias y m el número original de atributos; es decir, es $O(nm^2)$. La búsqueda de los atributos seleccionados necesita $(m^2 - m)/2$ operaciones.

5.3. El algoritmo MCBF

Nuestra propuesta, el algoritmo MCBF (*Min-Cut Base Feature Selection*), se basa en las ideas apuntadas en las secciones anteriores:

- Creamos un grafo que integre las correlaciones que subyacen al conjunto de datos de modo que el problema de encontrar el mejor subconjunto de atributos se reduzca a un problema de cálculo del flujo máximo en una red.
- Aplicamos el algoritmo del max-flow min-cut sobre ese grafo.
- Ahora los vértices del grafo han quedado separados en dos subconjuntos por el min-cut, que recordemos que es el conjunto de aristas de menor coste con la característica de que si se eliminan se interrumpe totalmente el flujo dentro del grafo.
- El subconjunto que queda un la parte del nodo fuente es el subconjunto de atributos que elegiremos.

Veamos con más detalle el algoritmo empleado, en el que se recibe como entrada un conjunto de datos etiquetados formado por instancias con m atributos $(f_1, f_2, \ldots, f_m)$ y una clase c.

1. A partir del conjunto de datos anterior se construye un grafo dirigido formado por $n + 2$ vértices: n (a los que llamaremos $v_1, v_2, \ldots v_n$) se corresponderán como veremos a continuación con los atributos del conjunto de datos; llamaremos s y t a los dos restantes.

2. Existirá un arco dirigido e_{ij} entre todos los pares de vértices $v_i \rightarrow v_j$. También habrá un arco dirigido entre s y cada uno de los v_i, que denotaremos e_{si} y un arco dirigido entre cada uno los v_i y t, que denotaremos e_{it}.

3. Se calculan las correlaciones entre la clase y cada uno de los atributos: denominaremos c_i a la correlación entre la clase y el atributo f_i. Se calcularán también las correlaciones entre todos los pares de atributos: llamaremos c_{ij} a la correlación entre los atributos f_i y f_j (obviamente, $c_{ij} = c_{ji}$).

4. Se asignan los valores inversos de las correlaciones entre los atributos $(1 - c_{ij})$ a los arcos e_{ij} $\forall i, j = 1 \ldots m, i \neq j$.

5. En principio, asignamos c_i a los arcos e_{si} y $1 - c_i$ a los arcos e_{it}. Como veremos en la subsección 5.4.2, experimentalmente se observó la conveniencia de aplicar unas funciones de transformación a los valores de las aristas de s a los vértices intermedios (e_{si}) y de éstos a t (e_{it}). Llamaremos $\mathrm{f_s}$ a la función de transformación que aplicamos a las primeras aristas (las que parten de s) y $\mathrm{f_t}$ a las últimas (las que llegan a t). El grafo resultante queda reflejado en la Figura 5.8.

6. Se calcula el $(s,t)-\mathrm{cut}$ de este grafo.

7. La interpretación del resultado obtenido es: Si un vértice v_i queda en S le damos el significado "se selecciona el atributo f_i".

Veamos a continuación el algoritmo con más detalle. Denotamos como $e(u, v, w)$ la arista dirigida ponderada que va del vértice u al vértice w con peso w, como $G = (V, E)$ el grafo dirigido ponderado compuesto por el conjunto de vértices V y el conjunto de aristas E; denotamos la matriz de correlaciones entre atributos $\boldsymbol{corr}$: es una matriz de $m \times m$ elementos donde el elemento $corr_{ij}$ denota la correlación entre el atributo f_i y el atributo f_j (o viceversa: es una matriz simétrica); sea, por último el vector $\boldsymbol{corrC}$ donde $corrC_i$ la correlación entre las clases y el atributo f_i; la matriz y el vector son los calculados por CFS; llamamos matCFS a la función que devuelve estas matrices. Con estas consideraciones, el algoritmo es el que vemos en el Algoritmo 5.1.

En la figura 5.9, podemos ver un ejemplo de todo esto aplicado a un pequeño grafo (en la experimentación no se han considerado conjuntos de datos con tan pocos atributos porque no los hemos considerado relevantes en un estudio sobre selección de atributos, pero mostramos este ejemplo con cuatro atributos para que se vea con mayor claridad). A la izquierda

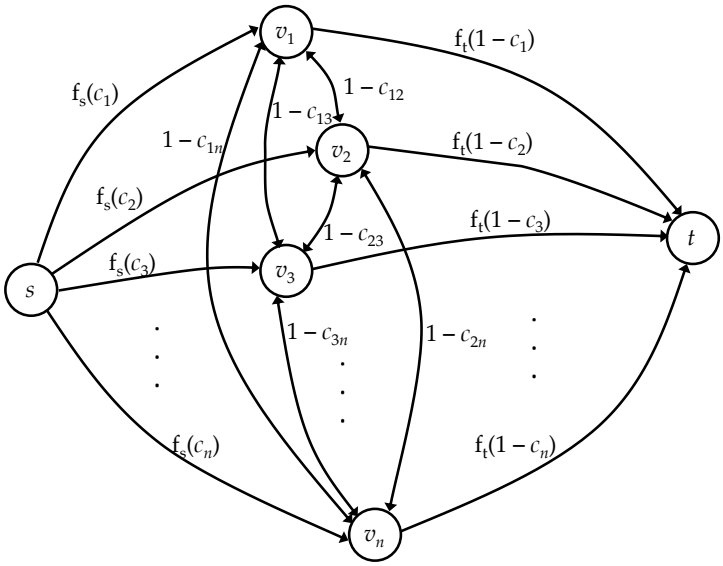

Figura 5.8 – Estructura del grafo generado. Por simplicidad se han representado las aristas entre cada par de vértices intermedios como una arista con dos sentidos. En realidad serán dos aristas; una en cada sentido, con el mismo peso.

podemos ver un grafo en el que los pesos de las aristas representan las capacidades correspondientes. El de la derecha representa el grafo con los mismos vértices, pero en este caso los pesos de las aristas son ahora el flujo máximo que puede circular por cada arco. La línea sinuosa representa el min-cut.

¿Por qué funciona nuestro algoritmo? Como vimos anteriormente, el min-cut representa la forma más económica de cortar el flujo desde s hasta t. Veamos qué significa esto en el grafo de correlaciones que hemos construido.

En nuestro grafo se penalizan tres tipos de aristas:

- Las que indican la selección de atributos que "no correlacionen"fuertemente con la clase, es decir, arcos con poco peso conectados con s (aquellos $v_i \in S$ con c_i bajo o, lo que es lo mismo, con $1 - c_i$ alto).

- Las que indican la "no selección" de atributos que correlacionan fuertemente con la clase, esto es, arcos con poco peso conectados con t (aquellos $v_i \in T$ con c_i alto, o, lo que es lo mismo, con $1 - c_i$ bajo).

- Las selecciones de pares de atributos que correlacionan fuertemente

Algoritmo 5.1: MCBF.

Input: D: Conjunto de Datos, ti: umbral de entrada, to: umbral de salida
Output: S: Subconjunto de Atributos

1 $\langle \boldsymbol{corr}, \boldsymbol{corrC} \rangle = \text{matCFS}(D)$
/* ***corr*** es la matriz de correlaciones entre atributos. ***corrC*** es el vector de correlaciones entre las clases y los atributos */
// A continuación se crea el grafo $G = (V, E)$
2 $V = \{v_1, v_2, \ldots, v_m, s, t\}$
/* El vértice v_i se corresponde con el atributo f_i, $i = 1 \ldots m$; s y t son el nodo fuente y el nodo sumidero, respectivamente */
3 **for** $i = 1$ **to** m **do**
4 | **if** $(corrC_i \geq ti)$ **then**
5 | | $E = E \cup \{e(s, v_i, (m-1) \cdot corrC_i)\}$
6 | **end if**
7 | **if** $((1 - corrC_i) \geq to)$ **then**
8 | | $E = E \cup \{e(v_i, t, (m-1) \cdot (1 - corrC_i))\}$
9 | **end if**
10 | **for** $j = i + 1$ **to** m **do**
11 | | **if** $(i \neq j)$ **then**
12 | | | $E = E \cup \{e(v_i, v_j, corr_{ij})\}$
13 | | | $E = E \cup \{e(v_j, v_i, corr_{ij})\}$
14 | | **end if**
15 | **end for**
16 **end for**
17 $C_s = \text{minCut}(G)$
/* minCut es el algoritmo que calcula el min-cut; C_s es el corte devuelto por el algoritmo que contiene a s */
18 $S = \emptyset$
19 **for** $i = 1$ **to** m **do**
| /* Se añaden los atributos al conjunto de salida */
20 | **if** $(v_i \in C_s)$ **then**
21 | | $S = S \cup \{f_i\}$
22 | **end if**
23 **end for**

entre sí (la información que pueda proporcionar uno ya la puede proporcionar el otro), esto es, los valores de c_{ij} altos.

Estamos, por tanto, ante un problema de optimización. Si se nos permite la licencia claramente comprensible en la notación de llamar c_{vt} a los pesos de las aristas que van de los nodos intermedios a t, c_{sv} a los de las que van de s a los nodos intermedios y c_{uv} a uno menos los pesos de las que van de un vértice intermedio a otro, se trata de minimizar

$$\sum_{v \in S} c_{vt} + \sum_{u \in T} c_{su} + \sum_{u \in S v \in T} c_{uv}$$

que es precisamente es lo que se minimiza en el cálculo del corte mínimo.

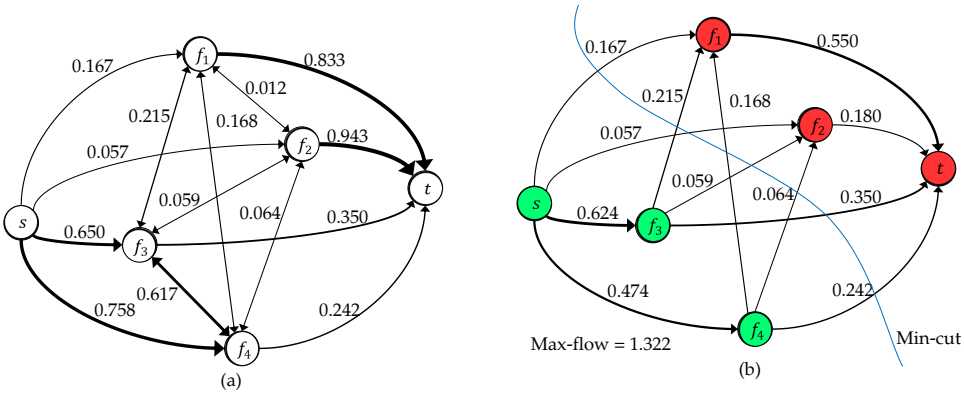

Figura 5.9 – A la izquierda, grafo de partida de ejemplo de un conjunto de datos con cuatro atributos. Las aristas entre los atributos son en realidad dos aristas, una en cada sentido. El grafo de la derecha es el resultante de aplicar el algoritmo max-flow min-cut. El min-cut se muestra por la línea sinuosa. Los atributos f_3 y f_4 quedan en el lado del nodo fuente; por tanto, son los seleccionados.

Como hemos dicho, con el fin de modular la influencia de las capacidades de las aristas desde el nodo fuente hasta los intermedios y desde éstos hasta el sumidero con los pesos de las aristas entre los nodos intermedios, incorporamos unas funciones que transforman los valores de los pesos de las aristas que van de s a los vértices intermedios y de éstos a t. Los valores de estos pesos se determinaron en la fase experimental y se verán en la subsección 5.4.2.

Clasificación del algoritmo MCBF

En la clasificación de (Kohavi y John, 1997) nuestra técnica es del tipo Filter. En cuanto a la clasificación Forward/Backward, no es ni de un tipo ni del otro: escoge en un solo paso los atributos que formarán parte de la selección.

En la clasificación del estudio de (Liu y Yu, 2005) nuestro algoritmo estaría ubicado entre los algoritmos de tipo Filter, usando medidas de dependencia, aunque no podría ser integrado en ninguna de las tres estrategias de búsqueda que se plantean.

Complejidad del algoritmo

La parte más compleja del algoritmo completo, y por tanto la que marca la complejidad total de éste, es el cálculo del min-cut. Como hemos indicado, hemos utilizado la implementación de (Cherkassky y Goldberg, 1994),

cuya complejidad es $(O(n^2\sqrt{m})$ siendo $n = |V|$ y $m = |E|$. En nuestro caso $m = n(n+1)$: n aristas de s a los v_i, otras n de los v_i a t y $n(n-1)$ entre las v_i y v_j, una en cada sentido. Por tanto, la complejidad de nuestro algoritmo es $O(n^2\sqrt{m}) = O(n^2\sqrt{n(n+1)}) \simeq O(n^3)$ (n es aquí el número de atributos del conjunto de datos), lo que se corresponde con los valores observados experimentalmente.

5.4. Experimentación

5.4.1. Bases de datos utilizadas y criterios de comparación

Para conseguir un resultado estadístico que corroborara la validez de nuestro algoritmo, experimentamos sobre treinta y ocho conjuntos de datos: 35 de los datasets fueron tomados de UCI (Frank y Asuncion, 2010), todos ellos con 10 o más atributos; los de menos no los consideramos interesantes para un estudio sobre reducción de atributos. También se utilizaron tres conjuntos de datos (Arcene, Gisette y Madelon) que fueron propuestos en la competición sobre selección de atributos NIPS 2003 (Thrun y otros, 2004). En la tabla 5.1 Se muestra el número de instancias y el número de atributos de los conjuntos de datos utilizados, junto con el número de clases de cada uno de ellos.

5.4.2. Ajuste de parámetros

En la implementación se normalizaron los valores de las correlaciones con la clase, es decir, los pesos de las aristas que van del nodo fuente a los vértices intermedios de manera que la media fuera 0.5. Como hemos mencionado anteriormente, se aplicaron unas transformaciones a los pesos de las aristas que van del nodo fuente a los nodos intermedios y de éstos al sumidero. Por otra parte, observamos experimentalmente que se obtenían mejores resultados cuando se equilibraba las capacidades de los nodos que van del fuente a los intermedios (que una vez normalizados sumaban lo mismo que los que van de los intermedios al sumidero) con las capacidades de las aristas que unen nodos intermedios: en un grafo que represente un conjunto de datos con n atributos, hay n aristas del fuente a los intermedios (y de éstos al sumidero) y hay $n(n-1)$ aristas entre los intermedios. Por esta razón multiplicamos los pesos de las aristas no intermedias por $n-1$.

También observamos experimentalmente que se obtenían mejores resultados si se eliminaban directamente las aristas del fuente a los inter-

Tabla 5.1 – Características de los conjuntos de datos usados en la experimentación. Número de instancias, atributos y clases.

Conjunto de datos	nº de instancias	nº de atributos	nº de clases
Ads	3279	1558	2
Anneal.ORIG	898	38	6
Arcene	100	10000	2
Autos	205	25	7
Colic	368	22	2
Colic.ORIG	368	27	2
Credit-g	1000	20	2
Cylinder-band	540	39	2
Flags	194	29	8
Gisette	6000	5000	2
Heart (Cleveland)	303	13	2
Heart (Hungarian)	294	13	2
Heart (Long Beach VA)	200	13	2
Heart (Statlog)	270	13	2
Heart (Swiss)	123	13	2
Hepatitis	155	19	2
Image segmentations	2310	19	7
Ionosphere	351	34	2
Led creator + 17	10000	24	10
Madelon	2000	500	2
Multifeatures	2000	649	10
Mushroom	8124	22	2
Musk2	6598	166	2
Optdigits	5620	64	7
Page blocks	5473	10	5
Promoters	106	57	2
Schizo	340	14	2
Sick	3772	29	2
Solar flare 1	323	12	2
Solar flare 2	1066	12	3
Sonar	208	60	2
Spambase	4601	57	2
Splice	3190	60	3
Sponge	76	45	3
Vehicle	846	18	4
Vote	435	16	2
Waveform	5000	40	3
Wine	178	13	3

medios que indicaban bajas correlaciones con la clase, y sus equivalentes entre los intermedios y el sumidero. Para esto establecimos dos umbrales que denominamos input threshold (it) y output threshold (ot).

- En la parte del nodo fuente, si la correlación de un atributo (recordemos que estaban normalizadas) superaba it se incorporaba la arista

multiplicada por $n-1$; en caso contrario, no se incorporaba la arista.

- En la parte del nodo sumidero, si uno menos la correlación del atributo superaba ot se incorporaba la arista multiplicada por $n-1$; en caso contrario, no se incorporaba la arista.

En casos sencillos, como Iris, esto significa algo muy obvio: los arcos que unen s con los atributos que correlacionan con la clase (en Iris, f_3 y f_4) tienen capacidad distinta de cero, al igual que los arcos que unen f_1 y f_2 con t. Evidentemente, el min-cut estará formado por f_3 y f_4. Hasta aquí el resultado puede parecer trivial, pero ¿qué ocurre cuando hay muchos atributos? Nuestro algoritmo descarta a priori aquellos atributos que no correlacionan con la clase, pero las correlaciones entre los atributos pueden provocar que en el conjunto final de atributos determinado por el min-cut final se incluyan algunos de los descartados inicialmente o se eliminen algunos de los que inicialmente eran candidatos.

Para ajustar los valores de los umbrales realizamos miles de ejecuciones sobre todos los conjuntos de datos de experimentación con distintos valores de it y ot, los mismos para todos los conjuntos, de modo que los valores que obtuviéramos no fueran específicos de un conjunto determinado, sino que fueran lo más genéricos posible. Llegamos finalmente a unos valores de $it = 0{,}52$ y $ot = 0{,}49$.

5.5. Resultados experimentales y análisis estadístico

Es difícil conseguir un equilibrio entre la reducción del tamaño del conjunto de atributos seleccionados y la obtención de la mayor precisión posible. Las figuras 5.10, 5.11 y 5.12 expresan gráficamente este compromiso. En ellas podemos observar la precisión y el porcentaje de atributos seleccionados por nuestro algoritmo comparado con el resto de los algoritmos analizados. Cada uno de los puntos representa el resultado obtenido para un conjunto de datos.

El eje horizontal representa la diferencia, en porcentaje sobre el número total de atributos, de los atributos seleccionados por nuestra técnica menos los seleccionados por la técnica correspondiente; esto significa que cuanto más a la izquierda se sitúa el punto, más reducción obtiene nuestra técnica. Como podemos observar, MCBF está bastante equilibrado con CFS pero obtiene peores resultados frente a FCBF, un algoritmo que es conocido por

obtener unas grandes tasas de reducción. En la gráfica en la que se compara con 1NN, naturalmente todos los puntos están en la izquierda del 0, puesto que 1NN no tiene ninguna reducción.

El eje vertical representa diferencia, en porcentaje, entre la precisión obtenida con los atributos seleccionados por nuestra técnica menos la obtenida por la técnica correspondiente. Esto significa que si el punto está por encima del eje horizontal nuestra técnica obtiene mejor precisión que la otra. Como podemos ver, nuestra técnica supera ampliamente a CFS y a FCBF, manteniéndose más equilibrada con 1NN en cuanto al número de puntos, aunque los puntos en la parte positiva están más alejados del eje que los puntos en la parte negativa. Queremos enfatizar el significado de la aparición de un punto en el cuadrante inferior derecho: significa que nuestra técnica es peor tanto en precisión como en reducción que la técnica con la que estamos comparando; pues bien, podemos observar que esto solo ocurre en 5 de los 38 conjuntos de datos en la comparación con CFS (incluyendo los que están sobre alguno de los ejes) y en 5 en la comparación con FCBF. En la comparación con el vecino más cercano, evidentemente, no hay ninguno. El resto de los casos está en los demás cuadrantes, en los que los puntos significan que nuestra técnica es mejor en precisión, en reducción, o en ambas medidas. En cualquier caso, esto simplemente da una idea visual de las prestaciones de nuestra técnica. Procedemos analizar con mayor exactitud los resultados de MCBF frente a los otros algoritmos en las siguientes subseciones, haciendo un análisis estadístico riguroso.

5.5.1. Precisión

La tabla 5.2 muestra los valores de las precisiones obtenidas por las técnicas que hemos comparado. Comparada con la técnica del vecino más cercano (debemos recordar que esta técnica no produce ninguna reducción en el número de atributos), MCBF obtiene mejor precisión en 25 de los conjuntos de datos, empatando en otros 3. Esto muestra que MCBF tiene un gran poder de edición.

Hemos realizado un análisis estadístico de estos resultados para poder valorar objetivamente la precisión del algoritmo MCBF. Hemos comparado los resultados de todas las técnicas usando el test de Friedman, que es el método adecuado para comparar las prestaciones de una nueva técnica frente a otras sobre diferentes conjuntos de datos (Demšar, 2006; García y Herrera, 2008). El test de Friedman ($\chi^2_F = 19{,}583$, $df = 3$, $p < 0{,}001$) muestra que hay diferencias significativas entre las precisiones de las técnicas analizadas (con $\alpha = 0{,}05$). Los rangos promedios del test de Friedman se

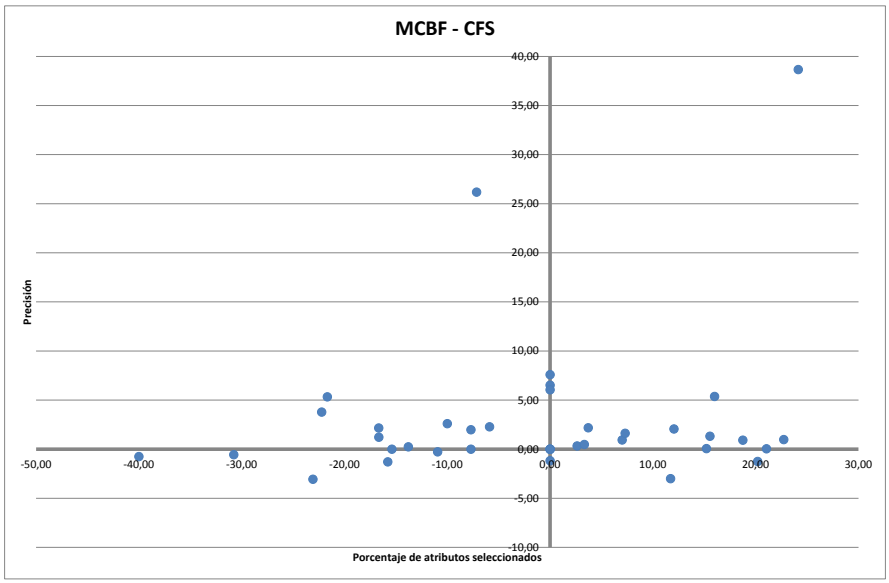

Figura 5.10 – Equilibrio precisión-tamaño: Comparación entre MCBF y CFS. El eje vertical representa la diferencia de la precisión, en porcentaje, obtenida con los atributos seleccionados por nuestra técnica y CFS; si un punto está sobre el eje horizontal se interpreta como que MCBF tiene mayor precisión. El eje horizontal representa la diferencia del porcentaje de atributos seleccionados, de modo que cuanto más a la izquierda esta un punto, mejor reducción ha obtenido MCBF.

muestran en la Tabla 5.3.

Las diferencias son significativas por lo que, siguiendo las indicaciones de Demšar (2006) y García y Herrera (2008), hemos efectuado un análisis post-hoc (Tabla 5.3). El significado de las columnas es el siguiente: "Avr. Rank" es el rango de Friedman, "$R_i - R_{MCBF}$" es la diferencia entre el rango de Friedman de la técnica correspondiente y el rango de Friedman de MCBF, "z"$= (R_i - R_{MCBF})/\sqrt{\frac{k(k+1)}{6N}}$ (donde $k = 4$ es el número de técnicas y $N = 38$ el número de conjuntos de datos), "p(uni)" es el p-value unilateral para z. De acuerdo con la corrección de Bonferroni-Dunn, las diferencias son significativas si p es menor que $\alpha/(k-1)$, esto es 0.0166667 para $\alpha = 0{,}05$. Podemos observar que $p(uni)$ está por debajo de ese valor ($p = 0{,}00184$ para 1NN, $p = 0{,}00002$ para FCBF y $p = 0{,}01409$ para CFS), lo que significa que las diferencias son estadísticamente significativas. Los

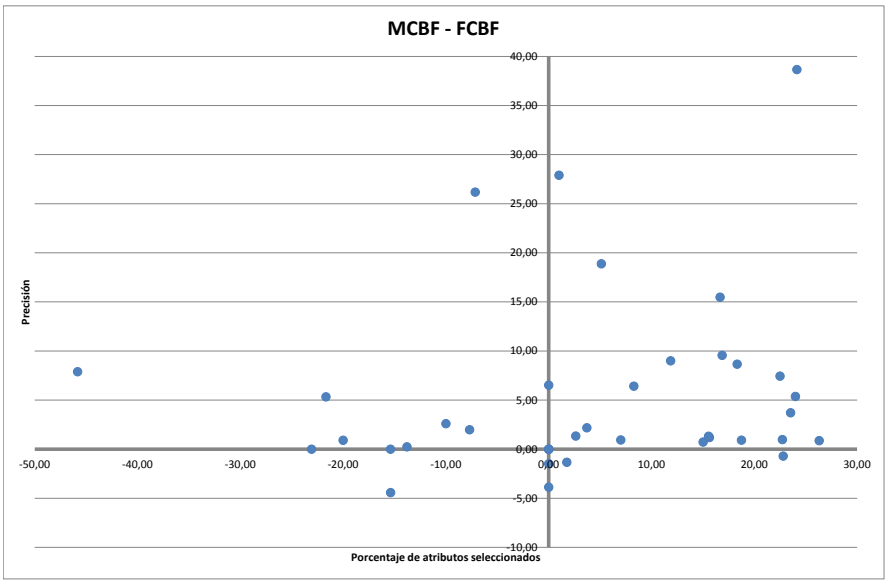

Figura 5.11 – Equilibrio precisión-tamaño: Comparación entre MCBF y FCBF. El eje vertical representa la diferencia de la precisión, en porcentaje, obtenida con los atributos seleccionados por nuestra técnica y FCBF; si un punto está sobre el eje horizontal se interpreta como que MCBF tiene mayor precisión. El eje horizontal representa la diferencia del porcentaje de atributos seleccionados, de modo que cuanto más a la izquierda esta un punto, mejor reducción ha obtenido MCBF.

rangos del test de Friedman de MCBF son mayores que los de las otras tres técnicas (recordemos que, en precisión, mayor es mejor), por lo que podemos afirmar que la precisión obtenida por MCBF es significativamente mejor que las obtenidas por las otras tres técnicas.

5.5.2. Reducción

La Tabla 5.4 muestra el tamaño de los conjuntos de atributos seleccionados por cada técnica en los conjuntos de datos. El test de Friedman aplicado a los valores de los tamaños ($\chi_F^2 = 81{,}780$, $df = 3$, $p < 0{,}001$) muestra diferencias significativas entre los resultados de los algoritmos. Los rangos medios de Friedman se muestran en la Tabla 5.5.

Siguiendo las recomendaciones de Demšar (2006) y García y Herrera

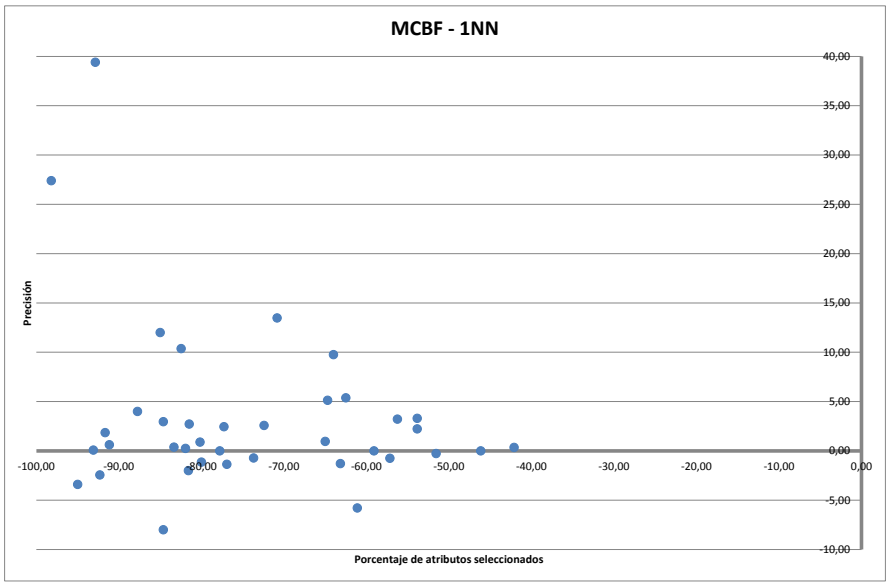

Figura 5.12 – Equilibrio precisión-tamaño: Comparación entre MCBF y 1NN. El eje vertical representa la diferencia de la precisión, en porcentaje, obtenida con los atributos seleccionados por nuestra técnica y 1NN; si un punto está sobre el eje horizontal se interpreta como que MCBF tiene mayor precisión. El eje horizontal representa la diferencia del porcentaje de atributos seleccionados, de modo que cuanto más a la izquierda esta un punto, mejor reducción ha obtenido MCBF. El porcentaje de atributos seleccionados por 1NN es siempre 100 %, obviamente.

(2008), hemos realizado también el análisis post-hoc, (Tabla 5.5). El significado de las columnas es el mismo que el de la tabla anterior.

Recordemos que de acuerdo con la corrección de Bonferroni-Dunn las diferencias son significativas si p es menor que $\alpha/(k-1)$, esto es, 0.0166667 para $\alpha = 0{,}05$. MCBF es significativamente mejor que 1NN ($p < 0{,}001$, y el rango de Friedman para 1NN es mayor que MCBF; en reducción, valores más bajos significan mejores resultados). Las diferencias con las otras dos técnicas no son significativas ($p = 0{,}01670$ para FCBF y $p = 0{,}50000$ para CFS).

Tabla 5.2 – Precisión de todas las técnicas frente MCBF. La tabla muestra la precisión alcanzada con cada técnica, expresada como el porcentaje de aciertos sobre el conjunto de test. Se ha usado validación cruzada estratificada de 10 pliegues.

Conjunto de datos	1NN	FCBF	CFS	MCBF
Ads	95.91	96.07	96.74	96.80
Anneal.ORIG	95.43	92.09	93.10	93.43
Arcene	81.00	76.00	88.00	85.00
Autos	74.63	79.02	79.02	84.39
Colic	80.16	76.09	76.09	82.61
Colic.ORIG	82.61	83.15	83.15	85.33
Credit-g	72.40	66.40	66.40	69.00
Cylinder-band	76.85	60.93	72.22	79.81
Flags	57.73	21.65	21.65	60.31
Gisette	95.75	89.95	94.75	96.37
Heart (Cleveland)	75.25	76.57	76.57	78.55
Heart (Hungarian)	78.23	81.29	79.93	76.87
Heart (Long Beach VA)	71.00	63.00	63.00	63.00
Heart (Statlog)	74.44	78.15	76.67	76.67
Heart (Swiss)	91.87	89.43	89.43	89.43
Hepatitis	81.94	84.52	81.94	80.65
Image segmentations	97.01	96.49	97.32	97.36
Ionosphere	86.61	88.03	89.46	91.74
Led creator + 17	51.29	56.88	63.55	64.77
Madelon	56.50	56.00	77.85	83.90
Multifeatures	98.00	97.95	98.50	97.25
Mushroom	100.00	99.02	99.02	100.00
Musk2	95.85	86.54	94.04	96.10
Optdigits	98.72	97.26	98.72	98.45
Page blocks	95.91	93.86	95.52	94.77
Promoters	79.25	88.68	88.68	89.62
Schizo	60.59	73.82	73.82	100.00
Sick	96.29	96.13	96.13	96.37
Solar flare 1	95.67	97.52	95.36	97.52
Solar flare 2	99.04	99.44	99.44	99.44
Sonar	85.58	77.88	86.06	86.54
Spambase	90.83	91.44	91.28	90.11
Splice	75.05	81.72	81.72	87.05
Sponge	94.74	93.42	93.42	94.74
Vehicle	70.09	48.82	60.52	64.30
Vote	91.72	94.02	94.02	94.94
Waveform	73.82	71.76	79.20	79.20
Wine	96.07	96.07	96.63	96.07
Media	83.52	81.50	83.92	86.66

Tabla 5.3 – Rangos del test de Friedman para las precisiones.

	Avg. Rank	$R_i - R_{MCBF}$	z	p (uni)
1NN	2.32	-0.86	-2.90369	0.00184
FCBF	1.97	-1.21	-4.08543	0.00002
CFS	2.53	-0.65	-2.19465	0.01409
MCBF	3.18	0.00	0.00000	

5.6. Resumen

Hemos visto en la sección precedente que el test de Friedman seguido del análisis post-hoc muestran que MCBF es significativamente mejor en precisión que los demás algoritmos analizados.

En términos de reducción, no hay diferencias significativas entre MCBF y los otros dos algoritmos de selección de atributos, aunque sí que la hay, como era de esperar, entre MCBF y 1NN, naturalmente, favorable para MCBF.

La precisión de nuestro algoritmo es notoriamente mayor que la del vecino más cercano (86.66 de media frente a 83.52, además de que, como hemos visto, las diferencias son estadísticamente significativas). Esto quiere decir que MCBF elimina atributos que empeoran la clasificación, bien porque aporten ruido, bien porque estén mal etiquetados, etcétera.

Tabla 5.4 – Tamaño de los conjuntos de atributos seleccionados por todas las técnicas y MCBF; los valores son la razón entre el número de atributos seleccionados respecto al total, expresado en porcentaje. Esto significa que a menor valor, mayor capacidad de reducción. Se ha usado validación cruzada estratificada con 10 pliegues.

Conjunto de datos	1NN	FCBF	CFS	MCBF
Ads	100.00	4.81	4.62	19.83
Anneal.ORIG	100.00	15.79	15.79	18.42
Arcene	100.00	0.39	0.53	12.25
Autos	100.00	12.00	20.00	36.00
Colic	100.00	22.73	22.73	22.73
Colic.ORIG	100.00	14.81	14.81	18.52
Credit-g	100.00	15.00	15.00	5.00
Cylinder-band	100.00	10.26	15.38	15.38
Flags	100.00	3.45	3.45	27.59
Gisette	100.00	0.56	1.54	8.84
Heart (Cleveland)	100.00	53.85	53.85	46.15
Heart (Hungarian)	100.00	38.46	46.15	23.08
Heart (Long Beach VA)	100.00	15.38	15.38	15.38
Heart (Statlog)	100.00	46.15	53.85	46.15
Heart (Swiss)	100.00	23.08	23.08	7.69
Hepatitis	100.00	36.84	52.63	36.84
Image segmentations	100.00	31.58	36.84	57.89
Ionosphere	100.00	11.76	41.18	35.29
Led creator + 17	100.00	75.00	45.83	29.17
Madelon	100.00	0.80	1.80	1.80
Multifeatures	100.00	20.03	22.65	42.84
Mushroom	100.00	18.18	18.18	40.91
Musk2	100.00	1.20	6.02	18.07
Optdigits	100.00	32.81	59.38	48.44
Page blocks	100.00	40.00	60.00	30.00
Promoters	100.00	10.53	10.53	17.54
Schizo	100.00	14.29	14.29	7.14
Sick	100.00	20.69	20.69	6.90
Solar flare 1	100.00	8.33	25.00	8.33
Solar flare 2	100.00	16.67	16.67	16.67
Sonar	100.00	16.67	31.67	35.00
Spambase	100.00	24.56	26.32	26.32
Splice	100.00	36.67	36.67	15.00
Sponge	100.00	6.67	6.67	22.22
Vehicle	100.00	22.22	61.11	38.89
Vote	100.00	25.00	25.00	43.75
Waveform	100.00	15.00	37.50	37.50
Wine	100.00	76.92	84.62	53.85
Media	100.00	22.08	27.56	26.26

Tabla 5.5 – Rangos del test de Friedman y análisis post-hoc para los valores de los tamaños de conjuntos de atributos seleccionados.

	Avg. Rank	$R_i - R_{MCBF}$	z	p (uni)
1NN	4.00	1.79	6.04374	$< 0{,}00001$
FCBF	1.58	-0.63	-2.12712	0.01670
CFS	2.21	0.00	0.50000	0.50000
MCBF	2.21	0.00	0.00000	

Parte IV

Selección de Instancias

Capítulo 6

Introducción a la Selección de Instancias

6.1. Introducción

La técnica del vecino o de los k-vecinos más cercanos obtiene muy buenos resultados en el problema de la clasificación usando un algoritmo muy simple. Estrictamente hablando no hay una etapa de aprendizaje, puesto que no se construye ningún modelo. Cada vez que hay que clasificar una nueva instancia, esta se compara con todas las instancias presentes en el conjunto de entrenamiento y se decide la clase que le corresponde en función de la de las instancias más parecidas. Esto permite que el algoritmo de clasificación tenga en cuenta regiones en el espacio de búsqueda que serían omitidas por otros algoritmos que sí realizan una generalización a partir de los datos, o lo que es lo mismo, que tienen una fase de aprendizaje. Pero no todo son beneficios; el coste de disponer de una técnica simple y potente se encuentra en los requerimientos de almacenamiento (al no existir un modelo hay que mantener en memoria todas las instancias) y de tiempo (hay que comparar cada instancia a clasificar con todas las del conjunto de datos) en la etapa de clasificación.

Para resolver este problema se han propuesto varias aproximaciones, que pueden ser agrupadas en dos categorías:

- Las que buscan acelerar la búsqueda de las instancias más pareci-

das con la ayuda de estructuras de datos e índices (Papadimitriu y Bentley, 1980; Vidal, 1986; Sproull, 1991; Yianilos, 1993; Brin, 1995; Gómez-Ballester y otros, 2006).

- Las que se centran en la reducción del número de instancias en el conjunto de datos intentando no perder calidad en la clasificación. Esta categoría, que se llama de Selección de Instancias, será la base de nuestro trabajo y es a la que dedicaremos el resto del capítulo.

En una primera aproximación a la selección de instancias podemos decir que esta consiste en quedarnos con un subconjunto (S) de las instancias de un conjunto de datos de entrenamiento (T) y trabajar con ellas en lugar de trabajar con el conjunto completo, lo que podemos ver reflejado en la Figura 6.1. ¿Qué ventajas obtenemos con esto? En primer lugar reducimos los costes de almacenamiento. Puede parecer que esta cuestión solo era importante hace unos años, cuando la capacidad de almacenamiento de los ordenadores era relativamente pequeña; puede pensarse que con el incremento de la capacidad de almacenamiento de los sistemas actuales esto ya no es un problema, pero esto es una falacia: ha aumentado aún más el tamaño de los datos a procesar.

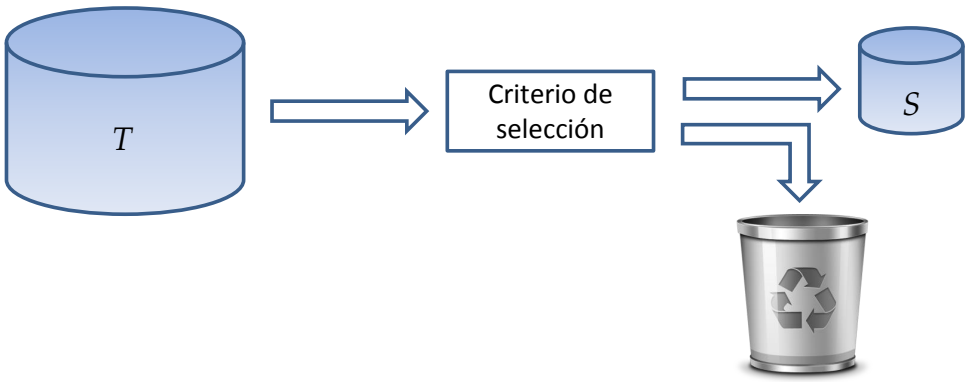

Figura 6.1 – Esquema de los algoritmos de selección de instancias. Se parte de un conjunto de datos de entrenamiento T del que el algoritmo descarta los que le parecen superfluos, dando como salida un subconjunto S de T.

Con la disminución del tamaño de los datos también reducimos los costes de procesamiento de esos datos: cualquier algoritmo de los empleados en minería de datos tendrá n, el cardinal del conjunto de datos con los que trabaja, en la descripción de su complejidad. Esto es especialmente

sensible en los algoritmos de clasificación "lazy", como el del vecino o los $k-$vecinos más cercanos: cuando se intenta clasificar un nuevo elemento este se debe comparar con todas las instancias del conjunto de entrenamiento para encontrar el vecino o los vecinos más cercanos, por lo que el coste computacional va a depender del tamaño de ese conjunto.

Si queremos disminuir el tamaño de los conjuntos de entrenamiento tendremos fundamentalmente que seleccionar aquellos que sean más relevantes y que representen mejor al conjunto completo, de manera que consigamos aproximadamente los mismo resultados que si trabajáramos con el conjunto de datos completo Michalski (1975).

Según (Liu y Motoda, 2001), el resultado ideal de la selección de instancias es una muestra mínima de los datos, independiente del modelo, que pueda conseguir las mismas tareas con poco o ningún deterioro en su rendimiento. Más formalmente, dado un algoritmo de minería de datos M, su rendimiento $\mathcal{P}$ usando un subconjunto S de instancias seleccionadas del conjunto de datos completo T ($S \subset T$) es aproximadamente

$$\mathcal{P}(M_S) \doteq \mathcal{P}(M_T),$$

esto es, el rendimiento obtenido por el algoritmo M usando S es aproximadamente igual que el obtenido usando T. Independiente del modelo quiere decir que para dos algoritmos cualquiera de minería de datos M_i y M_j, si denotamos $\Delta\mathcal{P}$ la diferencia entre el rendimiento obtenido usando S y el obtenido usando T. debe ser

$$\Delta\mathcal{P}(M_i) \doteq \Delta\mathcal{P}(M_j).$$

Obsérvese que con estas definiciones "nos conformamos" con que el rendimiento sea similar, o que no baje mucho. Sin embargo, veremos que en muchas ocasiones tendremos un rendimiento mejor del algoritmo de clasificación usando el subconjunto seleccionado que usando el conjunto completo. Esto es debido a que los algoritmos de selección de instancias consiguen, en ocasiones, eliminar *outliers* (valores atípicos), ruido (elementos mal medidos) y elementos redundantes, que sesgan los resultados, empeorándolos.

Siguiendo con (Liu y Motoda, 2001), la funciones que se persiguen, y se consiguen, con la selección de instancias son las siguientes:

Posibilitar Con la selección de instancias en ocasiones se consigue hacer posible lo imposible. Todos los algoritmos de minería de datos tienen alguna limitación en cuanto al tamaño de datos. Minorar el tamaño del conjunto de datos permite aplicar algoritmos sobre conjuntos de datos grandes con los que de otra manera no sería posible.

Enfocar Los datos de los que se dispone, en el supuesto de que estos datos son reales y no se han obtenido específicamente para hacer minería de datos sobre ellos, suelen tener mucha información que no es relevante para la minería. La reducción de los datos permite delimitar la parte más relevante de estos, de manera que están más enfocados y mejoran la calidad de los resultados.

Limpiar El principio de "si metes basura, sacas basura" (*Garbage In, Garbage Out* o *GIGO*) se cumple también en la minería de datos. Por esta razón es fundamental limpiar los datos antes de su procesamiento; los algoritmos de selección de instancias eliminarán los elementos irrelevantes o ruidosos, obteniendo por tanto unos datos de mejor calidad, con los que se obtendrán resultados mejores y a costes más reducidos.

En cualquier caso, la aplicación de un algoritmo de selección de instancias tiene también un coste, lo que hay que tener en consideración. Por otra parte, la reducción de los datos en ocasiones produce una merma en el rendimiento (por ejemplo, en la precisión obtenida por un clasificador), por lo que debe tenerse en cuenta el compromiso entre el tamaño del conjunto de datos y la calidad de los resultados. Desde este punto de vista, la selección de instancias puede contemplarse como un problema de optimización multiobjetivo: se intenta mantener la calidad de los resultados mientras se minimiza el conjunto de datos.

Encontrar el subconjunto óptimo S de T por fuerza bruta, esto es, probando todos los posibles subconjuntos de instancia tiene un coste computacional de orden $O(2^n)$ siendo n el cardinal de T (el número exacto de subconjuntos a comprobar es $2^n - 1$, puesto que el único subconjunto de T que no habría que probar es el conjunto vacío). Esto es, evidentemente, intratable, por lo que es necesario encontrar estrategias que resuelvan el problema en tiempo polinomial.

6.2. Taxonomía

Hay muchos tipos de algoritmos de selección de instancias: humorísticamente (Bezdek y Kuncheva, 2001) dicen que hay *zillions* de métodos. Existen también en la literatura muchas maneras de clasificar los algoritmos de selección de instancias.

6.2.1. Métodos selectivos y métodos de síntesis

Podemos hacer una primera gran distinción entre los que llamaremos métodos selectivos y los métodos de síntesis (Kim y Oommen, 2003). Los métodos selectivos, también conocidos como de edición, son los que hemos estado exponiendo en la sección anterior: consisten en seleccionar un subconjunto de los datos originales. Los métodos de síntesis (o creativos) crean nuevos elementos (que denominaremos prototipos aunque no hay uniformidad en la literatura) a partir de los de T, donde no todos (o ninguno) tienen que pertenecer a T. En nuestro trabajo nos centraremos en los métodos selectivos, aunque muchas de las clasificaciones que veremos a continuación serían de aplicación a los métodos de síntesis.

6.2.2. Dirección de búsqueda: métodos incrementales, decrementales y por lotes

Una clasificación de los métodos de selección de instancias podemos hacerla (al igual que en los métodos de selección de atributos) según la dirección de búsqueda. Wilson y Martinez (Wilson y Martinez, 2000) y otros muchos trabajos, distinguen entre métodos incrementales, decrementales o por lotes (*batch*):

- La búsqueda incremental parte de un conjunto vacío de instancias seleccionadas ($S = \emptyset$) y va añadiendo instancias que satisfacen determinados criterios. En este caso puede llegar a ser de importancia crítica el orden en el que se van considerando las instancias del conjunto T original, por lo que muchos de ellos presentan las instancias en orden aleatorio.

  Los algoritmos que utilizan la búsqueda incremental tienen varias ventajas: en primer lugar, usan menos espacio de almacenamiento y suelen ser más rápidos que los demás, puesto que en todo momento solo necesitan considerar las instancias que han sido realmente seleccionadas; esto significa que, si llamamos $n = |T|$ y $s = |S|$, en un algoritmo incremental el almacenamiento será probablemente $O(s)$ y el tiempo $O(ns)$, en lugar de $O(n)$ y $O(n^2)$ que podemos intuir que son el almacenamiento y el tiempo necesario en un algoritmo no incremental. Como ventaja debemos destacar que este tipo de algoritmos es más fácil de ser reutilizado en aquellos casos en los que el conjunto de entrenamiento sigue creciendo.

  Como desventajas debemos mencionar la ya apuntada sobre el orden de presentación de las instancias al algoritmo, que pueden sesgar el

resultado final: en las primeras fases del algoritmo, si este no esta bien diseñado, se pueden tomar decisiones basadas en relativamente poca información. Para soslayar esto, algunos algoritmos realizan un procesamiento previo sobre un pequeño subconjunto de instancias; otros actúan incrementalmente pero tienen en cuenta todas las instancias disponibles, con lo que no son, estrictamente hablando, incrementales.

- Los métodos decrementales parten del conjunto de todas las instancias ($S = T$) y van eliminando sucesivamente las que no se consideran pertinentes para el conjunto final. En este tipo de algoritmos también es importante el orden de presentación, pero en cualquier momento el algoritmo dispone de toda la información sobre los datos.

  Los algoritmos decrementales suelen ser más costosos en tiempo y almacenamiento que los incrementales. En cualquier caso, siempre debe tenerse en cuenta que lo verdaderamente importante en los algoritmos de selección de instancias es la calidad del conjunto final de instancias seleccionadas, por lo que un mayor coste en esta fase puede considerarse bien empleado si con ello se consigue una mejor clasificación final.

- Los métodos por lotes (*batch*) difieren la decisión sobre qué instancias eliminar o incluir hasta que se han considerado todas ellas. Esto, que en principio pudiera parecer ventajoso, no está exento de riesgos. Por ejemplo, un algoritmo por lotes podría usar la muy prometedora regla "elimina una instancia si su clase coincide con la de sus k vecinos más cercanos": en un algoritmo por lotes, se corre el riesgo de que esta regla elimine todas las instancias de un clúster bien definido; sin embargo, dentro de un esquema decremental siempre dejaría alguno, puesto que cuando hubiera eliminado todos los algoritmos del clúster menos $\lfloor k/2 \rfloor$, los k vecinos más cercanos de las instancias restantes ya no serían de la misma clase.

  En cualquier caso, los métodos por lotes también tienen en general un coste computacional y de almacenamiento mayor que los métodos incrementales.

6.2.3. Métodos de incremento de la competencia, de preservación de la competencia, híbridos

En (Brighton y Mellish, 2002) se definen los conceptos de mejora de la competencia y de preservación de la competencia:

Mejora de la competencia Mediante la eliminación de ciertas instancias, a menudo es posible mejorar la precisión de un algoritmo de clasificación. Esto es posible cuando existen instancias ruidosas o corruptas y son eliminadas.

Preservación de la competencia Una instancia superflua es aquella que cuando es eliminada no degrada la precisión de la clasificación. Por tanto se puede eliminar sin ninguna merma en la calidad de la clasificación.

Según estas definiciones los autores dividen las diversas técnicas en:

- Técnicas que mejoran la competencia. Son las que optan por eliminar las instancias que producen ruido; por ejemplo, las que están mal clasificadas según sus vecinos. El problema con estas técnicas es que solo pueden trabajar con un número pequeño de instancias ruidosas: si son muchas, dejan de aparecer como excepcionales y comienzan a ser bien clasificadas por otras instancias ruidosas. Los autores sugieren que pueden haber áreas, como el reconocimiento del habla, en el que los datos están formados por muchas pequeñas regiones de excepciones, en el que estos métodos perjudicarían la clasificación.

- Técnicas que preservan la competencia. Son las que optan por eliminar aquellas instancias que no son útiles para la clasificación. En esta categoría se encuadran la mayoría de los métodos clásicos de selección de instancias.

- Técnicas híbridas. Utilizan una combinación de las anteriores: eliminan las instancias ruidosas y eliminan las superfluas.

6.2.4. Puntos frontera versus puntos centrales

Wilson y Martinez (2000) hacen también una distinción entre aquellas técnicas que intentan preservar los puntos de la frontera entre clases, los que intentan preservar los puntos centrales, y aquellos que preservan otro tipo de conjuntos de puntos.

La intuición que hay detrás de los algoritmos que retienen los puntos frontera es que los puntos internos no afectan a los límites de decisión (no van a ser puntos más cercanos a aquellos a clasificar) y, por tanto, pueden ser eliminados sin que esto afecte a la clasificación.

En el otro extremo, otro tipo de técnicas intenta eliminar los puntos frontera: los que son ruido o los que no se corresponden con sus vecinos. Eliminan elementos de la frontera que están cerca de sus enemigos (elementos cercanos de distinta clase), suavizando de este modo la frontera de decisión. Estos algoritmos quizás no eliminen puntos centrales que no aporten nada a la decisión. En el caso extremo de $k = 1$ (el vecino más cercano), la eliminación de los puntos frontera obliga a una cuidadosa elección de los puntos seleccionados para que modelen adecuadamente los bordes entre clases.

6.2.5. Otras caracterizaciones

Filtro versus Wrapper

Existen muchas otras formas de clasificar las técnicas de selección de instancias. Por ejemplo, podemos aplicar la distinción de métodos de Filtro o Wrapper utilizada en el capítulo 4. La mayoría de los métodos de selección de instancias se basan en distintas variantes de analizar si la inclusión o no de una instancias o conjunto de instancias mejora o degrada la clasificación obtenida local o globalmente. Desde este punto de vista, pueden considerarse métodos de envoltura (*wrapper*). Existen también (siguiendo esa terminología) métodos de filtro, esto es, métodos que seleccionan las instancias por propiedades intrínsecas, independiente de la clasificación. Dentro de esta categoría podemos citar, por ejemplo, el trabajo de (Riquelme y otros, 2003) que proponen el algoritmo POP (*Pattern by Ordered Projections*) que se basa en el concepto de debilidad de una instancia, que es el número de veces que no es el borde de una clase en la proyección sobre los valores de cada atributo.

Ponderación de instancias

Algunos métodos asignan un peso a las instancias seleccionadas, de manera que se module la influencia que sobre la consideración de ser más cercano tenga una instancia. Esto permite modelar mejor las fronteras entre las regiones de distintas clases.

6.3. Estrategias de evaluación

Para poder comparar distintas técnicas de selección de instancias, hay que establecer unos criterios para destacar las bondades y las debilidades de cada algoritmo. Wilson y Martinez (2000) enumeran los siguientes:

Precisión en la generalización Este es uno de los parámetros fundamentales. Un buen algoritmo debe reducir el tamaño del conjunto de entrenamiento sin mucha pérdida en la clasificación. En muchas ocasiones, la precisión se verá incluso incrementada, al eliminar los elementos que producen ruido o al conseguir que las fronteras de las regiones se adapten mejor al modelo subyacente real.

Reducción de tamaño Como dijimos en la introducción, uno de los principales objetivos de los algoritmos de selección de instancias es la reducción del tamaño del conjunto de datos. Por tanto, este es otro de los parámetros fundamentales a considerar.

Incremento de la velocidad de la clasificación Otro de los objetivos primordiales es el de incrementar la velocidad de la clasificación. La reducción del tamaño normalmente conducirá a una reducción en el tiempo necesario para la clasificación.

Tolerancia al ruido Los algoritmos pueden tener distinto comportamiento cuando hay presencia de ruido. Cuando hay muchos elementos ruidosos pueden ocurrir dos problemas: por una parte, se eliminarán pocas instancias del conjunto de entrenamiento porque serán necesarias para modelar las fronteras producidas por las instancias ruidosas. Por otra, si por culpa del ruido se mantienen las instancias ruidosas y se eliminan instancias "buenas", la precisión se degradará considerablemente.

Velocidad de aprendizaje El proceso de aprendizaje, esto es, la generación del subconjunto de instancias, se realiza una sola vez, por lo que su importancia no es tan elevada como la de la velocidad de la clasificación. De todos modos, si la obtención del subconjunto de instancias toma demasiado tiempo, puede hacerlo impracticable en aplicaciones reales.

Incremental Un algoritmo incremental es uno capaz de adaptarse a la llegada de nuevas instancias. Es una característica deseable, pero no imprescindible. Por ejemplo, podemos aplicar una buena técnica no

Algoritmo 6.1: Algoritmo CNN

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1  S = ∅
2  foreach clase c de T do
3  |  Elige una instancia x_c de la clase c al azar
4  |  S = S ∪ {x_c}
5  end foreach
6  repeat
7  |  añadidos = false
8  |  Elige una instancia x ∈ T al azar
9  |  T = T − {x}
10 |  Sea s_c ∈ S tal que dist(x, s_c) = mín_{y∈S} dist(x, y)
11 |  if clase(x) ≠ clase(s_c) then
12 |  |  S = S ∪ {x}
13 |  |  añadidos = true
14 |  end if
15 until añadidos == false
```

incremental y posteriormente aplicar una técnica incremental cuando se dispongan de nuevos datos.

6.4. Algunas técnicas relevantes

Vamos a destacar en primer lugar que hemos dejado fuera muchos algoritmos. Por ejemplo, como dijimos en la introducción, no trataremos los algoritmos de síntesis; se puede ver un completo estudio de estos en (Triguero y otros, 2012). Veremos a continuación los que nos han parecido relevantes bien por su importancia histórica, bien por su calidad, bien por ser utilizados más adelante.

Algoritmo CNN

Se puede situar el comienzo de la historia de los algoritmos de selección de instancias en el algoritmo CNN (*Condensed Nearest Neighbor Rule*) (Hart, 1968).

Este algoritmo, cuyo pseudocódigo podemos ver en 6.1, deja en S un subconjunto de T tal que cada elemento de T está más cerca de un elemento de S de la misma clase que a un elemento de S de distinta clase.

Por tanto, los elementos de S clasifican correctamente (usando la regla del vecino más cercano, esto es, $k = 1$) todos los elementos de T correctamente, en el supuesto de que T es consistente: no hay dos elementos en T con los mismos valores de los atributos y distinta clase. El algoritmo clasifica correctamente todas las instancias de T, aunque no garantiza que S sea mínimo.

Este algoritmo es muy sensible al ruido: las instancias ruidosas son mal clasificadas por sus vecinos; por tanto el algoritmo las retiene. Por la misma razón (su cercanía a instancias de distinta clase), también retiene las instancias no ruidosas que estén cerca de las instancias ruidosas.

Es incremental, y en la clasificación de Brighton y Mellish es de preservación de la competencia.

Algoritmo SNN

El algoritmo SNN (*Selective Nearest Neighbor Rule*) (Ritter y otros, 1975) es una extensión del algoritmo CNN. En este método todos los elementos de T estarán más cerca a un elemento de S de su misma clase que que a cualquier miembro de T (no de S, como en CNN) de una clase diferente. Además, garantiza que es el subconjunto mínimo que satisface esta condición.

El algoritmo es muy complicado, en palabras de Wilson y Martinez, y tiene una complejidad, en tiempo, mayor que la mayoría de los demás algoritmos. Utiliza una matriz de $n \times n$ elementos booleanos ($n = |T|$) y complejas operaciones sobre ella.

Es, al igual que el anterior, de preservación de la competencia.

Algoritmo RNN

El algoritmo RNN (*Reduced Nearest Neighbor*) (Gates, 1972) es otra extensión de CNN, en este caso decremental. Parte de $S = T$ y elimina las instancias de S cuya eliminación no provoca que cualquier otra instancia de T sea mal clasificada por las instancias restantes de S. Es más costosa en tiempo que la técnica CNN, pero siempre obtiene un subconjunto de las instancias seleccionadas por esta. El algoritmo podemos verlo en 6.2.

Debido a que las instancias que se eliminan no tenían por qué ser clasificadas correctamente, el algoritmo elimina instancias ruidosas; elimina también instancias internas, pero mantiene las de las fronteras.

Es de preservación de la competencia en la clasificación de Brighton y Mellish.

Algoritmo 6.2: Algoritmo RNN

Input: T: Conjunto de instancias; S_{CNN} subconjunto de instancias seleccionadas por CNN
Output: S: Subconjunto de instancias seleccionadas

```
1 S = S_CNN
2 foreach x ∈ S do
3 |   S = S − {x}
4 |   Usa S para clasificar T
5 |   if algún elemento de T se clasifica incorrectamente then
6 |   |   S = S ∪ {x}
7 |   end if
8 end foreach
```

Algoritmo 6.3: Algoritmo ENN

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1 S = T
2 foreach x ∈ S do
3 |   Sean {x_1, x_2, ..., x_k} los k−vecinos más cercanos de x en S − {x}
4 |   Sea c la clase mayoritaria de {x_1, x_2, ..., x_k}
5 |   if clase(x) ≠ c then
6 |   |   S = S − {x}
7 |   end if
8 end foreach
```

Algoritmo ENN

La técnica ENN (*Edited Nearest Neighbor*) (Wilson, 1972). Este algoritmo, que podemos ver en 6.3, es decremental y va eliminando instancias de S si no tiene la misma clase que la mayoría de sus k vecinos más cercanos, con $k = 3$. Elimina elementos ruidosos y puntos de la frontera, suavizando los bordes de decisión. Una variante de este algoritmo es RENN (*Repeated Edited Nearest Neighbor*) (Tomek, 1976), que aplica ENN repetidamente hasta que las instancias que quedan en S tienen la mayoría de sus vecinos de su misma clase, lo que aún suaviza más los bordes de decisión.

En la clasificación de Brighton y Mellish, el algoritmo ENN es de incremento de la competencia. Naturalmente, RENN también lo es.

Algoritmo 6.4: Algoritmo All k-NN

Input: T: Conjunto de instancias; k: número de vecinos a considerar
Output: S: Subconjunto de instancias seleccionadas

```
1  S = T
2  foreach x ∈ S do
3      eliminable(x) = false
4      i = 1
5      while i ≤ k do
6          Sean NN(i, x) los i vecinos más cercanos de x en T − {x}
7          Sea c la clase mayoritaria de NN(i, x)
8          if clase(x) ≠ c then
               /* NN(i, x) clasifica incorrectamente a x */
9              eliminable(x) = true
10             i = k        // Para forzar la salida del bucle
11         end if
12         i = i + 1
13     end while
14     foreach x ∈ T do
15         if eliminable(x) then
16             S = S − {x}
17         end if
18     end foreach
19 end foreach
```

Algoritmo All k-NN

Este algoritmo, debido también a Tomek (Tomek, 1976), es otra extensión de ENN. Por cada elemento, determina cómo lo clasifica su vecino, sus dos vecinos, etc. hasta sus k vecinos más cercanos; si lo clasifican erróneamente, se marca. Una vez analizado todo el conjunto T, se eliminan todos los elementos marcados. El pseudocódigo se puede ver en 6.4.

Obtiene un resultado mucho mejor, en precisión, que RENN, que ya mejoraba los métodos anteriores. El problema es que deja elementos internos, por lo que su poder de reducción es limitado.

Es un método por lotes (ni incremental ni decremental) y, en la clasificación de Brighton y Mellish es de incremento de la competencia.

Algoritmos IBL (*"Instance-Based Learning"*)

En (Aha y otros, 1991) y (Aha, 1992) se presentan un conjunto de métodos de selección de instancias "Instance-based Learning" denominados IB1, IB2, IB3.

Todos los algoritmos usan una función de similitud definida como

$$sim(x, y) = -\sqrt{\sum_{i=1}^{n} f(x_i, y_i)},$$

donde $f(x_i, y_i) = (x_i - y_i)^2$ para los valores numéricos y $f(x_i, y_i) = (x_i \neq y_i)$ para los booleanos y discretos. Los valores perdidos se toman como de diferencia máxima con el presente y si los dos son perdidos se toma $f(x_i, y_i) = 1$.

IB1 No es más que la regla del vecino más cercano (excepto en que normaliza los rangos de los atributos, procesa las instancias incrementalmente y tiene una estrategia para tratar los valores perdidos. Es utilizado como *baseline*.

IB2 Es un método incremental. Comienza con $S = \emptyset$ y cada instancia de T se añade si no es clasificada correctamente por las instancias que ya están en S; la primera instancia de T siempre se añade. Es un método muy similar a CNN, excepto en que S no se siembra con un elemento de cada clase y no repite el proceso después de la primera pasada por el conjunto de entrenamiento (podemos ver su psedocódigo en 6.5). Esto significa que IB2 no necesariamente clasifica T correctamente.

Este algoritmo retiene los puntos frontera en S, eliminando los puntos internos que estén rodeados por elementos de su misma clase. Al igual que CNN, este algoritmo es muy sensible al ruido.

IB3 Este algoritmo es una extensión de IB2 en la que se intenta evitar seleccionar elementos ruidosos almacenando únicamente instancias mal clasificadas *aceptables*; este es un concepto que se utiliza en el algoritmo y que se explica a continuación. Podemos ver el pseudocódigo del algoritmo en 6.6.

Una instancia es *aceptable* si el límite inferior en su precisión es significativamente más alta, con un nivel de confianza del $90\,\%$, que el límite superior de la frecuencia de su clase. Análogamente, se elimina una instancia de S si el límite superior de su precisión está significativamente por

Algoritmo 6.5: Algoritmo IB2

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1  S = ∅
2  foreach x ∈ T do
3  |   foreach s ∈ S do
4  |   |   sim(s) = sim(s, x)
5  |   end foreach
6  |   s_max = algún s ∈ S con sim(s) máximo
7  |   if clase(x) ≠ clase(s_max then
8  |   |   S = S ∪ {x}
9  |   end if
10 end foreach
```

debajo (con un nivel de confianza del $70\,\%$) que el límite inferior de la frecuencia de su clase. Otras instancias de S se mantienen provisionalmente, y al final se eliminan si no demuestran ser aceptables.

La expresión para el cálculo de los límites superiores e inferiores del intervalo de confianza es:

$$\frac{p + z^2/2n \pm \sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n},$$

donde para la *precisión* de una instancia en S, n es el número de intentos de clasificación hasta que la instancia se introduce en S, p es la precisión de tales intentos (el número de veces que la instancia tuvo la misma clase que x, dividido entre n) y z el nivel de confianza, que como hemos dicho es $0{,}9$ para la aceptación y $0{,}7$ para el rechazo. En cuanto al cálculo de la *frecuencia* de una clase, p es la frecuencia, es decir, la proporción de instancias de la clase consideradas hasta el momento, n es el número de instancias consideradas y z es la confianza (de nuevo, $0{,}9$ para la aceptación y $0{,}7$ para el rechazo).

IB3 obtiene una gran tasa de reducción así como de precisión y es relativamente poco sensible al ruido.

Todos los algoritmos "Instance-based" son híbridos en la clasificación de Brighton y Mellish.

Algoritmo 6.6: Algoritmo IB3

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1  S = ∅
2  foreach x ∈ T do
3  |  Sea a la instancia aceptable más próxima a x en S
4  |  (Si no hay ninguna aceptable sea a una elegida al azar)
5  |  if clase(a) ≠ clase(x) then
6  |  |  S = S ∪ {x}
7  |  end if
8  |  foreach s ∈ S do
9  |  |  if s es al menos tan cercana a x como a then
10 |  |  |  Actualiza el registro de clasificación de s
11 |  |  |  if El registro de clasificación de s es significativamente
   |  |  |  pobre then
12 |  |  |  |  S = S − {s}
13 |  |  |  end if
14 |  |  end if
15 |  end foreach
16 end foreach
17 foreach s ∈ S do
18 |  if s no es aceptable then
19 |  |  S = S − {s}
20 |  end if
21 end foreach
```

Algoritmo Explore

Cameron-Jones presenta en (Cameron-Jones, 1995) dos métodos de selección de instancias basados en el concepto de heurística de longitud de codificación (*encoding length heuristic*). En primer lugar se presenta el algoritmo Pre/All, que añade una instancia a S si esto reduce el coste. El coste de un modelo (una situación) se define como

$$cost(m, n, x) = F(m, n) + m \log_2 c + F(x, n - m) + x \log_2 (c - 1),$$

donde n es el cardinal de T, m es el cardinal de S en cada paso del algoritmo, c es el número de clases y x es el número de *excepciones*, esto es, el número de instancias analizadas hasta el momento que son clasificadas erróneamente por las instancias de S. $F(m, n)$ es el coste de codificar

Algoritmo 6.7: Algoritmo Pre/All

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1  S = ∅
2  m = 0          // m es siempre el cardinal de S
3  n = |T|
4  costeAnterior = ∞
5  foreach x ∈ T do
6  |  S = S ∪ {x}
7  |  m = m + 1
8  |  coste = cost(m, n, x)
9  |  if coste < costeAnterior then   // S ∪ {x} mejora el coste
10 |  |  costeAnterior = coste
11 |  else
12 |  |  S = S − {x}
13 |  |  m = m − 1
14 |  end if
15 end foreach
```

qué m instancias de entre las n disponibles son retenidas; se define como:

$$F(m,n) = \log^*\left(\sum_{j=0}^{m} C_j^n\right) = \log^*\left(\sum_{j=0}^{m} \frac{n!}{j!(n-j)!}\right)$$

donde $\log^*(x)$ es la suma de $\log_2(x), \log_2(\log_2(x)), \ldots$

El algoritmo, que podemos ver en 6.7, es sensible al orden de presentación de los elementos. Es incremental, e híbrido en la clasificación de Brighton y Mellish.

En el mismo trabajo se presenta el algoritmo Explore. Este algoritmo aplica en primer lugar el algoritmo Pre/All y a continuación se aplican 1000 mutaciones sobre S: se añade una instancia a S, se elimina una instancia de S o se intercambia una instancia de S con una de $T - S$; se estima el coste del nuevo subconjunto obtenido y se mantiene la mutación si no se incrementa el coste de S.

Este algoritmo tiene las mismas características que Pre/All (incremental e híbrido). Obtiene experimentalmente buenos resultados en cuanto a precisión y muy buenos en cuanto a reducción, muy por encima de la mayoría de los demás algoritmos.

Algoritmos DROP

En (Wilson y Martinez, 2000), los autores plantean un conjunto de algoritmos denominados genéricamente DROP (*Decremental Reduction Optimization Procedure*); son DROP1, DROP2, DROP3, DROP4, DROP5 y DEL. Se basan en el concepto de *asociado*. Las instancias que tienen a x entre sus k vecinos más cercanos se denominan asociadas de x; denotaremos $asociados(x)$ a la lista de asociados de x, que supondremos ordenados de más cercano a más lejano.

DROP1 La primera versión es similar a RNN excepto en que la precisión se comprueba sobre S en lugar de sobre T. Se basa en el siguiente principio: elimina x si se clasifican correctamente con él al menos tantos de sus asociados en S que sin él.

El algoritmo lo podemos ver en 6.8. Es decremental e híbrido como todos los de esta familia de algoritmos. Los autores lo plantean como baseline.

DROP2 El algoritmo DROP1 tiende a acumular instancias ruidosas. Para mejorarlo se introduce DROP2, que es como el anterior, salvo que en este caso la regla es:

Elimina x si se clasifican correctamente con él al menos tantos de sus asociados en S que sin él.

Es decir, tiene en cuenta no solo las instancias seleccionadas hasta el momento sino todas. También se modifica el orden de consideración de las instancias: se ordenan las instancias según la distancia a su enemigo y se van considerando comenzando por las que están más lejos de su enemigo (instancia de distinta clase) más cercano. Esto tiende a eliminar en primer lugar las instancias más lejanas a las fronteras de decisión, lo que incrementa la probabilidad de mantener puntos frontera.

DROP3 El algoritmo DROP2 elimina puntos centrales antes que los de los bordes. Esto puede ser un problema, puesto que un punto ruidoso en el centro de un clúster causa que muchos puntos de ese clúster sean considerados puntos frontera y pueden permanecer en S después de que se haya eliminado el punto ruidoso. La solución que aporta DROP3 es hacer un filtrado de instancias ruidosas antes de ordenar las instancias de S. Se usa para ello la regla similar a ENN: se eliminan las instancias que sean incorrectamente clasificadas por su k vecinos más cercanos. Esto elimina las instancias ruidosas y las de los bordes, haciendo la frontera más suave, lo

Algoritmo 6.8: Algoritmo DROP1

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

$S = T$
foreach $x \in S$ **do**
  $asociados(x) =$ lista vacía
end foreach
foreach $x \in S$ **do**
  Sean $\{y_1, y_2, \ldots, y_{k+1}\}$ los $k+1$ vecinos más cercanos de x en S
  for $i = 1$ **to** $k+1$ **do**
    Añade x a $asociados(y_i)$
  end for
end foreach
foreach $x \in S$ **do**
  $with =$ número elementos de $asociados(x)$ correctamente clasificados con x como vecino
  $without =$ número de elementos de $asociados(x)$ correctamente clasificados sin x
  if $without - with \geq 0$ **then**
    $S = S - \{x\}$
    foreach $y \in asociados(x)$ **do**
      Elimina x de la lista de vecinos más cercanos de y
      Sea z un nuevo vecino más cercano de x
      Añade y a $asociados(y)$
    end foreach
    foreach vecino n de x **do**
      Elimina x de $asociados(n)$
    end foreach
  end if
end foreach

que rebaja la probabilidad de que se produzca sobreajuste. Una vez hecha esta limpieza, el algoritmo actúa como DROP2.

DROP4 En ocasiones DROP3 elimina demasiadas instancias en el paso previo. DROP4 es idéntico a DROP3 excepto en que en lugar de aplicar ENN sin más, en el paso previo de filtrado se elimina una instancia solo si

1. Es clasificada erróneamente por sus k vecinos más cercanos.

2. No degrada la clasificación de otra instancia.

Con esto se evita el borrado de más instancias de las necesarias, a costa de algo más de almacenamiento.

DROP5 Es como DROP2 excepto en que las instancias son consideradas en orden desde las que tienen más cerca a su enemigo más cercano hasta las que lo tienen más lejos. Por tanto, elimina en primer lugar los puntos frontera (además de los ruidosos), suavizando los límites de decisión. Después de este paso se aplica reiteradamente DROP2 siguiendo, ahora sí, el orden de enemigo más lejano a más cercano, hasta que no se produce ninguna mejora.

DEL El algoritmo DEL (*Decremental Encoding Length*) es como DROP3 excepto en que usa la heurística de longitud de codificación de Explore para decidir en cada caso si una instancia debe ser eliminada. Comienza con $S = T$ y aplica un paso de filtrado de ruido en el que una instancia es eliminada si

1. Es clasificada erróneamente por sus k vecinos más cercanos.

2. La eliminación de la instancia no incrementa el coste de longitud de codificación.

Después se ordenan las instancias restantes en orden según su distancia al enemigo más cercano, de más lejanas a más cercanas, y mientras haya alguna mejora se van eliminando las instancias si hacerlo no incrementa el coste de longitud de codificación.

Algoritmo ICF

Este algoritmo fue introducido en (Brighton y Mellish, 1999) y su estudio fue ampliado en (Brighton y Mellish, 2002). Se basan en un trabajo previo de Smyth y Keane (1995). Se basa en los conceptos de *alcance* y *cobertura*. Se define en primer lugar el $LocalSet$ (conjunto local) de una instancia x como el conjunto de instancias contenida en la hiperesfera más grande centrada en x que solo contiene instancias de la misma clase que x. Esto se refleja gráficamente en la figura 6.2.

Una vez definido esto, se puede definir el alcance y la cobertura de una instancia como

- $alcance(x) = LocalSet(x)$

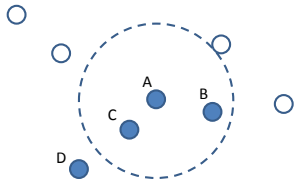

Figura 6.2 – $LocalSet(A) = \{A, B, C\}$.

- $cobertura(x) = \{y \in T | x \in alcance(y)\}$

El concepto es muy parecido al de vecinos y asociados de Wilson y Martinez excepto en que no hay k vecinos más cercanos, sino tantos como se encuentren hasta llegar al primer elemento de otra clase (enemigo).

La intuición que hay tras la heurística utilizada es que una instancia se puede eliminar si hay más instancias que pueden clasificar como ella de las que ella puede clasificar.

Este algoritmo deja delgadas líneas de instancias a ambos lados de las fronteras. Es decremental e híbrido.

6.5. Resumen

En este capítulo hemos definido el problema de la selección de instancias, dando algunas definiciones formales. Hemos visto las ventajas que puede reportar la selección de instancias en el problema de la clasificación. Hemos procedido a mostrar algunos criterios por los que se pueden clasificar los distintos (y numerosos) algoritmos que hay y también hemos establecido unas características para medir la bondad de estos. Por último, hemos estudiado algunos algoritmos por su interés intrínseco o porque serán empleados en el capítulo siguiente en la experimentación.

Algoritmo 6.9: Algoritmo ICF

Input: T: Conjunto de instancias
Output: S: Subconjunto de instancias seleccionadas

```
1  S = T
   // Se ejecuta la edición de Wilson
2  foreach x ∈ S do
3  |  if x es clasificado incorrectamente por sus k vecinos más
   |  cercanos then
4  |  |  Marca x para su eliminación
5  |  end if
6  end foreach
7  foreach x ∈ S do
8  |  if x está marcado para su eliminación then
9  |  |  S = S − {x}
10 |  end if
11 end foreach
12 repeat
13 |  foreach x ∈ S do
14 |  |  Calcula alcance(x)              // Conjunto de vecinos
15 |  |  Calcula cobertura(x)          // Conjunto de asociados
16 |  end foreach
17 |  progreso = false
18 |  foreach x ∈ S do
19 |  |  if (|alcance(x)| > |cobertura(x)| then
20 |  |  |  Marca x para su eliminación
21 |  |  |  progreso = true
22 |  |  end if
23 |  end foreach
24 |  foreach x ∈ S do
25 |  |  if x está marcado para su eliminación then
26 |  |  |  S = S − {x}
27 |  |  end if
28 |  end foreach
29 until progreso == false
```

Capítulo 7

Nuestras aportaciones: InstanceRank e ISR

7.1. Introducción

Como hemos visto en el Capítulo 6, en la técnica del vecino más cercanos no todo son beneficios: el coste de disponer de una técnica simple y potente se encuentra en los requerimientos de almacenamiento y, sobre todo, de tiempo (hay que comparar la instancia a clasificar con todas las demás) en la etapa de generalización.

Para resolver este problema, se han propuesto varias soluciones que se pueden encuadrar en dos categorías: las orientadas a reducir el número de instancias del conjunto de datos sin perder calidad en la clasificación y que hemos visto en el capítulo anterior, y las que buscan acelerar la búsqueda de las instancias más parecidas con la ayuda de estructuras de datos e índices (Papadimitriu y Bentley, 1980; Vidal, 1986; Sproull, 1991; Yianilos, 1993; Brin, 1995; Gómez-Ballester y otros, 2006).

En este capítulo presentamos nuestras dos aportaciones, publicadas en (Vallejo y otros, 2010): un ranking que permitirá caracterizar la relevancia de los elementos del conjunto de datos, al que llamaremos *InstanceRank*, y un algoritmo de reducción de instancias, que llamaremos *ISR* y que utiliza InstanceRank. Veremos que la precisión obtenida por nuestra técnica es estadísticamente equivalente a la obtenida por los mejores algoritmos

de selección de instancias, y que mejora significativamente a todas esas técnicas en capacidad de reducción excepto a una, con la que no tiene diferencias siginificativas.

7.2. InstanceRank

La idea general que nos ha conducido a en esta parte del trabajo es la consideración de que la medida de relevancia que aporta un ranking entre las instancias puede ser útil para determinar cuáles son más importantes dentro de una base de datos. En nuestro caso, nos hemos apoyado en una versión ponderada de PageRank similar a la utilizada por el algoritmo TextRank empleado para priorizar unidades lingüísticas en textos escritos en lenguaje natural.

7.2.1. PageRank ponderado

TextRank, debido a Mihalcea (Mihalcea y Tarau, 2004), es una adaptación del PageRank, visto en la subsección 3.5.1, aplicado en un contexto muy distinto del anterior: el procesamiento del lenguaje natural, concretamente a la extracción de palabras clave y a la realización de resúmenes. La idea central es que una palabra o frase (en general, un vértice del grafo) tiene mayor o menor importancia dependiendo de la influencia que tengan las demás sobre ella; esa importancia es la que denota el TextRank. La influencia de una palabra o frase sobre otra depende de su similitud; por esta razón, y a diferencia del PageRank, la influencia tiene una cierta ponderación: el vértice V_i influye sobre el vértice V_j con un determinado peso w_{ij} (por tanto, el grafo es ponderado).

Recordemos la expresión del PageRank:

$$PR(v) = (1-d) + d \sum_{w \in In(v)} \frac{PR(w)}{|Out(w)}.$$

Una vez introducidas las ponderaciones introducidas por Mihalcea, se obtiene una forma de PageRank ponderado:

$$TR(V_i) = (1-d) + d \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{v_k \in Out(V_j)} w_{jk}} TR(V_j)$$

La influencia de una palabra o frase sobre otra es recíproca, de ahí que el grafo que describe la situación sea no dirigido ($w_{ij} = w_{ji}$, $In(V) = Out(V)$), lo que simplifica mucho los cálculos.

La expresión del TextRank, al igual que la del PageRank, puede calcularse mediante un procedimiento iterativo, que experimentalmente se comprueba que converge.

7.2.2. InstanceRank

Esta es nuestra primera propuesta en este capítulo. Consideramos las instancias de una base de datos como vértices de un grafo, de modo análogo a lo que hacen PageRank y TextRank. Cada arista de este grafo será la similitud entre las instancias que representan los extremos de la arista; esta arista estará etiquetada con esa similitud. Este grafo, por su naturaleza, será completo y no dirigido. Hemos definido la similitud entre las instancias como una función de la distancia entre ellas; es un concepto en cierta manera inverso a la de la distancia. Una función de similitud $\mathrm{sim}(x, y)$ entre las instancias de un conjunto de datos T debe cumplir tres propiedades $\forall v_1, v_2 \in T$:

$$\begin{aligned} &0 \leq \mathrm{sim}(v_1, v_2) \leq 1 \\ &\mathrm{sim}(v_1, v_1) = 1 \\ &\mathrm{sim}(v_1, v_2) = \mathrm{sim}(v_2, v_1) \end{aligned}$$

Para la distancia entre dos instancias hemos utilizado la métrica HEOM (Heterogeneous Euclidean-Overlap Metric) **?**, vista en el apartado 2.4.2. Recordemos que es la distancia euclídea para los atributos continuos, ajustada a los valores máximo y mínimo; para los discretos se usa la distancia de solapamiento: 0 para los que tienen el mismo valor y 1 para los que tienen distinto valor. Finalmente, la distancia se normaliza, de manera que

$$\forall v_1, v_2 \in T \quad 0 \leq \mathrm{dist}(v_1, v_2) \leq 1.$$

Hemos experimentado con las siguientes funciones de similitud:

$$\begin{aligned} \mathrm{sim}(v_1, v_2) &= 1 - \frac{\mathrm{dist}(v_1, v_2)}{k} \\ \mathrm{sim}(v_1, v_2) &= \frac{1}{1 + k\,\mathrm{dist}(v_1, v_2)} \\ \mathrm{sim}(v_1, v_2) &= e^{-k\frac{\mathrm{dist}(v_1, v_2)}{(1-\mathrm{dist}(v_1, v_2))}} \\ \mathrm{sim}(v_1, v_2) &= e^{-k\,\mathrm{dist}(v_1, v_2)^2} \end{aligned}$$

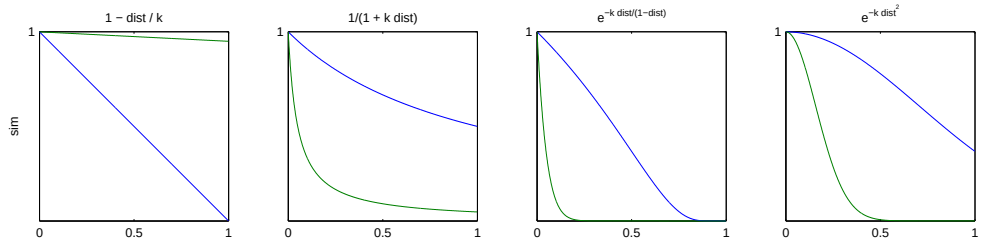

Figura 7.1 – Funciones de similitud con las que se ha experimentado en InstanceRank.

donde k $(k > 0)$ es un parámetro cuyo valor habrá que ajustar adecuadamente. Estas cuatro funciones cumplen las tres propiedades anteriores. La gráfica de estas funciones se puede ver en la Fig. 7.1, en la que hemos representado los valores para $k = 1$ y $k = 20$ para cada una de ellas, estando los demás valores incluidos en el área delimitada por estos dos.

La expresión del cálculo del InstanceRank del conjunto de instancias T, donde las instancias se expresan como $\{V_i, 1 \leq i \leq |T|\}$, es la siguiente:

$$IR(V_i) = (1 - d) + d \sum_{j=1}^{|T|} \frac{\text{sim}(V_j, V_i)}{\sum_{k=1}^{|T|} \text{sim}(V_j, V_k)} IR(V_j) \tag{7.1}$$

Se ha experimentado con dos tipos de InstanceRank:

GlobalInstanceRank No tiene en cuenta a qué clase pertenece cada instancia, por lo que es una medida de la densidad global de las instancias. $\text{sim}(V_i, V_j)$ es simplemente la similitud entre las dos instancias V_i y V_j. En la Figura 7.2 podemos ver los elementos del conjunto de datos Iris donde el tamaño de cada elemento indica el valor de su GlobalInstanceRank.

LocalInstanceRank Se distingue entre cada una de las clases, haciendo que la similitud entre dos instancias sea cero si ambas pertenecen a distintas clases, y su valor real si ambas pertenecen a la misma clase. Podemos ver una representación de los elementos de Iris en la Figura 7.3; el tamaño de la esfera representa el valor del LocalInstanceRank de cada uno de los elmenentos.

La interpretación que puede hacerse del LocalIstanceRank es la siguiente: un valor más alto del rango significa que la instancia correspondiente tiene una mayor densidad local de *instancias de su misma clase*. Por el

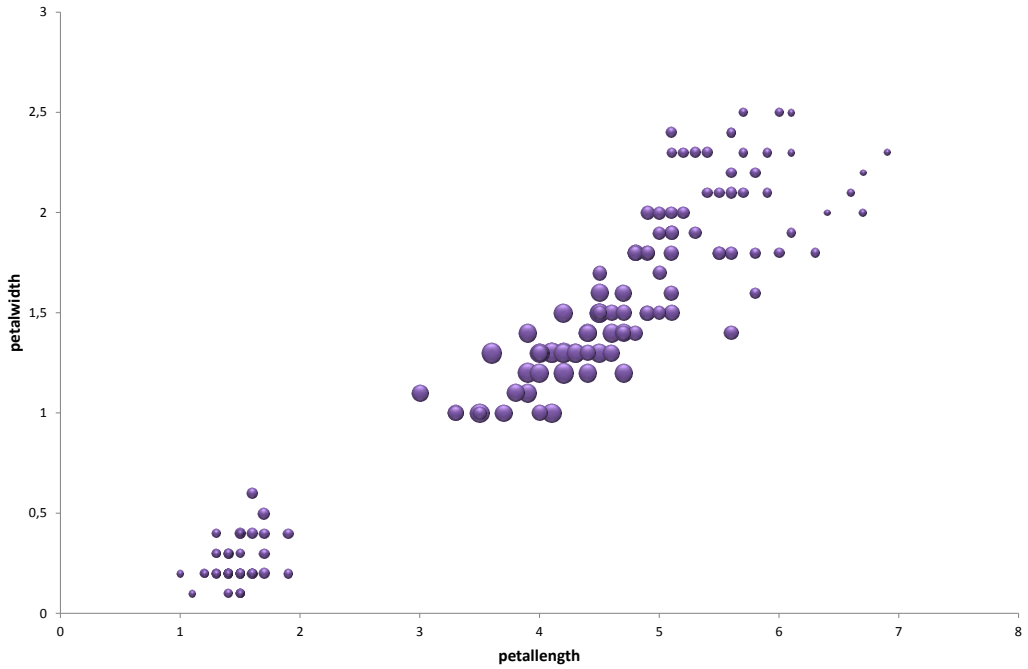

Figura 7.2 – Elementos de Iris con su GlobalInstanceRank. Se ha representado en el eje horizontal el valor del atributo *petallength* y en el vertical el del *sepallength* (todos los algoritmos de selección de atributos coinciden en que estos dos atributos son los más representativos).

contrario, un valor más pequeño significará, o bien que es una instancia periférica (situada en la frontera entre clases), o bien que es una instancia que está rodeada de instancias de otra clase, y que, por tanto, puede estar representando ruido.

En general, el InstanceRank es una medida de la importancia de la instancia dentro del conjunto de datos; por tanto, permite establecer un ranking entre las instancias. Hemos utilizado GlobalInstanceRank en un trabajo preliminar (Vallejo y otros, 2007), que también ha sido aplicado con éxito en el problema del clustering semisupervisado (Ruiz y otros, 2010). En el algoritmo ISR hemos utilizado LocalInstanceRank como criterio para seleccionar las instancias de la base de datos de entrenamiento porque es con la que mejores resultados se ha obtenido.

7.2.3. Justificación algebraica

Vamos a ver a continuación la justificación algebraica subyacente al InstanceRank. En este punto, y para simplificar la notación, vamos a lla-

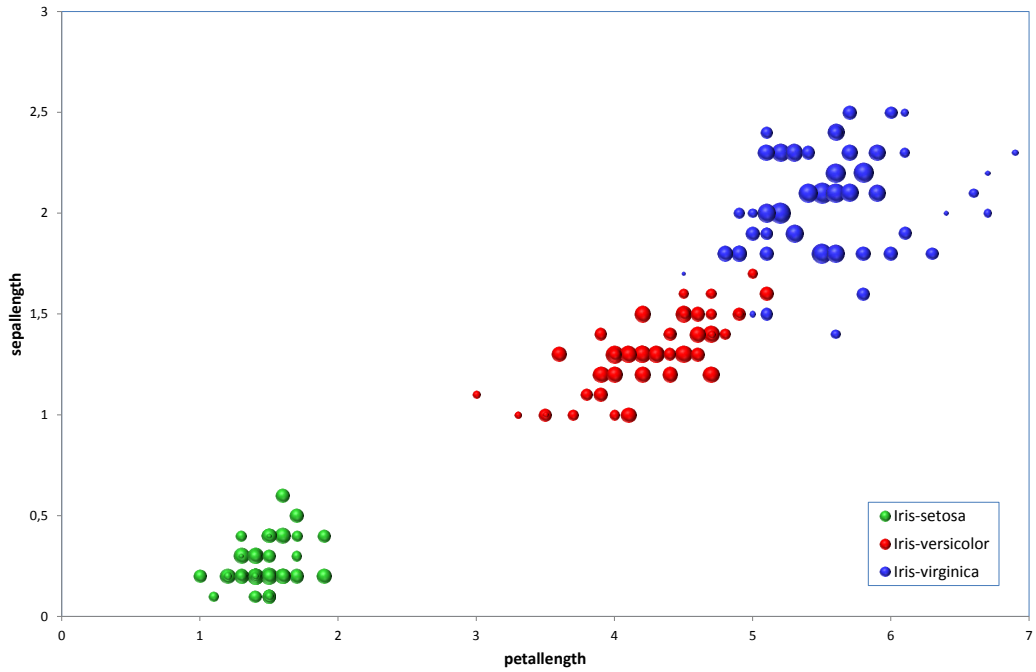

Figura 7.3 – Elementos de Iris con su LocalInstanceRank. Se ha representado en el eje horizontal el valor del atributo *petallength* y en el vertical el del *sepallength*. Se muestran por separado cada uno de los elementos de cada clase.

mar x_i a $IR(V_i)$ y $N = |T|$. Denotamos como $s_{ij} = \text{sim}(V_j, V_i)$ (aunque el orden sería indiferente, puesto que la similitud es simétrica) y $s_j = \sum_{k=1}^{N} s_{kj}$. Entonces, la expresión del cálculo del InstanceRank puede expresarse matricialmente como

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = (1-d) \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + d \begin{bmatrix} s_{11}/s_1 & s_{12}/s_2 & \dots & s_{1N}/s_N \\ s_{21}/s_1 & s_{22}/s_2 & \dots & s_{2N}/s_N \\ \vdots & \vdots & & \vdots \\ s_{N1}/s_1 & s_{N2}/s_2 & \dots & s_{NN}/s_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}.$$

Si en la expresión anterior llamamos $\boldsymbol{S}$ a la matriz cuadrada, y $\boldsymbol{u}$ al vector de $N \times 1$ unos, podemos reescribirla como

$$\boldsymbol{x} = (1-d)\boldsymbol{u} + d\boldsymbol{S}\boldsymbol{x}. \tag{7.2}$$

La matriz $\boldsymbol{S}$ es estocástica: por una parte, sus valores, que son los cocientes $s_{ij}/s_j = s_{ij}/\sum_{k=1}^{N} s_{kj}$, están obviamente comprendidos entre 0 y 1; por otra parte, la suma de los elementos de la columna i es

$$\sum_j \frac{s_{ji}}{s_i} = \frac{1}{s_i} \sum_j s_{ji} = \frac{1}{s_i} s_i = 1.$$

Si $\boldsymbol{x}$ está normalizado, de forma que $\|\boldsymbol{x}\|_1 = N$, es decir, que la suma de los componentes es N (si partimos de un vector con elementos no negativos, siempre se van a mantener no negativos, puesto que los elementos de $\boldsymbol{S}$ lo son), se tiene que

$$\boldsymbol{u}^T\boldsymbol{x} = N$$

con lo que la expresión 7.2 del cálculo de $\boldsymbol{x}$ se puede escribir como

$$\boldsymbol{x} = (1-d)\boldsymbol{u}\frac{1}{N}\boldsymbol{u}^T\boldsymbol{x} + d\boldsymbol{S}\boldsymbol{x}$$

o

$$\boldsymbol{x} = ((1-d)\boldsymbol{e}\boldsymbol{u}^T + d\boldsymbol{S})\boldsymbol{x},$$

donde $\boldsymbol{e}$ es un vector cuyos elementos son iguales a $1/N$.

$\boldsymbol{e}\boldsymbol{u}^T$ vuelve a ser una matriz estocástica: todos sus elementos están comprendidos entre 0 y 1 (son todos iguales a $1/N$) y sus columnas suman 1.

Si llamamos

$$\boldsymbol{M} = ((1-d)\boldsymbol{e}\boldsymbol{u}^T + d\boldsymbol{S})$$

el sistema se escribe como

$$\boldsymbol{x} = \boldsymbol{M}\boldsymbol{x}$$

$\boldsymbol{M}$ es la combinación convexa (combinación lineal con coeficientes positivos que suman 1) de dos matrices estocásticas, por tanto $\boldsymbol{M}$ también es estocástica: sus valores están comprendidos entre 0 y 1 y está normalizada (sus columnas suman 1), entonces, como consecuencia del teorema de Perron-Frobenius, tiene un valor propio 1 y el resto son menores que 1 en módulo (pueden ser complejos). La solución al sistema anterior es un autovector correspondiente al autovalor 1.

7.2.4. Convergencia del InstanceRank

El cálculo de $\boldsymbol{x}$ (es decir, el InstanceRank) puede hacerse a partir de la expresión 7.1 mediante un proceso iterativo, que converge:

Si

$$\boldsymbol{x}_{k+1} = (1-d)\boldsymbol{u} + d\boldsymbol{S}\boldsymbol{x}_k$$

y

$$\boldsymbol{x}_k = (1-d)\boldsymbol{u} + d\boldsymbol{S}\boldsymbol{x}_{k-1}$$

entonces

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| = d\|\boldsymbol{S}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})\| \leq d\|\boldsymbol{S}\|\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|$$

para cualquier norma vectorial y matricial compatible. Por ejemplo, usando la norma 1,

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{N} |x_i|,$$

y

$$\|\boldsymbol{S}\|_1 = \max_{j=1...N} \sum_{i=1}^{N} |s_{ij}|.$$

En este caso $\|\boldsymbol{S}\|_1 = 1$, y como $d < 1$, se tiene la convergencia asegurada.

7.3. Algoritmo ISR

A continuación presentamos ISR (*Instance Selection Ranking based*), un algoritmo de aprendizaje basado en instancias mediante el que podremos evaluar la utilidad de la ordenación de las instancias de una base de datos generada por InstanceRank aplicada al problema de la selección de instancias. En el diseño de este algoritmo se ha partido de la premisa de que el ranking que aporta InstanceRank puede ser útil para determinar qué instancias de una base de datos son más relevantes, de la misma forma que PageRank mide la importancia de una página web en Internet o TextRank la de una palabra o frase dentro de un texto.

Para el diseño de ISR nos hemos inspirado en las ideas del algoritmo WITS propuesto por Morring y Martinez (2004), un algoritmo de selección de instancias dependiente del orden en el que se procesan las instancias; este algoritmo las considera según un criterio denominado tipicidad. Este concepto fue propuesto por Zhang (1992), como una medida de representatividad de una instancia dentro de una clase y se define como el cociente entre la similitud media de la instancia con todas las de su misma clase (similitud intra-clase) y la similitud media con todas las de distinta clase (similitud extra-clase), donde la similitud de dos instancias x e y se define como $1 - dist(x, y)$.

De la misma forma que el algoritmo WITS se apoya en la tipicidad, ISR se apoya en el ranking producido por InstanceRank para determinar el orden en el que se procesan las instancias. Tomando como base este ranking, ISR produce como salida un conjunto S en el que se ponderan las instancias de una base de datos T. Este peso se puede interpretar como el radio de la esfera de influencia de la instancia y permite al algoritmo obtener un mejor ajuste a la superficie de los datos (Cost y Salzberg, 1993; Paredes, 2006). El pseudocódigo se puede ver en Algoritmo 7.1.

Algoritmo 7.1: ISR: Instance Selection based on Ranking

```
   Input: T: Conjunto de instancias de entrada
   Output: S: Subconjunto de instancias seleccionadas
1  computeInstanceRank(T)
2  sort(T)
3  numClasses := numClasses(T)
4  for i := 1 ... numClasses do bucket_i := EmptyQueue
5  foreach p ∈ T do enqueue(p, bucket_{p.class})
6  S := ∅
7  for i = 1 ... numClasses do bucketNumErrors_i := |bucket_i|
8  totalNumErrors := |T|
9  repeat
10     j := argmax_i(bucketNumErrors_i)
11     p := dequeue(bucket_j)
12     pNumErrors := ∞
13     for k = 0 ... 30 do
14        p.weight := b^k                                  /* b = 1,1 */
15        weightNumErrors := computeErrors(S ∪ p, T)
16        if weightNumErrors < pNumErrors then
17           pNumErrors := weightNumErrors
18           bestWeight := p.weight
19        end if
20     end for
21     if totalNumErrors − pNumErrors ≥ G · |T| then
22        p.weigth := bestWeight
23        S := S ∪ p
24        totalNumErrors := pNumErrors
25        for i := 1 ... numClasses do bucketNumErrors_i := recomputeErrors(bucket_i, S)
26     end if
27     maxBuckect := máx_i(|bucket_i|)
28 until maxBucket = 0 ∨ totalNumErrors = 0
```

El algoritmo trabaja incrementalmente, partiendo de un conjunto de instancias seleccionadas S vacío. La idea general es ir incluyendo en S los elementos que mejoran *significativamente* el número de instancias del conjunto T correctamente clasificadas.

Los principales pasos del algoritmo son:

1. Se calcula el InstanceRank de todas las instancias de T (línea 1).

2. Se ordenan los elementos de T en orden de InstanceRank inverso (línea 2): este será el orden de relevancia.

3. Se crea una cola vacía por cada clase del conjunto T de entrenamiento, a la que le llamamos *bucket*[1]. Se asigna cada instancia de T a su *bucket*, manteniendo el orden de relevancia (líneas 3 – 5).

4. Se realiza el siguiente proceso iterativamente hasta que no haya erro-

[1] *bucket* se traduce por cubeta, pero hemos preferido mantener la palabra original.

res (obviamente, no se puede mejorar S) o hasta que no queden instancias en las colas:

a) Se selecciona la instancia p más relevante del *bucket* j que tenga el mayor número de instancias incorrectamente clasificadas (líneas 10 – 11).

b) Se calcula el peso para p que minimiza el número de errores que se obtienen cuando se añade p al conjunto de instancias seleccionadas (líneas 13 – 20).

c) Si S mejora *significativamente* cuando se le añade p (ponderado con su mejor peso), se selecciona, es decir, se añade efectivamente a S (líneas 21 – 26). El parámetro G define qué es una mejora significativa. De este modo se evita incorporar instancias ruidosas o que produjeran un sobreajuste perjudicial. La mejora necesaria es proporcional al tamaño del conjunto de entrenamiento.

Fase de generalización En la fase de generalización se ponderan las distancias según los pesos asignados a las instancias. Se elije la clase del vecino más cercano ($k = 1$) como clase de la instancia que se desea clasificar, de acuerdo con esa distancia ponderada.

Complejidad del algoritmo Llamando n al número de instancias del conjunto de datos, la complejidad del algoritmo es la siguiente: el cálculo de InstanceRank (línea 1) es $O(n^2)$. El bucle de la línea 9 se ejecuta n veces y el cuerpo del bucle es $O(n)$; por tanto la complejidad total del algoritmo es $O(n^2)$. Los requisitos de almacenamiento son también $O(n^2)$.

7.3.1. Experimentación

Conjuntos de datos utilizados y criterios de comparación

Con el objetivo de conseguir estudiar consistentemente la bondad del algoritmo desarrollado se ha experimentado con las mismas 31 bases de datos con las que se realizó la experimentación del trabajo de Wilson y Martinez (2000); en él se presentó, entre otros, el algoritmo DROP3, que se ha evidenciado como uno de los mejores encontrados hasta la fecha de la familia de los que se apoyan en el vecino o los vecinos más cercanos. Las bases de datos se tomaron del Repositorio de Bases de Datos de Aprendizaje Automático de la Universidad de California en Irvine (Frank

Tabla 7.1 – Características de los conjuntos de datos de experimentación. La primera columna es el nombre del conjunto de datos utilizado en el presente trabajo (y en (Wilson y Martinez, 2000)), la segunda la utilizada en el repositorio; las otras tres columnas son el número de instancias, el número de atributos y el número de clases.

Nombre utilizado	Nombre UCI	#inst	#atrib	#clases
Anneal	Annealing	798	38	6
Australian	Statlog Australian Credit Approval	690	14	2
Breast Cancer (WI)	Breast Cancer Wisconsin	699	9	2
Bridges	Pittsburgh Bridges	108	11	7
Crx	Credit Approval	690	15	2
Echocardiogram	Echocardiogram	132	9	2
Flags	Flags	194	28	8
Glass	Glass Identification	214	9	7
Heart	Statlog Heart	270	13	2
Heart (Cleveland)	Heart Disease Cleveland	303	13	5
Heart (Hungarian)	Heart Disease Hungarian	294	13	5
Heart (Long Beach VA)	Heart Disease Long Beach VA	200	13	5
Heart (More)	Heart Disease New	1541	13	5
Heart (Swiss)	Heart Disease Switzerland	123	13	5
Hepatitis	Hepatitis	155	19	2
Horse Colic	Horse Colic	301	23	2
Image Segmentations	Image Segmentations	420	19	7
Ionosphere	Ionosphere	351	34	2
Iris	Iris	150	4	3
Led Creator+17	Led Display Domain	10000	24	10
LED Creator	Led Display Domain	1000	7	10
Liver (Bupa)	Liver Disorders	345	6	2
Pima Diabetes	Pima Indians Diabetes	768	8	2
Promoters	Molecular Biology (Promoter Gene Sequences)	106	57	2
Sonar	Connectionist Bench (Sonar, Mines vs. Rocks)	208	60	2
Soybean (Large)	Soybean (Large)	307	35	19
Vehicle	Statlog Vehicle Silhouettes	846	18	4
Voting	Congresional Voting Records	435	16	2
Vowel	Connectionist Bench (Vowel Recognition)	528	10	11
Wine	Wine	178	13	3
Zoo	Zoo	90	16	7

y Asuncion, 2010), cuyas características se detallan en el Cuadro 7.1. En la primera columna se indica el nombre que se ha usado en este trabajo, que es el mismo utilizado en (Wilson y Martinez, 2000), en la segunda el nombre utilizado en el Repositorio de la UCI. Las otras tres columnas son el número de instancias, el número de atributos y el número de clases. Cuando ha habido alguna discrepancia entre las características hemos tomado las usadas por los autores en el artículo referido.

Hemos comparado los resultados de ISR con los del vecino más cercano (1NN) y con las técnicas de selección de instancias que aparecen en (Wilson y Martinez, 2000), de donde hemos tomado los valores de precisión y reducción. En nuestros experimentos hemos utilizado validación cruzada estratificada con diez plegamientos (*stratified 10-fold cross validation*), que es la forma de medir más estandarizada en este ámbito de investigación. Hemos obtenido los valores de precisión de la técnica del vecino más cercano del algoritmo IB1 de la implementación de Weka (Witten y Frank, 2005). Hemos implementado también el algoritmo dentro del entorno Weka utilizando su API, lo que facilita la escritura del clasificador y, especialmente, su evaluación.

Hemos medido la precisión como la razón, expresada en porcentajes, entre el número de instancias correctamente clasificadas sobre el número total de instancias en el conjunto de entrenamiento. El tamaño lo hemos medido como el cociente (mostrado en porcentajes) entre el número de instancias seleccionadas usando la técnica y el número total de instancias en el plegamiento correspondiente del conjunto de entrenamiento. Esto significa que valores numéricos más pequeños indican que el tamaño del conjunto de instancias seleccionadas es también más pequeño y, por tanto, el algoritmo tiene mayor capacidad de reducción.

En el trabajo de Wilson y Martinez (2000) se establece la comparación entre 16 técnicas, mencionadas en el capítulo 4. La relación entre las medias sobre los 31 conjuntos de datos de la precisión obtenida por cada una de ellas junto con el tamaño del conjunto de instancias seleccionadas se puede ver gráficamente en la Figura 7.4. Como podemos ver, hay un grupo de técnicas, que hemos rodeado de un círculo, que están mucho más arriba (tienen más precisión) y a la izquierda (el conjunto de instancias seleccionadas es más pequeño) que las demás. Estas técnicas son IB3 (Aha y otros, 1991), Explore (Cameron-Jones, 1995), y DROP2, DROP3, DROP4, DROP5 y DEL (Wilson y Martinez, 2000). Compararemos nuestra técnica con estas, además de con la del vecino más cercano.

Ajuste de parámetros

En el desarrollo de ISR hemos tenido que ajustar varios parámetros hasta alcanzar los resultados óptimos:

- La posible influencia de una instancia sobre sí misma durante el cálculo del InstanceRank.
- El valor del parámetro d en ese cálculo.

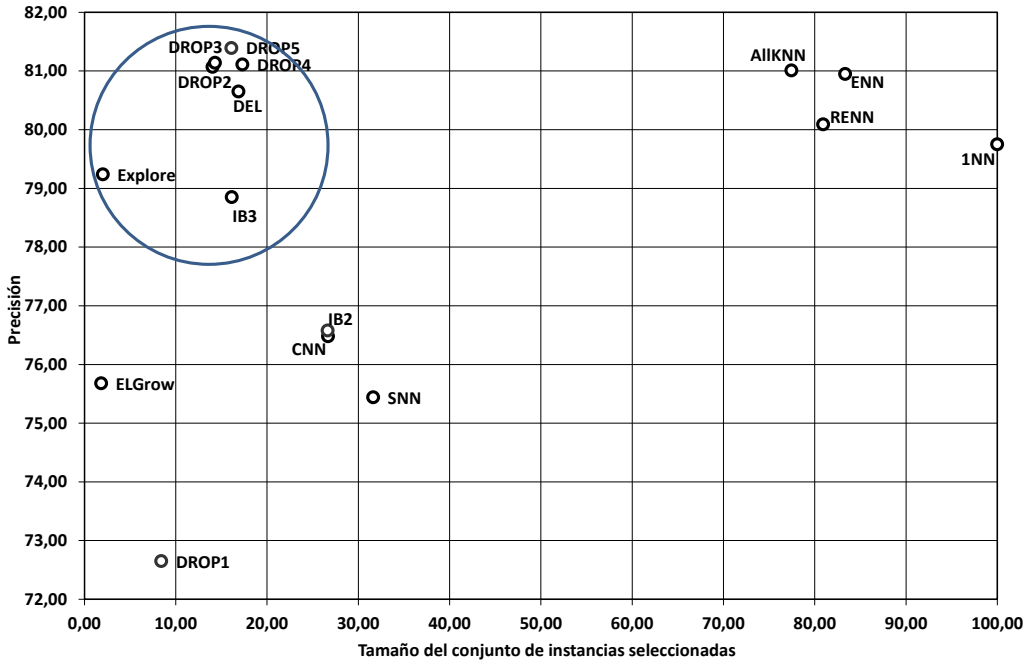

Figura 7.4 – Relación entre las medias de los tamaños de los conjuntos de instancias seleccionadas y precisión en el estudio de Wilson y Martinez (2000). Los valores están representados en porcentajes. Una técnica es mejor cuanto más a la izquierda (menor tamaño) y más arriba (mayor precisión) esté.

- La función de similitud.
- El valor del parámetro k en la función de similitud.
- El parámetro G a partir del que se entiende que una mejora es significativa.

Sobre la influencia de una instancia sobre sí misma observamos que se debía tomar un valor muy próximo a cero, pero no cero, puesto que en caso contrario se producían problemas de convergencia en el caso de que una clase tuviera solo una instancia. Para ajustar todos los demás parámetros se llevaron a cabo experimentos con cada una de las funciones de similitud con un amplio rango de valores de k (entre 1 y 20) y valores de d entre $0{,}65$ y $1{,}0$ con intervalos de $0{,}05$, analizando para cada uno de los casos los valores de la precisión y la reducción más elevadas, siempre tomando la media, para el mismo conjunto de parámetros, sobre los 31 conjuntos de datos considerados. Observamos que para los mejores valores de precisión los valores de reducción se mantenían aproximadamente estables, por lo

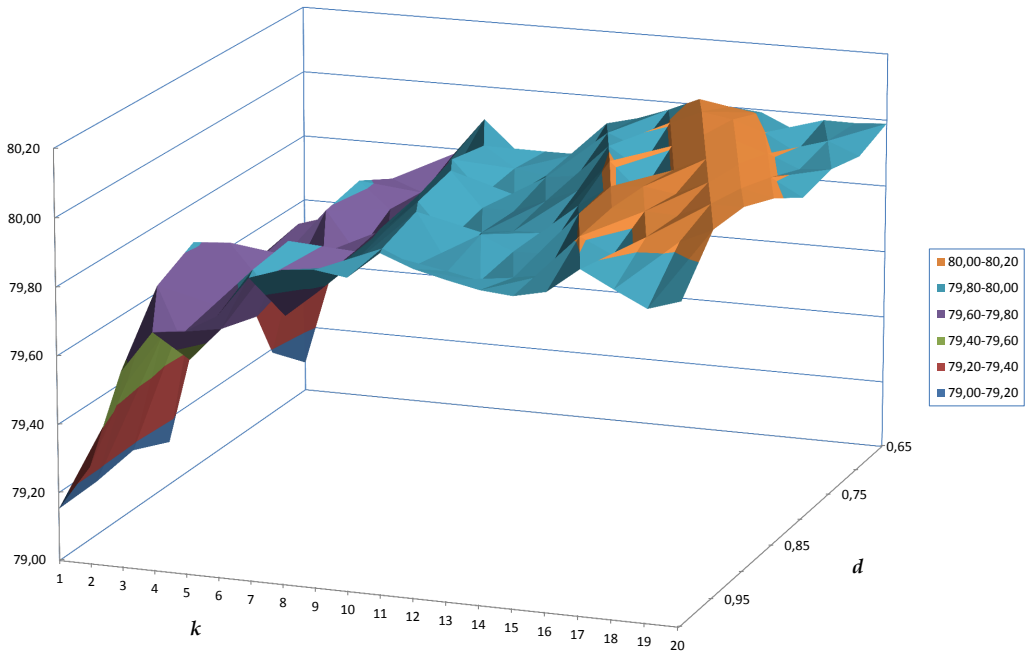

Figura 7.5 – Evolución de la precisión para distintos valores de k y d.

que se tomaron los valores de los parámetros que producían la mejor precisión media.

Los mejores resultados se dieron consistentemente para la función de similitud $\mathrm{sim}(v_1, v_2) = 1 - \mathrm{dist}(v_1, v_2)/k$. En la Figura 7.5 se puede ver la evolución de la precisión dependiendo de los valores de los parámetros k y d para esa función de similitud y en la Figura 7.6 se muestra la evolución de los tamaños de los conjuntos de datos seleccionados. A continuación procedimos a analizar los resultados para esta función de similitud usando intervalos más pequeños, llegando finalmente a la conclusión de que los mejores valores de precisión se obtenían para $k = 14{,}2$ y $d = 0{,}802$. Los resultados experimentales se han obtenido con esos valores.

Probamos también valores entre $0{,}05$ y $0{,}00001$ para el parámetro G, el parámetro que determina si una instancia mejora significativamente la clasificación. Finalmente se tomó el valor $0{,}005$, con el que se obtiene un buen equilibrio entre la precisión y el tamaño del conjunto de instancias seleccionadas. Con este valor, en conjuntos de datos pequeños (con menos de 200 instancias) una sola instancia bien clasificada más es una buena mejora, mientras que en otras más grandes, por ejemplo, Pima, una nueva instancia se acepta solo si con ella se clasifican correctamente al menos cuatro instancias más.

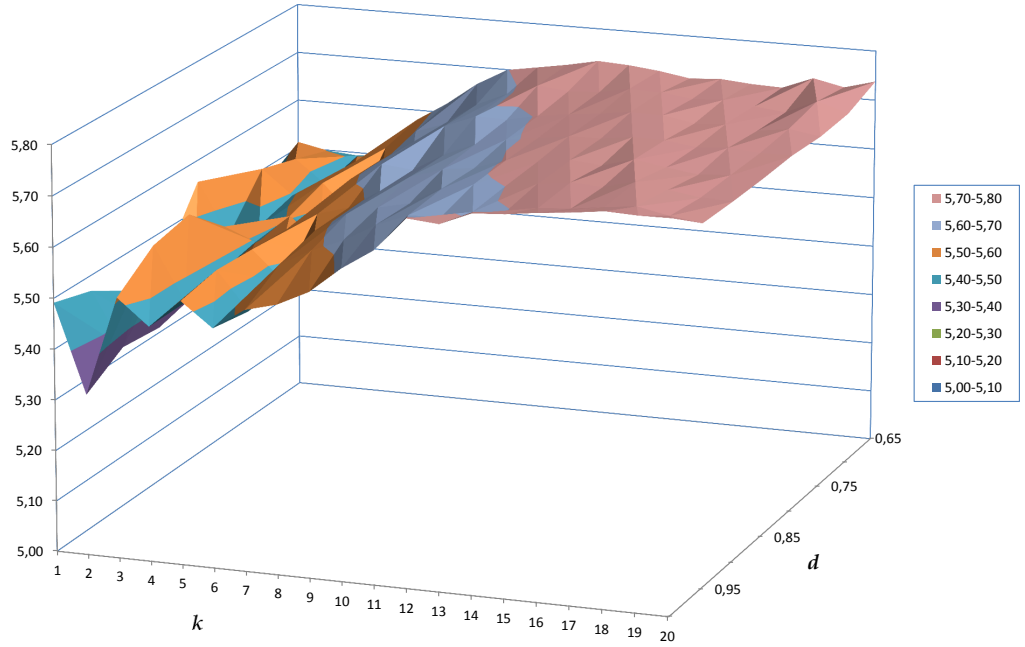

Figura 7.6 – Evolución del tamaño de los conjuntos de instancias seleccionadas para distintos valores de k y d.

También ajustamos el parámetro del cálculo iterativo del InstanceRank que indica el nivel aceptable de error por debajo del cual se detiene la iteración. Tomamos un valor que consigue suficiente precisión en los cálculos como para garantizar la correcta clasificación de las instancias. Analizamos el número de iteraciones necesarias para alcanzar tal nivel de precisión y los resultados mostraron que se mantenían a un nivel más que aceptable (entre tres y diez iteraciones).

7.3.2. Resultados experimentales y análisis estadístico

Precisión En la tabla 7.2 mostramos los valores de precisión obtenidos por cada una de las técnicas con las que hemos comparado la nuestra, que figura en la última columna; los valores están expresado como la razón entre el número de aciertos y el número total de instancias, en porcentaje. Podemos observar que ISR obtiene la mejor precisión en 11 de los 31 conjuntos de datos analizados, empatando en tres ocasiones.

En la comparación con la técnica del vecino más cercano (recordemos que en este caso no se produce ninguna reducción), ISR mejora los resultados en 19 casos, empatando en otros 3. Esto demuestra que ISR tiene un gran poder de edición.

Tabla 7.2 – Comparación de la precisión de todas las técnicas e ISR.

Conjunto de datos	1NN	IB3	Explore	DROP2	DROP3	DROP4	DROP5	DEL	ISR
Anneal	98.12	91.35	91.11	95.61	94.11	94.36	95.24	93.85	90.98
Australian	82.03	85.22	85.80	83.62	83.91	84.78	83.91	84.78	84.35
Breast Cancer (WI)	95.28	96.57	96.71	95.86	96.14	96.28	95.71	96.28	96.14
Bridges	54.72	64.73	57.18	61.18	56.36	57.36	62.82	64.27	64.76
Crx	81.16	86.09	85.51	84.64	85.80	85.51	83.77	83.62	84.78
Echocardiogram	89.19	72.86	94.82	94.82	93.39	94.82	93.39	93.39	90.54
Flags	56.70	49.47	56.16	62.79	61.29	59.58	58.13	56.18	58.25
Glass	70.56	62.14	63.98	65.04	65.02	65.91	65.45	69.59	66.36
Heart	75.19	80.00	81.85	81.85	83.33	81.85	81.11	78.89	83.70
Heart (Cleveland)	76.24	81.16	82.15	79.55	80.84	78.19	79.84	79.49	80.53
Heart (Hungarian)	76.87	79.20	82.30	78.52	80.29	79.22	79.60	77.18	81.97
Heart (Long Beach VA)	69.50	70.00	74.50	70.00	73.50	74.00	73.00	70.00	75.00
Heart (More)	72.10	76.31	73.13	73.98	76.38	74.36	74.63	75.15	76.83
Heart (Swiss)	91.06	93.46	93.46	93.46	93.46	93.46	92.63	92.69	93.50
Hepatitis	80.65	73.08	78.67	80.75	81.87	78.75	83.29	80.00	81.94
Horse Colic	67.44	66.75	67.09	70.74	70.13	67.73	68.45	67.73	80.16
Image Segmentations	93.10	92.14	89.76	92.86	92.62	94.05	89.29	91.90	91.67
Ionosphere	86.32	85.75	80.89	86.60	87.75	86.90	86.90	86.32	91.17
Iris	95.33	94.67	92.67	94.67	95.33	95.33	94.00	93.33	95.33
Led Creator+17	50.84	60.70	72.20	69.20	70.40	69.50	69.80	66.60	42.37
LED Creator	61.80	70.40	72.10	71.80	71.70	71.90	72.00	72.30	74.20
Liver (Bupa)	62.90	58.24	57.65	67.77	60.84	62.60	65.50	61.38	62.90
Pima Diabetes	70.18	69.78	75.27	70.44	75.01	72.53	73.05	71.61	75.78
Promoters	83.96	91.64	91.36	84.91	86.82	86.82	87.00	83.09	70.75
Sonar	86.54	69.38	70.29	80.88	78.00	82.81	79.88	83.29	83.17
Soybean (Large)	91.21	86.63	85.92	86.60	84.97	86.29	83.73	87.27	81.11
Vehicle	69.86	67.62	60.76	67.37	65.85	67.03	70.22	68.10	63.59
Voting	92.41	95.64	94.25	94.50	95.87	95.87	95.86	94.27	94.25
Vowel	99.43	89.57	57.77	91.08	89.56	90.70	93.36	93.17	71.59
Wine	94.94	91.50	95.46	93.24	94.93	94.93	96.08	94.38	96.07
Zoo	96.67	92.22	95.56	88.89	90.00	91.11	95.56	90.00	96.67
Media	79.75	78.85	79.24	81.07	81.14	81.11	81.39	80.65	80.01

En la tabla podemos destacar tres resultados especialmente pobres para nuestra técnica: en Led Creator + 17, Promoters y Vowel, la precisión cae más de un $16\,\%$ en comparación con DROP3. Hemos intentando buscar valores específicos de los parámetros de nuestro algoritmo que mejoraran esta caída, pero no ha sido posible encontrarlos, lo que nos sugiere que estos tres conjuntos de datos tienen alguna característica que los hacen especialmente inadecuados para nuestra técnica.

Hemos realizado un estudio estadístico de los resultados a fin de valorar objetivamente el algoritmo ISR. Comparamos los resultados, en cuanto a precisión de todas las técnicas usando el test de Friedman, que es un método que permite comparar las prestaciones de una nueva técnica frente a otras sobre diferentes conjuntos de datos (Demšar, 2006; García y Herrera, 2008). El test de Friedman ($\chi^2_F = 14{,}871$, $df = 8$, $p = 0{,}062$) muestra

Tabla 7.3 – Rangos del test de Friedman para las precisiones obtenidas.

Técnica	Rango medio
1NN	4.06
IB3	4.21
Explore	4.73
DROP2	4.95
DROP3	5.50
DROP4	5.63
DROP5	5.44
DEL	4.53
ISR	5.95

que no hay diferencias significativas entre las precisiones de las técnicas analizadas (con $\alpha = 0{,}05$). En la Tabla 7.3 se muestran los rangos del test de Friedman. Aunque no hay diferencias significativas entre las técnicas, estos rangos sugieren que el mejor algoritmo, en cuanto a precisión, es ISR, seguido de DROP4, DROP3 y DROP5.

Reducción En la Tabla 7.4 podemos ver el tamaño de los conjuntos de instancias seleccionadas por cada técnica (expresado como la razón entre el número de instancias seleccionadas y el total de instancias, en porcentaje).

El test de Friedman aplicado a los tamaños ($\chi^2_F = 183{,}744$, $df = 8$, $p < 0{,}001$) muestra que en este caso sí hay diferencias significativas entre las técnicas analizadas. Los rangos medios de Friedman se muestran en la Tabla 7.5. Obsérvese que ISR tiene el segundo mejor rango (en este caso valores más pequeños son mejores) tras Explore.

Siguiendo Demšar (2006) y García y Herrera (2008), en este caso debemos realizar un análisis post-hoc, que figura en la misma Tabla 7.5. El significado de cada columna es el siguiente: "Rango medio" es el rango de Friedman, "$R_i - R_{ISR}$" es la diferencia entre el rango de Friedman de la técnica y el rango de Friedman de ISR, "z"$= (R_i - R_{ISR})/\sqrt{\frac{k(k+1)}{6N}}$, donde $k = 9$ es el número de técnicas analizadas y $N = 31$ el número de conjuntos de datos, "p(uni)" es el p-value unilateral para z y "p(bil)" es el bilateral.

De acuerdo con la corrección de Bonferroni-Dunn, las diferencias son significativas si p es menor que $\alpha/(k-1)$, que en este caso es $0{,}00625$ para $\alpha = 0{,}05$. Por tanto ISR es significativamente mejor ($\alpha < 0{,}001$) que todas las técnicas excepto Explore. La diferencia entre ISR y Explore no es

Tabla 7.4 – Tamaño de los conjuntos de datos seleccionados por todas las técnicas e ISR.

Conjunto de datos	1NN	IB3	Explore	DROP2	DROP3	DROP4	DROP5	DEL	ISR
Anneal	100.00	9.79	0.75	8.08	8.65	11.67	9.93	9.30	1.21
Australian	100.00	4.78	0.32	7.28	5.96	7.99	9.18	2.56	0.98
Breast Cancer (WI)	100.00	3.47	0.32	3.13	3.58	4.05	4.07	1.89	0.48
Bridges	100.00	28.83	5.67	17.30	17.60	21.28	22.22	35.64	16.40
Crx	100.00	4.28	0.32	7.31	5.46	7.33	7.68	3.08	1.01
Echocardiogram	100.00	11.57	3.01	10.51	10.66	10.96	9.16	6.91	3.60
Flags	100.00	34.14	2.06	20.62	20.45	27.09	25.26	45.88	14.78
Glass	100.00	33.80	3.53	23.10	23.88	29.54	24.81	38.42	20.40
Heart	100.00	13.58	0.82	12.22	13.62	16.71	16.67	4.73	3.62
Heart (Cleveland)	100.00	11.11	0.73	11.92	12.76	15.26	15.37	13.64	3.34
Heart (Hungarian)	100.00	9.90	0.75	8.80	9.86	11.53	11.15	12.28	2.34
Heart (Long Beach VA)	100.00	4.89	1.11	11.83	4.50	11.72	14.94	19.28	4.22
Heart (More)	100.00	9.36	0.14	10.71	9.14	13.19	14.62	16.81	0.72
Heart (Swiss)	100.00	3.70	0.90	2.53	1.81	2.35	5.42	4.25	0.90
Hepatitis	100.00	5.09	1.29	10.54	7.81	9.75	9.39	7.59	8.46
Horse Colic	100.00	8.49	0.37	8.20	10.30	20.41	14.14	21.82	3.80
Image Segmentations	100.00	16.01	2.43	10.45	10.98	12.41	11.35	11.11	3.12
Ionosphere	100.00	14.59	0.63	7.79	7.06	10.60	9.78	12.88	3.99
Iris	100.00	19.78	2.30	14.22	14.81	14.89	12.15	9.56	6.37
LED Creator+17	100.00	32.31	1.40	12.98	12.66	16.37	14.96	20.90	0.29
LED Creator	100.00	22.04	1.52	11.85	11.93	13.71	12.33	13.92	1.17
Liver (Bupa)	100.00	10.66	0.64	24.77	24.99	32.56	31.08	38.36	2.96
Pima Diabetes	100.00	10.97	0.29	17.59	16.90	21.76	21.95	12.64	1.75
Promoters	100.00	18.12	2.10	13.63	16.67	16.67	12.58	7.34	13.63
Sonar	100.00	12.02	1.07	26.60	26.87	31.20	29.81	29.86	14.21
Soybean (Large)	100.00	30.33	7.78	22.77	25.26	28.41	25.44	24.76	11.15
Vehicle	100.00	28.36	2.47	21.49	23.00	27.88	26.71	32.51	4.36
Voting	100.00	5.44	0.51	4.90	5.11	5.36	7.13	2.02	3.35
Vowel	100.00	36.60	6.65	44.66	45.22	46.02	42.66	36.15	9.26
Wine	100.00	16.60	2.12	11.42	16.11	16.17	9.74	9.05	4.56
Zoo	100.00	29.38	8.40	15.80	20.00	21.60	17.16	18.27	10.74
Media	100.00	16.13	2.01	14.03	14.31	17.30	16.09	16.88	5.72

Tabla 7.5 – Rangos del test de Friedman y análisis post-hoc para los valores de los tamaños de los conjuntos de instancias seleccionadas.

	Rango medio	$R_i - R_{ISR}$	z	p (uni)	p (bil)
1NN	9.00	6.84	9.83312	$<$ 0,001	$<$ 0,001
IB3	5.77	3.61	5.18970	$<$ 0,001	$<$ 0,001
Explore	1.08	-1.08	-1.55260	0.060	0.121
DROP2	4.31	2.15	3.09082	$<$ 0,001	0,002
DROP3	4.66	2.50	3.59398	$<$ 0,001	$<$ 0,001
DROP4	6.63	4.47	6.42603	$<$ 0,001	$<$ 0,001
DROP5	6.00	3.84	5.52035	$<$ 0,001	$<$ 0,001
DEL	5.39	3.23	4.64342	$<$ 0,001	$<$ 0,001
ISR	2.16	0.00	0.00000		

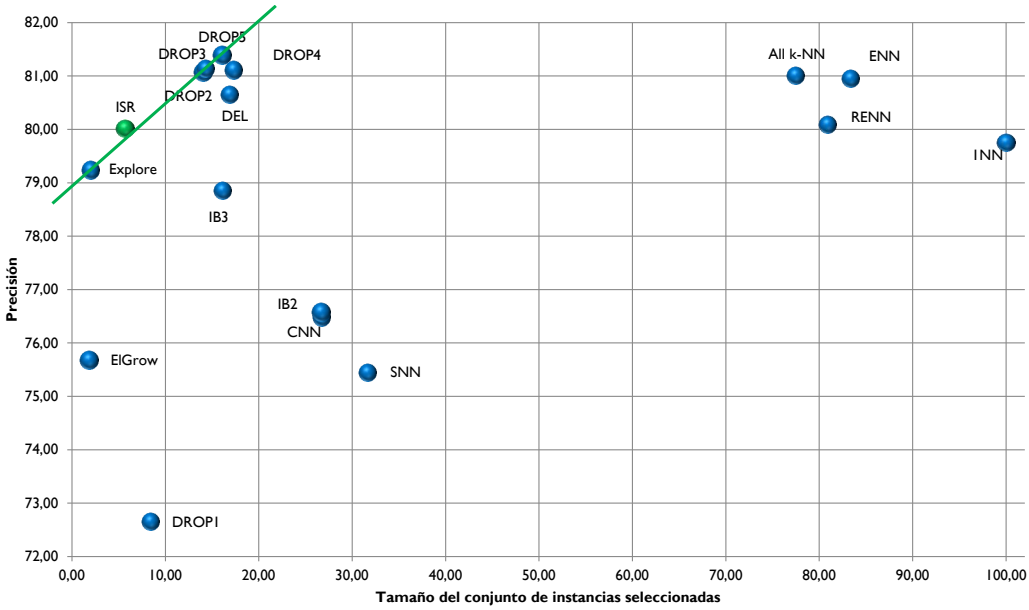

Figura 7.7 – Equilibrio precisión-tamaño de todas las técnicas analizadas, incluyendo ISR. Las técnicas son mejores cuanto más a la izquierda (mayor reducción) y más arriba (mayor precisión) estén. ISR está en la zona adecuada.

significativa ($p = 0{,}121$).

Los resultados de nuestra técnica pueden visualizarse gráficamente en la Figura 7.7; en ella podemos observar el buen resultado obtenido por nuestra técnica: está en un punto de equilibrio entre la reducción y la precisión.

Resumen de los resultados y comentarios adicionales Hemos visto en los apartados anteriores que el test de Friedman seguido del análisis post-hoc demuestran que ISR es significativamente mejor en reducción que todas las demás técnicas excepto Explore.

En términos de precisión, no hay diferencias significativas entre los resultados obtenidos por las técnicas, aunque los rangos del test de Friedman sugieren que la mejor técnica es ISR.

De las afirmaciones anteriores podemos concluir que ISR es mejor en precisión/reducción que todas las técnicas excepto Explore. Vamos a permitimos la licencia estadística de comparar entre sí Explore e ISR. En este caso debe usarse el test de los rangos de Wilcoxon (Demšar, 2006). Encontramos que, con $\alpha = 0{,}05$, ISR es significativamente mejor que Explore en precisión ($z = 1{,}861$, $p = 0{,}032$) aunque es significativamente peor en reducción ($z = 4{,}556$, $p < 0{,}001$).

Debe tenerse en consideración que los resultados de ISR (y los de 1NN) han sido obtenidos usando validación cruzada estratificada con 10 plegamientos, mientras que el resto de los resultados fueron tomados de (Wilson y Martinez, 2000), donde se usa validación cruzada con 10 plegamientos, pero sin estratificación. Hay una ligera diferencia en los resultados según se use o no la estratificación de los datos, pero los resultados estadísticos resultan ser exactamente los mismos: ISR sin usar estratificación tiene una precisión media de $79{,}32\,\%$ y un tamaño medio de los conjuntos de instancias seleccionadas de $5{,}51\,\%$. El test de Friedman muestra que, de nuevo, no hay diferencias significativas entre ISR y los demás algoritmos en precisión ($\chi^2_F = 8{,}839$, $df = 8$, $p = 0{,}356$). En cuanto a los tamaños, el test de Friedman vuelve a mostrar los mismos resultados que antes: hay diferencias significativas ($\chi^2_F = 183{,}983$, $df = 8$, $p < 0{,}001$) y el análisis post-hoc muestra que ISR es mejor que todos los demás algoritmos ($p < 0{,}001$) excepto Explore ($p = 0{,}124$). Como los resultados estadísticos son los mismos, hemos preferido mantener los resultados de ISR usando estratificación porque este es actualmente el estándar en nuestro campo de investigación y facilitar así posibles futuras comparaciones.

También hemos realizado experimentos en los que no utilizábamos distancias ponderadas. Encontramos que la distancia ponderada obtiene mejores resultados en precisión (media de $80{,}01\,\%$ frente a $79{,}39\,\%$) aunque ligeramente peores en reducción ($5{,}68\,\%$ frente a $5{,}57\,\%$). El análisis estadístico llega a las mismas conclusiones, independientemente de que se usen distancias ponderadas o sin ponderar.

La capacidad de reducción del algoritmo puede ser observada en el bien conocido conjunto de datos Pima. El subconjunto de instancias seleccionadas por ISR tiene un tamaño de $1{,}75\,\%$ (respecto del total) en la media de los 10 plegamientos, lo que significa que, de media, tiene aproximadamente 12 elementos en cada uno de los conjuntos seleccionados en los 10 plegamientos (comparados con los aproximadamente 691 elementos que hay en cada plegamiento en el conjunto de entrenamiento). La precisión media obtenida con ellos en la clasificación de los conjuntos de test correspondientes es $75{,}78\,\%$ (el vecino más cercano se queda en $70{,}18\,\%$). Extrapolando los resultados de tamaño al conjunto completo (768 instancias), el clasificador tendría 14 elementos.

Tiempos de ejecución Los tiempos de ejecución fueron acordes con la complejidad del algoritmo ($O(n^2)$), variando desde los 360 milisegundos en Zoo (90 instancias) hasta los 469 minutos en Led creator + 17 (10.000 intancias). El tiempo medio ($16{,}14$ minutos) está fuertemente sesgado por

este último conjunto de datos: si no lo tenemos en consideración, el tiempo medio se reduce a $60{,}82$ segundos.

Robustez de los parámetros En el proceso de ajuste de parámetros escogimos aquellos que obtenían los mejores resultados. En cualquier caso, como reflejan las Figuras 7.5 y 7.6, pequeñas variaciones en sus valores no afectan demasiado la precisión media o el tamaño medio de los conjuntos reducidos. Por ejemplo, para $k = 16{,}0$ y $d = 1{,}0$ la media de las precisiones obtenidas es $79{,}16\,\%$, solo $0{,}85\,\%$ por debajo del resultado para el mejor conjunto de parámetros, mientras que el tamaño medio resulta ser $5{,}65\,\%$, que es incluso ligeramente mejor – $0{,}03\,\%$ – que el obtenido por los mejores parámetros.

7.3.3. Análisis de las instancias seleccionadas

En el presente apartado vamos a analizar la naturaleza de las instancias que selecciona el algoritmo ISR. Vamos a estudiarlo en dos conjuntos de datos, Iris y Pima. Se han escogido estas dos conjuntos por ser ampliamente conocidos en el ámbito del aprendizaje automático.

En el caso de Iris, ISR clasifica el conjunto de datos completo utilizando sólo una media del $6{,}37\,\%$ de los elementos de la base original. Si consideramos el conjunto de datos completo, éste se clasifica con sólo 9 elementos (el 6 % de los elementos, o lo que es lo mismo, obteniendo una reducción del 94 % sobre el tamaño del conjunto original). En la Figura 7.8 se reflejan las instancias de la base de datos Iris proyectadas sobre dos de sus atributos, *petallength* y *petalwidth*, que son los más representativos (son los elegidos por los algoritmos de selección de atributos, como por ejemplo, Correlation-based Feature Subset Selection (CFS) (Hall, 1999). Los puntos huecos representan las 150 instancias, mientras que los rellenos son los nueve elementos seleccionados por ISR, que son los etiquetados como S-Iris-setosa, S-Iris-versicolor y S-Iris-virginica; ISR ha seleccionado una instancia de la clase Iris-setosa, tres de la Iris-versicolor y cinco de la Iris-virginica: Podemos observar que ISR selecciona instancias centrales en los clusters formados por las distintas clases; en este conjunto de datos están muy bien diferenciadas. Con las instancias seleccionadas, ISR consigue una precisión del 98.67 % cuando se evalúa sobre el propio conjunto de entrenamiento (un error de resustitución del 1.33 %).

El otro conjunto de datos que se ha escogido para mostrar gráficamente el comportamiento del algoritmo ISR es Pima. En la Figura 7.9 podemos ver la proyección de este conjunto de instancias sobre los atributos *plas* y

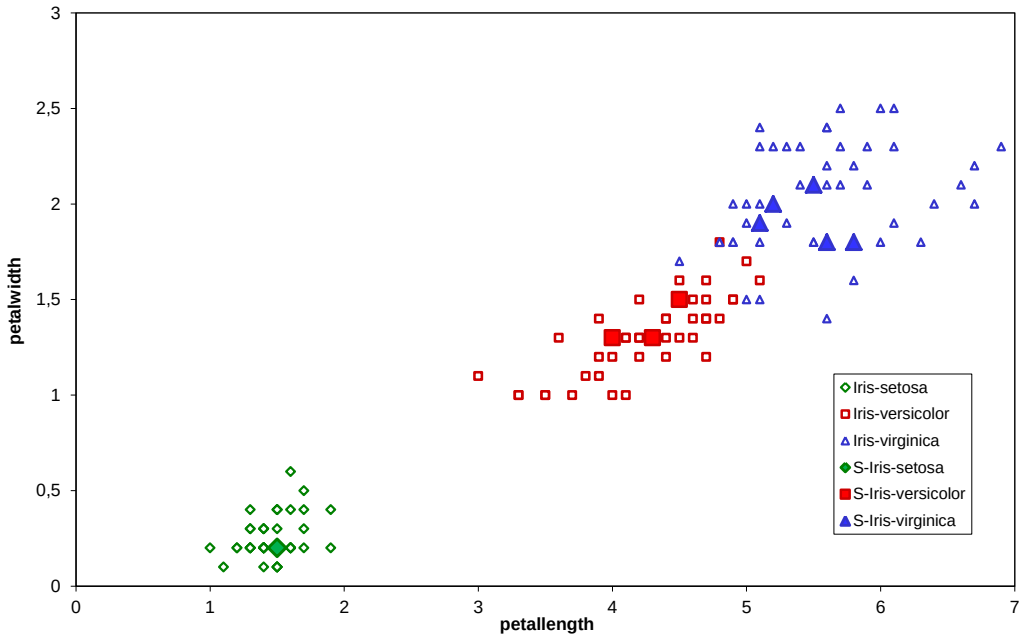

Figura 7.8 – Instancias seleccionadas por ISR en Iris. Se visualizan proyectadas sobre los atributos *petallength* y *petalwidth*. Las instancias con fondo sólido son las seleccionadas por ISR.

mass (el segundo y el sexto). Se han representado como puntos con fondo blanco todas las instancias, mientras que se han representado con fondo obscuro las instancias seleccionadas por el algoritmo (las etiquetadas S-positive y S-negative). Los algoritmos de selección de atributos sugieren el uso de los atributos número 2, 3, 6 y 8, es decir, *plas*, *pres*, *mass* y *pedi*; para representar el conjunto de datos sobre dos dimensiones se han tenido que elegir dos, por lo que se ha seleccionado de entre esos cuatro atributos los dos que seleccionan aquellos algoritmos de selección de atributos que permiten elegir un número concreto de atributos, como FCBF (Yu y Liu, 2003). En este caso son el segundo y el sexto, *plas* y *mass*; en cualquier caso, las fronteras entre las distintas regiones están muy poco definidas.

En el conjunto de datos hay 768 instancias, etiquetadas como negativas (500) o positivas (268). En las instancias seleccionadas por ISR hay 8 entre las negativas y 6 en las positivas, lo que hace un total de 14 instancias, con las que se logra una precisión, medida sobre las 768 instancias, del $81{,}12\,\%$ sobre el conjunto de entrenamiento, o lo que es lo mismo, un error de resustitución del $18{,}88\,\%$. Como se puede observar en la Figura 7.9, los puntos seleccionados son fundamentalemente centrales a las poco definidas regiones que conforman los puntos del conjunto de datos.

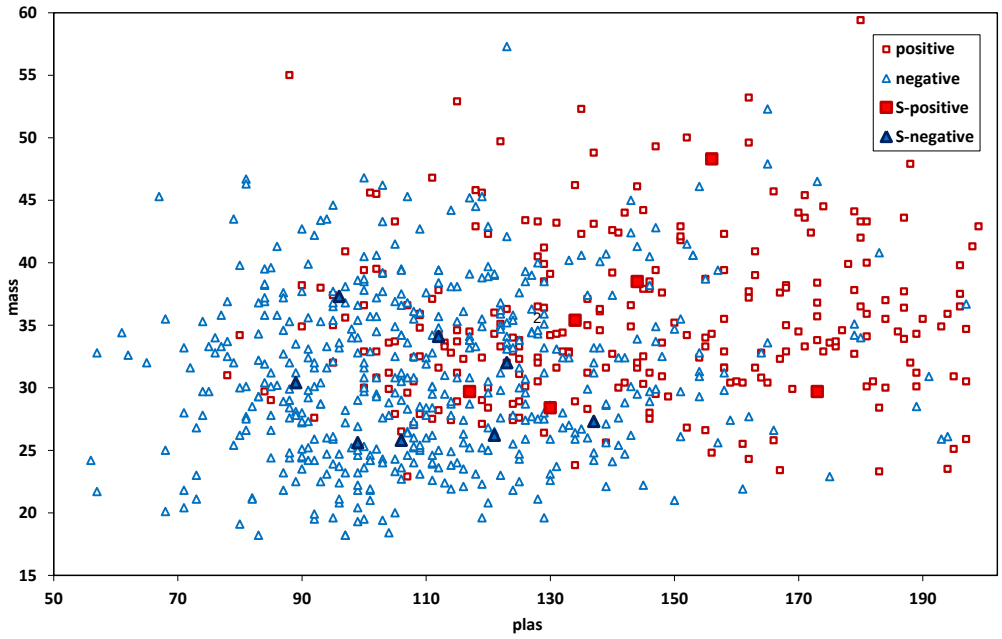

Figura 7.9 – Instancias seleccionadas por ISR en Pima. Se visualizan proyectadas sobre los atributos *plass* y *mass*. Las instancias con fondo sólido son las seleccionadas por ISR.

7.4. Otras aportaciones

Algunos de los resultados de nuestro trabajo sobre selección de instancias han sido aplicados en otros dominios. En concreto, y en relación con la formalización de InstanceRank, hemos colaborado en (Cruz y otros, 2012) en la demostración de la convergencia de su algoritmo PolarityRank, de aplicación, entre otras, en la extracción de opiniones. Nuestro trabajo ha consistido en justificar la existencia de solución y la convergencia de la expresión. En el Apéndice A puede verse el concepto de PolarityRank, la formalización de este junto con la justificación algebraica y la demostración de convergencia.

7.5. Resumen

En este capítulo hemos presentado las otras dos aportaciones que presentamos en esta memoria además del algoritmo de selección de atributos MCBF. Estas son un ranking entre elementos de un conjunto de datos, que hemos denominado InstanceRank, y un algoritmo de selección de instan-

cias, ISR, que utiliza la información proporcionada por el ranking anterior para determinar el orden en el que deben ser procesados los elementos del conjunto de datos de entrenamiento.

Los resultados de los experimentos realizados demuestran que esta aproximación es muy competitiva comparada con los algoritmos de selección de instancias más relevantes basados en el método de los vecinos más cercanos. El análisis estadístico muestra que no hay diferencias significativas entre las técnicas analizadas, lo que en este caso es un excelente resultado dado que se compara nuestra técnica con los mejores algoritmos de selección de instancias. En cuanto a la reducción del tamaño de los conjuntos, nuestra técnica obtiene un resultado excelente, con una tasa de reducción del $94{,}32\,\%$. El análisis estadístico muestra que ISR es significativamente mejor que todas las demás técnicas excepto Explore y que la diferencia entre ISR y Explore no es significativa. ISR consigue un buen equilibrio entre los dos objetivos que estos algoritmos se plantean: precisión y reducción.

Queremos enfatizar la capacidad de edición (la eliminación de instancias ruidosas) de la técnica desarrollada: reduciendo el número de instancias en el clasificador, mejora el resultado obtenido usando la técnica del vecino más cercano en 19 de los 31 conjuntos de datos considerados, empatando en otras tres.

Parte V

Consideraciones finales

Capítulo 8

Conclusiones y Oportunidades

8.1. Introducción

8.2. Resumen de resultados

A lo largo del presente trabajo hemos desarrollado dos algoritmos, uno de Selección de Atributos y otro de Selección de Instancias, basándonos para ello en la representación de los datos como grafos.

Para llegar hasta estos resultados en esta memoria de tesis hemos presentado la etapa, dentro de la Minería de Datos, a la que pertenecen los problemas que queríamos abordar, la del Preprocesamiento de Datos; hemos visto también el problema de la Clasificación, estudiando medidas de calidad de los clasificadores. Como sabíamos que tendríamos que medir distancias entre instancias, hemos dedicado una sección a estudiarlas.

Posteriormente hemos visto algunas definiciones de utilidad de la estructura de datos que íbamos a utilizar a lo largo de nuestro trabajo: los grafos, poniendo un énfasis especial en las medidas de centralidad y de ranking, que son las que nos resultarían de utilidad en una de nuestras aportaciones.

Con estos antecedentes, hemos abordado nuestras soluciones a los dos problemas planteados: la Selección de Atributos y la de Instancias (Capítulos 5 y 7), presentando previamente en cada uno de los dos casos una defi-

nición del problema a resolver, las características generales de las soluciones aportadas en cada ámbito y un estudio del estado del arte.

Vamos a ver con más detalle cada una de las soluciones aportadas, el algoritmo MCBF para la Selección de Atributos (Capítulo 5) y la medida de ranking InstanceRank y el algoritmo ISR para la Selección de Instancias (Capítulo 7).

8.2.1. Selección de Atributos

Esta es una de las formas que existen de acelerar el funcionamiento de muchos de los algoritmos de Minería de Datos: al reducir la dimensionalidad (a lo ancho) del conjunto de datos, se mejora el rendimiento. Está encuadrada dentro de la Etapa de Preprocesamiento de Datos.

En el Capítulo 4 hemos visto la definición del problema, algunos conceptos (relevancia y redundancia) en los que se basan normalmente las soluciones a este problema, la estructura general de estas, un repaso al estado del arte en este ámbito y una posible clasificación de las soluciones aportadas.

Una vez realizado este trabajo hemos planteado nuestro algoritmo, MCBF (*Min-Cut Base Feature Selection*), que se basa en los cortes mínimos en flujos. Para poder aplicar estos cortes mínimos hemos representado las correlaciones entre los atributos y entre estos y la clase dentro de un grafo. Por esta razón hemos empezado el capítulo viendo qué es un corte mínimo dentro de un grafo y cómo se calculan las correlaciones entre atributos y entre estos y la clase. Una vez realizado este trabajo preliminar, presentamos finalmente nuestro algoritmo.

Hemos estudiado los resultados obtenidos con nuestro algoritmo, comparándolos con otros dos y con el vecino más cercano. En la experimentación hemos utilizados 38 conjuntos de datos del repositorio de la UCI. En la tabla 8.1 presentamos las medias en tamaño y precisión obtenidos por el vecino más cercano (1NN), las dos técnicas de selección de atributos con las que nos hemos comparado (FCBF y CFS), y la nuestra (MCBF).

Hemos realizado el test de Friedman sobre los resultados, encontrando que había diferencias significativas tanto en precisión y como en reducción, por lo que en ambos casos pasamos a realizar un análisis post-hoc. En precisión (Tabla 5.3) nuestra técnica muestra diferencias significativas respecto a todas las demás. En cuanto a tamaño del conjunto de atributos seleccionados (Tabla 5.5), nuestra técnica es significativamente mejor que la del vecino más cercano, lo que resulta obvio, aunque no muestra diferencias significativas con las otras dos.

Tabla 8.1 – Resumen de las medias en precisión y en el tamaño de los conjuntos de atributos seleccionados por todas las técnicas frente a MCBF.

Técnica	Precisión	Tamaño del conjunto de atributos seleccionados
1NN	83,52	100,00
FCBF	81,50	22,08
CFS	83,92	27,56
MCBF	86,66	26,26

8.2.2. Selección de Instancias

Esta es la otra forma que existe de acelerar el funcionamiento de muchos algoritmos de Minería de Datos: reducir la dimensionalidad (a lo alto) del conjunto de datos. Está encuadrada, al igual que la anterior, en la Etapa de Preprocesamiento de Datos.

Antes de plantear nuestra solución hemos visto, en el Capítulo 6, una definición del problema, qué tipos de métodos hay, qué tipo de instancias se pueden querer eliminar/retener, cómo evaluar las distintas técnicas y el estado del arte.

En el Capítulo 7 hemos presentado en primer lugar un algoritmo que permite establecer un ranking de instancias, InstanceRank. Aunque los experimentos realizados demostraban que el algoritmo convergía siempre a una solución, hemos hecho una demostración matemática de la existencia de solución y de la convergencia del algoritmo.

Una vez obtenido este ranking, hemos presentado un algoritmo de Selección de Instancias, ISR (*Instance Selection Ranking based*), que se utiliza InstanceRank como orden de presentación y consideración de instancias. Hemos comparado los resultados obtenidos sobre 31 conjuntos de datos del repositorio de la UCI, tanto en precisión como en el tamaño del conjunto de instancias seleccionadas, con siete de los mejores algoritmos de selección de instancias existentes, además del vecino más cercano. Las medias de las precisiones y del número de instancias seleccionadas por cada técnica puede verse en la tabla 8.2.

Del análisis estadístico de los resultados obtenidos aplicando el test de Friedman a las precisiones podemos concluir no tiene diferencias significativas con las demás técnicas, aunque las medias de los rangos de Friedman sugieren que nuestra técnica es la cuarta mejor, tras 1NN, IB3 y Explore. En cuanto a las reducciones, sí se observan la nuestra es mejor significativamente que todas las demás excepto Explore, con la que no tiene diferencias

Tabla 8.2 – Resumen de las medias en precisión y en el tamaño de los conjuntos de instancias seleccionadas por todas las técnicas frente a ISR.

Técnica	Precisión	Tamaño del conjunto de instancias seleccionadas
1NN	79,75	100,00
IB3	78,85	16,13
Explore	79,24	2,01
DROP2	81,07	14,03
DROP3	81,14	14,31
DROP4	81,11	17,30
DROP5	81,39	16,09
DEL	80,65	16,88
ISR	80,01	5,72

significativas (Tabla 7.5). Nuestra técnica muestra un buen equilibrio entre las dos magnitudes a optimizar: la precisión y el tamaño del conjunto de instancias seleccionadas.

8.3. Validación de las hipótesis

Recordemos que nuestra primera hipótesis, la principal era

Los grafos tienen una potencia expresiva que permite modelar y resolver sobre ellos problemas aparentemente lejanos.

Vistos los resultados obtenidos, podemos afirmar que la hipótesis es cierta: hemos utilizado grafos para resolver con éxito cada uno de los dos problemas abordados.

En la selección de atributos hemos conseguido modelar el problema como un grafo, utilizando para ello las correlaciones entre atributos y entre estos y la clase, lo que valida la Hipótesis 2:

Las relaciones entre los atributos de un conjunto de datos pueden modelarse mediante un grafo; utilizaremos las correlaciones entre los atributos y entre estos y las clases como pesos de las aristas correspondientes.

Por otra parte, el algoritmo desarrollado compite favorablemente con otros algoritmos de selección de atributos, como hemos visto en el Capítu-

lo 5 y, más abreviadamente en el apartado 8.2.1. Por tanto podemos validar la Hipótesis 3:

A partir del grafo anterior desarrollaremos un algoritmo de selección de atributos.

En cuanto a la selección de instancias (Capítulo 7), hemos obtenido una medida de la representatividad de las instancias dentro de un conjunto de datos, InstanceRank, lo que valida nuestra Hipótesis 4:

Las instancias de un conjunto de datos se pueden modelar como grafo haciendo que cada una de ellas sea un nodo del grafo. Las aristas serán las relaciones de vecindad entre ellas. Este modelo permitirá establecer la importancia de los nodos dentro del grafo, o lo que es lo mismo, de las instancias en el conjunto de datos.

A partir de este ranking hemos desarrollado el algoritmo ISR que, como se ha visto en el Capítulo 5 y, más resumidamente, en el apartado 8.2.2, consigue unos resultados muy equilibrados entre precisión obtenida y tamaño del conjunto de instancias seleccionadas, lo que valida la Hipótesis 5:

Los nodos que sean más relevantes según esta medida serán más representativos del conjunto de nodos de su clase.

De la validación de las Hipótesis 2, 3, 4 y 5 se sigue la validez de la Hipótesis 1: el gran potencial de los grafos en este ámbito.

8.4. Oportunidades

A partir de los resultados del presente trabajo se abren una serie de oportunidades de desarrollo que pensamos que permitirían explotar y ampliar los resultados, así como rellenar los huecos que todo trabajo tiene que tener.

Vamos a detallar estas oportunidades según el algoritmo desarrollado, aunque hay una línea común a ambas: aunque es la práctica habitual aplicar estos algoritmos a los conjuntos de datos de UCI, que nos permiten establecer comparaciones, creemos que sería interesante aplicarlos a problemas del mundo real, para el que al fin y al cabo desarrollamos nuestro trabajo.

8.4.1. MCBF

En cuanto a la selección de atributos estamos trabajando en la integración de MCBF en el software Weka (Witten y Frank, 2005). Nos encontramos con el obstáculo de que el algoritmo utilizado para el cálculo del corte mínimo hi_pr) (Cherkassky y Goldberg, 1994), se distribuye actualmente por una empresa `http://www.igsystems.com/hipr/index.html` que lo licencia exclusivamente para uso comercial, aunque permite su utilización para pruebas. Nosotros lo hemos empleado en su versión original, en lenguaje C, pero también lo hemos traducido a Java para conseguir una mejor integración en Weka. En cualquier caso, para evitar los posibles inconvenientes que pudiéramos tener con la licencia, hemos pensado utilizar la clase StoerWagnerMinimumCut (Stoer y Wagner, 1994) del paquete `http://jgrapht.org`; este algoritmo fue implementado originalmente dentro de la librería LEDA (Mehlhorn y Näher, 1995).

Otra idea es aplicar el algoritmo casi-lineal para el cálculo del corte mínimo de Karger (Karger y Stein, 1996; Karger, 2000) para acelerar la velocidad del algoritmo.

Creemos que también sería oportuno explorar otras funciones de transformación de las correlaciones entre atributos y clase en pesos de las aristas; pensamos que con los valores adecuados se puede mejorar aún más la precisión y reducción del algoritmo.

8.4.2. InstanceRank e ISR

En cuanto a la selección de instancias, queremos destacar en primer lugar que el trabajo previo al publicado en Vallejo y otros (2010), basado en GlobalInstanceRank (Vallejo y otros, 2007) ha sido explotado con éxito en el campo del aprendizaje semisupervisado (Ruiz y otros, 2010).

En dos publicaciones se ha planteado la utilización de InstanceRank su utilización para eliminar los atributos no relevantes en un algoritmo para extraer información de la web (Fernández y Sleiman, 2011; Fernández y otros, 2011).

Estamos trabajando en la integración de ISR dentro de la distribución de Weka (Witten y Frank, 2005). Tenemos el problema de que este software no incluye ningún algoritmo de selección de instancias, por lo que la clase que realiza la evaluación (Evaluation) no da información sobre el número de instancias seleccionadas (en nuestro trabajo lo hemos resuelto incluyendo una clase MyEvaluation que sí daba esa información).

En cuanto a posibles mejoras en rendimiento de InstanceRank (y por tanto, de ISR), hemos pensado que establecer un umbral por debajo del

cual se consideren que no hay aristas, aceleraría el cálculo, al no ser el grafo completo (de hecho, pensamos que pasaría a ser bastante disperso).

Hemos pensado también incluir disimilitudes (Pękalska y otros, 2006; Kim, 2011) además de similitudes, lo que podría enriquecer los resultados, de un modo similar al utilizado en (Cruz y otros, 2012).

Los pobres resultados obtenidos por ISR en Led Creator + 17, Promoters y Wovel, nos han hecho pensar en otra posible línea de investigación puede abrirse utilizando InstanceRank, en combinación con otras medidas, como un medio para caracterizar la estructura interna de los conjuntos de datos.

Parte VI

Apéndice

Apéndice A

PolarityRank: Justificación algebraica y Convergencia

A.1. Introducción

Veremos a continuación la definición del PolarityRank, una medida de la influencia de los nodos de un grafo en el que los nodos pueden recibir recomendaciones positivas o negativas de otros nodos (Cruz y otros, 2012). Veremos la definición, la justificación algebraica y la demostración que su convergencia.

A.2. Definición

Sea un grafo dirigido $G = (V, E)$ donde V es un conjunto de nodos y E un conjunto de aristas dirigidas entre dos nodos. Cada arista de E contiene un valor real asociado o peso, distinto de cero, siendo p_{ji} el peso asociado a la arista que va del nodo v_j al v_i. Se define la operación $Out(v_i)$, que devuelve el conjunto de índices j de los nodos para los que existe una arista saliente desde v_i. Se definen las operaciones $In^+(v_i)$ y $In^-(v_i)$, que devuelven los conjuntos de índices j de los nodos para los que existe una arista entrante hacia v_i cuyo peso sea positivo o negativo, respectivamente. Definimos el PolarityRank positivo y negativo de un nodo v_i (fórmula A.1), donde los valores de e^+ son mayores que cero para ciertos nodos que actúan como semillas positivas y cero para el resto de nodos, y los valores de e^- son mayores que cero para ciertos nodos que actúan como semillas negativas y cero para el resto de nodos.

$$
\begin{aligned}
PR^+(v_i) &= (1-d)e_i^+ + \\
&\quad + d\Big(\sum_{j \in In^+(v_i)} \frac{p_{ji}}{\sum_{k \in Out(v_j)} |p_{jk}|} PR^+(v_j) + \\
&\quad + \sum_{j \in In^-(v_i)} \frac{-p_{ji}}{\sum_{k \in Out(v_j)} |p_{jk}|} PR^-(v_j) \Big) \\
PR^-(v_i) &= (1-d)e_i^- + \\
&\quad + d\Big(\sum_{j \in In^+(v_i)} \frac{p_{ji}}{\sum_{k \in Out(v_j)} |p_{jk}|} PR^-(v_j) + \\
&\quad + \sum_{j \in In^-(v_i)} \frac{-p_{ji}}{\sum_{k \in Out(v_j)} |p_{jk}|} PR^+(v_j) \Big)
\end{aligned}
\tag{A.1}
$$

La constante d es un factor de amortiguación, en todos los casos positivos y menor que 1. La suma de los valores de e^+ por un lado y de e^- por otro sea igual al número de nodos del grafo; de esta forma obtenemos valores de PolarityRank con magnitudes similares a las del algoritmo original de PageRank.

Obsérvese que para que las fracciones no sean indeterminadas, debe haber al menos un elemento no cero en cada fila, es decir, todos los nodos tienen que tener grado de salida positivo.

A.3. Justificación algebraica

Vamos a estudiar a continuación la fundamentación algebraica del PolarityRank. Sea $n = |V|$, el número de nodos del grafo. Denotamos $\boldsymbol{P}$ a la matriz de los (p_{ij}).

Llamemos $p_j = \sum_{k \in Out(v_j)} |p_{jk}|$, esto es, la suma de los pesos de las aristas que comienzan en v_j. En la matriz $\boldsymbol{P}$, p_j es la suma de los valores absolutos de la fila j; p_j puede escribirse como $p_j = \sum_{k=1}^{n} |p_{jk}|$

El PolarityRank puede ahora ser expresado como

$$
\begin{aligned}
PR^+(v_i) &= (1-d)e_i^+ + \\
&\quad + d\Big(\sum_{j \in In^+(v_i)} \frac{p_{ji}}{p_j} PR^+(v_j) + \sum_{j \in In^-(v_i)} \frac{-p_{ji}}{p_j} PR^-(v_j) \Big) \\
PR^-(v_i) &= (1-d)e_i^- + \\
&\quad + d\Big(\sum_{j \in In^+(v_i)} \frac{p_{ji}}{p_j} PR^-(v_j) + \sum_{j \in In^-(v_i)} \frac{-p_{ji}}{p_j} PR^+(v_j) \Big)
\end{aligned}
$$

Definamos ahora la matriz $\boldsymbol{Q} = \boldsymbol{P}^t$, es decir, la traspuesta de P (si la matriz $\boldsymbol{P}$ describe un grafo no dirigido entonces $\boldsymbol{Q} = \boldsymbol{P}$). Llamemos $q_j = \sum_{k=1}^{n} |q_{kj}|$, es decir, la suma de los elementos de la columna j de $\boldsymbol{Q}$. Obviamente, $p_j = q_j$.

Definamos ahora dos matrices $\boldsymbol{Q}^+ = (q_{ij}^+)$ y $\boldsymbol{Q}^- = (q_{ij}^-)$ como

$$\begin{aligned} q_{ij}^+ &= \begin{cases} q_{ij} & \text{si } \ q_{ij} > 0 \\ 0 & \text{en otro caso} \end{cases} \\ q_{ij}^- &= \begin{cases} -q_{ij} & \text{si } \ q_{ij} < 0 \\ 0 & \text{en otro caso} \end{cases} \end{aligned}$$

q_j puede verse como la suma de los elementos de la columna j de la matriz $\boldsymbol{Q}^+$ más la suma de los de la misma columna de la matriz $\boldsymbol{Q}^-$.

Ahora podemos expresar PolarityRank del siguiente modo:

$$\begin{aligned} PR^+(v_i) &= (1-d)e_i^+ + d\Big(\sum_{j=1}^{n} \frac{q_{ij}^+}{q_j} PR^+(v_j) + \sum_{j=1}^{n} \frac{q_{ij}^-}{q_j} PR^-(v_j)\Big) \\ PR^-(v_i) &= (1-d)e_i^- + d\Big(\sum_{j=1}^{n} \frac{q_{ij}^-}{q_j} PR^+(v_j) + \sum_{j=1}^{n} \frac{q_{ij}^+}{q_j} PR^-(v_j)\Big) \end{aligned}$$

Definimos a continuación la matriz $\boldsymbol{A}^+ = (a_{ij}^+)$ como $a_{ij}^+ = q_{ij}^+/q_j$ y la matriz $\boldsymbol{A}^- = (a_{ij}^-)$ como $a_{ij}^- = q_{ij}^-/q_j$, entonces

$$\begin{aligned} PR^+(v_i) &= (1-d)e_i^+ + d\Big(\sum_{j=1}^{n} a_{ij}^+ PR^+(v_j) + \sum_{j=1}^{n} a_{ij}^- PR^-(v_j)\Big) \\ PR^-(v_i) &= (1-d)e_i^- + d\Big(\sum_{j=1}^{n} a_{ij}^- PR^+(v_j) + \sum_{j=1}^{n} a_{ij}^+ PR^-(v_j)\Big) \end{aligned}$$

A continuación vamos a hacer algunas definiciones para simplificar la notación. Sea $m = 2n$. Definimos el vector $\boldsymbol{x}$ de $m \times 1$ elementos como

$$\boldsymbol{x} = \begin{bmatrix} PR^+(v_1) \\ \vdots \\ PR^+(v_n) \\ PR^-(v_1) \\ \vdots \\ PR^-(v_n) \end{bmatrix}$$

y el vector $\boldsymbol{e}$ de $m \times 1$ elementos como

$$\boldsymbol{e} = \begin{bmatrix} e_1^+ \\ \vdots \\ e_n^+ \\ e_1^- \\ \vdots \\ e_n^- \end{bmatrix}$$

Finalmente, definimos la matriz $\boldsymbol{A}$ de $m \times m$ elementos

$$\boldsymbol{A} = \left[\begin{array}{c|c} \boldsymbol{A}^+ & \boldsymbol{A}^- \\ \hline \boldsymbol{A}^- & \boldsymbol{A}^+ \end{array}\right]$$

Con los vectores y matrices auxiliares anteriores, podemos reescribir las ecuaciones de PolarityRank (A.1) mucho más fácilmente como

$$\boldsymbol{x} = (1-d)\boldsymbol{e} + d\boldsymbol{A}\boldsymbol{x} \tag{A.2}$$

$\boldsymbol{A}$ es una matriz estocástica: Todos los elementos de $\boldsymbol{A}$ están entre 0 y 1 (ambos incluidos):

$$a_{ij}^+ = \frac{q_{ij}^+}{\sum_{k=1}^n q_{kj}^+ + \sum_{k=1}^n q_{kj}^-} \quad \text{y} \quad a_{ij}^- = \frac{q_{ij}^-}{\sum_{k=1}^n q_{kj}^+ + \sum_{k=1}^n q_{kj}^-}$$

y, obviamente, la suma de los elementos de cada columna es 1.

e_i^+ y e_i^- se han elegido de manera que $\sum_{i=1}^n e_i^+ = \sum_{i=1}^n e_i^- = n$ y positivos. Por tanto $\|\boldsymbol{e}\|_1 = m$.

Definamos ahora el vector de $m \times 1$ elementos $\boldsymbol{f} = \boldsymbol{e}/m$ (esto es, los elementos de e_i^+ y e_i^- divididos por m), y el vector $\boldsymbol{u}$ como el vector de $m \times 1$ unos. Analicemos la matriz $\boldsymbol{f}\boldsymbol{u}^t$:

$$\boldsymbol{f}\boldsymbol{u}^t = \begin{bmatrix} e_1^+/m \\ \vdots \\ e_n^+/m \\ e_1^-/m \\ \vdots \\ e_n^-/m \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} e_1^+/m & \dots & e_1^+/m \\ \vdots & & \vdots \\ e_n^+/m & \dots & e_n^+/m \\ e_1^-/m & \dots & e_1^-/m \\ \vdots & & \vdots \\ e_n^-/m & \dots & e_n^-/m \end{bmatrix}$$

Sus elementos están comprendidos entre 0 y 1, y la suma de los elementos de cada columna es 1, por tanto $\boldsymbol{f}\boldsymbol{u}^t$ es una matriz estocástica.

Si $\boldsymbol{x}$ está normalizado, de modo que $\|\boldsymbol{x}\|_1 = m$ entonces $\boldsymbol{u}^t\boldsymbol{x} = m$, siempre que los elementos de $\boldsymbol{x}$ sean positivos; pero si comenzamos dándole valores no negativos, éstos siempre permanecerán siendo no negativos puesto que los elementos de $\boldsymbol{A}$ también lo son.

La ecuación (A.2) se puede escribir entonces como:

$$\begin{aligned}\boldsymbol{x} &= (1-d)\boldsymbol{e} + d\boldsymbol{A}\boldsymbol{x} = \\ &= (1-d)\boldsymbol{e}\frac{1}{m}\boldsymbol{u}^t\boldsymbol{x} + d\boldsymbol{A}\boldsymbol{x} = \\ &= (1-d)\boldsymbol{f}\boldsymbol{u}^t\boldsymbol{x} + d\boldsymbol{A}\boldsymbol{x} = \\ &= ((1-d)\boldsymbol{f}\boldsymbol{u}^t + d\boldsymbol{A})\boldsymbol{x}\end{aligned}$$

Llamemos $\boldsymbol{B} = ((1-d)\boldsymbol{f}\boldsymbol{u}^t + d\boldsymbol{A})$. La expresión que define el PolarityRank (A.1) se puede escribir entonces como

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{x} \tag{A.3}$$

$\boldsymbol{B}$ es la combinación convexa (combinación lineal donde los dos coeficientes son positivos y suman 1) de dos matrices estocásticas, por tanto $\boldsymbol{B}$ es también estocástica. De aquí, y como resultado del teorema de Perron-Frobenius la ecuación (A.3) tiene el autovalor 1, y el resto de sus autovalores tienen módulo menor que 1 (pueden ser complejos). El sistema anterior tiene una solución que es precisamente el autovector correspondiente al autovalor 1.

A.4. Convergencia

El cálculo de PolarityRank (el vector $\boldsymbol{x}$) puede realizarse a partir de la expresión (A.1) mediante un calculo iterado que vamos a demostrar que converge.

Como hemos visto, la expresión (A.1) equivale a la expresión (A.2): $\boldsymbol{x} = (1-d)\boldsymbol{e} + d\boldsymbol{A}\boldsymbol{x}$. Expresemos como $\boldsymbol{x}_k$ como el término k-ésimo del cálculo iterado del PolarityRank. Es

$$\boldsymbol{x}_{k+1} = (1-d)\boldsymbol{e} + d\boldsymbol{A}\boldsymbol{x}_k$$

y

$$\boldsymbol{x}_k = (1-d)\boldsymbol{e} + d\boldsymbol{A}\boldsymbol{x}_{k-1}$$

Por tanto

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| = d\|\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})\| \leq d\|\boldsymbol{A}\|\|(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})\|$$

para cualquier norma compatible con el vector y la matriz. Por ejemplo, usando la norma 1,

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{m} |x_i|$$

y

$$\|\boldsymbol{A}\|_1 = \max_{j=1\ldots m} \sum_{i=1}^{m} |a_{ij}|$$

Es $\|\boldsymbol{A}\|_1 = 1$ (recordemos que es estocástica), y $d < 1$, luego

$$\lim_{k\to\infty} \|x_{k+1} - x_k\| \to 0.$$

Por tanto la convergencia está asegurada.

Bibliografía

AHA, D. W.: «Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms». *International Journal of Man-Machine Studies*, 1992.

AHA, D. W.; KIBLER, D. y ALBERT, M. K.: «Instance-Based Learning Algorithms». *Machine Learning*, 1991, **6**, pp. 37–66.

APPLEGATE, DAVID L.; BIXBY, ROBERT E.; CHVATAL, VASEK y COOK, WILLIAM J.: *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

BAYES, THOMAS: «An essay towards solving a problem in the doctrine of chances». *Philosophical Transactions of the Royal Society of London*, 1763, **53**, pp. 370–418.

BELLMAN, RICHARD: *Dynamic Programming*. Dover Publications, 1957.

BEZDEK, JAMES C. y KUNCHEVA, LUDMILA I.: «Nearest prototype classifier designs: An experimental study». *International Journal of Intelligent Systems*, 2001, **16(12)**, pp. 1445–1473.

BLUM, AVRIM y CHAWLA, SHUCHI: «Learning from Labeled and Unlabeled Data using Graph Mincuts». En: *ICML*, pp. 19–26, 2001.

BLUM, AVRIM L. y LANGLEY, PAT: «Selection of relevant features and examples in machine learning». *Artificial Intelligence*, 1997, **97**, pp. 245–271.

BOYKOV, YURI y KOLMOGOROV, VLADIMIR: «An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision». *IEEE Transactions on PAMI*, 2004, **26(9)**, pp. 1124–1137.

BRAY, TIM: «Measuring the Web». En: *Proceedings of the fifth international World Wide Web conference on Computer networks and ISDN systems*, pp. 993–1005, 1996.

BRIGHTON, H. y MELLISH, C.: «On the consistency of information filters for lazy learning algorithms». En: J.M. Zytkow y J. Rauch (Eds.), *Principles of Data Mining and Knowledge Discovery, 3rd European Conference*, pp. 283–288. Prague, Czech Republic, 1999.

——: «Advances in Instance Selection in Instance-Based Learning Algorithms». *Data Mining and Knowledge Discovery*, 2002, **6(2)**, pp. 153–172.

BRIN, SERGEY: «Near Neighbor Search in Large Metric Spaces». En: *Proceedings of the 21st VLDB Conference*, pp. 574–584, 1995.

BRINKMEIER, MICHAEL: «Minimum Cuts of Simple Graphs in Almost Always Linear Time». En: *WALCOM*, pp. 226–237, 2009.

CAMERON-JONES, R. M.: «Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing». En: *Proc. Eighth Australian Joint Conf. Artificial Intelligence*, pp. 293–301, 1995.

CARUANA, R. y FREITAG, D.: «How Useful is Relevance?» En: *Proceedings of the 1994 AAAI Fall Symposium on Relevance*, , 1994.

CHEKURI, CHANDRA S.; GOLDBERG, ANDREW V.; KARGER, DAVID R.; LEVINE, MATTHEW S. y STEIN, CLIFF: «Experimental Study of Minimum Cut Algorithms». *Informe técnico 96–132*, NECI TR 96–132, 1996.

CHERKASSKY, BORIS V. y GOLDBERG, ANDREW V.: «On Implementing Push-Relabel Method For The Maximum Flow Problem». *Algorithmica*, 1994, **19**, pp. 390–410.

CORMEN, THOMAS H.; LEISERSON, CHARLES E.; RIVEST, RONALD L. y STEIN, CLIFFORD: *Introduction to Algorithms*. The MIT Press, 2ł edición, 2001.

CORSO, GIANNA M. DEL; GULLI, ANTONIO y ROMANI, FRANCESCO: «Fast PageRank Computation via a Sparse Linear System». *Internet Mathematics*, 2005, **2(3)**, pp. 251–273.

COST, SCOTT y SALZBERG, STEVEN: «A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features». *Machine Learning*, 1993, **10**, pp. 57–78.

COVER, T. M.: «Estimation by nearest neighbor rule». *IEEE Transactions on Information Theory*, 1968, **14**, pp. 50–55.

COVER, T. M. y HART, P. E.: «Nearest neighbor pattern classification». *IEEE Transactions on Information Theory*, 1967, **13(1)**, pp. 21–27.

CRUZ, FERMÍN L.; VALLEJO, CARLOS G.; ENRÍQUEZ, FERNANDO y TROYANO, JOSÉ A.: «PolarityRank: Finding an equilibrium between followers and contraries in a network». *Information Processing and Management*, 2012, **48(2)**, pp. 271–282.

DEMŠAR, JANEZ: «Statistical Comparisons of Classifiers over Multiple Data Sets». *Journal of Machine Learning Research*, 2006, **7**, pp. 1–30.

DIJKSTRA, EDSGER WYBE: «A note on two problems in connexion with graphs». *Numerische Mathematik*, 1959, **1(1)**, pp. 269–271.

DOMINGOS, PEDRO y PAZZANI, MICHAEL J.: «On the Optimality of the Simple Bayesian Classifier under Zero-One Loss». *Machine Learning*, 1997, **29(2–3)**, pp. 103–130.

EDMONDS, J. y KARP, R. M.: «Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems». *J. Assoc. Comput. Mach.*, 1972, **19**, pp. 248–264.

EULER, LEONARD: «Solutio problematis ad geometriam situs pertinentis». *Commentarii academiae scientiarum Petropolitanae*, 1741, **8**, pp. 128–140.

EVEN, SHIMON: *Graph Algorithms*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716780445.

FAYYAD, USAMA M. y IRANI, KEKI B.: «Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning». En: *IJCAI*, pp. 1022–1029, 1993.

FERNÁNDEZ, GRETEL; SLEIMAN, HASSAN S.; CORCHUELO, RAFAEL y FRANTZ, RAFAEL Z.: «On Mining DOM Trees to build Information Extractors». En: *International Conference on Internet Computing*, pp. 363–367, 2011.

FERNÁNDEZ, GRETEL y SLEIMAN, HASSAN: «An Experiment on Using Datamining Techniques to Extract Information from the Web». En: Juan Corchado; Javier Pérez; Kasper Hallenborg; Paulina Golinska y Rafael Corchuelo (Eds.), *Trends in Practical Applications of Agents and Multiagent*

Systems, volumen 90 de *Advances in Intelligent and Soft Computing*, pp. 169–176. Springer Berlin / Heidelberg, 2011.

FIX, E. y HODGES, J. L.: «Discriminatory analysis, nonparametric discrimination consistency properties». *Informe técnico 4*, US Air Force, School of Aviation Medicine, Randolph Field, TX, 1951.

——: «Discriminatory analysis, nonparametric discrimination: small sample performance». *Informe técnico 11*, US Air Force, School of Aviation Medicine, Randolph Field, TX, 1952.

FLAKE, GARY WILLIAM; TARJAN, ROBERT ENDRE y TSIOUTSIOULIKLIS, KOSTAS: «Graph Clustering and Minimum Cut Trees». *Internet Mathematics*, 2003, **1(4)**, pp. 385–408.

FLOYD, ROBERT W.: «Algorithm 97: Shortest path». *Communications of the ACM*, 1962, **5(6)**, p. 345. ISSN 0001-0782.

FORD, L. R., JR. y FULKERSON, D. R.: «Maximal flow through a network». *Canadian Journal of Math.*, 1956, **8**, pp. 399–404.

——: *Flows in Networks*. Princeton University Press, 1962.

FRANK, A. y ASUNCION, A.: «UCI Machine Learning Repository», 2010. `http://archive.ics.uci.edu/ml`

GARCÍA, SALVADOR y HERRERA, FRANCISCO: «An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets"for all Pairwise Comparisons». *Journal of Machine Learning Research*, 2008, **9**, pp. 2677–2694.

GATES, G.W.: «The reduced nearest neighbor rule». *IEEE Transactions on Information Theory*, 1972.

GHISELLI, EDWIN ERNEST: *Theory of psychological measurement*. McGraw-Hill, 1964.

GINSBERG, J.; MOHEBBI, M. H.; PATEL, R. S.; BRAMMER, L.; SMOLINSKI, M. S. y BRILLIANT, L.: «Detecting influenza epidemics using search engine query data». *Nature*, 2009.

GOLDBERG, A. V. y TARJAN, R. E.: «A New Approach to the Maximum-Flow Problem». *J. ACM*, 1988, **35(4)**, pp. 921–940.

GÓMEZ-BALLESTER, EVA; MICÓ, LUISA y ONCINA, JOSE: «Some approaches to improve tree-based nearest neighbour search algorithms». *Pattern Recognition*, 2006, **39(2)**, pp. 171–179.

GONZÁLEZ NAVARRO, FÉLIX FERNANDO: *Feature Selection in cancer research: Microarray Gene Expression and in vivo 1H-MRS domains*. Tesis doctoral, Universitat Politécnica de Catalunya, 2011.

GREIG, D.; PORTEOUS, B. y SEHEULT, A.: «Exact maximum a posteriori estimation for binary images». *Journal of the Royal Statistical Society, Series B*, 1989, **51(2)**, pp. 271–279.

HALL, M. A.: *Correlation-based Feature Subset Selection for Machine Learning*. Tesis doctoral, University of Waikato, Hamilton, New Zealand, 1999.

HALL, MARK A.: «Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning». En: *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, 2000.

HALL, MARK A. y SMITH, LLOYD. A.: «Practical feature subset selection for machine learning». En: C. McDonald (Ed.), *Proceedings of the 21st Australasian Computer Science Conference*, pp. 181–191. Springer, 1998.

HAN, JIAWEI; KAMBER, MICHELINE y PEI, JIAN: *Data Mining: Concepts and Techniques*. Morgan Kaufman Publishers, 2012.

HARRIS, T. E. y ROSS, F. S.: «Fundamentals of a Method for Evaluating Rail Net Capacities». *Informe técnico*, Research Memorandum RM-1573, The RAND Corporation, Santa Monica, California, 1955.

HART, P. E.: «The condensed nearest neighbor rule». *IEEE Transactions on Information Theory*, 1968, **12**, pp. 515–516.

HE, HUAHAI y SINGH, AMBUJ K.: *Managing and Mining Graph Data*. volumen 40 de *Advances in Database Systems*, Capítulo Query Language and Access Methods for Graph Databases, pp. 125–160. Springer, 2010.

HITCHCOCK, F. L.: «The distribution of a product form several sources to numerous localities». *Journal of Mathematics and Phisics*, 1941, **20**, pp. 224–230.

HOLLAND, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

HUGHES, G.: «On the mean accuracy of statistical pattern recognizers». *Information Theory, IEEE Transactions on*, 1968, **14(1)**, pp. 55–63.

HUNT, EARL B.; MARIN, JANET y STONE, PHILIP J.: *Experiments in induction*. Academic Press, 1966.

INOKUCHI, AKIHIRO; WASHIO, TAKASHI y MOTODA, HIROSHI: «An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data». En: *Principles of Data Mining and Knowledge Discovery, 4th European Conference*, volumen 1910 de *Lecture Notes in Computer Science*, pp. 13–23. Springer, 2000.

JOHN, GEORGE H.; KOHAVI, RON y PFLEGER, KARL: «Irrelevant Features and the Subset Selection Problem». En: *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13*, pp. 121–129. Morgan Kaufmann, 1994.

JOHNSON, TROY A.; EIGENMANN, RUDOLF y VIJAYKUMAR, T. N.: «Min-Cut Program Decomposition for Thread-Level Speculation». En: *PLDI'04*, pp. 59–70. ACM, Washington, DC, USA, 2004.

KAHN, A. B.: «Topological sorting of large networks». *Communications of the ACM*, 1962, **5(11)**, pp. 558–562. ISSN 0001-0782. doi: 10.1145/368996.369025.
http://doi.acm.org/10.1145/368996.369025

KAMVAR, SEPANDAR; HAVELIWALA, TAHER; MANNING, CHRISTOPHER y GOLUB, GENE: «Exploiting the Block Structure of the Web for Computing PageRank». *Technical Report 2003-17*, Stanford InfoLab, 2003.
http://ilpubs.stanford.edu:8090/579/

KARGER, DAVID R.: «Minimum cuts in near-linear time». *J. ACM*, 2000, **47(1)**, pp. 46–76.

KARGER, DAVID R. y STEIN, CLIFFORD: «A New Approach to the Minimum Cut Problem». *J. ACM*, 1996, **43(4)**, pp. 601–640.

KERNIGHAN, B. W. y LIN, S.: «An Efficient Heuristic Procedure for Partitioning Graphs». *The Bell system technical journal*, 1970, **49(1)**, pp. 291–307.

KIM, SANG-WOON: «An empirical evaluation on dimensionality reduction schemes for dissimilarity-based classifications». *Pattern Recognition Letters*, 2011, **32(6)**, pp. 816 – 823. ISSN 0167-8655. doi:

10.1016/j.patrec.2011.01.009.
`http://www.sciencedirect.com/science/article/pii/S0167865511000183`

KIM, SANG-WOON y OOMMEN, B. JOHN: «A brief taxonomy and ranking of creative prototype reduction schemes». *Pattern Anal. Appl.*, 2003, **6(3)**, pp. 232–244.

KIRA, KENJI y RENDELL, LARRY A.: «The Feature Selection Problem: Traditional Methods and a New Algorithm». En: *AAAI*, pp. 129–134, 1992a.

——: «A Practical Approach to Feature Selection». En: *Proceedings of the Ninth International Workshop on Machine Learning*, ML '92, pp. 249–256. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-247-X, 1992b.

KLEINBERG, JON M.: «Authoritative Sources in a Hyperlinked Environment». *Journal of the ACM*, 1999, **46(5)**, pp. 604–632.

KOHAVI, RON; BECKER, BARRY y SOMMERFIELD, DAN: «Improving Simple Bayes». En: *Proceedings of European Conference on Machine Learning (ECML'97)*, ECML'97, 1997. Poster.

KOHAVI, RON y JOHN, GEORGE H.: «Wrappers for Feature Subset Selection». *Artificial Intelligence*, 1997, **97(1)**, pp. 273–324.

KOLLER, D. y SAHAMI, M.: «Toward Optimal Feature Selection». En: Lorenza Saitta (Ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pp. 284–292. Morgan Kaufmann Publishers, 1996.

KONDOR, RISI IMRE y LAFFERTY, JOHN D.: «Diffusion Kernels on Graphs and Other Discrete Input Spaces». En: *ICML*, pp. 315–322, 2002.

KONONENKO, IGOR: «Estimating Attributes: Analysis and Extensions of RELIEF». En: *European Conference on Machine Learning*, pp. 171–182, 1994.

KRUSKAL, JOSEPH. B.: «On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem». *Proceedings of the American Mathematical Society*, 1956, **7(1)**, pp. 48–50.

KURAMOCHI, MICHIHIRO y KARYPIS, GEORGE: «Frequent Subgraph Discovery». En: *Proceedings of the 2001 IEEE International Conference on Data Mining*, pp. 313–320. IEEE Computer Society, 2001.

——: «Finding Frequent Patterns in a Large Sparse Graph». *Data Minining and Knowledge Discovery*, 2005, **11(3)**, pp. 243–271.

LANGLEY, P.: «Selection of relevant features in machine learning». En: *AAAI Fall Symposium on Relevance*, pp. 140–144., 1994.

LEE, MONG-LI; YANG, LIANG HUAI; HSU, WYNNE y YANG, XIA: «XClust: clustering XML schemas for effective integration». En: *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management*, pp. 292–299, 2002.

LIU, BING: *Web Data Mining*. Springer, 2007.

LIU, HUAN y MOTODA, HIROSHI: *Feature extraction, construction and selection : a data mining perspective*. Capítulo Less is More, pp. 3–12. Kluwer Academic, 1998a.

——: *Feature selection for knowledge discovery and data mining*. Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 1998b. ISBN 9780792381983.

——: *Instance Selection and Construction for Data Mining*. Capítulo Data Reduction Via Instance Selection, pp. 3–20. Kluwer Academic Publishers, Boston, USA, 2001.

LIU, HUAN; MOTODA, HIROSHI; SETIONO, RUDY y ZHAO, ZHENG: «Feature Selection: An Ever Evolving Frontier in Data Mining». *Journal of Machine Learning Research - Proceedings Track*, 2010, **10**, pp. 4–13.

LIU, HUAN y SETIONO, RUDY: «A Probabilistic Approach to Feature Selection - A Filter Solution.» En: *Ninth International Conference on Machine Learning*, pp. 319–327, 1996.

LIU, HUAN y YU, LEI: «Towards Integrating Feature Selection Algorithms for Classification and Clustering». *IEEE Transactions on Knowledge and Data Engineering*, 2005, **17(4)**, pp. 491–502.

MARCHIORI, MASSIMO: «The Quest for Correct Information on the Web: Hyper Search Engines». *Computer Networks*, 1997, **29(8-13)**, pp. 1225–1236.

MEHLHORN, KURT y NÄHER, STEFAN: «LEDA: a platform for combinatorial and geometric computing». *Commun. ACM*, 1995, **38(1)**, pp. 96–102. ISSN 0001-0782.

MICHALSKI, R. S.: «On the Selection of Representative Samples from Large Relational Tables for Inductive Inference». *Informe técnico M.D.C.1.1.9*, U. Illinois (Chicago circle), 1975.

MIHALCEA, RADA y TARAU, PAUL: «TextRank: Bringing Order into Texts». En: Dekang Lin y Dekai Wu (Eds.), *Proceedings of EMNLP 2004*, pp. 404–411. Association for Computational Linguistics, Barcelona, Spain, 2004.

MILGRAM, STANLEY: «The small-world problem». *Psychology Today*, 1967, **1(1)**, pp. 61–67.

MILLER, ALAN J.: «Selection of Subsets of Regression Variables». *Journal of the Royal Statistical Society Series A*, 1984, **147**, pp. 389–425.

MORRING, BRENT D. y MARTINEZ, TONY R.: «Weighted Instance Typicality Search (WITS): A Nearest Neighbor Data Reduction Algorithm». *Intelligent Data Analysis*, 2004, **8(1)**, pp. 61–78.

NARENDRA, P.M. y FUKUNAGA, K.: «A Branch and Bound Algorithm for Feature Subset Selection». *IEEE Transactions on Computers*, 1977, **26**, pp. 917–922. ISSN 0018-9340. doi: http://doi.ieeecomputersociety.org/10.1109/TC.1977.1674939.

NEUHAUS, M. y BUNKE, H.: «Automatic learning of cost functions for graph edit distance». *Information Sciences*, 2007, **177(1)**, pp. 239–247.

NEWMAN, M. E. J. y PARK, JUYONG: «Why social networks are different from other types of networks». *Physical Review E*, 2003, **68**, p. 036122. doi: 10.1103/PhysRevE.68.036122.

PAGE, LAWRENCE; BRIN, SERGEY; MOTWANI, RAJEEV y WINOGRAD, TERRY: «The PageRank Citation Ranking: Bringing Order to the Web». *Informe técnico*, Stanford Digital Library Technologies Project, 1998.

PAL, S.K. y PAL, A.: *Pattern recognition: from classical to modern approaches*. World Scientific, 2001. ISBN 9789810246846.

PANG, BO y LEE, LILLIAN: «A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Bases on Minimum Cuts». En: *Proceeding of the ACL*, pp. 271–278, 2004.

PAPADIMITRIU, C. H. y BENTLEY, J. L.: «A Worst-Case Analysis of Nearest Neighbor Searching by Projection». *Lecture Notes in Computer Science*, 1980, **85**, pp. 470–482.

PAREDES, ROBERTO: «Learning Weighted Metrics to Minimize Nearest-Neighbor Classification Error». *IEEE Trans. Pattern Anal. Mach. Intell.*, 2006, **28(7)**, pp. 1100–1110. ISSN 0162-8828.

PĘKALSKA, ELŻBIETA; DUIN, ROBERT P. W. y PACLÍK, PAVEL: «Prototype Selection for Dissimilarity-based Classifiers». *Pattern Recognition*, 2006, **39(2)**, pp. 189–208.

POLLACK, ANDREW: «Redefining Disease, Genes and All». The New York Times, 6 de Mayo, 2008.
`http://www.nytimes.com/interactive/2008/05/05/science/20080506_DISEASE.html`

PRIM, R. C.: «Shortest Connection Networks and Some Generalizations». *Bell System Technical Journal*, 1957, **36**, pp. 1389–1401.

QUINLAN, J. R.: «Induction of Decision Trees». En: Jude W. Shavlik y Thomas G. Dietterich (Eds.), *Readings in Machine Learning*, Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

QUINLAN, J. ROSS: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

RICH, ELAINE y KNIGHT, KEVIN: *Artificial intelligence (2. ed.)*. McGraw-Hill, 1991. ISBN 978-0-07-052263-3.

RIQUELME, JOSÉ C.; AGUILAR-RUIZ, JESÚS S. y TORO, MIGUEL: «Finding representative patterns with ordered projections». *Pattern Recognition*, 2003.

RITTER, G. L.; WOODRUFF, H. B.; LOWRY, S. R. y ISENHOUR, T. L.: «An algorithm for a selective nearest neighbor decision rule». *IEEE Transactions on Information Theory*, 1975.

RUIZ, CARLOS; VALLEJO, CARLOS G.; SPILIOPOULOU, MYRA y MENASALVAS, ERNESTINA: «Automated constraint selection for semi-supervised clustering algorithm». En: *CAEPIA'09: Proceedings of the Current topics in artificial intelligence, and 13th conference on Spanish association for artificial intelligence*, pp. 151–160. Springer-Verlag, Berlin, Heidelberg. ISBN 3-642-14263-X, 978-3-642-14263-5, 2010.

RUIZ, ROBERTO: *Heurísticas de Selección de Atributos para Datos de Gran Dimensionalidad*. Tesis doctoral, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, 2006.

RUIZ, ROBERTO; AGUILAR-RUIZ, JESÚS S. y RIQUELME, JOSÉ C.: *Computational Methods of Feature Selection*. Capítulo Efficient Incremental-Ranked Feature Selection in Massive Data. Chapman & Hall/CRC, 2008.

RUIZ, ROBERTO; RIQUELME, JOSÉ C. y AGUILAR-RUIZ, JESÚS S.: «Incremental wrapper-based gene selection from microarray data for cancer classification». *Pattern Recognition*, 2006, **39(12)**, pp. 2383 – 2392. ISSN 0031-3203. doi: 10.1016/j.patcog.2005.11.001.

RUIZ, ROBERTO; RIQUELME, JOSÉ C. y AGUILAR-RUIZ, JESÚS S.: «Projection-based measure for efficient feature selection». *J. Intell. Fuzzy Syst.*, 2002, **12(3,4)**, pp. 175–183. ISSN 1064-1246.

SAEYS, YVAN; INZA, IÑAKI y LARRAÑAGA, PEDRO: «A review of feature selection techniques in bioinformatics». *Bioinformatics*, 2007, **23(19)**, pp. 2507–2517.

SAIGO, HIROTO; NOWOZIN, SEBASTIAN; KADOWAKI, TADASHI; KUDO, TAKU y TSUDA, KOJI: «gBoost: a mathematical programming approach to graph classification and regression». *Machine Learning*, 2009, **75(1)**, pp. 69–89.

SCHRIJVER, ALEXANDER: «On the history of the transportation and maximum flow problems». *Mathematical Programming*, 2002, **91(3)**, pp. 437–445.

SHANNON, C.E. y WEAVER, W.: *The mathematical theory of communication*. Número v. 1 en The Mathematical Theory of Communication. University of Illinois Press, 1949.

SMYTH, B. y KEANE, M.T.: «Remembering to forget». En: C. S. Mellish (Ed.), *IJCAI-95, Proceedings of the Fourteenth International Conference on Artificial Intelligence*, volumen 1, pp. 377–382, 1995.

SPROULL, R. F.: «Refinements to Nearest-Neighbor Searching in k-Dimensional Trees». *Algorithmica*, 1991, **6**, pp. 579–589.

STANFILL, C. y WALTZ, D.: «Toward memory-based reasoning». *Communications of the ACM*, 1986, **29**, pp. 1213–1228.

STOER, M. y WAGNER, F.: *A simple min cut algorithm*. Freie Universität Berlin, Fachbereich Mathematik: Ser. B, Informatik. Freie Univ., Fachbereich Mathematik, 1994.

THRUN, SEBASTIAN; SAUL, LAWRENCE K. y SCHÖLKOPF, BERNHARD (Eds.): *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*. MIT Press, 2004. ISBN 0-262-20152-6.

TOMEK, I.: «An experiment with the edited nearest-neighbor rule». *IEEE Transactions on Systems, Man, and Cybernetics*, 1976.

TRIGUERO, ISAAC; DERRAC, JOAQUÍN; GARCÍA, SALVADOR y HERRERA, FRANCISCO: «A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification». *IEEE Transactions on Systems, Man, and Cybernetics—PART C: Applications and Reviews*, 2012.

TSAY, A. A.; LOVEJOY, W. S. y KARGER, DAVID R.: «Random Sampling in Cut, Flow, and Network Design Problems». *Mathematics of Operations Research*, 1999, **24(2)**, pp. 383–413. doi: 10.1287/moor.24.2.383. http://dx.doi.org/10.1287/moor.24.2.383

VALLEJO, CARLOS G.; TROYANO, JOSÉ A. y ORTEGA, F. JAVIER: «WIRS: Un Algoritmo de Reducción de Instancias Basado en Ranking». En: *CAEPIA'07: Proceeding of the 12th Conference of the Spanish Association for Artificial Intelligence*, pp. 327–336, 2007.

VALLEJO, CARLOS G.; TROYANO, JOSÉ A. y ORTEGA, F. JAVIER: «InstanceRank: Bringing order to datasets». *Pattern Recognition Letters*, 2010, **31(2)**, pp. 133 – 142. ISSN 0167-8655. doi: DOI:10.1016/j.patrec.2009.09.022.

VAN LEEUWEN, J.: «Graph Algorithms». En: J.van Leeuwen (Ed.), *Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity*, pp. 525–631. Elsevier, Amsterdam, 1990.

VAPNIK, V.: *The Nature of Statistical Learning Theory*. Springer, 1995.

VIDAL, E.: «An algorithm for finding nearest neighbours in (approximately) constant average time». *Pattern Recognition Letters*, 1986, **4**, pp. 145–157.

WANG, HAIXUN y AGGARWAL, CHARU C.: *Managing and Mining Graph Data*. volumen 40 de *Advances in Database Systems*, Capítulo A Suervey of Algorithms for Keyword Search on Graph Data, pp. 249–273. Springer, 2010.

WANG, HUI; BELL, DAVID y MURTAGH, FIONN: *Feature extraction, construction and selection: a data mining perspective*. Capítulo Relevance approach to feature subset selection, pp. 85–97. Kluwer Academic Publishers, 1998.

WARSHALL, STEPHEN: «A Theorem on Boolean Matrices». *Journal of the ACM*, 1962, **9(1)**, pp. 11–12. ISSN 0004-5411. doi: 10.1145/321105.321107. http://doi.acm.org/10.1145/321105.321107

WASSERMAN, S. y FAUST, K.: *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. ISBN 9780521387071.

WILSON, D. L.: «Asymptotic properties of nearest neighbor rules using edited data». *IEEE Transactions on Systems, Man and Cybernetics*, 1972, **2(3)**, pp. 408–421.

WILSON, D. RANDALL y MARTINEZ, TONY R.: «Improved heterogeneous distance functions». *Journal of Artificial Intelligence Research*, 1997, **6(1)**, pp. 1–34.

——: «Reduction Techniques for Instance-Based Learning Algorithms». *Machine Learning*, 2000, **38**, pp. 257–286.

WITTEN, IAN H. y FRANK, EIBE: *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2005. ISBN 0120884070.

YIANILOS, PETER N.: «Data structures and algorithms for nearest neighbor search in general metric spaces». En: *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pp. 311–321. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. ISBN 0-89871-313-7, 1993.

YU, JEFFREY XU y CHENG, JIEFENG: *Managing and Mining Graph Data*. volumen 40 de *Advances in Database Systems*, Capítulo Graph Reachability Querys: A Survey, pp. 181–215. Springer, 2010.

YU, LEI y LIU, HUAN: «Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution». En: *20th International Conference on Machine Learning*, pp. 856–863, 2003.

ZHANG, JIANPING: «Selecting typical instances in instance-based learning». En: *ML92: Proceedings of the ninth international workshop on Machine*

learning, pp. 470–479. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-5586-247-X, 1992.

ZHAO, ZHENG; MORSTATTER, FRED; SHARMA, SHASHVATA; ALELYANI, SALEM; ANAND, ANEETH y LIU, HUAN: «Advancing Feature Selection Research - ASU feature selection repository». *Informe técnico*, Arizona State University, 2010.

ZHOU, DENGYONG; BOUSQUET, OLIVIER; LAL, THOMAS NAVIN; WESTON, JASON y SCHÖLKOPF, BERNHARD: «Learning with Local and Global Consistency». En: *Advances in Neural Information Processing Systems (NIPS)*, pp. 912–919. MIT Press, 2003.