Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Estudio y aplicación de técnicas de aprendizaje automático orientadas al ámbito médico: estimación y explicación de predicciones individuales

Trabajo de Fin de Máster presentado para la obtención del título Máster en Ingeniería Informática y de Telecomunicaciones
(*major* en Inteligencia Computacional)

Autor
Javier Di Deco Sampedro

Directora
Julia Díaz García

Madrid, junio 2012

Resumen

El aprendizaje automático está adquiriendo cada vez más importancia en diversas ramas de la ciencia, la tecnología o los negocios. Este trabajo se articula, concretamente, en torno al ámbito médico y está enmarcado en el paradigma del aprendizaje supervisado.

En términos generales, un sistema automático debe ser capaz de proporcionar una respuesta apropiada cuando se introduce información al mismo. En el aprendizaje supervisado, el núcleo del sistema es un modelo predictivo que asigna uno o varios valores de salida a cada elemento de entrada, en base al conocimiento adquirido a partir de un conjunto de datos cuyas salidas son conocidas. Sin embargo, la información disponible no siempre se puede o se debe utilizar directamente para alimentar al modelo. Por ello, suele ser necesaria una fase denominada de preprocesamiento para adecuar las entradas, en aras de maximizar el rendimiento del modelo. Asimismo, la salida que ofrece el modelo puede no ser satisfactoria o quizás resulte difícil de interpretar. Por lo tanto, en una última fase, de postprocesamiento, se evalúa el rendimiento del modelo y, en la medida de lo posible, se adaptan las salidas a las necesidades del usuario.

En este trabajo se abordan distintos aspectos de cada una de las fases mencionadas. En primer lugar, se han revisado los principales algoritmos de entrenamiento de modelos de aprendizaje automático supervisado, valorando cualitativamente sus características y su grado de adecuación para ser aplicados a problemas del ámbito médico. A continuación se estudian en profundidad algunas de las técnicas empleadas en las otras dos fases. En la de preprocesamiento, se analizan en detalle los métodos de selección de atributos *Relief*, *ReliefF* y *RReliefF* y se comparan con *Minimum Redundancy Maximum Relevance* y *Quadratic Programming Feature Selection*, tanto a nivel teórico como experimental. En la de postprocesamiento, el análisis se ha focalizado en las predicciones individuales, por tener especial relevancia en la medicina. Se estudian algunas medidas de estimación de fiabilidad en problemas de regresión y, en base a ellas, se proponen e implementan algunos métodos para tratar de corregir los errores de predicción. Finalmente, se estudia un método general, basado en la teoría de juegos cooperativos, para explicar las predicciones de un modelo en problemas de clasificación. Se implementa dicho método y se muestra cómo contribuye a hacer más transparentes los modelos considerados tradicionalmente como una *caja negra*.

Agradecimientos

Me gustaría agradecer a mis tutoras, Julia Díaz García y Ana González Marcos, los conocimientos, el apoyo y la confianza que me han transmitido durante estos casi dos años de máster. He aprendido mucho trabajando con ellas y no solamente a nivel académico. La normativa establece que únicamente puede firmar un director este trabajo pero, sin duda, ambas merecerían figurar en ese lugar.

Doy las gracias a toda la gente que *está ahí*, empezando por mi familia: a mis padres, a mis abuelos y a mi (no ya tan) pequeño hermano Sergio. A Susana, por todo. A mis amigos de *siempre*, en especial a Jorge y Quique. A mis amigos de la uni, entre ellos a Álex, Álvaro, David, Jose y María, que ya es como si les conociera de toda la vida. A mis amigos y compañeros de baloncesto, porque no hay nada mejor para desconectar. A mis compañeros del Instituto de Ingeniería del Conocimiento, Álvaro, Jorge y Sandra (por los buenos ratos del comedor), Alberto, Álvaro, Ana, Ana, Ángela, Carlos, David, Esther, José, Pablo, Sara y Sergio (con los que he compartido despacho y/o momentos de relax), al equipo de Bioinferencia, al de fútbol y a todos los que siempre están dispuestos a echar una mano o alegrarte el día. Aunque ya no están en el IIC, no me puedo olvidar de Irene y Jaime, que han seguido pendientes de cómo me iba y se les echa de menos por aquí.

Quiero dar las gracias también a todos los profesores que he tenido, con un recuerdo especial para Julio.

Finalmente, mi agradecimiento a los doctores Ana Frank y Juan Álvarez-Linera y al Instituto de Ingeniería del Conocimiento, por haberme permitido amablemente emplear el conjunto de datos recogido por el grupo DEMCAM (DEMencias en la Comunidad Autónoma de Madrid) para el estudio de la Enfermedad de Alzheimer mediante imágenes de resonancia magnética.

Índice general

Capítulo 1

Introducción

1.1. Motivación y objetivos

La creciente informatización del mundo permite recoger grandes volúmenes de datos y surge la necesidad de explotarlos eficaz y eficientemente. Por ello, el aprendizaje automático está adquiriendo cada vez más importancia en diversas ramas de la ciencia, la tecnología o los negocios. En concreto, este trabajo se centra en el ámbito médico, que es un campo de interés para mí, tanto profesional como personalmente.

El primer objetivo del trabajo es realizar una revisión de las aplicaciones de aprendizaje automático en la medicina. Se presentan las características de los modelos más exitosos, poniendo de manifiesto sus ventajas e inconvenientes. Además, se analizan detalladamente algunas técnicas que ayudan a potenciar el rendimiento de los modelos o facilitan su uso por parte de los especialistas médicos.

El segundo objetivo consiste en comprobar experimentalmente algunos de los resultados establecidos por la teoría. Se sigue el procedimiento habitual de realizar las primeras pruebas sobre conjuntos de datos muy sencillos y después sobre otros más complejos pero bien conocidos. Finalmente, se aplican las técnicas estudiadas sobre nuevos problemas reales del ámbito médico y se valoran los resultados obtenidos.

1.2. Conceptos fundamentales

Es difícil proporcionar una definición única y precisa de lo que significa *aprender*, ya que existen teorías que defienden posturas diferentes, así como la existencia de distintos tipos de aprendizaje. No obstante, cualquiera que sea el proceso de adquisición de conocimientos o habilidades, si lo realiza una máquina se denomina *aprendizaje automático*.

El objetivo principal del aprendizaje automático es la creación de un sistema capaz de dar una respuesta satisfactoria cuando se introduce información al mismo. En función de la realimentación que reciba el sistema se distinguen varios paradigmas. Este trabajo se enmarca en el ámbito del aprendizaje supervisado, que se caracteriza por el conocimiento de cuáles son las salidas esperadas para cierto conjunto de datos de entrada (datos etiquetados). Lo que se pretende es que el sistema generalice esas asociaciones para responder apropiadamente cuando reciba nuevas entradas. Otros paradigmas son el semisupervisado, en el que hay una parte de los datos iniciales cuya salida esperada se desconoce, y el no supervisado, en el que no se tiene ningún conocimiento a priori sobre el etiquetado de los datos iniciales.

La construcción del sistema suele comprender varias etapas que, a grandes rasgos, se pueden agrupar en tres: preprocesamiento de los datos, entrenamiento de uno o más modelos y postprocesamiento de los resultados. Los datos que recibe el sistema pueden estar incompletos, contener información redundante o irrelevante, sufrir perturbaciones a causa del ruido o presentar cualquier otra dificultad que afecte al rendimiento. Por ello, en la etapa de preprocesamiento se deben solventar todos esos inconvenientes. En la fase de entrenamiento se recibe la información ya depurada y se construye el núcleo del sistema: un modelo. Existen diferentes tipos de modelos, pero su función esencial es procesar una entrada y ofrecer una respuesta en consonancia con el conocimiento extraído de los datos etiquetados. En la fase de postprocesamiento se debe evaluar la idoneidad de las respuestas proporcionadas por el/los modelo/s para determinar su rendimiento y escoger el mejor, si procede. Además, se pueden llevar a cabo otras acciones para facilitar la presentación de las salidas al usuario del sistema, incluyendo información adicional si es necesario.

A continuación, se definen los elementos fundamentales del sistema: las entradas, las salidas y el modelo. La notación correspondiente se puede consultar en la sección 1.3.

Cada posible entrada al sistema es un vector de *atributos* y se denomina *instancia*. Los atributos pueden ser *nominales*, si toman valores dentro de un conjunto finito cualquiera, o *numéricos*, si toman valores reales (el conjunto de posibles valores puede ser finito, infinito numerable o no numerable).

Existen dos tipos de modelos atendiendo a las posibles salidas que ofrecen. Si los posibles valores de la salida se restringen a un conjunto finito, entonces se trata de un problema de *clasificación*. Si las salidas pueden tomar valores dentro de un conjunto infinito de la recta de números reales, entonces es un problema de *regresión*. En este trabajo aparecen ambos tipos.

El modelo se puede interpretar como una *aplicación* que asocia a cada instancia de entrada una salida [1]. Dependiendo del modelo, dicha aplicación puede resultar extremadamente complicada de escribir explícitamente pero a nivel teórico basta con una notación sencilla, ya que en la práctica el ordenador se encarga de realizar los cálculos.

1.3. Notación

En esta sección se especifica la notación general que se emplea a lo largo del trabajo. Si algún apartado requiere notación adicional, se indica donde corresponda pero respetando siempre la presentada aquí.

Una instancia genérica se representa con la letra I. Para referenciar diferentes instancias se emplean subíndices $(I_1, I_2, \ldots, I_i, \ldots)$. Se reserva el subíndice i para indexar o recorrer conjuntos de instancias. A cada instancia le corresponde una salida esperada o etiqueta t (o con el subíndice que proceda, t_i). Es un valor que indica la clase a la que pertenece la instancia (clasificación) o el valor objetivo (regresión). Cuando se desea hacer explícito el tipo de etiqueta, se emplea, en lugar de t, $\Omega(I)$ en el caso de clasificación y $\mathcal{T}(I)$ en el de regresión. En clasificación hay c posibles etiquetas C_k, $k = 1, \ldots, c$. Es decir, $\Omega(I) \in \{C_1, \ldots, C_c\}$. En regresión, $\mathcal{T}(I) \in S \subseteq \mathbb{R}$.

De manera análoga a las instancias, se reserva una letra, la A, para los atributos y el subíndice j para indexar los a atributos que forman cada instancia, $I = (A_1, A_2, \ldots, A_j, \ldots, A_a)$. Para indicar el valor que toma un atributo A en una instancia I, se emplea la notación $\Psi(A, I)$. Los m_j posibles valores que toma un atributo A_j en un conjunto de instancias $\{I_i\}_{i=1}^{n}$, se denotan por α_{s_j}, $s_j = 1, \ldots, m_j$. Es decir, $\Psi(A_j, I_i) \in \{\alpha_{s_j}\}_{s_j=1}^{m_j}$, $\forall\, i, j$.

El conjunto de datos que se utiliza para entrenar un modelo se representa con la letra E y está compuesto de n instancias etiquetadas: $E = \{(I_i, t_i)\}_{i=1}^{n}$, donde cada instancia es un vector de atributos $I_i = (A_j^{(i)})_{j=1}^{a}$.

Un modelo es una aplicación $f : \mathcal{A} \to \Upsilon$, donde $\mathcal{A}$ es el espacio completo de atributos (contiene todas las posibles instancias) y Υ es el espacio de posibles valores de la etiqueta ($\Upsilon = \{C_1, \ldots, C_c\}$ en clasificación y $\Upsilon \subseteq \mathbb{R}$ en regresión). Dada una instancia I, la predicción que ofrece el modelo f es $f(I)$. En problemas de clasificación es posible que el modelo proporcione las probabilidades a posteriori para las c clases: $\hat{f}(I) = (\mathbb{P}(C_1 \mid I), \ldots, \mathbb{P}(C_c \mid I))$. En ese caso, la

[1] A lo largo del trabajo se emplea el término *modelo* para designar también a los paradigmas o a los algoritmos de entrenamiento que permiten construir la aplicación concreta en cada conjunto de datos. No obstante, este pequeño abuso del lenguaje no supone ningún problema para la comprensión del texto.

predicción final viene dada por $f(I) = \underset{k}{\operatorname{argmax}}\, \hat{f}_k(I) = \underset{k}{\operatorname{argmax}}\, \mathbb{P}(C_k \mid I)$. Es decir, se selecciona la clase cuya probabilidad a posteriori es máxima según el modelo.

1.4. Contenidos y organización del trabajo

Como eje central y conductor de este trabajo se seleccionan algunas de las publicaciones firmadas por el profesor doctor Igor Kononenko y varios colegas del Departamento de Inteligencia Artificial de la Universidad de Liubliana, Eslovenia. El motivo de esta elección es que mis áreas de interes coinciden con las líneas de investigación principales de dicho departamento. No obstante, se consultan y estudian otras publicaciones para complementar o ampliar la información disponible y comparar algunos resultados. A continuación se exponen brevemente los contenidos del trabajo.

En el **Capítulo 2**, *Aprendizaje automático en medicina*, se analizan varios modelos de aprendizaje automático que se han aplicado en biomedicina. En su trabajo en [1], Kononenko realiza un estudio del estado del arte y recoge las características principales y los requisitos que debe cumplir un modelo para tener éxito en dicho campo. Destaca los siguientes modelos: *árboles de decisión, Naive Bayes, vecinos más cercanos, redes neuronales y máquinas de vectores de soporte.*

Previamente a la aplicación de un modelo (fase de preprocesamiento), conviene recurrir a métodos de selección de atributos, que eliminan la información redundante e irrelevante. Existen técnicas sencillas, como establecer un ranking basado en correlaciones o información mutua con la variable objetivo. Sin embargo, su rendimiento no es satisfactorio en problemas complejos como los que suelen aparecer en el ámbito médico. Por ello, en el **Capítulo 3**, *Selección de atributos*, se estudian métodos como Relief [9] y sus extensiones [10], [11], [12], que son capaces de encontrar fuertes dependencias condicionales entre los atributos. Además, se han analizado otros dos métodos que también toman en consideración las dependencias entre atributos a la hora de seleccionar los más informativos: mRMR (*Minimum Redundancy Maximum Relevance*) [13] y QPFS (*Quadratic Programming Feature Selection*) [15].

Tras el entrenamiento del modelo (fase de postprocesamiento) se debe evaluar su rendimiento. En general, se emplean medidas globales, promediadas sobre todos los casos analizados. En el ámbito médico, sin desestimar dichas medidas, es interesante conocer los detalles concernientes a cada predicción individual cuando se trata de apoyar la toma de decisiones sobre el tratamiento o el diagnóstico de un paciente. En el **Capítulo 4**, *Análisis de predicciones*

individuales, se abordan dos problemas: la estimación de la fiabilidad de una predicción y la capacidad de proporcionar una explicación basada en los valores de los atributos.

Para estimar la fiabilidad de una predicción, se puede aplicar el *análisis local de sensibilidad* generando perturbaciones en el conjunto de aprendizaje, como se propone en [16]. Lo interesante es que no se requiere conocimiento previo sobre el valor de la variable objetivo de la instancia que se analiza.

Existen diversas técnicas que tratan de dotar de mayor transparencia a los modelos, especialmente aquellos considerados tradicionalmente como *de caja negra* (p. ej. *redes neuronales*). Se ha estudiado la aproximación propuesta en [19], que valora la influencia de cada atributo en la predicción, mediante una técnica que se denomina *descomposición de la predicción*. Pronto se ve que tiene una limitación importante: no detecta las situaciones en las que un cambio en el valor de la predicción requiere de la ocurrencia de cambios en el valor de más de un atributo simultáneamente. Por ello, se analiza [21], donde se supera dicha limitación pero con un coste computacional excesivamente elevado. Una solución satisfactoria, propuesta por Strumbelj y Kononenko en [22], proviene de la teoría de juegos cooperativos.

La teoría de juegos cooperativos está desarrollada desde mediados del siglo XX pero no ha sido hasta principios del siglo XXI cuando se ha empezado a aplicar al aprendizaje automático. Para conocer las bases teóricas se han estudiado los artículos de Keinan y col. [23] y Cohen y col. [24]. Ambos están basados en el célebre artículo de Shapley [26], autor que da nombre a uno de los conceptos centrales de la teoría. Para poder realizar eficientemente los cálculos necesarios que sustentan la aplicación desarrollada en [22], se ha estudiado el algoritmo de coste polinómico presentado en [25].

En el **Capítulo 5**, *Experimentos*, se presenta el trabajo práctico realizado. Se divide en tres secciones, cada una con un objetivo diferente y bien definido. En la primera se amplía el alcance de algunos de los experimentos reportados en [10], extendiendo la comparativa a los algoritmos de selección de variables mRMR y QPFS. En la segunda, se implementa el sistema de estimaciones de fiabilidad propuesto en [16] y se propone una extensión para intentar corregir los errores de predicción en problemas de regresión. Por último, en la tercera, se implementa el método de explicación presentado en [22], se valida su funcionamiento empleando un conjunto de datos bien conocido y se aplica a un conjunto de datos real sobre un estudio reciente del Alzheimer mediante resonancia magnética.

Finalmente, el **Capítulo 6** recoge las conclusiones más relevantes.

Capítulo 2

Aprendizaje automático en medicina

El aprendizaje automático puede aplicarse para resolver problemas de muy diversos campos del conocimiento, siempre y cuando se disponga de datos con los que alimentar a los modelos. Existe una gran variedad de modelos y cada uno tiene sus propias características, por lo que no todos son adecuados para resolver cualquier tipo de problema. Concretamente, en el ámbito médico la toma de decisiones es un punto crítico, ya que está en juego la salud de las personas. Por ello, a la hora de emplear un modelo automático para apoyar los diagnósticos médicos deben considerarse detenidamente las ventajas y los inconvenientes, así como los riesgos que conlleva.

2.1. Requerimientos del modelo

En [1] se revisan las técnicas y modelos de aprendizaje automático que se han aplicado tradicionalmente en medicina y se recogen los requerimientos que deben cumplir para tener éxito en este campo. A continuación se exponen dichos requerimientos y se comenta, para cada modelo, en qué medida los cumplen, así como otras características particulares.

2.1.1. Buen rendimiento

El rendimiento de un modelo se puede medir de varias formas. Para cualquiera de ellas, si se conoce (o se estima) el rendimiento que ofrece el diagnóstico realizado por el médico, es deseable que el modelo lo supere. De no ser así, es fundamental que al menos sea similar y que ofrezca otras ventajas a la práctica médica como ahorro de tiempo o costes.

2.1.2. Capacidad de explicar un diagnóstico

Si el sistema automático proporciona un diagnóstico diferente al que el profesional médico espera, se produce un conflicto en el que la decisión última la toma siempre el médico. Si el sistema no ofrece explicación alguna, el médico será muy reacio a cambiar de opinión (salvo, quizás, si se ha contrastado que el rendimiento del modelo es claramente superior al del diagnóstico clínico). En cambio, si el sistema justifica su decisión razonadamente, el médico puede analizar los motivos de la decisión y juzgar si estaba inicialmente equivocado o no.

2.1.3. Transparencia del conocimiento adquirido

Es deseable que un modelo sea capaz de ofrecer, de alguna forma comprensible para el especialista en salud, el conocimiento adquirido sobre el problema. De este modo, el médico puede estudiarlo y tener acceso a otro punto de vista que quizás ayude a complementar sus conocimientos o a descubrir relaciones entre los datos que han pasado desapercibidas hasta el momento. Además, si el conocimiento extraído es aprobado por un especialista, el sistema puede servir para apoyar el aprendizaje de los estudiantes o de otros profesionales médicos no especializados en el tema. Del mismo modo, esto sirve para consolidar una base de datos robusta.

2.1.4. Manejo de valores perdidos y ruido

En los registros de datos de los pacientes es posible que, por diversas circunstancias, no se hayan cumplimentado todos los campos o que se haya introducido algún valor incorrecto. Técnicamente, estos problemas se conocen como la existencia de valores perdidos (desconocimiento del valor que tienen algunos atributos) y ruido (datos erróneos). El modelo debe ser lo suficientemente robusto para tolerar unos niveles aceptables de falta o incorrección de información, sin que su comportamiento se vea afectado significativamente.

2.1.5. Reducción del volumen de datos

En el dominio médico es muy costoso recoger información valiosa de los pacientes. Por ello, un sistema automático que sea capaz de ofrecer un buen comportamiento con menos información es preferible a otros que requieran más datos para conseguir un comportamiento similar. Esta es la versión estadística del principio conocido como *Navaja de Occam*, entendiendo menos información como menor número de atributos, menor número de instancias o ambas.

2.2. Modelos más utilizados

En esta sección se presentan cinco modelos de aprendizaje automático que gozan de amplia difusión entre los estudiosos de esta disciplina y que se aplican exitosamente al dominio médico. Se explican brevemente su funcionamiento y sus características principales y se realiza una valoración en términos de los requerimientos expuestos en la sección 2.1.

Es importante tener en cuenta que ningún modelo es mejor que los demás en todas las circunstancias posibles, sino que cada uno tiene sus ventajas e inconvenientes dependiendo del problema al que se aplica. No obstante, es muy difícil conocer a priori qué modelo es más adecuado para un caso determinado. Por ello, es muy aconsejable probar siempre varios modelos de diferentes paradigmas y comparar los resultados.

2.2.1. Árboles de decisión

Los árboles de decisión se ubican dentro de una rama del aprendizaje automático denominada aprendizaje simbólico, en la que también se encuentran los modelos de reglas de decisión, estrechamente relacionados con los árboles. Un árbol de decisión consta de nodos, ramas y hojas, que son como los nodos pero no surge ninguna rama de ellas. De cada nodo nacen dos o más ramas en las que se formulan proposiciones lógicas excluyentes que particionan el espacio de atributos. Esto quiere decir que cada instancia, dependiendo de los valores de los atributos implicados en la proposición lógica de un nodo, cumple la condición correspondiente a una de las ramas. De este modo, comenzando en el nodo raíz, cada instancia se dirige hasta una de las hojas. A todas las instancias pertenecientes a la misma hoja se les asigna una etiqueta común (válido tanto en problemas de clasificación como de regresión).

La explicación anterior se enfoca a un árbol ya construido. Previamente, tiene que existir una fase de entrenamiento en la que se decide qué preguntas se van a formular en cada nodo. Para ello, existen diversos algoritmos, entre los que se comentan *ID3*, *C4.5*, *Assistant* y *Lookahead Feature Selection (LFC)* [1], [2].

El algoritmo *ID3*, propuesto por Quinlan, fue uno de los primeros métodos de construcción de árboles de decisión. Es recursivo y se basa en medidas de entropía y de ganancia de información para determinar cuál es el atributo más adecuado para la proposición lógica de un nodo. Según los valores del atributo escogido, se particiona el conjunto de datos de entrenamiento en varios subconjuntos, que corresponden a cada una de las ramas que surgen del nodo. Se evalúan los atributos restantes sobre cada subconjunto y se formulan las proposiciones lógicas de los nodos del siguiente nivel del árbol. El proceso se aplica

recursivamente sobre cada nuevo nodo hasta que no queden más atributos que evaluar, no queden más datos que cumplan las condiciones que conducen a dicho nodo o todos los datos tengan la misma etiqueta. Ante cualquiera de dichas circunstancias, el nodo pasa a ser una hoja del árbol.

ID3 es relativamente sencillo de implementar pero tiene algunas limitaciones importantes:

- Solamente permite atributos discretos y que la variable clase sea binaria (+,–).
- No soporta valores desconocidos.
- No poda, es decir, no limita o recorta de ningún modo la profundidad del árbol, lo cual empeora su capacidad de generalización.

El algoritmo *C4.5* es del mismo autor que el *ID3* y es una extensión natural del mismo. Supera todas las limitaciones mencionadas anteriormente, por lo que puede aplicarse a un mayor número de problemas, es más robusto y ofrece una mayor capacidad de generalización.

El algoritmo *Assistant*, propuesto por Kononenko, también surge a raíz del *ID3*. Permite atributos continuos pero la principal diferencia respecto al *ID3* es que, en los problemas de clasificación, las clases forman una jerarquía. Esta novedad puede resultar útil en problemas médicos. Por ejemplo, si se quieren distinguir pacientes sanos de pacientes diabéticos pero también se quieren considerar subtipos de diabetes (tipos 1 o 2). Se podría resolver con clases disjuntas, considerando el conjunto {sano, tipo 1, tipo 2}, pero puede ser más conveniente registrar una superclase (diabetes) que agrupe a todos los enfermos. De este modo, los posibles valores de la etiqueta forman una jerarquía: {sano, diabetes:{tipo 1, tipo 2}}. *Assistant* tiene dos variantes: *Assistant-I* y *Assistant-R*. La primera utiliza el criterio de ganancia de información para decidir cómo se ramifica cada nodo del árbol (igual que *ID3*), mientras que la segunda emplea el algoritmo *ReliefF*, que se explica en la sección 3.1.3.

El algoritmo *LFC* solamente maneja atributos binarios, por lo que deben umbralizarse todos los que no lo sean. La particularidad de este algoritmo es que genera nuevos atributos en cada nivel del árbol. Lo hace aplicando funciones lógicas (conjunción, disyunción y negación) a diversas combinaciones de los atributos originales o heredados de niveles superiores. La principal ventaja es que permite descubrir combinaciones de atributos que ofrezcan mejores particiones que los atributos originales. Las desventajas son: una mayor carga computacional y la posibilidad de que aparezcan expresiones muy complejas que dificulten la interpretación del árbol.

Se ha visto que cada algoritmo tiene sus peculiaridades. No obstante, se puede hacer una valoración cualitativa global del grado de cumplimiento de los requerimientos establecidos en la sección 2.1. En general, los árboles de decisión presentan un rendimiento apropiado, así como un manejo aceptable de valores perdidos y ruido. También reducen la cantidad de datos necesaria para evaluar a un paciente, ya que no suelen aparecer todos los atributos de entrenamiento en el árbol construido. Sin embargo, las claves del éxito de los árboles de decisión son su gran transparencia (se puede observar toda la estructura del conocimiento extraído de los datos) y su capacidad de explicar cada una de las predicciones (registrando cómo la instancia recorre el árbol hasta llegar a la hoja correspondiente). Adicionalmente, en producción no es necesario realizar ningún cálculo y se pueden utilizar sin el ordenador, lo que es muy bien recibido por los profesionales médicos no familiarizados con la informática.

2.2.2. Naive Bayes

Es un modelo probabilístico [3] que debe su nombre a dos circunstancias: aplicación del Teorema de Bayes para extraer la regla de clasificación y la simplicidad de suponer que todos los atributos son condicionalmente independientes dada la clase (*naive* ≡ ingenuo, simple). Dicha suposición está bastante alejada de la realidad en la mayoría de problemas. No obstante, este modelo ofrece un rendimiento sorprendentemente bueno debido a que no hay que estimar distribuciones de probabilidad conjuntas, evitando así sufrir la *maldición de la dimensionalidad* [I].

La función de clasificación del método *Naive Bayes* se define basándose en la regla del *Máximo A Posteriori (MAP)* de las probabilidades de cada clase:

$$f(I) = \underset{k}{\operatorname{argmax}}\, \mathbb{P}\left(\Omega(I) = C_k \mid I\right) = \underset{k}{\operatorname{argmax}}\, \mathbb{P}\left(\Omega(I) = C_k \mid A_1, \ldots, A_a\right).$$

Desarrollando la expresión anterior mediante el Teorema de Bayes y aplicando la suposición de independencia se obtiene:

$$f(I) = \underset{k}{\operatorname{argmax}}\, \mathbb{P}(C_k) \prod_{j=1}^{a} \mathbb{P}\left(\Psi(A_j, I) \mid C_k\right).$$

Los términos $\mathbb{P}(C_k)$ y $\mathbb{P}(\Psi(A_j, I) \mid C_k)$ representan las probabilidades a priori de que una instancia I sea de la clase C_k y de que el atributo A_j tome cierto valor para las instancias de la clase C_k, respectivamente. Se pueden calcular con facilidad a partir de los datos de entrenamiento, haciendo un recuento del

[I] *Maldición de la dimensionalidad* es el nombre con el que se conoce comúnmente al siguiente hecho: el número de instancias necesarias para estimar convenientemente una distribución conjunta crece exponencialmente con el número de atributos.

número de instancias de cada clase y, entre ellas, el número de veces que aparece cada valor del atributo. Finalmente, se divide cada cantidad entre el total de casos posibles (total de instancias para $\mathbb{P}(C_k)$ y total de instancias de la clase C_k para $\mathbb{P}(\Psi(A_j, I) \mid C_k)$). Si los atributos son continuos se pueden discretizar y tratarlos como tal, o bien ajustar los parámetros de una distribución conocida, típicamente una *Gaussiana*, para estimar las probabilidades buscadas. La primera opción es más recomendable si se dispone de gran cantidad de datos.

Existen extensiones del modelo *Naive Bayes* que tratan de relajar la suposición de independencia entre atributos [4]. Entre ellas, figura el método denominado *Semi-Naive Bayes*, cuya función de clasificación es:

$$f(I) = \underset{k}{\operatorname{argmax}}\, \mathbb{P}(C_k) \prod_{\hat{j}=1}^{\hat{a}} \mathbb{P}\left(\Psi(\hat{A}_{\hat{j}}, I) \mid C_k\right).$$

Donde $\hat{A}_{\hat{j}}$ representa un vector de uno o más atributos, de tal forma que cada atributo original A_j solamente puede pertenecer a un $\hat{A}_{\hat{j}}$ (por tanto, $\hat{a} <= a$). Cada $\hat{A}_{\hat{j}}$ agrupa los atributos A_j dependientes entre sí y se asume independiente de cualquier otro $\hat{A}_{\hat{l}}$, $\hat{l} \neq \hat{j}$.

El inconveniente de estas nuevas aproximaciones es que la complejidad computacional aumenta exponencialmente con el número de atributos. El orden exponencial se debe a la búsqueda exhaustiva de subconjuntos óptimos entre los 2^a posibles. Se puede reducir drásticamente el tiempo de computación si se aplica un algoritmo codicioso [II], con la contrapartida de que los subconjuntos encontrados pueden no ser los óptimos.

Los modelos Bayesianos son los que ofrecen mayor capacidad para explicar sus predicciones. Determinan en qué medida los valores de cada atributo contribuyen positiva o negativamente a la decisión tomada. Este hecho, unido a que presentan un rendimiento bastante bueno, manejan con solvencia valores perdidos [III], toleran niveles aceptables de ruido y tienen un buen nivel de transparencia (aunque no tanto como los árboles de decisión), es lo que hace de los modelos Bayesianos los preferidos en el ámbito médico. Tanto es así, que Kononenko afirma en [1] que deben probarse siempre en primer lugar y solamente utilizar otros modelos si los resultados son claramente superiores.

[II] Se dice que un algoritmo de optimización es codicioso si, en lugar de realizar una búsqueda exhaustiva en el espacio de soluciones, construye la solución final basándose en criterios locales de optimalidad.

[III] Si se desconoce el valor de un atributo para una instancia, el método Naive Bayes lo trata como si fuese aleatorio, repartiendo las probabilidades entre todos los posibles valores del mismo.

La necesidad de explicar cómo los atributos intervienen en las predicciones da pie a que, en la sección 4.2, se expongan métodos generales que sean aplicables a un gran número de modelos. Entre ellos, se encuentran todos los mencionados en este trabajo y otros de extendido uso en sistemas de aprendizaje automático. Estos métodos de explicación se ejecutan después de que el modelo predictivo haya sido entrenado y consiguen extraer las contribuciones de cada atributo y expresarlas de forma similar a la de los modelos Bayesianos.

2.2.3. Vecinos más cercanos

Es un modelo perteneciente al paradigma de *aprendizaje perezoso*, que se caracteriza por la ausencia de una fase de extracción de conocimiento de los datos. El funcionamiento está dirigido por la demanda de clasificación de una nueva instancia. En ese momento se analiza el entorno de dicha instancia y se emite una predicción basada, como el propio nombre del algoritmo indica, en las instancias vecinas más cercanas a la demandada.

El modelo se puede aplicar tanto a problemas de clasificación como de regresión. El algoritmo de predicción es muy sencillo. En primer lugar se fija un natural (b [IV]) que determina el número de vecinos que se tendrán en cuenta y se hallan la b instancias más cercanas a una dada, empleando alguna distancia (euclídea, Mahalanobis, etc.). A continuación, si se trata de un problema de clasificación, se hace un recuento de cuántos de los b-vecinos pertenecen a cada clase y se asigna a la nueva instancia la clase mayoritaria; si se trata de un problema de regresión, se calcula el promedio de los valores de las etiquetas de los b-vecinos y se asigna el resultado a la etiqueta de la nueva instancia. En ambos casos es posible ponderar la influencia de cada vecino con una función dependiente de la distancia, de modo que, cuanto más cercano sea el vecino, más peso tenga en la decisión.

En general, b-vecinos más cercanos maneja aceptablemente valores perdidos y ruido pero no presenta un rendimiento claramente mejor que ninguno de los modelos anteriores. Su transparencia es muy pobre (por no decir inexistente) ya que, por ser un algoritmo de *aprendizaje perezoso*, no extrae conocimiento sino que se limita únicamente a emitir una predicción como respuesta a una demanda. El factor diferenciador que hace interesante el uso de este modelo, a pesar de su falta de transparencia, es la forma de explicar las predicciones. Una nueva instancia se etiqueta en función de cómo están etiquetadas sus vecinas. Haciendo una analogía, se puede identificar esta aproximación con el método de diagnóstico clínico basado en la experiencia adquirida al tratar casos similares ($\equiv$ vecinos más cercanos).

[IV] Habitualmente se emplea la letra k pero a lo largo de este trabajo se utiliza b por motivos de notación.

2.2.4. Redes neuronales

Inicialmente, las redes neuronales surgieron por la motivación de tratar de reproducir el funcionamiento del cerebro. En 1943, McCulloch y Pitts elaboraron un modelo artificial de una neurona, que simulaba el procesamiento de información que tenía lugar en una neurona real [5]. A partir de entonces, se ha tratado de construir redes cada vez más complejas pero, de momento, el objetivo de emular el comportamiento del cerebro está lejos de alcanzarse. No obstante, en este proceso de investigación se han descubierto algunos modelos que resultan muy útiles para abordar determinadas tareas, como la resolución de problemas de clasificación o regresión.

Uno de los primeros modelos fue el *Perceptrón de Rosenblatt* [6], que permite establecer un hiperplano en un espacio a-dimensional mediante el ajuste de los parámetros ($\equiv$ pesos sinápticos) de las neuronas artificiales. Con este modelo se pueden resolver problemas lineales. Sin embargo, como la mayoría de problemas reales no son lineales, se desarrolló más este enfoque incluyendo varias capas de neuronas artificiales y generando un nuevo algoritmo de aprendizaje (*Backpropagation*) para ajustar los pesos de la red [7]. El aprendizaje se basa en *descenso por gradiente* del error de predicción. Este nuevo modelo se denomina *Perceptrón Multicapa* y se demostró que es una *Familia de Aproximación Universal*, lo que significa que puede aproximar cualquier función continua con tanta precisión como se desee (siempre que se disponga de un número suficiente de instancias).

En la práctica, dado un conjunto de datos etiquetados (problema de clasificación o regresión), un *Perceptrón Multicapa* es capaz de *memorizarlos*. Sin embargo, el coste computacional puede ser muy elevado y, aunque no lo fuera, no es conveniente que el modelo se ajuste excesivamente a los datos de entrenamiento (*overfitting*) ya que puede reducir notablemente su capacidad de generalización para nuevos casos. Para reducir el *overfitting* se puede modificar el algoritmo de entrenamiento de dos maneras:

- Penalizando el número de pesos de la red (*Backpropagation with weight elimination*).
- Penalizando el tamaño de los pesos (*Backpropagation with weight decay*).

Aunque el *Perceptrón Multicapa* no ha alcanzado la precisión perfecta, como se pudo pensar en un principio, ofrece un rendimiento muy bueno (mejor, en general, que los modelos expuestos anteriormente). Los principales inconvenientes para su aplicación en el ámbito médico son su falta de transparencia y su escasa capacidad de explicar las predicciones emitidas. Se han desarrollado algunos métodos para tratar de solventar estas deficiencias, como la extracción de reglas a partir de los pesos de la red. Sin embargo, como se cita en [1], las

reglas extraídas suelen ser bastante más largas y complicadas de interpretar que las que se derivan de un árbol de decisión. En la sección 4.2 se exponen algunas técnicas que permiten explicar las predicciones, incluso considerando el modelo como una *caja negra*. De esta manera, se mitiga el impacto negativo de los puntos débiles de las redes neuronales para poder aprovechar mejor su excelente rendimiento.

2.2.5. Máquinas de vectores de soporte

Este modelo es parecido a las redes neuronales en cuanto a su objetivo de ajustar un conjunto de parámetros, que permiten establecer fronteras en el espacio a-dimensional y aproximar funciones o separar patrones en diferentes regiones del espacio de atributos. La diferencia radica en el método de entrenamiento para ajustar los parámetros. Como se vio en 2.2.4, el *Perceptrón Multicapa* entrenado mediante *Backpropagation* se basa en *descenso por gradiente* del error de predicción. En cambio, las *máquinas de vectores de soporte* basan su entrenamiento en la maximización del margen existente entre el hiperplano separador y las instancias de las dos clases (inicialmente, este modelo se diseñó para resolver problemas de clasificación de dos clases pero hay extensiones para problemas multi-clase y para problemas de regresión) [8].

En términos de los requerimientos establecidos en la sección 2.1, las *máquinas de vectores de soporte* se comportan de manera muy similar a las redes neuronales. Presentan un rendimiento muy bueno pero el hecho de que el entrenamiento consista en el ajuste de unos parámetros *ocultos* y la predicción consista únicamente en un resultado numérico, hace que tanto la transparencia como la capacidad de explicación sean muy pobres. Este último aspecto se mejora mediante la aplicación de técnicas como las que se explican en la sección 4.2.

Capítulo 3

Selección de atributos

Cuando se utiliza una base de datos para tratar de dar respuesta a una cuestión o para realizar determinadas predicciones, es muy posible que exista información *irrelevante* o *redundante*. Generalmente, un atributo (o combinación de atributos) no se considera categóricamente como irrelevante o no, sino que se puede establecer un grado de relevancia. Además, conviene observar que un atributo puede resultar muy poco relevante para emitir determinadas predicciones pero ser muy valioso para otras. Si no se detecta y elimina la información con menor valor predictivo antes de entrenar un modelo de aprendizaje automático, su rendimiento puede verse mermado; bien por una reducción de la precisión, bien por un aumento del coste computacional (por operar con más información de la necesaria) o ambos simultáneamente.

Durante la resolución de un problema de aprendizaje automático existe una fase, dentro de la etapa de preprocesado, denominada *reducción de dimensionalidad*, que se ocupa de la tarea expuesta en el párrafo anterior. Hay numerosos métodos para reducir la dimensionalidad de un problema pero se clasifican en dos categorías principales:

- *Selección*. Escogen un subconjunto de los atributos iniciales o bien establecen un ranking de todos los atributos y el usuario escoge los $\hat{a}$ primeros de acuerdo a algún criterio establecido, $\hat{a} \leq a$.

- *Extracción*. Generan nuevos atributos mediante combinaciones (lineales o no-lineales) o proyecciones de los atributos originales.

El título de este capítulo corresponde a la primera categoría, debido a que en el ámbito médico son preferibles los métodos de *selección*. Los métodos de *extracción* dan lugar a atributos que ya no se corresponden con ninguna medida del mundo real, dificultando su interpretación. Por contra, los métodos de *selección* mantienen esa correspondencia intacta, simplemente indicando los que más información útil aportan para resolver el problema abordado.

En la medicina moderna existen aparatos que toman un gran número de medidas del mundo real y se hace necesaria la automatización del procesado de esa información. La *selección* de las medidas más importantes, además de facilitar el ajuste de un modelo, aporta, en sí misma, un valor añadido a la investigación biomédica. Permite descubrir factores asociados a determinadas enfermedades o dependencias desconocidas entre diversos atributos.

En este capítulo se estudian en profundidad los algoritmos de la familia *Relief* [10] y se comparan con otros algoritmos de *selección* de atributos como mRMR [13] y QPFS [15]. Todos ellos son algoritmos de tipo *filter*, es decir, no requieren el entrenamiento de ningún modelo para decidir cuáles son los atributos más importantes (los que sí lo requieren se denominan algoritmos *wrapper*).

3.1. Relief y sus extensiones

Los algoritmos de la familia *Relief* son métodos de estimación de atributos. Generalmente tienen una aplicación directa para *seleccionar* los mejores atributos en un problema de aprendizaje automático, aunque también sirven para determinar cómo ramificar un árbol de decisión o se pueden aplicar en programación lógica inductiva. Los algoritmos *Relief* son capaces de detectar dependencias condicionales entre los atributos a escala local.

El algoritmo *Relief* original fue propuesto, para problemas de clasificación binaria, por Kira y Rendell en [9]. Kononenko desarrolló varias extensiones que superan ciertas limitaciones del original, pudiéndose aplicar también a problemas multiclase [11]. Entre dichas extensiones, comprobó que la mejor es la denominada *ReliefF*. Posteriormente, Robnik-Sikonja y Kononenko, en su trabajo en [12], proponen una adaptación para problemas de regresión: *RReliefF* (Regressional *ReliefF*). Por último, en [10], estudian en profundidad todos estos algoritmos, desde los puntos de vista teórico y práctico.

En primer lugar, se presenta la notación específica que se emplea en esta sección. A continuación se explican los algoritmos de la familia *Relief*: *Relief* (original), *ReliefF* y *RReliefF*. Posteriormente, se realizan varias observaciones relevantes que afectan a los tres algoritmos y, por último, se analiza la complejidad computacional de los mismos.

3.1.1. Notación

Los algoritmos de la familia *Relief* necesitan hallar los vecinos más cercanos a las instancias que analizan. Para referenciarlos, se escribe H (*Hit*) si pertenece a la misma clase o M (*Miss*) si pertenece a la otra clase ($M(C_k)$ para

problemas multi-clase, siendo C_k la clase a la que pertenece M). En problemas de regresión, se utiliza la letra J. Si se considera más de una instancia cercana se indicará mediante el subíndice l: H_l, $M_l(C_k)$ o J_l, con $l = 1, \ldots, b$.

De las n instancias del conjunto de entrenamiento E, se deben escoger aleatoriamente $\hat{n}$ ($\hat{n} \leq n$). Para ello, se genera una permutación π de n elementos y se toman los $\hat{n}$ primeros, referenciando a las instancias de E como $I_{\pi(\hat{i})}$, de modo que $\hat{i} = 1, \ldots, \hat{n}$ y $1 \leq \pi(\hat{i}) \leq n$.

3.1.2. Relief

Propuesto por Kira y Rendell en [9], *Relief* (Algoritmo 1) pretende estimar la calidad de los atributos en base a lo bien que ayudan a distinguir la clase entre instancias cercanas en el espacio de atributos. Solamente es aplicable a problemas de clasificación binaria pero los atributos pueden ser tanto nominales como numéricos. Tiene otra limitación importante, ya que no soporta valores perdidos.

Sean I_1, I_2, dos instancias cercanas cualesquiera. La idea intuitiva subyacente a *Relief* es la siguiente: un atributo A que tome valores distintos para las instancias dadas ($\Psi(A, I_1) \neq \Psi(A, I_2)$) es *bueno* si las instancias son de distinta clase y *malo* si las instancias son de la misma clase. En ambos casos, la diferencia en el valor del atributo ayuda a separar las instancias en el espacio de atributos, que es lo deseable en el primer caso pero no en el segundo.

Algoritmo 1 Relief

Entrada: $E = \{(I_i, t_i)\}_{i=1}^{n}$. Conjunto de n instancias etiquetadas.
Salida: $W \in [-1, 1]^a$. Vector de estimaciones de la calidad de cada atributo.
Procedimiento:

1: Inicializar los pesos $W[j] \leftarrow 0$; $j = 1, 2, \ldots, a$.
2: Generar una permutación aleatoria π, de n elementos.
3: **for** $\hat{i} = 1 \rightarrow \hat{n}$ ($\hat{n} \leq n$) **do**
4: Encontrar la instancia más cercana a $I_{\pi(\hat{i})}$ de la misma (H) y de distinta (M) clase.
5: **for** $j = 1 \rightarrow a$ **do**
6: $W[j] \leftarrow W[j] - \frac{diff(A_j, I_{\pi(\hat{i})}, H)}{\hat{n}} + \frac{diff(A_j, I_{\pi(\hat{i})}, M)}{\hat{n}}$.
7: **end for**
8: **end for**
9: **return** W

En el Algoritmo 1 falta determinar cómo se hallan las instancias más cercanas y cómo se calcula la función $diff$. Para la primera tarea, se debe escoger

una medida de distancia en el espacio de atributos. Como dicha distancia depende de la función $diff$, se define ésta previamente. Se toman dos instancias y un atributo genéricos, I_1, I_2 y A_j. Deben distinguirse dos casos, en función de la naturaleza del atributo:

- Atributo nominal.

$$diff(A_j, I_1, I_2) = \begin{cases} 0 & \text{si } \Psi(A_j, I_1) = \Psi(A_j, I_2) \\ 1 & \text{en otro caso} \end{cases}.$$

- Atributo numérico.

$$diff(A_j, I_1, I_2) = \frac{|\Psi(A_j, I_1) - \Psi(A_j, I_2)|}{max\{\alpha_{s_j}\}_{s_j=1}^{m_j} - min\{\alpha_{s_j}\}_{s_j=1}^{m_j}}.$$

En el denominador de la expresión correspondiente a atributos numéricos, se calcula la diferencia entre los valores máximo y mínimo que toma el atributo. m_j indica el número de valores distintos (α_{s_j}) que toma el atributo A_j en la muestra. Se puede objetar que, si existen *outliers* para algún atributo A_j, dicha diferencia será grande y $diff$ (y en consecuencia $W[j]$) será comparativamente menor que para otros atributos que no presenten *outliers* en sus valores. No obstante, los autores otorgan prioridad a la obtención de unos pesos finales normalizados en el intervalo $[-1, 1]$. Por ello, definen $diff$ de esa manera y no de otras más robustas como, por ejemplo, dividiendo por la mediana del atributo. Por el mismo motivo se divide por $\hat{n}$ en el paso de actualización de pesos del algoritmo. Es aconsejable, pues, eliminar los *outliers* antes de ejecutar *Relief*.

Finalmente, hay que definir la distancia entre dos instancias, $\delta(I_1, I_2)$, que permite hallar los vecinos más cercanos en el espacio a-dimensional de atributos. Para ello, se toma como distancia en una sola dimensión (correspondiente a un atributo) el valor de la función $diff$. Para el espacio completo se emplea la distancia de Manhattan:

$$\delta(I_1, I_2) = \sum_{j=1}^{a} diff(A_j, I_1, I_2).$$

3.1.3. ReliefF

En [11], Kononenko revisa el algoritmo *Relief* y propone varias extensiones con el fin de hacerlo más robusto y superar algunas de las limitaciones que tiene. En primer lugar, aborda el problema de los valores perdidos y presenta cuatro alternativas, a las que bautiza añadiendo a *Relief* los sufijos A, B, C o D. Una vez superado ese punto, plantea dos extensiones basadas en *ReliefD* (que es la mejor aproximación de las cuatro iniciales) para dar soporte a problemas multiclase, nombrándolas con los sufijos E y F. En sus experimentos comprueba que la que mejor funciona es la F. Dicha versión, *ReliefF*, es la que aquí se presenta (Algoritmo 2), explicando también las modificaciones respecto a *Relief*.

Algoritmo 2 ReliefF

Entrada: $E = \{(I_i, t_i)\}_{i=1}^{n}$. Conjunto de n instancias etiquetadas.
Salida: $W \in [-1, 1]^a$. Vector de estimaciones de la calidad de cada atributo.
Procedimiento:

1: Inicializar los pesos $W[j] \leftarrow 0$; $j = 1, 2, \ldots, a$.
2: Generar una permutación aleatoria π, de n elementos.
3: **for** $\hat{i} = 1 \rightarrow \hat{n}$ $(\hat{n} \leq n)$ **do**
4: Encontrar las b instancias más cercanas a $I_{\pi(\hat{i})}$ de la misma clase (H_l, $l = 1, \ldots, b$).
5: **for all** $C_k \neq \Omega(I_{\pi(\hat{i})})$ **do**
6: Encontrar las b instancias de clase C_k más cercanas a $I_{\pi(\hat{i})}$ ($M_l(C_k)$, $l = 1, \ldots, b$).
7: **end for**
8: **for** $j = 1 \rightarrow a$ **do**
9: $$W[j] \leftarrow W[j] - \frac{\sum_{l=1}^{b} diff(A_j, I_{\pi(\hat{i})}, H_l)}{(\hat{n} \cdot b)} + \frac{\sum_{C_k \neq \Omega\left(I_{\pi(\hat{i})}\right)} \left[\frac{\mathbb{P}(C_k)}{1 - \mathbb{P}\left(\Omega\left(I_{\pi(\hat{i})}\right)\right)} \sum_{l=1}^{b} diff(A_j, I_{\pi(\hat{i})}, M_l(C_k))\right]}{(\hat{n} \cdot b)}.$$
10: **end for**
11: **end for**
12: **return** W

Al observar el Algoritmo 2, la primera diferencia que llama la atención respecto a *Relief* es que ahora, en lugar de considerar únicamente el vecino más cercano a la instancia analizada en cada iteración, se consideran b vecinos más cercanos. Inmediatamente después se aprecia la siguiente diferencia, considerar varias clases en lugar de solamente dos. Se toman los b vecinos más cercanos de cada una de las c clases posibles (hay, por tanto, $b \cdot (c-1)$ instancias pertenecientes a las $c-1$ clases distintas a $\Omega(I_{\pi(\hat{i})})$).

Tomar b vecinos en lugar de uno, aporta mayor robustez al método. Sin embargo, no funciona mejor cuanto mayor sea b. Si se consideran demasiados vecinos, se pierde la perspectiva *local* del método y dejan de detectarse ciertas dependencias entre atributos. El valor más apropiado para b depende del problema y debería ser elegido, entre varios candidatos, mediante *validación cruzada*.

La actualización de los pesos $W[j]$ es, en esencia, igual para c clases que para 2 clases. Los cambios son consecuencia natural de considerar b vecinos y de exigir que $W[j] \in [-1, 1]$. Al considerar más vecinos, deben sumarse las aportaciones de todos ellos y dividir por $(\hat{n} \cdot b)$, en lugar de hacerlo solamente por $\hat{n}$, para respetar la restricción impuesta a los valores de $W[j]$. Además, aparece un nuevo término de normalización $\left(\frac{\mathbb{P}(C_k)}{1 - \mathbb{P}(\Omega(I_{\pi(\hat{i})}))}\right)$ para ajustar los

pesos en función de las probabilidades a priori de cada clase.

Lo que no se aprecia en el Algoritmo 2 es cómo tratar los valores perdidos. Esta tarea queda enmascarada en la función $diff$. Si se conoce el valor del atributo para las dos instancias que se analicen, $diff$ se calcula igual que en *Relief* (sección 3.1.2). Sin embargo, si dicho valor es desconocido para alguna o ambas instancias, $diff(A_j, I_1, I_2)$ se calcula de la siguiente manera:

- Supongamos, sin pérdida de generalidad, que es la instancia I_1 para la que se desconoce el valor de A_j, entonces

$$diff(A_j, I_1, I_2) = 1 - \mathbb{P}(\Psi(A_j, I_2) \mid \Omega(I_1)).$$

- Si el valor de A_j es desconocido para ambas instancias,

$$diff(A_j, I_1, I_2) = 1 - \sum_{s_j=1}^{m_j} \left(\mathbb{P}(\alpha_{s_j} \mid \Omega(I_1)) \times \mathbb{P}(\alpha_{s_j} \mid \Omega(I_2))\right).$$

Los autores no dan más detalles pero las expresiones anteriores solamente permiten el manejo de valores perdidos para atributos nominales. Para abordar el caso de atributos continuos existen al menos dos opciones:

- Discretizarlos para poder aplicar las fórmulas anteriores.

- Suponer que los valores del atributo siguen una determinada distribución de probabilidad y estimar sus parámetros a partir del conjunto de entrenamiento.

3.1.4. RReliefF

Se ha visto en la sección 3.1.3 que la extensión de *Relief* para soportar problemas multi-clase y manejo de valores perdidos no requiere ningún cambio en la esencia de los cálculos del algoritmo. Sin embargo, la situación es diferente en problemas de regresión, ya que la variable objetivo es continua y, por tanto, no determina clases de manera categórica. Para abordar esta nueva extensión, se parte desde un enfoque teórico. La estimación que hace *Relief* sobre la calidad de un atributo A_j es una aproximación de la siguiente diferencia de probabilidades:

$$\begin{aligned} W[j] \simeq\ & \mathbb{P}(\Psi(A_j, I_1) \neq \Psi(A_j, I_2) \mid vec(I_1, I_2),\ \Omega(I_1) \neq \Omega(I_2)) \\ & - \mathbb{P}(\Psi(A_j, I_1) \neq \Psi(A_j, I_2) \mid vec(I_1, I_2),\ \Omega(I_1) = \Omega(I_2)). \end{aligned} \tag{3.1}$$

El suceso $vec(I_1, I_2)$ indica que las instancias I_1 e I_2 se encuentran lo suficientemente cerca en el espacio de atributos y se consideran vecinas. Con I_1 e I_2 no se hace referencia a instancias concretas, sino a cualquier par de instancias en general. Expresado con palabras, lo que se está estimando es la diferencia entre la probabilidad de que el valor del atributo A_j sea diferente en instancias vecinas de distinta clase y, la probabilidad del mismo suceso, en el caso de vecinas de igual clase. De esta manera, se evalúa *localmente* la capacidad de discriminación del atributo A_j.

En problemas de regresión ya no se puede hablar en términos de clases, sino de valor objetivo ($\mathcal{T}(I)$). Por tanto, en lugar de los sucesos $\Omega(I_1) \neq \Omega(I_2)$ y $\Omega(I_1) = \Omega(I_2)$ (*diferente/misma clase*), se emplean en regresión $\mathcal{T}(I_1) \neq \mathcal{T}(I_2)$ y $\mathcal{T}(I_1) = \mathcal{T}(I_2)$ (*diferente/mismo valor objetivo($\mathcal{T}$)*), considerando una precisión ϵ.

Para no extender en exceso el desarrollo de la ecuación 3.1, se abrevian los sucesos y determinadas probabilidades con la siguiente *notación* (se prescinde del subíndice j, ya que A representa un atributo cualquiera):

- $vec(I_1, I_2)$ (vecinas) $\rightarrow vec$.
- $\mathcal{T}(I_1) \neq \mathcal{T}(I_2)$ / $\mathcal{T}(I_1) = \mathcal{T}(I_2)$ (diferente/mismo $\mathcal{T}$) $\rightarrow dif\mathcal{T}/mis\mathcal{T}$.
- $\Psi(A_j, I_1) \neq \Psi(A_j, I_2)$ (valor diferente de A_j) $\rightarrow difA$.
- $\mathbb{P}_{difA} = \mathbb{P}(difA \mid vec)$.
- $\mathbb{P}_{dif\mathcal{T}} = \mathbb{P}(dif\mathcal{T} \mid vec)$.
- $\mathbb{P}_{dif\mathcal{T}|difA} = \mathbb{P}(dif\mathcal{T} \mid difA,\ vec)$.
- $\mathbb{P}_{dif\mathcal{T}\&difA} = \mathbb{P}(dif\mathcal{T},\ difA \mid vec)$.

Aplicando sucesivamente el Teorema de Bayes sobre la ecuación 3.1, así como algunas propiedades básicas de la probabilidad (teniendo en cuenta que los sucesos $dif\mathcal{T}$ y $mis\mathcal{T}$ son complementarios), se obtiene que:

$$\begin{aligned} W[j] &= \mathbb{P}(difA \mid dif\mathcal{T},\ vec) - \mathbb{P}(difA \mid mis\,\mathcal{T},\ vec) \\ &= \frac{\mathbb{P}(difA,\ dif\mathcal{T},\ vec)}{\mathbb{P}(dif\mathcal{T},\ vec)} - \frac{\mathbb{P}(difA,\ mis\,\mathcal{T},\ vec)}{\mathbb{P}(mis\,\mathcal{T},\ vec)} \\ &= \frac{\mathbb{P}(dif\mathcal{T} \mid\ difA,\ vec)\mathbb{P}(difA,\ vec)}{\mathbb{P}(dif\mathcal{T} \mid vec)\mathbb{P}(vec)} \\ &\quad - \frac{\mathbb{P}(mis\,\mathcal{T} \mid\ difA,\ vec)\mathbb{P}(difA,\ vec)}{\mathbb{P}(mis\,\mathcal{T} \mid vec)\mathbb{P}(vec)} \\ &= \frac{\mathbb{P}(dif\mathcal{T} \mid\ difA,\ vec)\mathbb{P}(difA,\ vec)}{\mathbb{P}(dif\mathcal{T} \mid vec)\mathbb{P}(vec)} \end{aligned}$$

$$
\begin{aligned}
& -\frac{(1-\mathbb{P}(dif\mathcal{T} \mid\ difA,\ vec))\mathbb{P}(difA,\ vec)}{(1-\mathbb{P}(dif\mathcal{T} \mid vec))\mathbb{P}(vec)} \\
= \quad & \frac{\mathbb{P}_{dif\mathcal{T}|difA}\mathbb{P}(difA \mid vec)\mathbb{P}(vec)}{\mathbb{P}_{dif\mathcal{T}}\mathbb{P}(vec)} \\
& -\frac{(1-\mathbb{P}_{dif\mathcal{T}|difA})\mathbb{P}(difA \mid vec)\mathbb{P}(vec)}{(1-\mathbb{P}_{dif\mathcal{T}})\mathbb{P}(vec)} \\
= \quad & \frac{\mathbb{P}_{dif\mathcal{T}|difA}\mathbb{P}_{difA}}{\mathbb{P}_{dif\mathcal{T}}} - \frac{(1-\mathbb{P}_{dif\mathcal{T}|difA})\mathbb{P}_{difA}}{1-\mathbb{P}_{dif\mathcal{T}}} \\
= \quad & \frac{\mathbb{P}_{dif\mathcal{T}\&difA}}{\mathbb{P}_{dif\mathcal{T}}} - \frac{\mathbb{P}_{difA}-\mathbb{P}_{dif\mathcal{T}\&difA}}{1-\mathbb{P}_{dif\mathcal{T}}}.
\end{aligned}
$$

El algoritmo *RReliefF* (Algoritmo 3) se presenta como una extensión de la familia *Relief* para cubrir los problemas de regresión. Se identifica con el desarrollo teórico expuesto anteriormente. De hecho, se puede observar que las tres variables $N_{d\mathcal{T}}$, N_{dA} y $N_{d\mathcal{T}\&dA}$, empleadas por *RReliefF*, son las estimaciones muestrales de $\mathbb{P}_{dif\mathcal{T}}$, $\mathbb{P}_{difA}$ y $\mathbb{P}_{dif\mathcal{T}\&difA}$, respectivamente.

Algoritmo 3 RReliefF

Entrada: $E = \{(I_i, t_i)\}_{i=1}^{n}$. Conjunto de n instancias etiquetadas.
Salida: $W \in [-1,1]^a$. Vector de estimaciones de la calidad de cada atributo.
Procedimiento:

Inicializar $N_{d\mathcal{T}}, N_{dA}[j], N_{d\mathcal{T}\&dA}[j], W[j] \leftarrow 0$; $j = 1, 2, \ldots, a$.
Generar una permutación aleatoria π, de n elementos.
for $\hat{i} = 1 \rightarrow \hat{n}$ $(\hat{n} \leq n)$ **do**
  Encontrar las b instancias más cercanas a $I_{\pi(\hat{i})}$ $(J_l,\ l = 1, \ldots, b)$.
  for $l = 1 \rightarrow b$ **do**
    $N_{d\mathcal{T}} \leftarrow N_{d\mathcal{T}} + diff(\mathcal{T}(\cdot), I_{\pi(\hat{i})}, J_l) \cdot d(\hat{i}, l)$.
    for $j = 1 \rightarrow a$ **do**
      $N_{dA}[j] \leftarrow N_{dA}[j] + diff(A_j, I_{\pi(\hat{i})}, J_l) \cdot d(\hat{i}, l)$.
      $N_{d\mathcal{T}\&dA}[j] \leftarrow N_{d\mathcal{T}\&dA}[j]$
        $+diff(\mathcal{T}(\cdot), I_{\pi(\hat{i})}, J_l) \cdot diff(A_j, I_{\pi(\hat{i})}, J_l) \cdot d(\hat{i}, l)$.
    end for
  end for
end for
for $j = 1 \rightarrow a$ **do**
  $W[j] \leftarrow \frac{N_{d\mathcal{T}\&dA}[j]}{N_{d\mathcal{T}}} - \frac{(N_{dA}[j]-N_{d\mathcal{T}\&dA}[j])}{(\hat{n}-N_{d\mathcal{T}})}$.
end for
return W

En el Algoritmo 3 se aplica la función $diff$ a las etiquetas de las instancias, además de a los atributos. No se le da otro nombre porque la forma de calcularla

para una etiqueta es similar a la de un atributo numérico. Únicamente varía la notación para acceder a los valores, como se observa en la siguiente expresión:

$$diff(\mathcal{T}(\cdot), I_1, I_2) = \frac{|\mathcal{T}(I_1) - \mathcal{T}(I_2)|}{max\{t_i\}_{i=1}^{n} - min\{t_i\}_{i=1}^{n}}.$$

El término $d(\hat{i}, l)$ se emplea para ponderar la influencia que ejercen las instancias vecinas en la actualización de los pesos. La influencia de un vecino debe ser menor cuanto más alejado se encuentre de la instancia analizada. Además, dicha influencia se normaliza en función de la suma de las influencias de los b-vecinos. Hay muchas alternativas para el cálculo de $d(\hat{i}, l)$, siendo una posibilidad:

$$d(\hat{i}, l) = \frac{d_1(\hat{i}, l)}{\sum_{\hat{l}=1}^{b} d_1(\hat{i}, \hat{l})},$$

$$d_1(\hat{i}, l) = e^{-\left(\frac{rank\left(I_{\pi(\hat{i})}, J_l\right)}{\sigma}\right)^2}.$$

Donde, $rank(I_{\pi(\hat{i})}, J_l)$ indica la posición que ocupa J_l en una lista de instancias ordenadas de menor a mayor distancia a $I_{\pi(\hat{i})}$ (calculada mediante la función δ de la sección 3.1.2) y σ es un parámetro que se debe adaptar a la muestra, usando para su determinación validación cruzada.

En lugar de la función *rank*, se puede utilizar directamente la distancia entre ambas instancias. En cuanto a la función d_1, se pueden emplear otras funciones decrecientes. Dependiendo del problema abordado, funcionan mejor unas configuraciones u otras.

En el algoritmo *ReliefF* (sección 3.1.3), ya existe una ponderación encubierta de la influencia de los vecinos, equivalente a tomar $d_1 = 1/b$. No cumple la condición de ser decreciente con la distancia, ya que no surge de la intención de ponderar, sino porque es la forma más sencilla de normalizar los pesos de *ReliefF*. No obstante, se puede incluir en dicho algoritmo la función de ponderación d, con cualquiera de las alternativas para d_1 propuestas en esta sección.

El número óptimo de vecinos (b^*) depende del problema. Si se escoge b excesivamente grande, el algoritmo relega a un segundo plano las dependencias locales y prevalece una estimación global [1], que no es lo deseable en este caso.

[1] Los autores, con ingenio y quizás con cierto tono humorístico, califican el fenómeno diciendo que el algoritmo se vuelve *miope*.

3.1.5. Observaciones

3.1.5.1. Interpretación teórica

Hay otra manera de interpretar el significado de los pesos obtenidos mediante *Relief* o *ReliefF*, además de la diferencia de probabilidades desarrollada en la sección 3.1.3. Para ello, se define el concepto de *responsabilidad*: se dice que un atributo A es *responsable* del cambio de valor de la etiqueta t de una instancia I, si el cambio de su valor ($\Psi(A, I)$) es uno de los cambios mínimos necesarios para que se produzca un cambio en el valor de t.

Se puede cuantificar la *responsabilidad*, asignando una cuota a cada instancia y repartiéndola a partes iguales entre los conjuntos de atributos responsables de los cambios en su etiqueta. Los valores de los atributos de cada conjunto deben cambiar simultáneamente para que se modifique el valor de la etiqueta, por lo que cada uno recibe la cuota total del conjunto [II]. La *cuota de responsabilidad* de un atributo se obtiene sumando las cantidades recibidas en todas las instancias.

En el caso de *Relief*, se define el *ratio de responsabilidad* del atributo A_j como:

$$R_{A_j} \equiv \frac{\text{cuota de responsabilidad del atributo } A_j}{\text{nº de instancias examinadas}}.$$

Si se dispone de n instancias libres de ruido (recordar que solamente se examinan $\hat{n}$, $\hat{n} \leq n$, de dichas instancias), se cumple:

$$\lim_{n\to\infty} W[j] = R_{A_j}.$$

En el caso de *ReliefF*, los cambios en la etiqueta se distribuyen en función de cuál es la clase original de una instancia y cuál es la nueva clase tras cambiar el valor del atributo A_j. De esta forma, se puede desglosar R_{A_j} en función de la transferencia entre clases (original, modificada) y se obtienen varios ratios de responsabilidad, $R_{A_j}(k, \hat{k})$. En cada uno de esos ratios solamente se considera la cuota de responsabilidad de A_j correspondiente a los cambios en los que la etiqueta pase de clase C_k a $C_{\hat{k}}$. Si se emplean las probabilidades a priori de las clases correspondientes para ponderar, se cumple:

$$\lim_{n\to\infty} W[j] = \sum_{k=1}^{c} \sum_{\hat{k}\neq k}^{c} \frac{\mathbb{P}(C_k)\mathbb{P}(C_{\hat{k}})}{1 - \mathbb{P}(C_k)} R_{A_j}(k, \hat{k}).$$

[II] Por ejemplo, si los cambios mínimos deben producirse en $\{A_1, A_2\}$ o $\{A_1, A_3\}$, a cada conjunto le corresponde la mitad de la cuota de la instancia. A continuación, los atributos reciben la cuota completa de los conjuntos a los que pertenecen. Por tanto, A_1, suma dos mitades porque pertenece a los dos conjuntos y recibe en total la cuota completa asignada a la instancia, mientras que A_2 y A_3 reciben la mitad cada uno porque solamente pertenecen a uno de los conjuntos.

En problemas pequeños y cuyo dominio es perfectamente conocido [III] se pueden determinar con exactitud los *ratios de responsabilidad* de cada atributo y se ha comprobado experimentalmente que el valor de los pesos se va aproximando a la expresión obtenida teóricamente, a medida que crece el número de instancias. En el caso de problemas reales es prácticamente imposible trabajar con estos ratios porque, ni se suele conocer completamente el dominio del problema, ni se suele disponer de un número suficiente de instancias libres de ruido.

3.1.5.2. Métrica

Se ha visto en la sección 3.1.2 que, para el cálculo de las instancias más cercanas, se usa la distancia de Manhattan tomando como base la función $diff$. Sin embargo, es posible emplear otras métricas:

- Añadir un término de ponderación a cada atributo como, por ejemplo, el propio peso que va estimando el algoritmo en cada momento.

$$\delta(I_1, I_2) = \sum_{j=1}^{a} \omega(j) diff(A_j, I_1, I_2).$$

- Considerar la distancia euclídea, o en general una norma L^p (L^1 es precisamente la distancia de Manhattan y L^2 la euclídea).

$$\delta(I_1, I_2) = \left(\sum_{j=1}^{a} diff(A_j, I_1, I_2)^p \right)^{\frac{1}{p}}.$$

3.1.5.3. Función $diff$

La forma de definir la función $diff$ (sección 3.1.2) hace que se infraestime un atributo numérico frente a otro nominal que contenga esencialmente la misma información. Esta circunstancia es provocada por la normalización aplicada a los atributos numéricos, al dividir por la diferencia entre el máximo y el mínimo de sus valores.

Para solucionarlo, se propone una nueva definición de la función $diff$ para atributos numéricos:

$$diff(A_j, I_1, I_2) = \begin{cases} 0 & d \leq \gamma_1 \\ 1 & d > \gamma_2 \\ \frac{d-\gamma_1}{\gamma_2-\gamma_1} & \gamma_1 < d \leq \gamma_2 \end{cases}.$$

Donde,

[III] En inglés se suele emplear el término *toy example* para estos casos. No se corresponden con la inmensa mayoría de problemas reales pero permiten un estudio más exhaustivo en un entorno controlado.

- $d = |\Psi(A_j, I_1) - \Psi(A_j, I_2)|$. Es la diferencia (en valor absoluto) entre los valores del atributo A_j en las instancias I_1 e I_2.
- $\gamma_1 \equiv$ umbral de precisión. Si d es menor que γ_1, los valores del atributo en las dos instancias se consideran exactamente iguales.
- $\gamma_2 \equiv$ umbral de diferencia. Si d es mayor que γ_2, los valores del atributo en las dos instancias se consideran completamente diferentes.

De esta forma, si $d \leq \gamma_1$ o $d > \gamma_2$, el atributo numérico recibe el mismo tratamiento que un nominal. Es fácil observar que si se toma $\gamma_1 = 0$ y $\gamma_2 = max\{\alpha_{s_j}\}_{s_j=1}^{m_j} - min\{\alpha_{s_j}\}_{s_j=1}^{m_j}$, se obtiene la antigua definición de $diff$. Si se tienen conocimientos específicos sobre el dominio del problema se pueden escoger otros umbrales o incluso otras funciones no tan abruptas, como, por ejemplo, una sigmoidal. En caso contrario, se propone como norma general fijar:

- $\gamma_1 = 0{,}05 \cdot (max\{\alpha_{s_j}\}_{s_j=1}^{m_j} - min\{\alpha_{s_j}\}_{s_j=1}^{m_j})$.
- $\gamma_2 = 0{,}10 \cdot (max\{\alpha_{s_j}\}_{s_j=1}^{m_j} - min\{\alpha_{s_j}\}_{s_j=1}^{m_j})$.

3.1.6. Complejidad computacional

Siendo n el número de instancias de entrenamiento, a el número de atributos y $\hat{n}$ el número de iteraciones del bucle principal en los Algoritmos 1, 2 y 3, la complejidad computacional de todos ellos es $O(\hat{n} \cdot n \cdot a)$. En los siguientes párrafos se justifica esta afirmación.

En los tres algoritmos, en cada iteración del bucle principal, es necesario calcular las distancias entre la instancia seleccionada y todas las demás. Estas operaciones conllevan un coste de $O(n \cdot a)$, ya que el cálculo de cada distancia implica la realización de un sumatorio sobre todos los atributos. Por tanto, al multiplicar dicho coste por el número de iteraciones del bucle, se obtiene que la complejidad es, al menos, $O(\hat{n} \cdot n \cdot a)$. A continuación se muestra que el coste del resto de operaciones del bucle es inferior a $O(n \cdot a)$.

En *Relief*, encontrar la instancia más cercana de cada una de las clases tiene coste $O(1)$ si se computa al mismo tiempo que se calculan las distancias. El bucle de actualización de pesos tiene coste $O(a)$. Ambos órdenes de complejidad son claramente inferiores a $O(n \cdot a)$.

En *ReliefF*, el bucle de actualización de pesos también tiene coste $O(a)$. Sin embargo, el hecho de tener que encontrar las b instancias más cercanas de cada una de las clases, implicaría una mayor complejidad que la esperada si no se implementa eficientemente. Una solución posible consiste en utilizar la

estructura de datos *min-heap*. Para cada una de las c clases se construye un *min-heap* con las distancias entre la instancia seleccionada y las demás. Esta operación tiene coste $O(c \cdot n) \equiv O(n)$ [IV]. La extracción del primer elemento del *heap* tiene coste $O(log\ n)$ (debido al procesamiento posterior a la extracción, que permite mantener la estructura). Dicho elemento corresponde al de menor distancia. Por tanto, para las c clases, el coste total de encontrar los b-vecinos más cercanos es de $O(c \cdot b \cdot log\ n) \equiv O(log\ n)$. Nuevamente, los costes son inferiores a $O(n \cdot a)$.

En *RReliefF*, el doble bucle de estimación de $N_{d\mathcal{T}}$, N_{dA} y $N_{d\mathcal{T}\&dA}$ tiene coste $O(b \cdot a) \equiv O(a)$. Para seleccionar las b instancias que se consideran en dicho doble bucle, se emplea la misma estrategia que en *ReliefF* y, como ahora no se distinguen c clases, el coste es de $O(b \cdot log\ n) \equiv O(log\ n)$. Por tanto, se cumple la afirmación enunciada inicialmente.

El parámetro $\hat{n}$ corresponde al número de instancias que se examinan, se fija externamente por el usuario y su valor óptimo puede variar para cada problema abordado. Lo que se deduce de los experimentos expuestos en [10] es que dicho valor óptimo se sitúa en el rango $log\ n << \hat{n} \leq n$.

3.2. Otros métodos

Existen otros algoritmos de selección de variables, muchos de los cuales no tienen en cuenta las relaciones entre los atributos, sino simplemente la relación de cada atributo con la variable objetivo. Los problemas del ámbito médico suelen ser complejos y, por tanto, es necesario valorar las dependencias entre los atributos, como hacen los algoritmos de la familia *Relief*.

Los métodos *Minimum Redundancy Maximum Relevance (mRMR)* [13] y *Quadratic Programming Feature Selection (QPFS)* [15] son capaces de detectar dichas dependencias. El enfoque de ambos es global, es decir, evalúan todas las instancias en lugar de solamente los entornos de un subconjunto de ellas. No obstante, existe una diferencia fundamental entre *mRMR* y *QPFS*: el primero sigue una estrategia codiciosa para detectar dependencias en subconjuntos de atributos, mientras que el segundo toma en consideración todas las posibles interacciones entre atributos. A continuación se explica muy brevemente el funcionamiento de estos dos algoritmos, se comparan los costes computacionales con los de la familia *Relief* y se propone una forma de evaluar el comportamiento de todos ellos frente a atributos ruidosos.

[IV] La construcción eficiente de un *heap* tiene coste $O(n)$. Si se construyese mediante inserciones sucesivas, el coste sería de $O(n \cdot log\ n)$.

3.2.1. *Minimum Redundancy Maximum Relevance*

Abreviado por las siglas mRMR, este método de selección de variables, como su propio nombre da a entender, trata de obtener un subconjunto de atributos de tal manera que sean lo más diferentes posible entre sí (mínima redundancia) y que aporten la máxima información posible sobre la variable objetivo (máxima relevancia) [13]. Para cuantificar las relaciones entre las variables se emplean medidas de similitud, como el *coeficiente de correlación de Pearson* o la *información mutua*.

El algoritmo calcula, en primer lugar, la relevancia de los a atributos con la variable objetivo, obteniendo una puntuación numérica para cada uno. A partir de ese momento, el funcionamiento sigue un esquema codicioso, ya que el cálculo de las redundancias de todos los posibles subconjuntos de atributos conlleva una excesiva complejidad computacional. Por tanto, se escoge el atributo cuya relevancia con la variable objetivo sea máxima y se calcula la redundancia con cada uno de los otros $a - 1$ atributos. A mayor redundancia, menor puntuación se otorga al subconjunto correspondiente. Se toma ahora el subconjunto con mayor puntuación, nuevamente se calculan las redundancias con los demás atributos $(a - 2)$ y se repite el proceso hasta alcanzar el número de atributos deseado.

El coste computacional es $O(n \cdot a^2)$, ya que, para incluir un nuevo atributo en el conjunto seleccionado, se deben calcular las redundancias con todos los restantes, lo que implica recorrer todas las instancias para el cálculo de la medida de similitud.

3.2.2. *Quadratic Programming Feature Selection*

Abreviado por las siglas QPFS, el objetivo principal del algoritmo es similar al de *mRMR*. Es decir, minimizar las redundancias entre los atributos seleccionados a la vez que se maximiza la relevancia respecto a la variable objetivo. De hecho, se pueden emplear las mismas medidas de similitud. Sin embargo, *QPFS* sí considera todas las dependencias entre atributos [15].

En el algoritmo se plantea un problema de minimización cuadrático, basado en una matriz de redundancias entre cada par de atributos y un vector de relevancias de cada atributo con la variable objetivo. Resolver dicho problema con exactitud dispararía el coste computacional, por lo que se descartan los autovalores menos significativos en el proceso de diagonalización de la matriz de redundancias y se alcanza una solución aproximada.

Aún así, la complejidad del método sigue siendo elevada en el caso de problemas con un gran número de atributos. Por ello, se aplica la técnica de *Nyström*

para aproximar matrices y sus autovalores y autovectores [15]. Dicha técnica permite diagonalizar más eficientemente la matriz de redundancias, ya que requiere el cálculo exacto de solamente $p \cdot a$ filas de la matriz de redundancias ($0 < p \leq 1$). El inconveniente es la introducción de un error adicional al que ya se produce por la resolución aproximada del problema de minimización. No obstante, ambos errores están acotados, lo cual garantiza un rendimiento aceptable del método.

La complejidad computacional de *QPFS* depende de la proporción de atributos (a) frente a instancias (n), ya que existen dos partes bien diferenciadas que tienen costes diferentes. Por un lado se debe calcular inicialmente la matriz de redundancias y, por otro lado, se resuelve el problema de minimización para obtener el ranking de atributos. El coste computacional de *QPFS* es $max(O(n \cdot a^2), O(a^3))$ y el de *QPFS + Nyström* es $max(O(n \cdot p \cdot a^2), O(p^2 \cdot a^3))$.

3.2.3. Comparativa con familia *Relief*

Se desea comparar dos aspectos de los algoritmos *mRMR*, *QPFS* y la familia *Relief*: la complejidad computacional y el comportamiento que presentan ante atributos ruidosos.

3.2.3.1. Coste computacional

La Tabla 3.1 recoge la complejidad computacional de los algoritmos de selección de atributos presentados. En ella se observa que *QPFS* es siempre igual o más costoso que *mRMR*, mientras que *QPFS+Nyström* es más eficiente que *mRMR* si $n > a$. En el caso de que existan más variables que instancias ($a > n$), se puede disminuir el valor de p para que *QPFS+Nyström* siga siendo más eficiente que *mRMR* pero teniendo en cuenta que, a menor valor de p, mayor error se introduce en el cálculo del ranking de atributos.

Tabla 3.1: Complejidad computacional de varios algoritmos de selección de atributos. n es el número de instancias, a el número de atributos, $\hat{n}$ el número de iteraciones de los algoritmos *Relief* y p la porción de filas de la matriz de redundancias que se debe calcular con exactitud para el método de Nyström.

Método	Complejidad
Relief (y extensiones)	$O(n \cdot a \cdot \hat{n})$
mRMR	$O(n \cdot a^2)$
QPFS	$max(O(n \cdot a^2), O(a^3))$
QPFS+Nyström	$max(O(n \cdot a^2 \cdot p), O(a^3 \cdot p^2))$

En los algoritmos de la familia *Relief* aparece el parámetro $\hat{n}$, correspondiente al número de instancias que se analizan (iteraciones del algoritmo). Se sabe por los experimentos de [10] que dicho valor se debe fijar dentro del rango $log\ n << \hat{n} \leq n$ para lograr resultados satisfactorios. Por tanto, en los problemas en los que $n >> a$, *Relief* es más costoso que el resto de algoritmos. En cambio, en los problemas con $a >> n$, es el algoritmo más eficiente, ya que depende linealmente del número de atributos mientras que los demás dependen cuadrática o cúbicamente.

3.2.3.2. Comportamiento

La cualidad fundamental en cualquier método de selección de atributos, es que sea capaz de discriminar entre los atributos que contienen información útil y aquellos que se pueden considerar simplemente ruido. A partir de las puntuaciones asignadas por un algoritmo a cada atributo (el vector de pesos $W \in \mathbb{R}^a$), se puede analizar la eficacia de los algoritmos de selección de variables a la hora de distinguir los atributos importantes (G) de los aleatorios o ruidosos (B). Sea $W_{G\,\{mejor,\,peor\}} = \{max,\ min\}(W_g)_{g \in G}$ y $W_{b\,mejor} = max(W_b)_{b \in B}$, donde W_g y W_b son los pesos correspondientes a atributos importantes y aleatorios, respectivamente. Entonces, se definen los siguientes dos conceptos.

- Separabilidad: es la diferencia entre la estimación más baja de los atributos importantes y la más alta de los aleatorios.

$$s = W_{G\,peor} - W_{B\,mejor}.$$

- Usabilidad: es la diferencia entre la estimación más alta de los atributos importantes y la más alta de los aleatorios.

$$u = W_{G\,mejor} - W_{B\,mejor}.$$

Es trivial observar que $u \geq s$. Además, lo ideal para cualquier método de selección de atributos, es que se obtenga $s > 0$, lo que significa que se aísla completamente el ruido. No obstante, basta con tener $u > 0$ para que el método sea útil, ya que distinguiría alguno de los atributos más importantes respecto al ruido.

En la sección 5.1 se realizan algunos experimentos para comparar los algoritmos de la familia *Relief* con mRMR y QPFS, en términos de usabilidad y separabilidad.

Capítulo 4

Análisis de predicciones individuales

Los modelos de aprendizaje automático se suelen analizar y evaluar globalmente, es decir, promediando ciertas medidas de error, eficiencia, etc. sobre conjuntos completos de datos. Sin embargo, cuando se aplican a ciertos dominios es más conveniente un análisis a nivel de instancia. Si se trata de diagnosticar un paciente, el riesgo de error puede no corresponderse con el error global del modelo. Además, si cuando el sistema emite una predicción, la acompaña de información adicional que la justifique, el usuario podrá valorar adecuadamente la decisión final, en lugar de solamente aceptar o rechazar, a ciegas, el resultado del modelo.

Por ello, en este capítulo se estudiarán algunas técnicas para estimar la fiabilidad de las predicciones individuales en modelos de regresión (sección 4.1) y para ofrecer explicaciones de la predicción, en modelos de clasificación, basadas en la contribución de cada atributo (sección 4.2).

4.1. Estimación de fiabilidad

Bosnic y Kononenko presentan en [16] una aproximación a la estimación de la fiabilidad de predicciones individuales en modelos de regresión. Se basa en dos ideas preexistentes en el aprendizaje automático:

- Análisis local de sensibilidad. Se utiliza para determinar la estabilidad de un modelo.
- Perturbaciones del conjunto de aprendizaje para mejorar el rendimiento de los modelos.

Cabe destacar que, para realizar las estimaciones de fiabilidad, no es necesario conocer a priori la etiqueta de la instancia, es decir, se puede aplicar en predicción de nuevas instancias.

Tras un análisis teórico que formaliza y justifica la intuición inicial (sección 4.1.1), se propone una metodología y algunas medidas de fiabilidad que se aplican a varios modelos de regresión conocidos (sección 4.1.2): *regresión lineal*, *regresión ponderada localmente*, *árboles de regresión*, *redes neuronales artificiales* y *máquinas de vectores de soporte*. Se observa que para los dos primeros se puede determinar de forma sencilla y explícita cómo afecta una perturbación a la predicción. Por tanto, carece de interés aplicar la metodología expuesta sobre ellos. En cambio, aplicada sobre los tres últimos permite obtener información valiosa.

4.1.1. *Minimum Description Length*

Fundamentado en la teoría de la información y de la probabilidad, el principio de *MDL* formaliza la suposición de que la mejor hipótesis (modelo) es aquella que permite una mayor compresión de los datos (*Navaja de Occam*). Se pueden definir nociones de fiabilidad y se puede estudiar lo que sucede al añadir nuevas instancias al conjunto de entrenamiento y cómo afecta al modelo y a la capacidad de compresión.

4.1.1.1. Notación

Se dispone de un conjunto $E = \{(I_i, t_i)\}_{i=1}^{n}$, de instancias de entrenamiento etiquetadas, donde cada par (I_i, t_i) se puede abreviar por e_i. Si se añade un nuevo ejemplo $e_{n+1} = (I_{n+1}, t_{n+1})$ a E, se obtiene el conjunto $\overline{E} = E \cup \{e_{n+1}\}$. Un modelo (o hipótesis) entrenado sobre E, se denota por H y si se entrena sobre $\overline{E}$, se denota por $\overline{H}$. La letra B representa el conocimiento a priori que se tiene sobre un problema (*background*). Por último, Z es una codificación y $Z(\cdot)$ indica el número de bits necesarios para representar cierta información. Por ejemplo, $Z(E \mid H, B)$ es el número de bits necesarios para codificar los ejemplos de entrenamiento siendo H y B conocidos.

4.1.1.2. Modelo óptimo

El modelo óptimo (H_{opt}), dados E y B, es el que maximiza la probabilidad condicionada o, en términos de teoría de la información, el que minimiza el opuesto del logaritmo de dicha probabilidad:

$$H_{opt} = \underset{H}{\operatorname{argmax}}\, \mathbb{P}(H \mid E,\ B) = \underset{H}{\operatorname{argmin}}\ -\log_2 \mathbb{P}(H \mid E,\ B).$$

Aplicando el Teorema de Bayes, propiedades de logaritmos y sabiendo que $\mathbb{P}(E \mid B)$ no depende de H (y por tanto no afecta a la minimización), se obtiene:

$$H_{opt} = \underset{H}{\operatorname{argmin}}\ \left[-\log_2 \mathbb{P}(E \mid H,\ B) - \log_2 \mathbb{P}(H \mid B)\right].$$

El primer término ($\mathbb{P}(E \mid H, B)$) corresponde a la minimización del error respecto a los ejemplos observados y el segundo ($\mathbb{P}(H \mid B)$) a la minimización de la complejidad del modelo, primando la sencillez del mismo.

Para que la formulación anterior se pueda emplear con absoluta precisión, es necesario conocer la codificación óptima para describir los ejemplos y el modelo, lo cual es prácticamente imposible. En su lugar, se define $\tilde{H}$ como una aproximación de H_{opt}, basándose en una codificación prefijada y computable (Z).

$$\tilde{H} = \underset{H}{\operatorname{argmin}} \ [Z(E \mid H,\ B) + Z(H \mid B)]\,.$$

Introducir el uso de un modelo conlleva un coste adicional pero lo esperable es que el número de bits necesarios para codificar los ejemplos se reduzca. En caso contrario, no tiene sentido usar el modelo. Por tanto, se debe cumplir:

$$Z(E \mid H,\ B) + Z(H \mid B) < Z(E \mid B).$$

Basándose en dicha desigualdad y en la suposición de que el mejor modelo (más fiable) es el que permite una mayor compresión de la información, se definen las fiabilidades (globales) absoluta y relativa de un modelo.

- Fiabilidad absoluta (R):

$$R(H \mid E,\ B) = Z(E \mid B) - Z(E \mid H,\ B) - Z(H \mid B).$$

- Fiabilidad relativa (R_{rel}):

$$R_{rel}(H \mid E,\ B) = 1 - \frac{Z(E \mid H,\ B) + Z(H \mid B)}{Z(E \mid B)} = \frac{R(H \mid E,\ B)}{Z(E \mid B)}.$$

4.1.1.3. Inclusión de nuevas instancias

Se desea estudiar ahora la fiabilidad a nivel de una predicción individual, en lugar de a nivel de modelo. Para ello, se define la fiabilidad para un nuevo ejemplo e_{n+1}.

- Fiabilidad individual (R_1):

$$R_1(e_{n+1}, \overline{H}) = R(\overline{H} \mid \overline{E},\ B) - R(H \mid E,\ B).$$

Para simplificar la expresión anterior se realizan las siguientes suposiciones:

- $Z(H \mid B) \approx Z(\overline{H} \mid B)$.
- $Z(e_i \mid H,\ B) \approx Z(e_i \mid \overline{H},\ B)$, $i = 1, \ldots, n$.

- Todos los ejemplos (e_i, $i = 1, \ldots, n+1$) son independientes. Entonces se cumplen
  - $Z(E \mid B) = Z(e_1 \mid B) + \ldots + Z(e_n \mid B)$.
  - $Z(E \mid H,\ B) = Z(e_1 \mid H,\ B) + \ldots + Z(e_n \mid H,\ B)$.
  - $Z(\overline{E} \mid \overline{H},\ B) = Z(e_1 \mid \overline{H},\ B) + \ldots + Z(e_n \mid \overline{H},\ B) + Z(e_{n+1} \mid \overline{H},\ B)$.

De esta manera, se puede desarrollar R_1 y se obtiene:

$$R_1(e_{n+1}, \overline{H}) = Z(e_{n+1} \mid B) - Z(e_{n+1} \mid B,\ \overline{H}).$$

Se observa que R_1 no depende del modelo original (H) y que el término $Z(e_{n+1} \mid B)$ tampoco depende de $\overline{H}$. Es decir, la fiabilidad de una predicción individual depende únicamente del número de bits que resulten de codificar el nuevo ejemplo con el nuevo modelo ($Z(e_{n+1} \mid B,\ \overline{H})$). Existen dos casos extremos:

- $Z(e_{n+1} \mid B,\ \overline{H}) = 0$. Esto quiere decir que $\overline{H}$ contiene la máxima información para codificar e_{n+1}.
- $Z(e_{n+1} \mid B,\ \overline{H}) = Z(e_{n+1} \mid B)$. En este caso, $\overline{H}$ no incluye información adicional que ayude a codificar e_{n+1} de manera más compacta que la obtenida con el conocimiento a priori (B).

En el primer caso ($Z(e_{n+1} \mid B,\ \overline{H}) = 0$), la fiabilidad relativa del modelo $\overline{H}$ es mayor que la del modelo H. Se comprueba empleando las definiciones y las suposiciones enunciadas:

$$R_{rel}(\overline{H} \mid \overline{E},\ B) = \frac{R(\overline{H} \mid \overline{E},\ B)}{Z(\overline{E} \mid B)} = \frac{R(\overline{H} \mid E,\ B) + R_1(e_{n+1}, \overline{H})}{Z(E \mid B) + Z(e_{n+1} \mid B)}$$

$$\approx \frac{R(H \mid E,\ B) + Z(e_{n+1} \mid B)}{Z(E \mid B) + Z(e_{n+1} \mid B)} > \frac{R(H \mid E,\ B)}{Z(E \mid B)} = R_{rel}(H \mid E,\ B).$$

En el segundo ($Z(e_{n+1} \mid B,\ \overline{H}) = Z(e_{n+1} \mid B)$), la desigualdad es la contraria ($R_{rel}(\overline{H} \mid \overline{E},\ B) < R_{rel}(H \mid E,\ B)$) pero se comprueba de forma análoga.

Teorema 1. *Sean e_{n+1}, H, $\overline{H}$ y B. Si $Z(e_{n+1} \mid H,\ B) = 0$ para cierto algoritmo de aprendizaje óptimo, entonces*

$$Z(e_{n+1} \mid \overline{H},\ B) = 0.$$

Demostración. Según la definición de modelo óptimo:

- $H = \underset{T}{\operatorname{argmin}}\ [Z(E \mid T,\ B) + Z(T \mid B)]$.
- $\overline{H} = \underset{T}{\operatorname{argmin}}\ \big[Z(\overline{E} \mid T,\ B) + Z(T \mid B)\big]$.

Por tanto,

$$Z(\overline{E} \mid \overline{H},\ B) + Z(\overline{H} \mid B) \leq Z(\overline{E} \mid H,\ B) + Z(H \mid B).$$

Se puede separar el ejemplo e_{n+1} aplicando la suposición de independencia entre ejemplos:

$$Z(E \mid \overline{H},\ B) + Z(e_{n+1} \mid \overline{H},\ B) + Z(\overline{H} \mid B)$$

$$\leq Z(E \mid H,\ B) + Z(e_{n+1} \mid H,\ B) + Z(H \mid B).$$

Por reducción al absurdo, si fuese $Z(e_{n+1} \mid \overline{H},\ B) > 0$, entonces, como una hipótesis del Teorema es $Z(e_{n+1} \mid H,\ B) = 0$, se tiene:

$$Z(E \mid \overline{H},\ B) + Z(\overline{H} \mid B) < Z(E \mid H,\ B) + Z(H \mid B)$$

y se llega a una contradicción, ya que H no sería el modelo óptimo para E. □

El Teorema 1 afirma que si el ejemplo e_{n+1} ya está perfectamente contenido en el modelo H ($Z(e_{n+1} \mid H,\ B) = 0$), entonces también lo está en $\overline{H}$. Si se produce esta situación, aunque la fiabilidad relativa aumente, en realidad el modelo $\overline{H}$ no tiene la oportunidad de aprender nada nuevo. Por este motivo, cuando se pretende mejorar un modelo añadiendo nuevos ejemplos, conviene tomar aquellos que el modelo inicial (H) no cubra adecuadamente, es decir, ($Z(e_{n+1} \mid H,\ B) > 0$). En este principio se basan la mayoría de métodos de *boosting* [1], como por ejemplo *AdaBoost* [17].

4.1.2. Análisis local de sensibilidad

El análisis de sensibilidad es una técnica de estimación de la robustez de un modelo estadístico, basada en el estudio de las variaciones de los valores de salida respecto a los de entrada. El término *local* hace referencia al hecho de que no se analizan cambios simultáneos en los valores de varias instancias, sino únicamente sobre una instancia. Se dice que un modelo es *estable* si, ante pequeñas variaciones de los datos de entrada, las salidas también varían poco. En problemas de regresión, se considera que, si el modelo es localmente estable, la predicción individual de la instancia analizada será fiable. A continuación se formaliza esta idea y se proporcionan varias medidas que permiten aproximar la fiabilidad de una predicción.

[1] Los métodos de *boosting* son métodos de combinación de clasificadores que seleccionan un subconjunto de entrenamiento para cada clasificador. Se muestrean con mayor probabilidad los ejemplos clasificados erróneamente por los clasificadores entrenados previamente.

4.1.2.1. Perturbaciones en el conjunto de aprendizaje

Se dispone de un conjunto de instancias etiquetadas $E = \{e_i\}_{i=1}^{n}$, $e_i = (I_i, t_i)$. Además, se dispone de una nueva instancia de la que se desconoce su etiqueta ($e_{n+1} = (I_{n+1}, \_)$). El objetivo es estimar la fiabilidad de un modelo H, entrenado sobre E, en la predicción de I_{n+1}. Dicha predicción se representa como $f_H(I_{n+1}) = K$, se denomina *predicción inicial* y sirve de referencia para abordar el análisis de sensibilidad.

La *perturbación* de los datos de entrada consiste en construir un nuevo conjunto de entrenamiento $\overline{E} = E \cup \{(I_{n+1}, K+\gamma)\}$, donde se ha incluido la nueva instancia, asignándole como etiqueta una modificación de la predicción inicial K. El valor de γ es proporcional a la longitud del intervalo $[a, b]$ en el que están contenidas las etiquetas de las instancias de E, es decir, $\gamma = \epsilon(b-a)$, $\epsilon \in \mathbb{R}$. Se entrena un modelo $\overline{H}$ sobre $\overline{E}$ y se obtiene la predicción para la nueva instancia ($f_{\overline{H}}(I_{n+1}) = K_\epsilon$), denominada *predicción de sensibilidad*.

Se pueden generar tantas predicciones de sensibilidad como se desee, basta escoger diferentes valores de ϵ. Sea $0 < \epsilon_l \in \{\epsilon_1, \ldots, \epsilon_m\}$. Se entrenan los $2 \cdot m$ modelos correspondientes a generar $\overline{E}$ con las etiquetas correspondientes a los ϵ_l y $-\epsilon_l$, $l = 1, \ldots, m$. Con dichos modelos se generan las predicciones de sensibilidad $K_{\epsilon_1}, K_{-\epsilon_1}, \ldots, K_{\epsilon_m}, K_{-\epsilon_m}$.

Suponiendo, sin pérdida de generalidad, $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_m$, lo deseable es que se cumpla que:

1. $K_{-\epsilon_m} < \ldots < K_{-\epsilon_1} < K < K_{\epsilon_1} < \ldots < K_{\epsilon_m}$.
2. Los intervalos $[K_{-\epsilon_l}, K_{\epsilon_l}]$ sean lo más pequeños posible.
3. La predicción inicial K esté centrada en cada uno de dichos intervalos.

En tal caso, el modelo sería estable (y, por tanto, la predicción fiable), ya que pequeñas variaciones en las entradas producen pequeñas variaciones en las salidas y no provocan sesgos notables.

4.1.2.2. Estimaciones empíricas de fiabilidad

Para valorar en qué medida se cumplen las condiciones anteriores, se proponen tres estimaciones de fiabilidad, denominadas RE_1, RE_2, y RE_3:

- $RE_1 = \frac{\sum_{l=1}^{m} |K_{\epsilon_l} - K_{-\epsilon_l}|}{m}$.
- $RE_2 = \frac{\sum_{l=1}^{m} |K_{\epsilon_l} - K| + |K_{-\epsilon_l} - K|}{2 \cdot m}$.
- $RE_3 = \frac{\sum_{l=1}^{m} (K_{\epsilon_l} - K) + (K_{-\epsilon_l} - K)}{2 \cdot m}$.

Se desea analizar la bondad de estas estimaciones sobre diferentes modelos de regresión. Se escogen dos de ellos simples y tres complejos.

- Simples: *regresión lineal* y *regresión ponderada localmente*.
- Complejos: *árboles de regresión*, *redes neuronales artificiales* y *máquinas de vectores de soporte*.

Los simples tratan de modelar todo el espacio de entrada a la vez, mientras que los complejos hacen una preselección antes de modelar, bien dividiendo el espacio o bien escogiendo los ejemplos más significativos.

A partir de las ecuaciones de los modelos simples, se pueden determinar con exactitud las predicciones de sensibilidad, conocidas la predicción inicial y la nueva instancia. En cambio, en los modelos complejos, la inclusión de una nueva instancia puede afectar a la estructura del modelo, haciendo inviable la obtención de una fórmula explícita para las predicciones de sensibilidad. Por ello, únicamente se realizan experimentos con los modelos complejos, puesto que para los simples ya se ha resuelto analíticamente el problema.

En los experimentos realizados en [16], se estudian las correlaciones entre las tres estimaciones de fiabilidad (RE_1, RE_2, RE_3) y el error real de predicción (en el caso de RE_1 y RE_2 se toma el valor absoluto del error real). Se emplea el *Coeficiente de Correlación de Pearson* como medida de correlación. Para determinar si las correlaciones halladas son significativas se emplea el test t de *student*.

Los autores reportan que la mejor estimación es RE_3, con un 48 % de correlaciones positivas significativas, frente a un 3 % de correlaciones negativas. Además, teniendo en cuenta que RE_3 tiene signo, sugieren su utilización para crear un mecanismo corrector de errores de predicción. En este trabajo, en la sección experimental 5.2 se aborda dicha tarea.

Por otra parte, se analizan las correlaciones de otra medida conocida para estimar la fiabilidad: la densidad del espacio de instancias. Es esperable que un modelo se comporte mejor en aquellas zonas del espacio donde existan más instancias de entrenamiento, ya que se habrá ajustado mejor. Entonces, en este caso lo adecuado es que la densidad correlacione negativamente con el error real de predicción.

En los resultados mostrados en [16], se obtienen un 33 % de correlaciones negativas significativas, frente a un 8 % de correlaciones positivas para la estimación de densidad calculada mediante ventanas de Parzen [18]. Se concluye, por tanto, que la estimación de fiabilidad mediante RE_3 es mejor (correlaciona

más significativamente con el error real de predicción) que la estimación de densidad.

4.2. Explicación de la clasificación

En esta sección se presentan dos métodos generales de explicación de la clasificación de una instancia concreta. El término *generales* hace referencia a que no dependen de ningún modelo en concreto, sino que se pueden aplicar sobre cualquier modelo que sea capaz de proporcionar la probabilidad a posteriori de cada una de las clases. Estos métodos pretenden estimar cuál es la aportación de cada atributo en la decisión de asignar una determinada clase a una instancia. Incluso se pueden aplicar a modelos que tradicionalmente se consideran como una *caja negra*.

En ciertos campos de aplicación, como la medicina, el profesional que emplea modelos de aprendizaje automático demanda sistemas con capacidad de explicar sus predicciones. De este modo, puede valorar más información antes de tomar su decisión final y, quizás, adquirir nuevos conocimientos sobre el problema abordado. Además, el hecho de disponer de métodos generales permite presentar al usuario una interfaz común, independientemente del modelo que se esté aplicando. Por tanto, el sistema automático puede escoger el modelo que ofrezca mejor rendimiento en cada caso, con total transparencia para el usuario.

Hay que distinguir dos niveles de explicación:

- *Nivel de dominio*. Consiste en tratar de determinar con exactitud las verdaderas relaciones causales entre todas las variables del problema. Este nivel es prácticamente inalcanzable, salvo que se estudie un dominio artificial perfectamente conocido (*toy example*).

- *Nivel de modelo*. Trata de hacer transparente el proceso de predicción, mostrando cómo contribuye el valor de cada atributo a la predicción emitida. Conviene remarcar que la buena calidad de una explicación no implica una gran precisión del modelo y viceversa.

Dada la extrema dificultad de alcanzar una explicación satisfactoria a nivel de dominio para la inmensa mayoría de problemas, se estudia únicamente el nivel de modelo. No obstante, cuanto mejor sea el modelo y más correcta la explicación de sus predicciones, más se aproximarán dichas explicaciones al nivel de dominio.

4.2.1. Descomposición de la predicción

Robnik-Sikonja y Kononenko [19] proponen un método que trata de asignar un peso numérico al valor que toma cada atributo de una instancia, en función de la contribución de dicho valor a la predicción de una clase determinada. Teniendo en cuenta las contribuciones de todos los valores de un atributo, también se puede valorar la influencia que tiene dicho atributo en las predicciones. Para ello, lo único que se requiere es un modelo capaz de proporcionar probabilidades a posteriori de las clases.

La descomposición de la predicción para el atributo A_j consiste en analizar la variación de la predicción de la instancia I cuando se conoce el valor de todos los atributos y cuando se supone desconocido el valor de A_j ($I \backslash A_j$):

$$predDiff_j(I) = \hat{f}(I) - \hat{f}(I \backslash A_j), \qquad j = 1, \ldots, a.$$

Recordando que $\hat{f}(I) = (\mathbb{P}(C_1 \mid I), \ldots, \mathbb{P}(C_c \mid I))$ y a es el número de atributos.

Cuanto mayor sea $predDiff_j$, mayor será la influencia del atributo A_j en la predicción de I. El cálculo se realiza componente a componente de $\hat{f}(\cdot)$, por lo que, a partir de ahora se supone fijada una clase cualquiera (C_k) para la que se desea obtener la explicación. Los autores proponen tres maneras de evaluar $predDiff_j$.

- Diferencia de información:

$$infDiff_j(C_k \mid I) = \log_2 \mathbb{P}(C_k \mid I) - \log_2 \mathbb{P}(C_k \mid I \backslash A_j).$$

- Peso de la evidencia:

$$WE_j(C_k \mid I) = \log_2(odds(C_k \mid I)) - \log_2(odds(C_k \mid I \backslash A_j)),$$
$$\text{siendo } odds(z) = \frac{\mathbb{P}(z)}{\mathbb{P}(\neg z)} = \frac{\mathbb{P}(z)}{1 - \mathbb{P}(z)}.$$

- Diferencia directa de probabilidades:

$$probDiff_j(C_k \mid I) = \mathbb{P}(C_k \mid I) - \mathbb{P}(C_k \mid I \backslash A_j).$$

Las tres aproximaciones son cualitativamente similares, aunque cada una tiene sus ventajas e inconvenientes. La *diferencia de información* representa la ganancia de información respecto a la predicción de I si se utiliza o no el valor del atributo A_j. Un inconveniente es que se obtienen resultados asimétricos para probabilidades complementarias, debido al uso del logaritmo. El *peso de la evidencia* corrige esa asimetría pero, al igual que la anterior, presenta los problemas de la división por cero o tomar el logaritmo de cero, que aparecen cuando las probabilidades valen 0 o 1. Por último, la *diferencia directa de probabilidades* no tiene ninguno de los inconvenientes anteriores pero, según los autores, es difícil evaluar e interpretar correctamente probabilidades, especialmente cuando son cercanas a 0 o a 1.

Por tanto, se recomienda el uso del *peso de la evidencia (WE)* pero incorporando una pequeña corrección para evitar los casos problemáticos. La corrección empleada se conoce como *corrección de Laplace* y consiste en sustituir cualquier probabilidad p que aparezca en los cálculos, por la expresión

$$\frac{p \cdot n + 1}{n + c}, \text{ siendo } \begin{cases} n \equiv \text{nº de instancias.} \\ c \equiv \text{nº de clases.} \end{cases}$$

La información que hay que obtener del modelo para determinar la contribución del valor de un atributo A_j a la predicción de la clase C_k, consta de las dos probabilidades siguientes: $\mathbb{P}(C_k \mid I)$ y $\mathbb{P}(C_k \mid I \backslash A_j)$. La primera es proporcionada directamente por la salida del modelo, como se observó en la definición de $\hat{f}$. Para extraer la segunda es necesario realizar procesamiento adicional, que se explica a continuación.

4.2.1.1. Obtención de $\mathbb{P}(C_k \mid I \backslash A_j)$

Si se trabaja con un modelo que soporte predicción con valores desconocidos (como *Naive Bayes*), se obtiene la probabilidad buscada directamente. No obstante, esta solución no es apropiada porque no todos los modelos soportan predicción con valores desconocidos y se pretende que el método de explicación sea general, independiente del modelo.

Por tanto, hay que simular la falta de información sobre el valor de A_j, para lo que se propone la siguiente fórmula. Solamente se puede aplicar a atributos nominales, por lo que los atributos continuos deben discretizarse, por ejemplo, tomando el punto medio de cada intervalo (α_{s_j}) como valor representativo.

$$\mathbb{P}(C_k \mid I \backslash A_j) = \sum_{s_j=1}^{m_j} \mathbb{P}(A_j = \alpha_{s_j} \mid I \backslash A_j) \cdot \mathbb{P}(C_k \mid I \leftarrow A_j = \alpha_{s_j}),$$

donde $I \leftarrow A_j = \alpha_{s_j}$ corresponde a reemplazar, en la instancia I, el valor original de A_j ($\Psi(A_j, I)$) por el valor α_{s_j}.

Para calcular $\mathbb{P}(C_k \mid I \leftarrow A_j = \alpha_{s_j})$, basta invocar el modelo para obtener la predicción $\hat{f}(I \leftarrow A_j = \alpha_{s_j})$ y seleccionar la componente correspondiente a C_k. En cambio, $\mathbb{P}(A_j = \alpha_{s_j} \mid I \backslash A_j)$ es difícil de computar y se aproxima mediante la probabilidad a priori de cada valor, resultando

$$\mathbb{P}(C_k \mid I \backslash A_j) \simeq \sum_{s_j=1}^{m_j} \mathbb{P}(A_j = \alpha_{s_j}) \cdot \mathbb{P}(C_k \mid I \leftarrow A_j = \alpha_{s_j}).$$

La expresión anterior es equivalente al cálculo que hace el modelo *Naive Bayes* para predecir con valores desconocidos. Este hecho hace que la fórmula propuesta se considere razonable y se pueda interpretar como una generalización para otros modelos, aplicada a la explicación de las predicciones en lugar de a la predicción en sí misma.

4.2.1.2. Interpretación de resultados

Al calcular $predDiff_j(C_k \mid I)$ en cualquiera de sus formas ($infDiff$, WE o $probDiff$), se está computando la influencia que tiene el valor que toma el atributo A_j en la instancia I ($\Psi(A_j, I)$) para que la clase de dicha instancia sea C_k (que no tiene por qué coincidir con su clase real ni con la predicha por el modelo).

Se define la contribución del valor α_{s_j} de un atributo A_j hacia la predicción de la clase C_k como $\mathop{\mathbb{E}}_{I}\left[predDiff_j(C_k \mid I) \mid \Psi(A_j, I) = \alpha_{s_j}\right]$. El promedio de $predDiff_j(C_k \mid I)$ sobre todas las instancias cuyo valor de A_j sea α_{s_j} es un estimador de dicha esperanza.

Se define la contribución global de un atributo A_j a que el modelo prediga la clase C_k como $\mathop{\mathbb{E}}_{\alpha_{s_j}}\left[\mathop{\mathbb{E}}_{I}\left[predDiff_j(C_k \mid I) \mid \Psi(A_j, I) = \alpha_{s_j}\right]\right]$. El promedio de las contribuciones de todos los valores posibles de un atributo (discretizado si originalmente fuera continuo) es un estimador de dicha esperanza.

Conviene normalizar las puntuaciones otorgadas de manera que la suma total sea 1 (u otro valor establecido). De este modo, se pueden comparar las contribuciones provenientes de diferentes modelos. Además, si se analizan muchos atributos y valores posibles de los mismos, se puede establecer un umbral para visualizar los más significativos, ya que, comparativamente, muchos tendrán una influencia irrelevante. No existe una regla fija para determinar dicho umbral. Una posibilidad es tomar los atributos cuyas contribuciones sumen el 95 % del total.

La herramienta de explicación así construida muestra claras similitudes con la práctica médica habitual de establecer puntuaciones en los criterios de diagnóstico clínico. Dichas puntuaciones, en función de las circunstancias de cada paciente, juegan a favor o en contra de determinado pronóstico, suelen estar normalizadas y sirven para apoyar la decisión final del profesional médico.

4.2.1.3. Calidad de las explicaciones

Como se adelantó en la introducción de esta sección, es prácticamente imposible que las explicaciones alcancen el nivel de dominio de un problema. En

[19], se propone un marco experimental en el que comprobar si las explicaciones ofrecidas se ajustan bien al modelo. En tal caso, cuanto mejor capture un modelo la realidad del problema, más se acercará el nivel de modelo al nivel de dominio y, por tanto, más se parecerán las explicaciones a las reales.

Para ello se analizan los cinco modelos presentados en la sección 2.2: *Naive Bayes*, *árboles de decisión*, *vecinos más cercanos*, *máquinas de vectores de soporte* y *redes neuronales artificiales*. Se construyen cinco dominios artificiales, de modo que cada uno pueda ser aprendido mejor por cierto modelo que por los demás, aprovechando las características particulares que los diferencian. Finalmente se define una medida de distancia entre la que sería la explicación correcta y la que ofrece el modelo:

$$d_{exp}(I) = \frac{1}{C}\left(\sum_{j=1}^{a}\left(\frac{trueExpl_j(I)}{\sum_{\hat{j}=1}^{a}|trueExpl_{\hat{j}}(I)|} - \frac{predDiff_j(I)}{\sum_{\hat{j}=1}^{a}|predDiff_{\hat{j}}(I)|}\right)^2\right)^{\frac{1}{2}}$$

$trueExpl_j(I)$ representa la contribución *verdadera* del atributo A_j a la predicción de la instancia I. Solamente se puede determinar dicha contribución *verdadera* si el dominio es perfectamente conocido (*toy example*). C es una constante de normalización.

Se define $\overline{d_{exp}} = \mathop{\mathbb{E}}_{I}[d_{exp}(I)]$, cuyo estimador es el promedio de $d_{exp}(I)$ sobre todas las instancias de test. Se calcula $\overline{d_{exp}}$ en los 25 casos de estudio, consistentes en la aplicación de cada uno de los cinco modelos a los cinco dominios. También se calcula la precisión de los modelos en cada situación. Se observa que, en cada dominio, el modelo de mayor precisión es justamente para el que $\overline{d_{exp}}$ es menor. Es decir, las explicaciones ofrecidas se aproximan más a las reales cuanto mejor capture el modelo el dominio del problema.

4.2.1.4. Limitaciones

A continuación se citan algunas limitaciones del método de explicación basado en la descomposición de la predicción:

- La principal limitación del método expuesto es su incapacidad para detectar los casos en los que es necesario que se produzcan cambios en los valores de varios atributos para provocar un cambio en la predicción. Esto se debe a que, al descomponer la predicción, solamente se analiza la variación de un atributo cada vez. Una posible solución que se plantea es la exploración del espacio de subconjuntos de atributos, simulando la falta de información en todos ellos. En la siguiente sección (4.2.2) se ahonda en esta cuestión.

- El método es sensible a la discretización de atributos continuos. Si no se dispone de información a priori que permita realizar una discretización eficaz, se recomienda utilizar la discretización más fina posible (teniendo en cuenta el incremento de coste computacional que ello conlleva).

- Por último, aunque no se trata de una limitación intrínseca al método, conviene observar que si las probabilidades a posteriori ofrecidas por los modelos predictivos no están bien calibradas, la calidad de las explicaciones se verá afectada. Los modelos más proclives a presentar este problema (si no se les aplica una adecuada calibración posterior) son *Naive Bayes*, las *máquinas de vectores de soporte* y la combinación mediante la técnica de *boosting* [20].

4.2.2. Mejora: aplicación de teoría de juegos cooperativos

El método de explicación basado en la descomposición de la predicción (sección 4.2.1) tiene una limitación fundamental: no detecta las situaciones en las que el cambio del valor de predicción requiera cambios en los valores de más de un atributo. En [21], Strumbelj, Kononenko y Robnik-Sikonja presentan un algoritmo que supera dicha limitación mediante la exploración del espacio de subconjuntos de atributos (tal y como se propuso para trabajo futuro en [19]).

No obstante, surgen nuevos inconvenientes, como la complejidad exponencial debido a la explosión combinatoria del análisis de subconjuntos. Strumbelj y Kononenko encontraron la forma de reducir la complejidad computacional, apoyándose en la teoría de juegos cooperativos. En esta sección se expone el método que plantean en [22].

4.2.2.1. Nociones básicas de teoría de juegos cooperativos

A mediados del siglo XX, Shapley, entre otros, contribuyó notablemente al desarrollo de la teoría matemática sobre juegos cooperativos, especialmente con la publicación en 1953 [26]. Sin embargo, hasta principios del siglo XXI no se comenzó a aplicar al campo del aprendizaje automático (Keinan et al. [23], Cohen et al. [24]). La idea básica consiste en identificar los jugadores con atributos (o los valores que estos toman) y las coaliciones con subconjuntos de atributos. De este modo se pueden emplear los resultados conocidos de la teoría de juegos cooperativos para determinar la influencia que tienen los atributos sobre las predicciones de un modelo. A continuación se presentan los conceptos de la teoría de juegos cooperativos necesarios para el desarrollo del método de explicación.

Definición 1. Juego Cooperativo.
Un juego cooperativo es una tupla $\langle \mathbb{A}, \nu \rangle$ *donde* $\mathbb{A} = \{1, \ldots, a\}$ *es un conjunto*

de a jugadores (atributos)[II] *y* $\nu : \mathcal{P}(\mathbb{A})$ [III] $\rightarrow \mathbb{R}$ *(con* $\nu(\emptyset) = 0$*) es el valor de la recompensa asignada a cada posible coalición de jugadores (subconjunto de atributos).*

El objetivo que se persigue es repartir la recompensa total ($\nu(\mathbb{A})$, *gran coalición*) de forma *justa* entre todos los jugadores, teniendo en cuenta la influencia de cada jugador en todas las posibles sub-coaliciones en las que participe. El resultado es un vector $\phi(\nu) = (\phi_1, \ldots, \phi_a) \in \mathbb{R}^a$ donde ϕ_j es la recompensa asignada al jugador j. Dado un juego cooperativo, existen infinitas formas de asignar $\phi(\nu)$ pero solamente se consideran *justas* las que cumplan determinadas condiciones, que se recogen en los siguientes cuatro axiomas:

Axioma 1. Eficiencia.
Se reparte toda la recompensa entre los jugadores, es decir, las recompensas individuales deben asignarse de manera que su suma iguale a la recompensa de la gran coalición*:* $\sum_{j\in\mathbb{A}} \phi_j(\nu) = \nu(\mathbb{A})$.

Axioma 2. Simetría.
Si $\nu(S \cup \{j\}) = \nu(S \cup \{\hat{j}\})$, $\forall S \subset \mathbb{A}$, *tal que* $j, \hat{j} \notin S$, *entonces, la recompensa asignada a ambos jugadores debe ser la misma:* $\phi_j(\nu) = \phi_{\hat{j}}(\nu)$.

Axioma 3. Irrelevancia.
Si $\nu(S \cup \{j\}) = \nu(S)$, $\forall S \subset \mathbb{A}$, *tal que* $j \notin S$, *ese jugador no aporta valor añadido a las coaliciones en las que participa y su recompensa debe ser nula:* $\phi_j(\nu) = 0$.

Axioma 4. Aditividad.
Para dos juegos cualesquiera (ν, ω), *se debe cumplir* $\phi(\nu + \omega) = \phi(\nu) + \phi(\omega)$, *con* $(\nu + \omega)(S) = \nu(S) + \omega(S)$, $\forall S \subset \mathbb{A}$.

En [26], Shapley demuestra que solamente existe una manera de asignar $\phi(\nu)$ que cumpla los axiomas anteriores. A dicha solución se la denomina *Valor de Shapley* y es uno de los conceptos básicos de la teoría de juegos cooperativos. El siguiente teorema recoge dicho resultado.

Teorema 2. Valor de Shapley.
Dado el juego cooperativo $\langle \mathbb{A}, \nu \rangle$, *existe una única solución que satisface los cuatro axiomas presentados anteriormente:*

$$\phi_j(\nu) = Sh_j(\nu) = \sum_{S \subseteq \mathbb{A}\setminus\{j\}} \frac{(a-s-1)!\; s!}{a!} \left(\nu(S \cup \{j\}) - \nu(S)\right),$$

con $a = |\mathbb{A}|$, $s = |S|$, $j = 1, \ldots, a$.

[II] En esta sección, para remarcar la analogía entre jugadores y atributos, se representa el conjunto de jugadores por $\mathbb{A}$ y el número de ellos por a.

[III] $\mathcal{P}(\mathbb{A})$ denota el conjunto de todos los subconjuntos de $\mathbb{A}$.

4.2.2.2. Generalización de la descomposición de la predicción

En la sección 4.2.1 se simula la falta de información para un solo atributo. Sin embargo, ahora es necesario considerar cualquier subconjunto posible de atributos.

Definición 2. Diferencia en la predicción.
Sea $S \subseteq \mathbb{A}$ un subconjunto de atributos. Suponiendo conocidos los valores de los atributos de S y desconocidos los de $\mathbb{A}\backslash S$, se define la diferencia en la predicción de la clase C_k para una instancia I como

$$\Delta(S) = \frac{1}{|\mathcal{A}_{\mathbb{A}\backslash S}|} \sum_{J \in \mathcal{A}_{\mathbb{A}\backslash S}} \hat{f}_k(\lambda(I, J, S)) - \frac{1}{|\mathcal{A}_{\mathbb{A}}|} \sum_{J \in \mathcal{A}_{\mathbb{A}}} \hat{f}_k(J),$$

donde:

- $\mathcal{A}_{\mathbb{A}}$ *representa el espacio completo de atributos, es decir, el espacio determinado por todas las posibles instancias.*
- $\mathcal{A}_{\mathbb{A}\backslash S}$ *es el subespacio de* $\mathcal{A}_{\mathbb{A}}$ *correspondiente a fijar los valores de los atributos de* S *y explorar todas las combinaciones de valores de los atributos de* $\mathbb{A}\backslash S$*.*
- λ *construye una nueva instancia, combinando la original (*I*) con otra tomada de* $\mathcal{A}_{\mathbb{A}\backslash S}$ *(*J*). Los atributos de* S *toman el valor que tienen en* I*, mientras que los que no están en* S *toman el valor que tienen en* J*.*

$$\lambda(I, J, S) = (\alpha_1, \alpha_2, \ldots, \alpha_a) \qquad \alpha_j = \begin{cases} \Psi(A_j, I) & j \in S \\ \Psi(A_j, J) & j \notin S \end{cases}$$

- $\hat{f}_k(I) = \mathbb{P}(C_k \mid I)$ *representa la predicción correspondiente a la clase* C_k*, ofrecida por el modelo para la instancia* I*.*

En [21], Strumbelj y Kononenko habían propuesto una definición alternativa:

$$\Delta(S) = \tilde{f}_{k,S}(I) - \tilde{f}_{k,\emptyset}(I),$$

siendo $\tilde{f}_{k,W}(I)$ la predicción de la instancia I, para la clase C_k, resultante de entrenar al clasificador únicamente con los atributos de W. De esta manera, se evita la complejidad computacional asociada a la exploración del espacio de instancias.

No obstante, se deben reentrenar los modelos y lo deseable para el método de explicación es operar exlusivamente en la etapa de postproceso, sobre un modelo ya entrenado. Además, tampoco se consigue evitar la complejidad exponencial debida a la exploración de todos los subconjuntos de $\mathbb{A}$. Por lo tanto, esta definición alternativa no satisface las necesidades del método buscado y se desecha.

4.2.2.3. Conexión con la teoría de juegos cooperativos

La Definición 2 otorga puntuaciones numéricas a subconjuntos de atributos, por lo que se puede interpretar, en términos de la teoría de juegos cooperativos, como una función de recompensa. A continuación se propone una manera de determinar la contribución individual de cada atributo.

En primer lugar, se reescribe la diferencia en la predicción como

$$\Delta(S) = \sum_{W \subseteq S} \xi(W), \qquad S \subseteq \mathbb{A},$$

donde se está definiendo implícitamente el concepto de interacción entre atributos de un subconjunto ($\xi(\cdot)$). De esta manera, las interacciones de los atributos de los $2^{|S|}$ subconjuntos de S, contribuyen a formar $\Delta(S)$. Asumiendo que $\xi(\emptyset) = 0$, se obtiene la siguiente definición recursiva:

$$\xi(S) = \Delta(S) - \sum_{W \subset S} \xi(W).$$

En segundo lugar, se deben distribuir las contribuciones entre los a atributos. Para ello, se asume que en una interacción todos los atributos son igualmente importantes, ya que, de lo contrario, la interacción no se produciría. A cada atributo j, se le asigna la porción correspondiente del valor de todas las interacciones en las que participa, siendo su contribución total:

Definición 3. *Contribución teórica que el método de explicación otorga al atributo j.*

$$\varphi_j(\Delta) = \sum_{W \subseteq \mathbb{A} \setminus \{j\}} \frac{\xi(W \cup \{j\})}{|W \cup \{j\}|}, \qquad j = 1, 2, \ldots, a.$$

El siguiente teorema identifica el reparto propuesto con el concepto de *Valor de Shapley.*

Teorema 3. *Sea $\langle \mathbb{A}, \Delta \rangle$, un juego cooperativo y $\varphi(\Delta) = (\varphi_1, \ldots, \varphi_a)$ con φ_j como en la Definición 3. Entonces, φ corresponde al* Valor de Shapley *del juego $\langle \mathbb{A}, \Delta \rangle$, $Sh(\Delta) = \varphi(\Delta)$.*

Demostración. Ver apéndice A. □

Si el método de explicación asigna las contribuciones de cada atributo según la Definición 3, entonces, gracias a la relación establecida por el Teorema 3, tendrá las siguientes propiedades basadas en los axiomas que cumple el *Valor de Shapley*:

- Por el *Axioma 1, eficiencia*, se dispone de un mecanismo sencillo para normalizar las contribuciones, dividiendo por la recompensa otorgada a la *gran coalición*. Por tanto, se pueden comparar las explicaciones de diferentes instancias o modelos.
- Por el *Axioma 2, simetría*, a los atributos que tienen la misma influencia en la predicción se les asigna la misma contribución.
- Por el *Axioma 3, irrelevancia*, a un atributo que no influya en la predicción se le asigna una contribución de 0.

Dichas propiedades son deseables para cualquier método de explicación general y, en este caso, son consecuencia directa de la aplicación de la teoría de juegos cooperativos.

4.2.2.4. Reducción de la complejidad computacional

La aproximación basada en la Definición 2, junto con la exploración de todos los subconjuntos de atributos, hace que el cálculo de las explicaciones tenga un coste computacional excesivo. Una vez demostrada su conexión con la teoría de juegos cooperativos, se pueden aprovechar los resultados de [25] para reducir el coste computacional del método.

La formulación original del *Valor de Shapley*, basada en subconjuntos, dificulta su cálculo. Sin embargo, existe una formulación alternativa (Proposición 1) que se basa en permutaciones, las cuales son más sencillas de manejar por los algoritmos de computación.

Proposición 1. Formulación alternativa del Valor de Shapley.
La formulación del Valor de Shapley *que se establece en el Teorema 2 es equivalente a la siguiente:*

$$Sh_j(\Delta) = \frac{1}{a!} \sum_{\pi \in \Pi(\mathbb{A})} \left(\Delta(Pre^j(\pi) \cup \{j\}) - \Delta(Pre^j(\pi)) \right), \qquad j = 1, \ldots, a.$$

Siendo

- $\Pi(\mathbb{A})$ *el conjunto de las permutaciones del conjunto* $\mathbb{A} = \{1, \ldots, a\}$.
- $Pre^j(\pi)$ *el conjunto de elementos (atributos) que preceden a* j *en la permutación* π.

Demostración. Fijado un subconjunto $S \subseteq \mathbb{A} \backslash \{j\}$, se define Π_S como el subconjunto de permutaciones π de $\Pi(\mathbb{A})$ para las que se cumple $S = Pre^j(\pi)$. Tomando los Π_S de todos los posibles $S \subseteq \mathbb{A} \backslash \{j\}$, se tiene una partición disjunta de $\Pi(\mathbb{A})$, ya que cualquier permutación $\pi \in \Pi(\mathbb{A})$ pertenece a algún Π_S y una misma permutación no puede pertenecer a Π_{S_1} y Π_{S_2}, con $S_1 \neq S_2$.

Por otro lado, siendo $s = |S|$ y $a = |\mathbb{A}|$, el número de permutaciones de Π_S es $s!\,(a-s-1)!$, ya que las permutaciones son de la forma $(\mu_1, \ldots, \mu_s, j, \eta_1, \ldots, \eta_{a-s-1})$, con $\mu_l \in S$, $l = 1, \ldots, s$ y $\eta_{\hat{l}} \in \mathbb{A}\backslash\{S \cup \{j\}\}$, $\hat{l} = 1, \ldots, a-s-1$.

Aplicando los dos pasos anteriores se llega desde la formulación alternativa a la original, demostrando su equivalencia.

$$Sh_j(\Delta) = \frac{1}{a!} \sum_{\pi \in \Pi(\mathbb{A})} \left(\Delta(Pre^j(\pi) \cup \{j\}) - \Delta(Pre^j(\pi))\right) =$$

$$= \frac{1}{a!} \sum_{S \subseteq \mathbb{A}\backslash\{j\}} \sum_{\pi \in \Pi_S} \left(\Delta(S \cup \{j\}) - \Delta(S)\right) =$$

$$= \sum_{S \subseteq \mathbb{A}\backslash\{j\}} \frac{(a-s-1)!\; s!}{a!} \left(\Delta(S \cup \{j\}) - \Delta(S)\right).$$

□

Antes de introducir $\Delta(S)$ en la formulación alternativa del *Valor de Shapley*, se reescribe de forma ligeramente diferente a la presentada en la Definición 2. Teniendo en cuenta que $|\mathcal{A}_{\mathbb{A}}| = |\mathcal{A}_{\mathbb{A}\backslash S}| \cdot |\mathcal{A}_S|$, se puede escribir

$$\Delta(S) = \frac{1}{|\mathcal{A}_{\mathbb{A}}|} \sum_{J \in \mathcal{A}_{\mathbb{A}}} \left(\hat{f}_k(\lambda(I, J, S)) - \hat{f}_k(J)\right),$$

donde $\hat{f}_k(\lambda(I, J, S))$ aparece $|\mathcal{A}_S|$ veces.

Incorporando $\Delta(S)$ en la formulación alternativa del *Valor de Shapley*, se obtiene la contribución teórica que el método de explicación otorga a cada atributo.

$$\begin{aligned}
\varphi_j(\Delta) &= \frac{1}{a!} \sum_{\pi \in \Pi(\mathbb{A})} \left[\frac{1}{|\mathcal{A}_{\mathbb{A}}|} \sum_{J \in \mathcal{A}_{\mathbb{A}}} \left(\hat{f}_k(\lambda(I, J, Pre^j(\pi) \cup \{j\})) - \hat{f}_k(J)\right) \right. \\
&\quad \left. - \frac{1}{|\mathcal{A}_{\mathbb{A}}|} \sum_{J \in \mathcal{A}_{\mathbb{A}}} \left(\hat{f}_k(\lambda(I, J, Pre^j(\pi))) - \hat{f}_k(J)\right) \right] \\
&= \frac{1}{a!\;|\mathcal{A}_{\mathbb{A}}|} \sum_{\pi \in \Pi(\mathbb{A})} \sum_{J \in \mathcal{A}_{\mathbb{A}}} \left[\hat{f}_k(\lambda(I, J, Pre^j(\pi) \cup \{j\})) \right. \\
&\quad \left. - \hat{f}_k(\lambda(I, J, Pre^j(\pi))) \right].
\end{aligned}$$

Se define la variable aleatoria $X : \Pi(\mathbb{A}) \times \mathcal{A}^{\text{IV}} \to \mathbb{R}$ como

$$X_{\pi \in \Pi(\mathbb{A}),\, J \in \mathcal{A}} = \hat{f}_k(\lambda(I, J, Pre^j(\pi) \cup \{j\})) - \hat{f}_k(\lambda(I, J, Pre^j(\pi))).$$

Se observa que $\varphi_j = \mathbb{E}\left[X_{\pi,\, J}\right]$ y se define $\hat{\varphi}_j = \frac{1}{m}\sum_{l=1}^{m} X_l$, donde $\{X_l\}_{l=1}^{m}$ es un conjunto de variables aleatorias independientes e idénticamente distribuidas (v.a. iid) con distribución igual a la de X y m es el cardinal de dicho conjunto. Entonces, por el Teorema del Límite Central, $\hat{\varphi}_j$ tiende a una distribución normal de media φ_j y varianza $\frac{\sigma_j^2}{m}$. Por tanto, $\hat{\varphi}_j$ es un estimador de φ_j no sesgado ($\mathbb{E}(\hat{\varphi}_j) = \varphi_j$) y consistente ($\frac{\sigma_j^2}{m} \to 0$).

Basándose en ello, se propone el Algoritmo 4 para estimar las contribuciones de cada atributo.

Algoritmo 4 Estimación mediante teoría de juegos cooperativos de la contribución de un atributo

Entrada: Modelo entrenado para la clase C_k ($\hat{f}_k$), instancia I.
Salida: Valor estimado de la contribución del atributo A_j para clasificar I como perteneciente a la clase C_k.
Procedimiento:

1: Fijar m.
2: $\varphi_j \leftarrow 0$.
3: **for** $l = 1 \to m$ **do**
4: Seleccionar aleatoriamente una permutación $\pi \in \Pi(\mathbb{A})$.
5: Seleccionar aleatoriamente una instancia $J \in \mathcal{A}$.
6: $\nu_1 \leftarrow \hat{f}_k(\lambda(I, J, Pre^j(\pi) \cup \{j\}))$.
7: $\nu_2 \leftarrow \hat{f}_k(\lambda(I, J, Pre^j(\pi)))$.
8: $\varphi_j \leftarrow \varphi_j + (\nu_1 - \nu_2)$.
9: **end for**
10: $\varphi_j \leftarrow \frac{\varphi_j}{m}$.
11: **return** φ_j

El valor de m se fija en función del error máximo que se desee tolerar (e). Al tratarse de un muestreo aleatorio, no se puede garantizar que el 100 % de las contribuciones $\hat{\varphi}_j$ calculadas mediante el Algoritmo 4 tengan error menor que e. Sin embargo, en virtud del Teorema del Límite Central se puede escoger una probabilidad $1 - \alpha$ con la que sí se cumpla, como se muestra a continuación:

$$\mathbb{P}\left(|\hat{\varphi}_j - \varphi_j| < e\right) = 1 - \alpha \Rightarrow \mathbb{P}\left(\left|\frac{1}{m}\left(\sum_{l=1}^{m} X_l\right) - \varphi_j\right| < e\right) = 1 - \alpha$$

[IV] $\mathcal{A}_{\mathbb{A}} \equiv \mathcal{A}$.

$$\Rightarrow \mathbb{P}\left(\frac{|(\sum_{l=1}^{m} X_l) - m \cdot \varphi_j|}{\sigma_j\sqrt{m}} < \frac{m \cdot e}{\sigma_j\sqrt{m}}\right) = 1 - \alpha.$$

Tomando z de forma que $\mathbb{P}(\mathcal{N}(0,1) \leq z) = 1 - \frac{\alpha}{2}$ y sustituyendo sobre la expresión anterior:

$$z = \frac{m \cdot e}{\sigma_j\sqrt{m}} \Rightarrow m = \frac{z^2 \cdot \sigma_j^2}{e^2}.$$

σ_j no se conoce a priori pero se puede observar que la cota superior es $\sigma_j = 1$, en el caso de que X estuviese uniformemente distribuida entre sus dos extremos (-1 y 1). Por ejemplo, para que el 99 % de las contribuciones tengan un error menor que 0,01, z es 2,57 y $m = \frac{2{,}57^2}{0{,}01^2} \simeq 65000$ iteraciones.

Se ha implementado este método y en la sección 5.3 se presentan algunos de los resultados que permite obtener, poniendo de manifiesto cómo refleja el conocimiento que ha adquirido un modelo, contribuyendo a aumentar la transparencia del mismo y facilitando la interpretación de sus predicciones.

Capítulo 5

Experimentos

En este capítulo se expone el trabajo experimental llevado a cabo para complementar el estudio teórico desarrollado en los capítulos anteriores. Se divide en tres secciones: Selección de atributos (sección 5.1), Estimaciones de fiabilidad en regresión (sección 5.2) y Explicaciones en clasificación (sección 5.3). Todas ellas siguen una estructura similar. En primer lugar se explica el objetivo del experimento y cómo se lleva a cabo. En segundo lugar se describen los conjuntos de datos utilizados. A continuación se muestran los resultados obtenidos en forma de gráficas o tablas y, por último, se realiza una valoración de dichos resultados.

En cada sección se proporciona información específica sobre la implementación pero hay algunos elementos comunes a todos los experimentos que se citan a continuación. El sistema operativo empleado es Linux Ubuntu 11.04. Todo el código se lanza mediante scripts de Bash [32], que en algunas ocasiones actúa como un mero intermediario para invocar a otros programas pero en otras también ejecuta parte de los algoritmos y técnicas desarrollados. Entre otras herramientas ofrecidas por GNU [31] que se emplean, destaca Gawk [33]. Para generar las gráficas que se muestran en las secciones de resultados, se emplea R [30].

5.1. Selección de atributos

En [10] se reportan varios experimentos cuya finalidad es analizar el comportamiento de los algoritmos de la familia *Relief* en los términos de *usabilidad* y *separabilidad* definidos en la sección 3.2.3.2. Además, se realiza una comparación con un método de selección de atributos basado en la ganancia de información. En este trabajo se pretende extender la comparativa a los algoritmos mRMR y QPFS.

La implementación utilizada de los algoritmos mRMR [27] y QPFS [15] es la que proporcionan los respectivos autores. Los algoritmos de la familia *Relief* se ejecutan a través de la herramienta WEKA [28]. Para cada uno de los conjuntos de datos descritos en la sección 5.1.1 se ejecutan los tres algoritmos con los siguientes parámetros:

- ReliefF. 10 vecinos más cercanos.
- RReliefF. 70 vecinos más cercanos, ponderación decreciente exponencialmente según distancia con $\sigma = 2$.
- mRMR. Umbrales de discretización situados en ± 1 desviaciones típicas respecto a la media de cada atributo, cálculo de los pesos por *MID (Mutual Information Difference)*.
- QPFS. Umbral de discretización de ± 1 desviación típica, Información Mutua como medida de similitud.

Adicionalmente, los algoritmos *Relief* se pueden ejecutar tanto para atributos nominales como numéricos, independientemente de si la variable objetivo se interpreta como correspondiente a un problema de clasificación o a uno de regresión.

5.1.1. Conjuntos de datos

El cálculo apropiado de la *usabilidad* y la *separabilidad* requiere conocer con certeza qué atributos contribuyen a la formación de la variable objetivo. No basta con añadir atributos aleatorios a conjuntos de datos tomados del mundo real, ya que se desconoce si todos los atributos existentes son verdaderamente importantes. Por tanto, el análisis debe abordarse generando artificialmente los conjuntos de datos para poder determinar con seguridad cuáles son los atributos importantes y cuáles los aleatorios.

En la Tabla 5.1 se describen los conjuntos de datos utilizados en esta sección. La letra I en la nomenclatura de los conjuntos indica el número de atributos importantes del mismo y la letra p el módulo. En el problema *modulo-p-I*, los atributos pueden tomar cualquier valor entero del intervalo $[0, p)$, en los problemas *linEq-I* y *linInc-I* toman valores numéricos en el intervalo $[0, 1]$ y en los problemas *monk* depende de cada atributo, por lo que se citan a continuación:

- Atributos A_1, A_2 y A_4. Posibles valores 1, 2 o 3.
- Atributos A_3 y A_6. Posibles valores 1 o 2.
- Atributos A_5. Posibles valores 1, 2, 3 o 4.

Además, en *monk-M3* se introduce ruido en la variable objetivo, generando el 5 % de las instancias mal clasificadas.

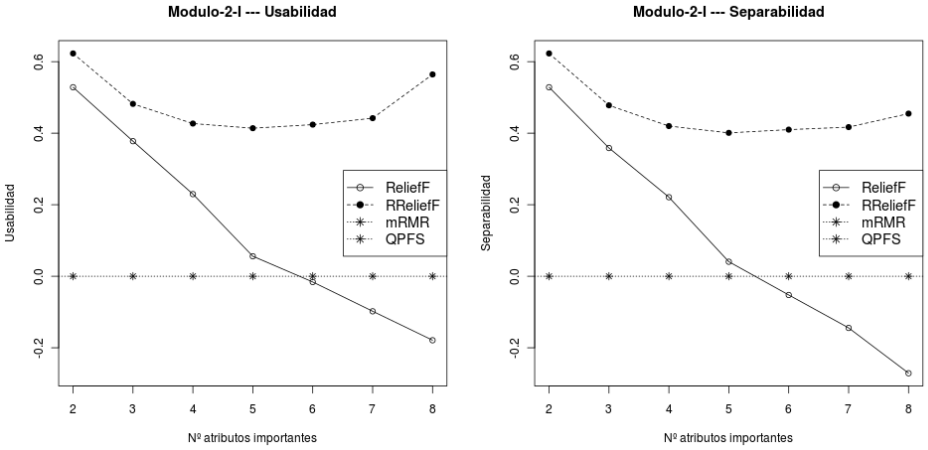
Tabla 5.1: Conjuntos de datos empleados en los experimentos de selección de variables. La columna etiquetada con n indica el número de instancias, con a el número de atributos, la columna siguiente indica el tipo de los mismos y la última el modo de calcular la variable objetivo. Los valores de los atributos se generan aleatoriamente, salvo en el problema *modulo-p-I* que se cubre exhaustivamente el dominio si $p = 2$. De los a atributos, aquellos que no participen en el cálculo de la variable objetivo son los considerados como aleatorios.

Nombre	**n**	**a**	**Tipo**	**Variable objetivo**
modulo-p-I	512	9	nominal	$\left(\sum_{j=1}^{I} A_j\right) \bmod p$
monk-M1	124	6	nominal	$(A_1 = A_2)$ o $(A_5 = 1)$
monk-M2	169	6	nominal	1 si exactamente dos atributos toman el valor 1. 0 en caso contrario.
monk-M3	122	6	nominal	$((A_5 = 3)$ y $(A_4 = 1))$ o $((A_5 \neq 4)$ y $(A_2 \neq 3))$
linEq-I	1000	$I + 10$	numérico	$\left(\sum_{j=1}^{I} j \cdot A_j\right)$
linInc-I	1000	$I + 10$	numérico	$\left(\sum_{j=1}^{I} A_j\right)$

5.1.2. Resultados

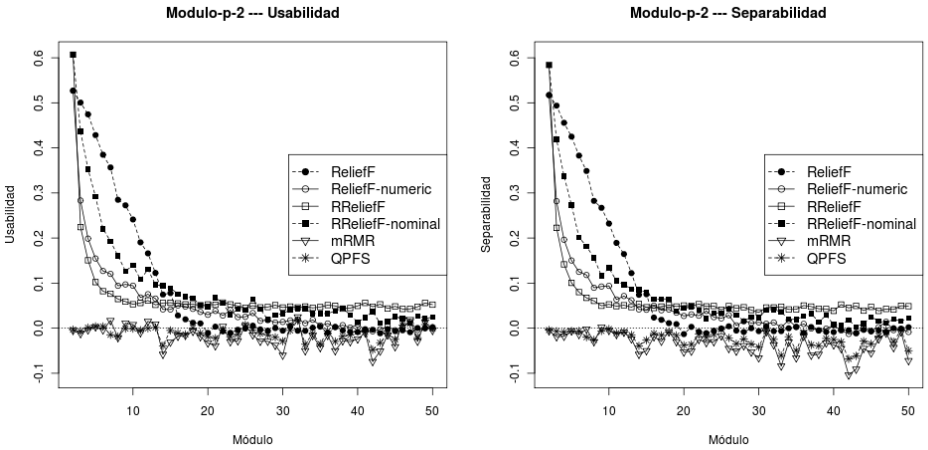
A continuación se muestra una serie de gráficas y tablas que recogen los resultados obtenidos. En el caso de los problemas *linEq-I* y *linInc-I*, se representan las gráficas de usabilidad y separabilidad para varios valores de I (número de atributos importantes). Para el problema *modulo-p-I*, se revisa desde dos puntos de vista: fijar $p = 2$, variando el valor de I y fijar $I = 2$, variando el valor de p (módulo). Para los problemas *monk* no tiene sentido una representación gráfica, por lo que los pesos asignados a cada atributo se presentan en forma de tabla y se explicitan los valores de separabilidad y usabilidad obtenidos por cada algoritmo. Dichos pesos se calculan promediado 10 ejecuciones sobre sendos muestreos aleatorios de instancias.

Al pie de cada figura o cabecera de tabla solamente se proporciona información descriptiva de la misma. La interpretación y valoración de resultados se realiza en la sección 5.1.3. En las otras dos secciones dedicadas a experimentos se sigue el mismo esquema.



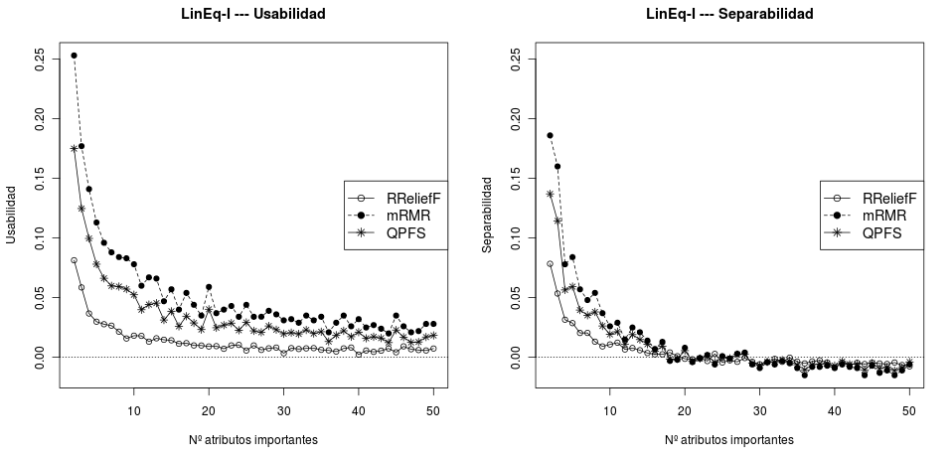
(a) Usabilidad **(b)** Separabilidad

Figura 5.1: Gráficas de usabilidad y separabilidad para el problema modulo-2-I. Las líneas de mRMR y QPFS se representan de manera idéntica ya que son ambas constantes iguales a 0.



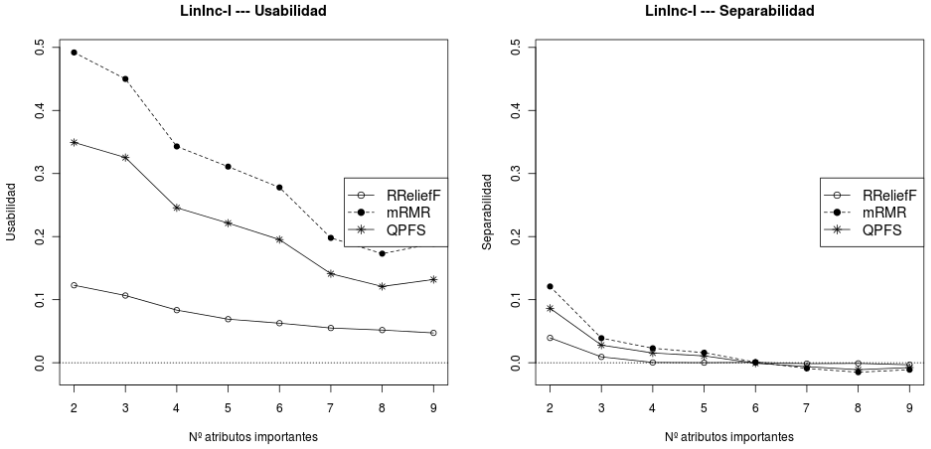
(a) Usabilidad **(b)** Separabilidad

Figura 5.2: Gráficas de usabilidad y separabilidad para el problema modulo-p-2. Tanto la usabilidad como la separabilidad de RReliefF y RReliefF-nominal (atributos interpretados como nominales) son siempre positivas, aunque el comportamiento asintótico del primero es más estable. La separabilidad de mRMR y QPFS siempre es negativa. La usabilidad y la separabilidad de ReliefF y ReliefF-numeric son positivas hasta 20 y 41 atributos importantes, respectivamente, y a partir de ese punto oscilan en torno a 0.



(a) Usabilidad **(b)** Separabilidad

Figura 5.3: Gráficas de usabilidad y separabilidad para el problema linEq-I. La usabilidad es siempre positiva para los tres algoritmos. La separabilidad se hace negativa por primera vez con 18 atributos importantes para mRMR y QPFS y con 20 para RReliefF. Tras algunas oscilaciones, a partir de 29 atributos importantes siempre es negativa para los tres algoritmos.



(a) Usabilidad **(b)** Separabilidad

Figura 5.4: Gráficas de usabilidad y separabilidad para el problema linInc-I. La usabilidad siempre es positiva para los tres algoritmos. La separabilidad con 6 atributos importantes es muy próxima a 0 (ligeramente negativa para QPFS) y, a partir de 7, siempre es negativa para los tres algoritmos.

Tabla 5.2: Resultados de selección de atributos para los problemas *monk*. En la primera columna de cada subtabla se indica el nombre del atributo y, entre paréntesis, una I si es importante o una R (de *random*) si no influye en la variable objetivo. En las otras tres columnas se muestran las puntuaciones otorgadas por los algoritmos ReliefF, mRMR y QPFS, respectivamente, promediando las ejecuciones realizadas sobre 10 muestreos aleatorios de instancias. En las últimas dos filas de las subtablas de monk-M1 y monk-M3 se calculan los valores de la usabilidad y separabilidad. Nótese que para el problema monk-M2 no se pueden calcular los valores de usabilidad y separabilidad, ya que todos los atributos son importantes. Los atributos más contribuyentes de cada método se resaltan en **negrita**.

monk-M1			
Atributo	**ReliefF**	**mRMR**	**QPFS**
$A_1(I)$	0,180	−0,008	0,106
$A_2(I)$	0,169	−0,010	0,107
$A_3(R)$	−0,027	0,003	0,182
$A_4(R)$	−0,068	−0,009	0,108
$A_5(I)$	**0.223**	**0.310**	**0.321**
$A_6(R)$	−0,046	−0,002	0,175
Usabilidad	0,249	0,308	0,139
Separabilidad	0,196	−0,012	−0,076

monk-M2			
Atributo	**ReliefF**	**mRMR**	**QPFS**
$A_1(I)$	0,040	0,004	0,142
$A_2(I)$	**0.065**	**0.012**	0,142
$A_3(I)$	0,047	0,003	**0.221**
$A_4(I)$	0,048	0,003	0,138
$A_5(I)$	0,037	−0,005	0,142
$A_6(I)$	0,058	−0,003	**0.216**

monk-M3			
Atributo	**ReliefF**	**mRMR**	**QPFS**
$A_1(R)$	−0,010	−0,004	0,090
$A_2(I)$	**0.297**	**0.248**	**0.267**
$A_3(R)$	−0,021	−0,007	0,143
$A_4(I)$	−0,004	−0,005	0,097
$A_5(I)$	**0.293**	**0.248**	**0.262**
$A_6(R)$	−0,030	−0,008	0,141
Usabilidad	0,307	0,253	0,124
Separabilidad	0,006	−0,001	−0,046

5.1.3. Discusión

En la Figura 5.1, correspondiente al problema modulo-2-I, se observa que tanto la usabilidad como la separabilidad de los algoritmos mRMR y QPFS es constante e igual a 0. Esto se debe al muestreo exhaustivo del espacio de atributos, que hace que la información mutua de cualquier atributo con la variable objetivo o de cualquier par de atributos entre sí sea 0. Por lo tanto, este experimento pone de manifiesto una diferencia fundamental entre ReliefF, que realiza un análisis local del espacio de atributos, y mRMR y QPFS, que se basan en una medida global (la información mutua) que involucra a todas las instancias. En este caso, dicha diferencia supone una ventaja evidente para ReliefF, ya que le permite valorar los atributos, mientras que mRMR y QPFS son incapaces de hacerlo. Es importante recordar que si se toman excesivos vecinos cercanos en ReliefF, el algoritmo se vuelve *miope*, perdiendo la visión local del dominio y quedándose únicamente con la global.

En este problema, interpretar los atributos como numéricos o nominales no supone ninguna diferencia, ya que existen únicamente dos valores posibles. Por lo tanto, solamente se representa una línea para ReliefF (interpretando el problema como clasificación) y otra para RReliefF (interpretándolo como regresión). Se observa que RReliefF tolera perfectamente el aumento de atributos importantes, mientras que ReliefF, a partir de 6, no logra usabilidad ni separabilidad positivas.

En la Figura 5.2, correspondiente al problema modulo-p-2, los algoritmos ReliefF y RReliefF se han ejecutado tanto con atributos nominales como numéricos [I]. El mejor comportamiento lo tienen RReliefF y RReliefF-nominal, que logran siempre usabilidad y separabilidad positivas. No obstante, el primero es asintóticamente más estable. ReliefF y ReliefF-numeric logran resultados positivos hasta $p = 20$ y $p = 41$, respectivamente, y para valores mayores, la usabilidad y la separabilidad oscilan en torno a 0. Finalmente, mRMR y QPFS son incapaces de seleccionar los atributos importantes. Obtienen siempre separabilidad negativa y casi siempre, excepto para algunos valores puntuales de p, usabilidad negativa.

Se puede concluir que mRMR y QPFS no son capaces de abordar con efectividad problemas de tipo *modulo*, a diferencia de los algoritmos de la familia *Relief*. Dentro de estos últimos, se obtienen mejores resultados si se interpreta el dominio como un problema de regresión y se consideran los atributos como numéricos.

[I] Los valores de los atributos en realidad no cambian pero sí la forma interna de interpretarlos por el algoritmo de selección.

Los dos siguientes problemas (linEq-I y linInc-I) son de regresión y los algoritmos mRMR y QPFS solamente se pueden aplicar a problemas de clasificación. Por tanto, la variable objetivo se ha discretizado, asignando cada valor al entero inmediatamente menor [II]. El algoritmo RReliefF se ejecuta sin realizar dicha discretización.

En las Figuras 5.3 y 5.4 se muestran los resultados de los problemas linEq-I y linInc-I, respectivamente. En ambos, el comportamiento de los tres algoritmos es cualitativamente similar, en el sentido de que no es tan relevante el valor concreto de la usabilidad y separabilidad, sino en qué punto cruzan la línea 0. La usabilidad de los tres algoritmos es siempre positiva y presentan una tendencia decreciente asintóticamente. La separabilidad de todos ellos pasa a ser negativa para, aproximadamente, el mismo número de atributos importantes (I). Dicho número es considerablemente mayor en el problema linEq-I ($\sim 18, 20$) que en linInc-I ($\sim 6, 7$), debido a que la ponderación realizada en este último provoca que los atributos con menor peso se confundan antes con los atributos aleatorios.

En la Tabla 5.2 se recogen los resultados de los problemas *monk*. En monk-M1, los tres algoritmos detectan sin problemas la importancia del atributo A_5, logrando puntuaciones positivas para la usabilidad. No obstante, no ocurre lo mismo para A_1 y A_2, que influyen en la variable objetivo según la expresión lógica $A_1 = A_2$. ReliefF es capaz de detectar esta fuerte dependencia condicional y su separabilidad es positiva, mientras que mRMR y QPFS no, obteniendo una puntuación negativa para la separabilidad.

En el problema monk-M2, la variable objetivo es 1 cuando exactamente dos atributos toman el valor 1 y 0 en cualquier otro caso. Dado que los valores de los atributos se generan aleatoriamente con una distribución uniforme, es razonable esperar que los atributos con menos valores posibles son más influyentes porque toman el valor 1 más veces. Según esta suposición, los atributos más importantes son A_3 y A_6 (con solo dos valores posibles), seguidos de A_1, A_2 y A_4 (tres valores) y, finalmente A_5 (cuatro valores). QPFS otorga la mayor puntuación correctamente a los atributos A_3 y A_6 pero no es capaz de discernir más allá. Por contra, mRMR y ReliefF detectan el atributo menos influyente (A_5) pero otorgan la máxima puntuación al atributo A_2, sin que exista un motivo aparente para ello. Este es el único experimento, de todos los realizados en esta sección, en el que el comportamiento de mRMR es cualitativamente diferente al de QPFS, aproximándose más al de ReliefF.

[II] Este proceso se realiza automáticamente al ejecutar cada uno de los dos algoritmos, dado que en la lectura de datos se fuerza a la variable objetivo a ser discreta y se toma el suelo (entero inmediatamente menor) de los valores numéricos.

Por último, en el problema monk-M3, todos los algoritmos tienen una usabilidad positiva, ya que puntúan acertadamente los atributos A_2 y A_5. Por otro lado, se puede deducir de la expresión lógica que forma la variable objetivo, que el atributo A_4 tiene una influencia muy pequeña. No obstante, ReliefF es capaz de distinguirlo de los otros tres atributos aleatorios, mientras que mRMR y QPFS no, por lo que estos últimos presentan una separabilidad negativa.

5.2. Estimaciones de fiabilidad en regresión

En la sección 4.1.2.2 se presentan tres medidas para estimar la fiabilidad de las predicciones en un problema de regresión, tomadas de [16]. En dicha publicación se analiza la correlación de las mencionadas medidas con el error real de predicción y se sugiere la posibilidad de crear un mecanismo corrector de errores basado en ellas. En este trabajo se proponen algunas aproximaciones a dicho objetivo y se aplican a cuatro conjuntos de datos para valorar los resultados.

En primer lugar se implementa el sistema para generar las predicciones iniciales y las predicciones de sensibilidad, tal y como se expone en la sección 4.1.2.1. Se escogen dos modelos diferentes: árboles de regresión y máquinas de vectores de soporte, en concreto se toman REPTree de WEKA [28] y LIBSVM [29], respectivamente. Se realiza validación cruzada de 10 hojas para obtener las predicciones de todas las instancias. Finalmente, mediante Gawk [33], se computan las medidas $RE1$, $RE2$ y $RE3$ expuestas en la sección 4.1.2.2.

Para explicar los métodos de corrección propuestos en este trabajo, se comienza definiendo el error real de la predicción de la instancia I_i como $err_i = \mathcal{T}(I_i) - f(I_i)$, donde $\mathcal{T}(I_i)$ es el valor real de la variable objetivo y $f(I_i)$ la predicción realizada inicialmente por el modelo. El objetivo es encontrar un valor $\theta(RE1, RE2, RE3)$ que se aproxime al error err_i. De este modo, se puede generar una predicción corregida añadiendo θ a la predicción inicial ($f^*(I_i) = f(I_i) + \theta$).

$RE3$ tiene signo y en [16], se muestra que suele correlacionar positivamente con el error de predicción, por lo que se toma como un factor base de θ. Por otro lado, se sugiere que cuanto menores sean $RE1$ y $RE2$, menor es el valor absoluto del error de predicción. Teniendo en cuenta estas ideas, se proponen tres fórmulas sencillas:

- $f^*(I_i) = f(I_i) + RE3(I_i, f)$.
- $f^*(I_i) = f(I_i) + RE3(I_i, f) \cdot RE1(I_i, f)$.
- $f^*(I_i) = f(I_i) + RE3(I_i, f) \cdot RE2(I_i, f)$.

El grado de correlación de las medidas de fiabilidad con los errores de predicción varía en función del conjunto de datos utilizado. Por ello, se plantea una alternativa adicional, más dinámica que las fórmulas anteriores. La idea es entrenar otro modelo ($\hat{f}$) para tratar de cuantificar el error de la predicción. Dado un conjunto de instancias de test (T), se divide el conjunto de entrenamiento disponible (E) en dos subconjuntos (E_1 y E_2). Empleando el conjunto E_1 se generan las predicciones iniciales, de sensibilidad y las medidas de fiabilidad para las instancias de E_2 y T. A continuación, se entrena $\hat{f}$ empleando como atributos los valores de $RE1$, $RE2$ y $RE3$ de las instancias de E_2 y como variable objetivo el error real de predicción. Finalmente, se aplica dicho modelo sobre el conjunto de test y se escoge θ para cada instancia igual al valor de salida del modelo:

- $f^*(I_i) = f(I_i) + \hat{f}(RE1(I_i, f), RE2(I_i, f), RE3(I_i, f))$.

Para aplicar este sistema con validación cruzada, se debe hacer la división del conjunto de entrenamiento de cada hoja (E) en los E_1 y E_2 descritos anteriormente. Este es, a priori, un punto débil del sistema, especialmente para conjuntos de datos con pocas instancias, ya que es necesario fraccionarlo más aún. En los experimentos realizados se han incluido en E_1 y E_2 el 60 % y el 40 % de las instancias de E, respectivamente. No obstante, el valor óptimo de estos porcentajes puede ser objeto de un futuro estudio.

5.2.1. Conjuntos de datos

En la Tabla 5.3 se citan los conjuntos de datos empleados, todos ellos problemas de regresión. *Breast cancer Wisconsin*, *Concrete* y *Wine quality* se han descargado del repositorio UCI [34]. El primero tiene como variable objetivo el número de meses que tarda en recurrir un cáncer de mama, el segundo, la fuerza compresiva que soporta un bloque de hormigón y el tercero, la puntuación de calidad de un vino otorgada por un experto.

El último conjunto de datos (denominado DEMCAM), está compuesto por numerosos atributos numéricos obtenidos tras el postprocesamiento de imágenes del cerebro capturadas mediante resonancia magnética. Las imágenes fueron tomadas en las instalaciones de la Fundación CIEN (Centro de Investigaciones Neurológicas) [35] para un estudio de la Enfermedad de Alzheimer, cuyos investigadores principales son los doctores Ana Frank y Juan Álvarez-Linera. Ambos pertenecen al grupo de investigación DEMCAM (DEMencias de la Comunidad Autónoma de Madrid), del que se ha tomado el nombre para citar el conjunto de datos en este trabajo. Tanto DEMCAM como la Fundación CIEN reciben el apoyo de la Fundación Reina Sofía [36]. El director del equipo encargado del tratamiento de las imágenes es el doctor Juan Hernández

Tamames. Dicho tratamiento incluye diversas técnicas: Volumetría, Espectroscopía, Tensor de Difusión y Perfusión mediante Etiquetado de Spines (*Arterial Spin Labelling*). El Instituto de Ingeniería del Conocimiento (IIC) [37] se incorporó al proyecto y tiene acceso a los datos, totalmente anonimizados, en cumplimiento de la L.O.P.D. (Ley Orgánica de Protección de Datos). A través de mi vinculación laboral al IIC, he solicitado y recibido los permisos oportunos para utilizar estos datos en el presente trabajo.

La variable objetivo es el estado cognitivo de una persona entre 4 posibles, que van desde sano hasta enfermo de Alzheimer, pasando por dos estados intermedios de deterioro cognitivo de diferente consideración. Desde el punto de vista diagnóstico, se debe interpretar como un problema de clasificación. No obstante, la Enfermedad de Alzheimer es un proceso progresivo, por lo que desde el punto de vista de la investigación, también tiene sentido tratarlo como un problema de regresión. En este trabajo se consideran ambas opciones, según las técnicas aplicadas. En esta sección se interpreta como una regresión y en la sección 5.3 como un problema de clasificación.

Tabla 5.3: Conjuntos de datos sobre los que se aplican mecanismos correctores de errores para problemas de regresión. n indica el número de instancias, a el número de atributos y la última columna indica el tipo de los mismos.

Nombre	n	a	Tipo
Breast cancer Wisconsin	47	32	numérico
Concrete	1030	8	numérico
Wine quality	1599	11	numérico
DEMCAM	78	238	numérico

5.2.2. Resultados

Para evaluar la calidad de las predicciones y así poder comparar las iniciales con las corregidas, se emplean cuatro medidas de error calculadas a partir de los errores individuales (err_i) definidos al comienzo de la sección.

- Error absoluto medio: $ErrAbs = \frac{\sum_i |err_i|}{n}$.
- Error absoluto relativo: $ErrAbsRel = \frac{100 \cdot ErrAbs}{\left(max\{\mathcal{T}(I_i)\}_{i=1}^{n} - min\{\mathcal{T}(I_i)\}_{i=1}^{n}\right)}$.
- Raíz del error cuadrático medio: $RMSE = \sqrt{\frac{\sum_i err_i^2}{n}}$.

- *RMSE* relativo: $RMSErel = \frac{100 \cdot RMSE}{\left(max\{\mathcal{T}(I_i)\}_{i=1}^{n} - min\{\mathcal{T}(I_i)\}_{i=1}^{n}\right)}$.

Para cada uno de los conjuntos de datos, se calculan las cuatro medidas de error tanto para las predicciones iniciales como para todas las resultantes de corregir las iniciales según las diferentes fórmulas expuestas al comienzo de esta sección. Se calcula también el Coeficiente de Correlación de Pearson de cada una de las tres medidas de fiabilidad ($RE1$, $RE2$ y $RE3$) con el error real de predicción (en el caso de $RE3$) o su valor absoluto (en los casos de $RE1$ y $RE2$).

Los modelos empleados para el sistema automático de corrección de las predicciones son los mismos que para las predicciones iniciales (árboles REPTree [28] y máquinas de vectores de soporte LIBSVM [29]). En la Tabla 5.5 se recogen los resultados correspondientes a la generación de las predicciones iniciales mediante árboles de regresión y en la Tabla 5.6 los correspondientes a las máquinas de vectores de soporte. Dentro de cada una de ellas existen dos filas (*Árbol* y *SVM*) que indican que el método de corrección de predicciones es automático y está basado en el entrenamiento del modelo homónimo. En la Tabla 5.4 se muestran las correlaciones de las medidas de fiabilidad con el error de predicción.

Tabla 5.4: Coeficientes de Correlación de Pearson de las medidas de fiabilidad con el error real de predicción en problemas de regresión. Cada subtabla corresponde a un conjunto de datos diferente. La primera columna indica el modelo utilizado para generar las predicciones de sensibilidad. Las siguientes columnas corresponden a las correlaciones de cada medida con el error o su valor absoluto.

Breast cancer Wisconsin			
Modelo	$Corr(RE1, \|err\|)$	$Corr(RE2, \|err\|)$	$Corr(RE3, err)$
Árbol	0,245	0,136	0,073
SVM	−0,275	−0,275	−0,089

Concrete			
Modelo	$Corr(RE1, \|err\|)$	$Corr(RE2, \|err\|)$	$Corr(RE3, err)$
Árbol	0,105	0,162	0,334
SVM	0,072	0,072	0,019

Wine quality			
Modelo	$Corr(RE1, \|err\|)$	$Corr(RE2, \|err\|)$	$Corr(RE3, err)$
Árbol	0,036	0,155	0,270
SVM	0,145	0,147	0,031

DEMCAM			
Modelo	$Corr(RE1, \|err\|)$	$Corr(RE2, \|err\|)$	$Corr(RE3, err)$
Árbol	0,165	0,216	0,284
SVM	−0,105	−0,106	−0,004

Tabla 5.5: Resultados de los experimentos para intentar mejorar las predicciones en problemas de regresión. Las predicciones iniciales se generan mediante árboles de regresión. Cada subtabla corresponde a un conjunto de datos diferente. La primera columna indica cómo se calcula θ ($f^*(I) = f(I) + \theta$). Las siguientes columnas corresponden a las medidas de error. Se resalta una casilla en **negrita** cuando el error sea menor que el de las predicciones iniciales.

Breast cancer Wisconsin				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	19,30	25,07	24,09	31,29
$RE3$	19,95	25,91	24,51	31,83
$RE3 \cdot RE1$	86,71	112,61	134,68	174,91
$RE3 \cdot RE2$	63,64	82,65	82,90	107,67
Árbol	19,54	25,38	25,29	32,85
SVM	**18,52**	**24,05**	25,50	33,12

Concrete				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	5,22	6,51	6,91	8,61
$RE3$	5,79	7,21	7,49	9,33
$RE3 \cdot RE1$	113,80	141,77	166,84	207,84
$RE3 \cdot RE2$	66,26	82,55	97,79	121,82
Árbol	5,99	7,46	7,93	9,89
SVM	6,03	7,51	8,19	10,20

Wine quality				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	0,525	10,50	0,700	13,99
$RE3$	0,541	10,83	**0,700**	**13,98**
$RE3 \cdot RE1$	0,552	11,04	0,728	14,55
$RE3 \cdot RE2$	0,534	10,67	0,701	14,01
Árbol	0,551	11,01	0,712	14,24
SVM	**0,523**	**10,46**	**0,691**	**13,81**

DEMCAM				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	0,812	27,07	0,990	32,99
$RE3$	**0,808**	**26,92**	**0,983**	**32,76**
$RE3 \cdot RE1$	0,815	27,17	1,000	33,32
$RE3 \cdot RE2$	**0,796**	**26,52**	0,994	33,15
Árbol	1,001	33,38	1,234	41,13
SVM	0,887	29,56	1,16	38,52

Tabla 5.6: Resultados de los experimentos para intentar mejorar las predicciones en problemas de regresión. Las predicciones iniciales se generan mediante máquinas de vectores de soporte. Cada subtabla corresponde a un conjunto de datos diferente. La primera columna indica cómo se calcula θ ($f^*(I) = f(I)+\theta$). Las siguientes columnas corresponden a las medidas de error. Se resalta una casilla en **negrita** cuando el error sea menor que el de las predicciones iniciales.

Breast cancer Wisconsin				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	17,45	22,67	24,54	31,88
$RE3$	17,47	22,68	**24,52**	**31,85**
$RE3 \cdot RE1$	17,48	22,70	**24,52**	**31,84**
$RE3 \cdot RE2$	17,47	22,68	**24,53**	**31,86**
Árbol	19,32	25,09	**23,39**	**30,37**
SVM	18,48	24,01	25,00	32,47

Concrete				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	9,12	11,36	11,30	14,08
$RE3$	**9,12**	**11,36**	**11,30**	**14,08**
$RE3 \cdot RE1$	**9,12**	**11,36**	11,30	14,08
$RE3 \cdot RE2$	**9,12**	**11,36**	11,30	14,08
Árbol	10,03	12,50	12,53	15,60
SVM	10,04	12,51	12,44	15,50

Wine quality				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	0,497	9,93	0,650	13,01
$RE3$	0,497	9,93	**0,650**	**13,00**
$RE3 \cdot RE1$	0,497	9,93	**0,650**	**13,01**
$RE3 \cdot RE2$	0,497	9,93	**0,650**	**13,01**
Árbol	0,525	10,49	0,678	13,57
SVM	0,501	10,03	0,658	13,16

DEMCAM				
θ	**ErrAbs**	**ErrAbsRel**	**RMSE**	**RMSErel**
0	0,673	22,42	0,845	28,18
$RE3$	0,673	22,42	0,846	28,19
$RE3 \cdot RE1$	**0,672**	**22,40**	**0,845**	**28,18**
$RE3 \cdot RE2$	**0,673**	**22,41**	**0,845**	**28,18**
Árbol	0,807	26,89	0,970	32,34
SVM	0,750	25,02	0,938	31,27

5.2.3. Discusión

En la Tabla 5.4 se muestran las correlaciones de las tres medidas de fiabilidad ($RE1$, $RE2$ y $RE3$) con el error real de predicción. Se observa que ninguna de las correlaciones es muy elevada, aunque en el caso de los árboles es positiva para los cuatro conjuntos mientras que en el caso de las máquinas de vectores de soporte depende del conjunto de datos. Esta percepción coincide con los resultados mostrados en [16], donde los árboles de regresión se comportan mejor, en términos de las medidas analizadas, que las máquinas de vectores de soporte. No obstante, se puede afirmar que las medidas no son tan robustas como sería deseable, ya que dependen del conjunto de datos y del modelo utilizados.

A continuación se valoran los resultados de los métodos de corrección de errores propuestos en este trabajo.

En las Tablas 5.5 y 5.6 se recogen las medidas de error (definidas en la sección 5.2.2) para las predicciones resultantes de aplicar las cinco fórmulas de corrección de errores propuestas anteriormente. En la primera de las tablas se emplean árboles de regresión para generar las predicciones iniciales y en la segunda, máquinas de vectores de soporte.

Se puede observar que, en varias situaciones, se logra mejorar la predicción inicial (valores resaltados con negrita). Sin embargo, dicha mejoría suele ser muy pequeña, incluso aparentemente nula debido al número de cifras significativas reportadas. La mayor diferencia se produce en la Tabla 5.5, en el conjunto Breast cancer Wisconsin, donde al aplicar la corrección mediante SVM, el error absoluto pasa de 19,30 a 18,52. No obstante, no supone una mejora efectiva, ya que en la Tabla 5.6 se observa que el error absoluto de las predicciones iniciales para el mismo conjunto pero con máquinas de vectores de soporte es 17,45.

Ninguno de los cinco mecanismos correctores produce mejoras consistentemente en los diversos conjuntos de datos. Tampoco se puede concluir que ninguno de ellos sea mejor que los demás. Lo que sí se observa es el riesgo que tienen las fórmulas $RE1 \cdot RE3$ y $RE2 \cdot RE3$ de disparar el error, como sucede en los dos primeros conjuntos de la Tabla 5.5.

Finalmente, se sugiere como trabajo futuro la inclusión de nuevos atributos en el modelo entrenado para la corrección de errores de predicción. Por ejemplo, pueden incluirse los propios valores de los atributos de las instancias, junto a los existentes $RE1$, $RE2$ y $RE3$.

5.3. Explicaciones en clasificación

El objetivo fundamental de esta sección consiste en implementar el Algoritmo 4 expuesto en la sección 4.2.2.4 y mostrar las ventajas que puede ofrecer, tanto a expertos en modelos como a cualquier usuario que utilice un sistema de aprendizaje automático como apoyo a su labor. Para ello, se valida la implementación realizada empleando un conjunto de datos bien conocido que ayude a verificar que el funcionamiento es correcto y, finalmente, se aplica a un nuevo conjunto de datos del ámbito médico.

Los modelos entrenados son árbol de clasificación (J48 de WEKA [28]) y máquina de vectores de soporte (LIBSVM [29]). El primero sirve para contrastar las explicaciones ofrecidas por el algoritmo con las que se pueden deducir del propio árbol, gracias a su transparencia. La aplicación sobre una máquina de vectores de soporte muestra el verdadero potencial del algoritmo, que consiste en proporcionar explicaciones de las predicciones de un modelo de tipo *caja negra*, haciendo que sea más transparente.

El algoritmo de generación de las explicaciones se implementa mediante scripts Bash [32] y Gawk [33]. Se hace validación cruzada de 10 hojas para obtener las explicaciones de las predicciones de todas las instancias de un conjunto de datos. Además, las explicaciones de las predicciones individuales se promedian para obtener las contribuciones de un valor (o rango de valores) de un atributo y, a su vez, promediando éstas se generan las contribuciones globales del atributo hacia la predicción de una clase u otra. A la hora de promediar, se consideran por separado las contribuciones positivas y las negativas, ya que, de lo contrario, la contribución de un valor o de un atributo puede resultar casi nula cuando en realidad es muy influyente en ambos sentidos. Se implementa en R [30] un sistema de visualización de resultados similar al que aparece en [19] y [22], facilitando su interpretación.

5.3.1. Conjuntos de datos

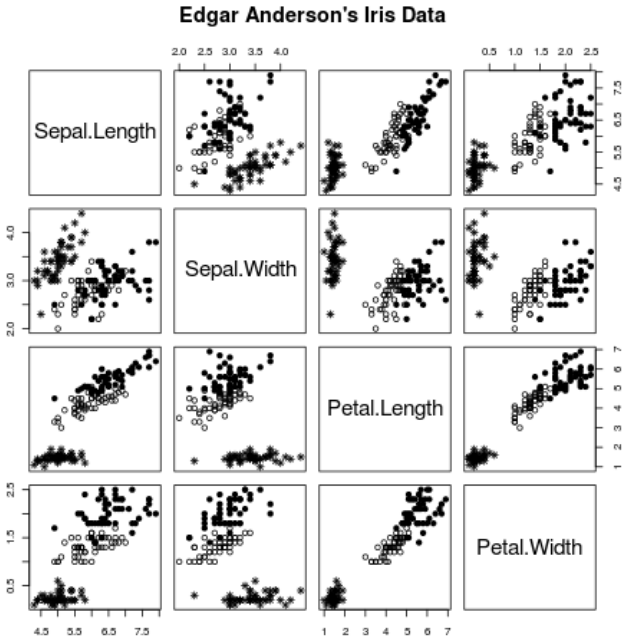
En la Tabla 5.7 se muestran los conjuntos de datos empleados en esta sección. El conjunto Iris, tomado del repositorio UCI [34], es ampliamente conocido en el mundo del aprendizaje automático y se escoge precisamente por ello. Así, es más sencillo e intuitivo comprobar la validez de las explicaciones obtenidas mediante la aplicación del Algoritmo 4. El otro conjunto, denominado DEMCAM, ya se ha presentado en la sección 5.2 por lo que simplemente cabe añadir que en esta sección se toman únicamente cuatro de sus atributos más informativos, para simplificar el análisis y la visualización de los resultados.

Tabla 5.7: Conjuntos de datos empleados en los experimentos para explicar las predicciones en problemas de clasificación. n indica el número de instancias, a el número de atributos y la última columna indica el tipo de los mismos.

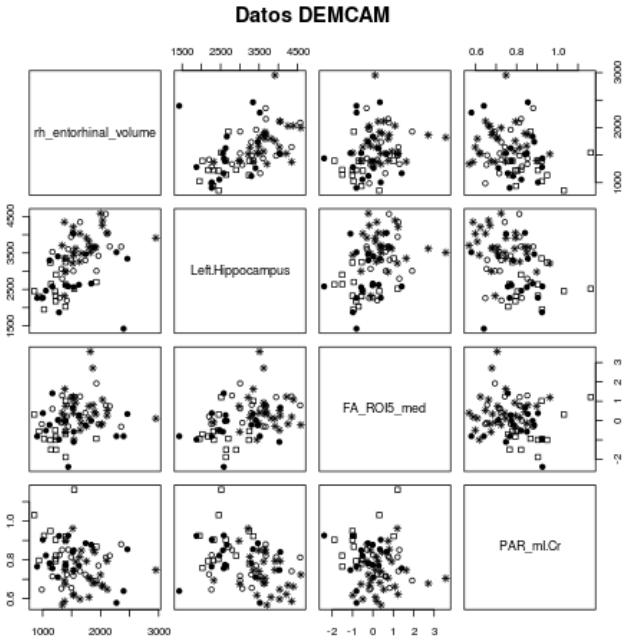
Nombre	n	a	Tipo
Iris	150	4	numérico
DEMCAM	78	238 (se toman 4)	numérico

5.3.2. Resultados

En primer lugar, en la Figura 5.5 se muestran las proyecciones de los conjuntos de datos Iris y DEMCAM sobre los subespacios definidos por cada par de atributos. A continuación, en la Figura 5.6 se representa el árbol de clasificación entrenado sobre una de las hojas de la validación cruzada realizada para el conjunto Iris. Dicho árbol está acompañado de dos gráficas que recogen las explicaciones globales de los cuatro atributos hacia la predicción de dos de las clases del conjunto Iris. A continuación, se presentan varias gráficas sobre más explicaciones generadas en diversas situaciones. En cada una de ellas se especifica si las explicaciones corresponden a una instancia concreta, a las contribuciones de los valores de un atributo o a las contribuciones globales de todos los atributos.

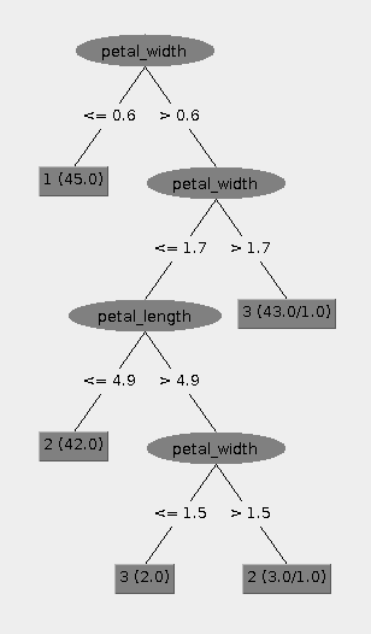


(a) Iris

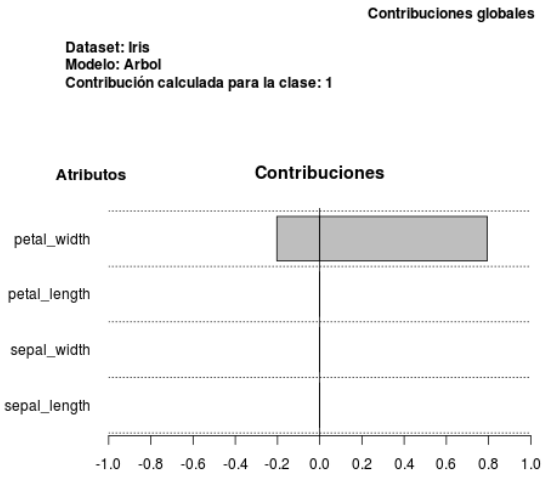


(b) DEMCAM

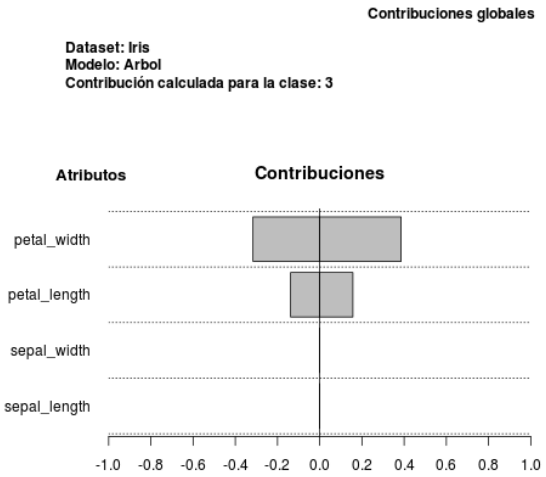
Figura 5.5: Representaciones bidimensionales de los conjuntos de datos Iris y DEMCAM. En cada recuadro se sitúan las instancias del conjunto de datos en función de los valores que toman los dos atributos implicados. Ambos son problemas de clasificación, por lo que se representan las clases a las que pertenecen las instancias con símbolos diferentes.



(a) Árbol de clasificación.



(b) Contribuciones a la predicción de la clase 1.



(c) Contribuciones a la predicción de la clase 3.

Figura 5.6: Árbol de clasificación y contribuciones globales de los atributos para el conjunto de datos Iris. El árbol se ha entrenado para la hoja correspondiente al conjunto de test al que pertenecen las instancias que se analizan posteriormente.

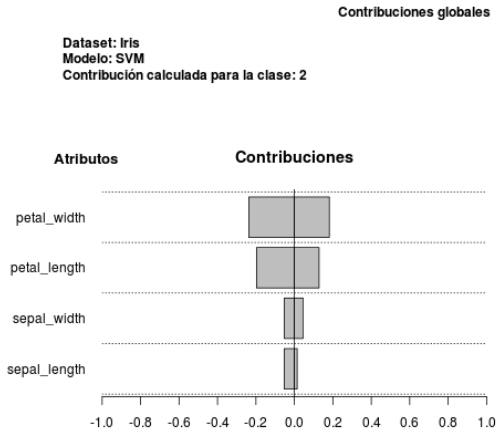
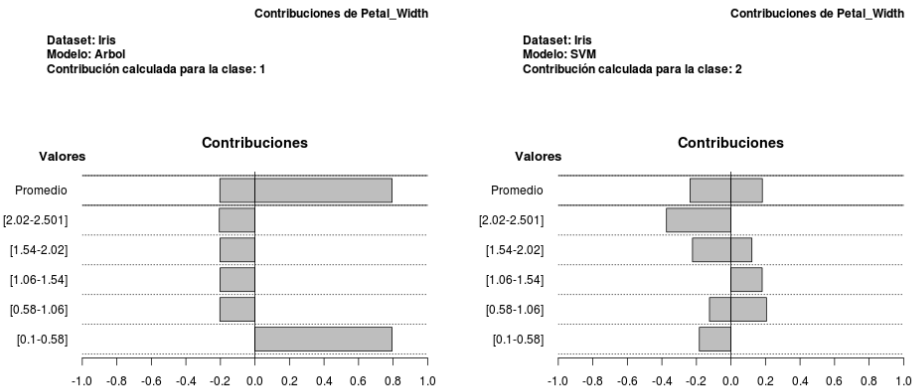


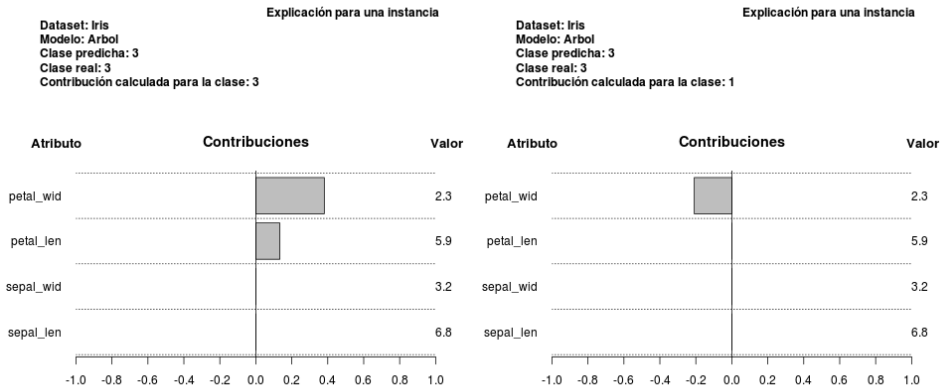
Figura 5.7: Contribuciones globales de los atributos para un modelo de máquinas de vectores de soporte.



(a) Contribuciones a la clase 1 para un árbol.

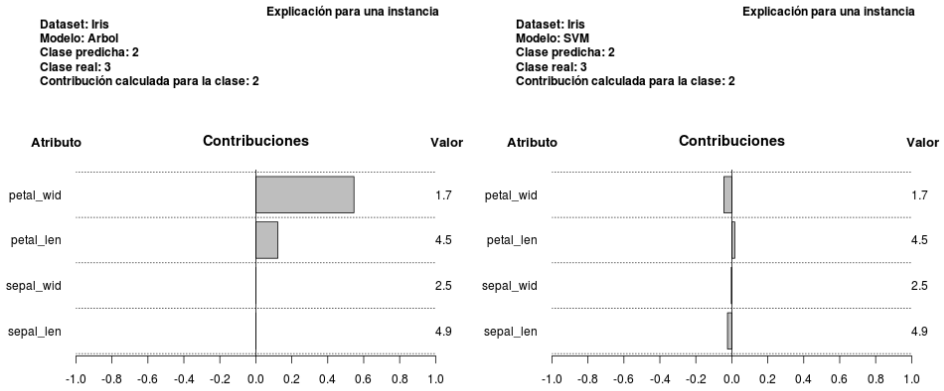
(b) Contribuciones a la clase 2 para una SVM.

Figura 5.8: Contribuciones de un atributo a la predicción de dos clases diferentes del conjunto Iris. La primera barra (Promedio) representa el promedio de las contribuciones de todos los rangos de valores.



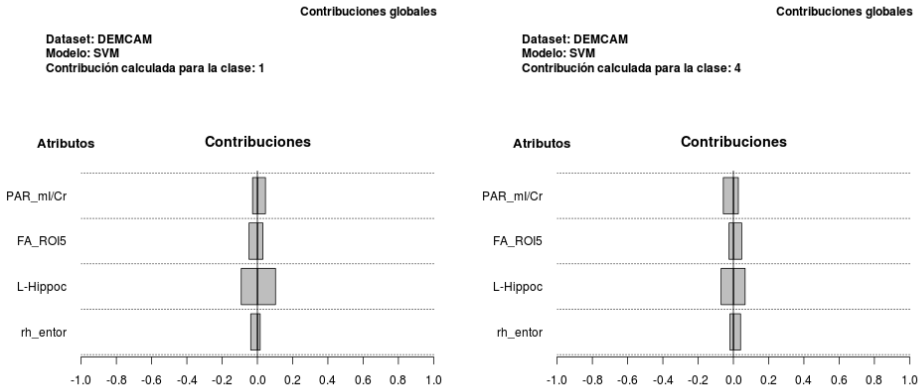
(a) Contribuciones para un árbol. **(b)** Contribuciones para una SVM.

Figura 5.9: Explicaciones de la predicción de una instancia bien clasificada del conjunto Iris mediante árboles. Las contribuciones se calculan para la clase predicha (que coincide con la real, 3) y para la clase 1.



(a) Contribuciones para un árbol. **(b)** Contribuciones para una SVM.

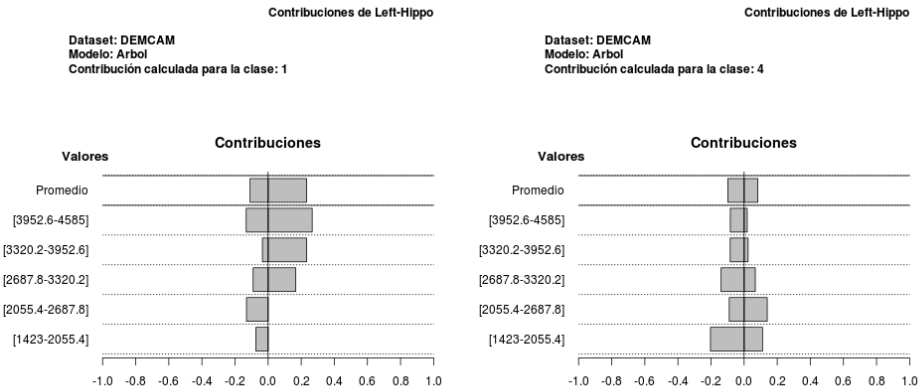
Figura 5.10: Explicaciones de la predicción de una instancia mal clasificada del conjunto Iris mediante dos modelos diferentes. La instancia pertenece realmente a la clase 3 pero se clasifica como 2. Las contribuciones se calculan para la clase 2.



(a) Contribuciones a la predicción de la clase 1.

(b) Contribuciones a la predicción de la clase 4.

Figura 5.11: Contribuciones globales de los atributos a la predicción de dos clases diferentes en el conjunto de datos DEMCAM.



(a) Contribuciones a la predicción de la clase 1.

(b) Contribuciones a la predicción de la clase 4.

Figura 5.12: Contribuciones de los diferentes valores de un atributo a la predicción de dos clases diferentes en el conjunto de datos DEMCAM. La primera barra (Promedio) representa el promedio de las contribuciones de todos los rangos de valores.

5.3.3. Discusión

En la Figura 5.5 se puede apreciar la diferente complejidad de los dos conjuntos de datos empleados. El conjunto Iris tiene tres clases, de las cuales una de ellas está claramente separada de las demás y las otras dos, salvo una pequeña zona de *contacto* son separables incluso por modelos lineales. En cambio, el conjunto DEMCAM tiene cuatro clases que no son tan fácilmente separables, como se puede observar en la gráfica. Por ello, se emplea el primero para explicar las distintas circunstancias que se pueden analizar con el método de explicación implementado.

En la Figura 5.6 se presenta el árbol de clasificación entrenado para el conjunto Iris, que sirve para contrastar los resultados de las contribuciones calculadas con lo que realmente ha aprendido el modelo. Se observa que solamente intervienen dos de los cuatro atributos a la hora de emitir las predicciones. Al lado de dicho árbol se muestran dos gráficas correspondientes a las contribuciones globales de los cuatro atributos hacia la predicción de la clase 1 y de la clase 3. Se verifica que las contribuciones de los dos atributos que no aparecen en el árbol valen exactamente 0. Adicionalmente, en el caso de la clase 1, la contribución del atributo *petal_length* también es nula. Observando el árbol, se comprueba que la decisión de pertenencia o no a la clase 1 se toma únicamente en el primer nivel, en función del valor de *petal_width*, siendo los demás irrelevantes.

En la Figura 5.7 se reportan las contribuciones globales de los atributos del conjunto Iris en las predicciones de un modelo de máquinas de vectores de soporte. A diferencia del árbol de clasificación, en el caso de las SVM no se dispone de una representación del conocimiento extraído por el modelo con la que contrastar los resultados. Precisamente el objetivo del método es aumentar la transparencia de estos modelos de tipo *caja negra*. Se aprecia que en este caso todos los atributos tienen cierta influencia, aunque la de los dos que no aparecen en el árbol de la Figura 5.6 es significativamente menor que la de los otros dos.

En la Figura 5.8 se representan las contribuciones de los diferentes rangos de valores del atributo *Petal_Width*, así como el promedio de las positivas y las negativas para todos ellos. Se ha establecido una división en cinco rangos, aunque la implementación realizada permite especificar cualquier número de rangos en los que dividir el intervalo de posibles valores de un atributo. Las contribuciones de la imagen de la izquierda se han calculado para el modelo de árbol y para la clase 1 y las de la derecha para el modelo de SVM y para la clase 2. La primera refleja lo que sucede en el árbol de la Figura 5.6, que todas las instancias de la clase 1 están confinadas en la zona de valores pequeños del atributo *Petal_Width*. De la segunda, se deduce que es más probable que

el modelo SVM asigne la clase 2 a una instancia para valores intermedios del atributo, mientras que en los valores extremos la contribución es fuertemente negativa, ya que corresponde a las zonas donde se concentran las instancias de las otras dos clases.

Es importante remarcar que las contribuciones reflejan lo que ha aprendido el modelo. No hacen referencia directa a la verdadera distribución de las instancias en el espacio de atributos. Lo que sucede para el conjunto Iris es que los modelos entrenados han capturado muy bien el dominio real del problema y las contribuciones, además de explicar el funcionamiento del modelo, reflejan con bastante fidelidad dicho dominio.

En la Figura 5.9 se recogen las explicaciones de la predicción de una instancia bien clasificada. Se calculan las contribuciones de los valores de sus atributos para predecir las clases 3 (la que realmente predice el modelo y es, además, la correcta) y 1. En el caso de la clase predicha, las contribuciones son positivas y en el otro negativas. De hecho, para descartar la pertenencia a la clase 1, solamente influye el valor de *Petal_Width*, como se comentó anteriormente.

En la Figura 5.10 se muestran las explicaciones para una instancia mal clasificada tanto por el árbol como por la máquina de vectores de soporte. Las contribuciones se calculan para la clase que predicen ambos modelos (que no coincide con la clase real). En el caso del árbol, los valores de *Petal_Width* y *Petal_Length* para la instancia analizada conducen claramente a una hoja en la que se predice la clase 2, por lo que sus contribuciones son fuertemente positivas, reflejando lo aprendido en el árbol pero, en este caso, la predicción es errónea. Las bajas contribuciones en el caso de la SVM pueden indicar que la instancia se encuentra cerca de las fronteras de decisión. Si se localiza la instancia en el diagrama de la Figura 5.5, se ve que está en la zona donde se mezclan ligeramente las clases 2 y 3 y donde, probablemente, la SVM haya establecido una frontera cercana.

Una vez revisado el funcionamiento del método y habiendo explicado cómo se pueden interpretar los diferentes resultados que ofrece, se muestran dos figuras, a modo de ejemplo, resultantes de la aplicación del algoritmo al conjunto DEMCAM.

En la Figura 5.11 se recogen las contribuciones globales de los atributos para hacia la predicción de las clases 1 (sano) y 4 (enfermo de Alzheimer) mediante máquinas de vectores de soporte. En ambos casos, el atributo más influyente en las decisiones del modelo es *L-Hippoc*, aunque con mayor claridad en el caso de la clase 1. Dicho atributo corresponde al volumen del hipocampo izquierdo. En la experiencia de trabajo junto a los profesionales del equipo

DEMCAM, aprendimos que es una de las variables más importantes en el estudio del Alzheimer, por lo que se valora positivamente haber llegado a las mismas conclusiones mediante métodos de aprendizaje automático, sin tener ningún conocimiento a priori sobre el problema.

En la Figura 5.12 se muestran las contribuciones de varios rangos de valores del citado atributo hacia la predicción de las clases 1 y 4. Se observa que los valores más altos son los influyentes para la clase 1, correspondiente a pacientes sanos, mientras que para los enfermos (clase 4), el hipocampo ya está muy dañado y ha sufrido una pérdida importante de volumen. No obstante, la existencia de contribuciones tanto positivas como negativas para casi todos los rangos de valores, pone de manifiesto la extrema complejidad del problema debido a la influencia de muchos otros factores en la enfermedad.

En el ámbito médico, la implantación de sistemas de aprendizaje automático para apoyar las labores de diagnóstico e investigación, sería mejor recibida por los profesionales si se incorporan técnicas como la aquí presentada, que permitan una visualización e interpretación de resultados más sencilla e intuitiva.

Capítulo 6

Conclusiones

El trabajo presentado se puede dividir en dos grandes bloques: una primera parte de estudio teórico sobre modelos, algoritmos de selección de atributos y métodos de análisis de predicciones individuales empleados en problemas de aprendizaje automático y una segunda, en la que se lleva a cabo un trabajo práctico implementando alguna de las técnicas estudiadas, ampliando el alcance de ciertos experimentos, proponiendo un sistema corrector de errores de predicción y valorando todos los resultados obtenidos.

El estudio teórico comienza introduciendo los conceptos básicos que aparecen en todo problema de aprendizaje automático, así como las fases necesarias para abordarlo. El núcleo de todo sistema de aprendizaje automático es un modelo capaz de ajustarse o aprender de un conjunto de datos de entrenamiento para emitir una predicción sobre nuevos datos que reciba en un futuro. Se han revisado los principales modelos de aprendizaje automático supervisado, valorando cualitativamente sus características y su adecuación para ser aplicados a problemas del ámbito médico. A continuación se estudian en profundidad algunas de las técnicas empleadas en otras fases, como son la de selección de atributos y la de postprocesamiento, para evaluar y presentar los resultados generados por el modelo.

Se han estudiado en detalle los algoritmos de selección de atributos Relief, ReliefF y RReliefF, que analizan el espacio de atributos localmente, detectando aquellos que mejor permiten discernir entre las clases a las que pertenece cada instancia. Se presentan brevemente otros métodos de selección de atributos, mRMR y QPFS, comparando las complejidades computacionales de todos ellos y planteando una forma de evaluar su comportamiento, la cual se emplea en la sección experimental correspondiente.

Sobre la fase de postprocesamiento, el análisis se ha focalizado en las predicciones individuales. Normalmente, la valoración del rendimiento de los modelos

se hace mediante medidas globales pero en el ámbito médico cobra especial relevancia una evaluación individualizada. Para ello se ha estudiado un método para estimar la fiabilidad de una predicción para problemas de regresión. A partir de esas ideas, se propone y se implementa en la sección de experimentos un sistema para tratar de corregir las predicciones emitidas por un modelo.

Se ha estudiado una técnica, también correspondiente a la etapa de postprocesamiento pero para problemas de clasificación, que permite conocer cómo contribuye el valor de cada atributo a la predicción de una instancia concreta. Agregando la información se pueden obtener las contribuciones globales de cada valor de un atributo y del atributo en sí. Surge de una idea sencilla pero con serias limitaciones y, para superarlas, se recurre a la teoría matemática de juegos cooperativos. Durante el desarrollo teórico que da lugar al algoritmo final, se proporciona una demostración alternativa a uno de los teoremas importantes que intervienen (ver Apéndice A). En la sección experimental se han implementado el algoritmo y un sistema apropiado para visualizar los resultados.

El trabajo práctico realizado aborda las tareas que se han ido mencionando anteriormente. En cada una de las secciones en las que se divide se han discutido en detalle los resultados obtenidos. A continuación se resumen las conclusiones más relevantes de cada uno.

Los algoritmos de la familia *Relief* analizan localmente el espacio de atributos, lo que les permite encontrar dependencias que pasan desapercibidas para los algoritmos mRMR y QPFS por su enfoque global. Esto supone una ventaja para los *Relief* en ciertos dominios. En los demás analizados, la capacidad de distinguir los atributos importantes de los aleatorios ha sido muy similar para todos los algoritmos.

Las medidas estudiadas para estimar la fiabilidad de las predicciones dependen en exceso del problema abordado y del modelo utilizado. Por lo tanto no son tan robustas como sería deseable, aunque en algunos casos sí resultan de utilidad. El sistema automático que se ha propuesto para corregir los errores de predicción no ha presentado un comportamiento satisfactorio en los conjuntos de datos sobre los que se ha aplicado. No obstante, se plantean algunas ideas para mejorarlo en futuros trabajos, como la inclusión de atributos adicionales.

Los resultados obtenidos tras la implementación y aplicación del algoritmo de explicación de las predicciones han sido muy satisfactorios. Se ha comprobado que las contribuciones calculadas explican cómo emite un modelo las predicciones a partir de los valores de los atributos, reflejando el conocimiento

que ha adquirido. Esto permite hacer más transparentes los modelos considerados tradicionalmente como una *caja negra* y facilitan su uso, por ejemplo, para especialistas médicos que empleen un sistema de aprendizaje automático como apoyo a sus labores de diagnóstico e investigación. Finalmente, se ha aplicado a un conjunto de datos procedente de dicho ámbito y que recoge información sobre resonancias magnéticas tomadas para el estudio de la Enfermedad de Alzheimer.

Apéndice A

Demostración del Teorema 3

Demostración. Según la Definición 2, $\Delta(\emptyset) = 0$. Por tanto, se puede considerar el juego cooperativo $\langle \mathbb{A}, \Delta \rangle$. La función que asigna un valor a la interacción entre los atributos de un subconjunto S ($\xi(S)$) se ha definido recursivamente y no es operativa. El primer paso consiste en obtener una definición no recursiva de $\xi(S)$.

Lema 1. *La forma no recursiva de $\xi(S)$ es:*

$$\xi(S) = \sum_{W \subseteq S} \left((-1)^{|S|-|W|} \Delta(W) \right).$$

Demostración. Se prueba por inducción matemática sobre el número de elementos de S ($|S|$).

El caso base ($S = \emptyset$) es trivial. Según ambas definiciones (la recursiva y la no recursiva):

$$\xi(\emptyset) = \Delta(\emptyset) = 0.$$

Para el caso general se asume que la definición no recursiva se cumple para $|S| \leq a$. Hay que demostrar que se cumple para $|S| = a + 1$. Agrupando los subconjuntos de S según su tamaño, se puede escribir la definición recursiva (omitiendo el caso $j = 0$ porque $\xi(\emptyset) = 0$) como:

$$\xi(S) = \Delta(S) - \sum_{j=1}^{a} \sum_{\substack{W \subset S \\ |W|=j}} \xi(W).$$

Aplicando la hipótesis de inducción se tiene:

$$\xi(S) = \Delta(S) - \sum_{j=1}^{a} \sum_{\substack{W \subset S \\ |W|=j}} \sum_{V \subseteq W} \left((-1)^{j-|V|} \Delta(V) \right).$$

Agrupando los subconjuntos de W según su tamaño:

$$\xi(S) = \Delta(S) - \sum_{j=1}^{a} \sum_{\substack{W \subset S \\ |W|=j}} \sum_{\hat{j}=1}^{j} \sum_{\substack{V \subset W \\ |V|=\hat{j}}} \left((-1)^{j-\hat{j}} \Delta(V) \right).$$

Reordenando términos se tiene:

$$\xi(S) = \Delta(S) - \sum_{j=1}^{a} \sum_{\hat{j}=1}^{j} (-1)^{j-\hat{j}} \sum_{\substack{W \subset S \\ |W|=j}} \sum_{\substack{V \subset W \\ |V|=\hat{j}}} \Delta(V).$$

En la expresión $\sum_{\substack{W \subset S \\ |W|=j}} \sum_{\substack{V \subset W \\ |V|=\hat{j}}} \Delta(V)$, el término $\Delta(V)$ correspondiente a un subconjunto de S de tamaño $\hat{j}$, aparece repetido tantas veces como está contenido V en subconjuntos de S de tamaño j ($j \geq \hat{j}$). El número combinatorio $\binom{a+1-\hat{j}}{j-\hat{j}}$ indica dicho número de repeticiones. Por tanto:

$$\xi(S) = \Delta(S) - \sum_{j=1}^{a} \sum_{\hat{j}=1}^{j} (-1)^{j-\hat{j}} \binom{a-\hat{j}+1}{j-\hat{j}} \sum_{\substack{W \subset S \\ |W|=\hat{j}}} \Delta(W).$$

Cambiando los índices de sumación:

$$\xi(S) = \Delta(S) - \sum_{\hat{j}=1}^{a} \sum_{j=\hat{j}}^{a} (-1)^{j-\hat{j}} \binom{a-\hat{j}+1}{j-\hat{j}} \sum_{\substack{W \subset S \\ |W|=\hat{j}}} \Delta(W).$$

Se puede reescribir el término

$$\sum_{j=\hat{j}}^{a} (-1)^{j-\hat{j}} \binom{a-\hat{j}+1}{j-\hat{j}} = \sum_{j=0}^{a-\hat{j}} (-1)^{j} \binom{a-\hat{j}+1}{j},$$

y aplicando el Teorema del Binomio de Newton se obtiene:

$$\sum_{j=0}^{a-\hat{j}} (-1)^{j} \binom{a-\hat{j}+1}{j} = -(-1)^{a-\hat{j}+1} + \sum_{j=0}^{a-\hat{j}+1} (-1)^{j} \binom{a-\hat{j}+1}{j} = (-1)^{a-\hat{j}}.$$

Por tanto:

$$\xi(S) = \Delta(S) - \sum_{\hat{j}=1}^{a} (-1)^{a-\hat{j}} \sum_{\substack{W \subset S \\ |W|=\hat{j}}} \Delta(W) = \Delta(S) - \sum_{\hat{j}=1}^{a} \sum_{\substack{W \subset S \\ |W|=\hat{j}}} (-1)^{a-|W|} \Delta(W).$$

Finalmente, se deshace la agrupación por tamaños de W y se incluye $\Delta(S)$ en el sumatorio (recordando que $|S| = a+1$):

$$\xi(S) = \sum_{W \subseteq S} \left((-1)^{|S|-|W|} \Delta(W) \right).$$

□

Con la expresión no recursiva de ξ obtenida en el Lema 1, se puede escribir $\varphi_j(\Delta)$ sin emplear ξ:

$$\varphi_j(\Delta) = \sum_{W \subseteq \mathbb{A}\backslash\{j\}} \frac{\xi(W \cup \{j\})}{|W \cup \{j\}|} = \sum_{W \subseteq \mathbb{A}\backslash\{j\}} \frac{\sum_{V \subseteq (W \cup \{j\})} \left((-1)^{(|W \cup \{j\}| - |V|)} \Delta(V)\right)}{|W \cup \{j\}|}.$$

Se pretende demostrar que

$$\varphi_j(\Delta) = Sh_j(\Delta) = \sum_{S \subseteq \mathbb{A}\backslash\{j\}} \frac{(a-s-1)!\, s!}{a!} \left(\Delta(S \cup \{j\}) - \Delta(S)\right),$$

donde $a = |\mathbb{A}|$ y $s = |S|$.

Por tanto, fijado un subconjunto $S \subseteq \mathbb{A}\backslash\{j\}$, se debe contar el número de apariciones de $\Delta(S \cup \{j\})$ y $\Delta(S)$ en $\varphi_j(\Delta)$. Dado que el elemento $\{j\}$ se suprime en la selección de subconjuntos del sumatorio externo y se incluye en el interno, $\Delta(S \cup \{j\})$ y $\Delta(S)$ aparecen el mismo número de veces pero con el signo cambiado, debido a la diferente paridad del cardinal de los conjuntos $(S \cup \{j\})$ y S.

Fijado un subconjunto $W \subseteq \mathbb{A}\backslash\{j\}$ (es decir, se escoge un sumando concreto del sumatorio externo de $\varphi_j(\Delta)$), $\Delta(S \cup \{j\})$ y $\Delta(S)$ aparecen una vez si y solo si $S \subseteq W$ [1]. Entonces, hay que contar cuántas veces se cumple $(S \subseteq W \subseteq \mathbb{A}\backslash\{j\})$.

El número de subconjuntos de tamaño w contenidos en $\mathbb{A}\backslash\{j\}$ y que a su vez contienen a S, viene dado por el número combinatorio $\binom{a-s-1}{w-s}$. Sumando sobre todos los posibles tamaños de W para cada S, se tiene

$$\varphi_j(\Delta) = \sum_{S \subseteq \mathbb{A}\backslash\{j\}} \sum_{\hat{j}=s}^{a-1} \frac{(-1)^{\hat{j}-s}}{\hat{j}+1} \binom{a-s-1}{\hat{j}-s} \left(\Delta(S \cup \{j\}) - \Delta(S)\right).$$

Lema 2.

$$\sum_{\hat{j}=s}^{a-1} \frac{(-1)^{\hat{j}-s}}{\hat{j}+1} \binom{a-s-1}{\hat{j}-s} = \frac{(a-s-1)!\, s!}{a!}.$$

Demostración. Se prueba por inducción matemática sobre las variables s y a. Se debe cubrir la región del plano cartesiano discreto $(\mathbb{Z} \times \mathbb{Z})$ correspondiente a $0 \leq s \leq a-1$. Para ello, se toma como caso base la *diagonal* $s = a-1$:

$$\sum_{\hat{j}=a-1}^{a-1} \frac{(-1)^{\hat{j}-a+1}}{\hat{j}+1} \binom{a-(a-1)-1}{\hat{j}-a+1} = \frac{1}{a} = \frac{(a-(a-1)-1)!\, (a-1)!}{a!}.$$

[1] Obsérvese que si $S \not\subseteq W$, es imposible que $S = V$, ya que $V \subseteq (W \cup \{j\})$. Por otro lado, el sumatorio interno recorre los subconjuntos de $W \cup \{j\}$ y no puede aparecer más de una vez cada uno.

Para probar el caso general, se asume que la proposición es verdadera para ciertos s y a, $s-1$ y $a-1$ $(s<a)$ y se demuestra que es cierta para $s-1$ y a. Intuitivamente, se parte del caso base ya demostrado, $s=a-1$, y se cubre toda la región citada de arriba hacia abajo y de izquierda a derecha (con a en el eje de abscisas y s en el de ordenadas).

Aplicando la propiedad de los números combinatorios $\binom{\mu}{\eta}=\binom{\mu-1}{\eta}+\binom{\mu-1}{\eta-1}$, se tiene:

$$\sum_{\hat{j}=s-1}^{a-1}\frac{(-1)^{\hat{j}-s+1}}{\hat{j}+1}\binom{a-s}{\hat{j}-s+1} = \sum_{\hat{j}=s-1}^{a-2}\frac{(-1)^{\hat{j}-s+1}}{\hat{j}+1}\binom{a-s-1}{\hat{j}-s+1} \quad \text{(A.1)}$$

$$+\sum_{\hat{j}=s}^{a-1}\frac{(-1)^{\hat{j}-s+1}}{\hat{j}+1}\binom{a-s-1}{\hat{j}-s}. \quad \text{(A.2)}$$

Aplicando la hipótesis de inducción correspondiente a $s-1$ y $a-1$ sobre (A.1), se tiene:

$$\text{(A.1)} = \sum_{\hat{j}=(s-1)}^{(a-1)-1}\frac{(-1)^{\hat{j}-(s-1)}}{\hat{j}+1}\binom{(a-1)-(s-1)-1}{\hat{j}-(s-1)} = \frac{(a-s-1)!\,(s-1)!}{(a-1)!}.$$

Extrayendo (-1) como factor común de (A.2) y aplicando la hipótesis de inducción correspondiente a s y a:

$$\text{(A.2)} = -\sum_{\hat{j}=s}^{a-1}\frac{(-1)^{\hat{j}-s}}{\hat{j}+1}\binom{a-s-1}{\hat{j}-s} = -\frac{(a-s-1)!\,s!}{a!}.$$

Finalmente,

$$\begin{aligned}
\text{(A.1)}+\text{(A.2)} &= \frac{(a-s-1)!\,(s-1)!}{(a-1)!}-\frac{(a-s-1)!\,s!}{a!}\\
&= \frac{a\,(a-s-1)!\,(s-1)!}{a!}-\frac{s\,(a-s-1)!\,(s-1)!}{a!}\\
&= \frac{(a-s)\,(a-s-1)!\,(s-1)!}{a!}\\
&= \frac{(a-s)!\,(s-1)!}{a!}.
\end{aligned}$$

□

Aplicando el Lema 2 sobre $\varphi_j(\Delta)$ se concluye:

$$\varphi_j(\Delta)=\sum_{S\subseteq\mathbb{A}\setminus\{j\}}\frac{(a-s-1)!\,s!}{a!}.\,(\Delta(S\cup\{j\})-\Delta(S))=Sh_j(\Delta).$$

□

Bibliografía

[1] I. Kononenko. *Machine learning for medical diagnosis: history, state of the art and perspective*. Artificial Intelligence in Medicine, 23(1), 89–109, 2001.

[2] J.R. Quinlan. *Induction of decision trees*. Machine Learning 1, 81–106, 1986.

[3] D.A. Morales, E. Bengoetxea, P. Larrañaga, M. García, Y. Franco, M. Fresnada, M. Merino. *Bayesian classification for the selection of in vitro human embryos using morphological and clinical data*. Computer Methods and Programs in Biomedicine 90, 104–116, 2008.

[4] R. Blanco, I. Inza, M. Merino, J. Quiroga, P. Larrañaga. *Feature selection in Bayesian classifiers for the prognosis of survival of cirrhotic patients treated with TIPS*. Journal of Biomedical Informatics 38, 376–388, 2005.

[5] W. McCulloch, W. Pitts. *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 7, 115–133, 1943.

[6] F. Rosenblatt. *The Perceptron: a probabilistic model for information storage and organization in the brain*. Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, 386–408, 1958.

[7] C.M. Bishop. *Neural Networks for Pattern Recognition*. University, 2005.

[8] J. Shawe-Taylor, N. Cristianini. *Support Vector Machines*. Cambridge University Press, 2000.

[9] K. Kira, L.A. Rendell. *A practical approach to feature selection*. Machine Learning: Proceedings of International Conference (ICML92), 249–256, 1992.

[10] M. Robnik-Sikonja, I. Kononenko. *Theoretical and empirical analysis of ReliefF and RReliefF*. Machine Learning, 53(1-2), 23–69, 2003.

[11] I. Kononenko. *Estimating attributes: analysis and extensions of RELIEF*. ECML 1994, 171–182.

[12] M. Robnik-Sikonja, I. Kononenko. *An adaptation of Relief for attribute estimation in regression*. ICML 1997, 296–304.

[13] C. Ding, H. Peng. *Minimum redundancy feature selection from microarray gene expression data*. Journal of Bioinformatics and Computational Biology, 3(2), 185–205, 2005.

[14] Y. Zhang, C. Ding, T. Li. *Gene selection algorithm by combining RELIEFF and mRMR*. BMC Genomics, 9(2), S27, 2008.

[15] I. Rodríguez-Luján, R. Huerta, C. Elkan, C. Santa Cruz. *Quadratic Programming Feature Selection*. Journal of Machine Learning Research 11, 1491–1516, 2010.

[16] Z. Bosnic, I. Kononenko. *Estimation of individual prediction reliability using the local sensitivity analysis*. Applied Intelligence, 29(3), 187–203, 2008.

[17] Y. Freund, R.E. Schapire. *A short introduction to boosting*. Journal of Japanese Society for Artificial Intelligence, 14(5), 771–780, 1999.

[18] E. Parzen. *On estimation of a probability density function and mode*. Annals of Mathematical Statistics 33, 1065–1076, 1962.

[19] M. Robnik-Sikonja, I. Kononenko. *Explaining classifications for individual instances*. IEEE Transactions on Knowledge and Data Engineering, 20(5), 589–600, 2008.

[20] A. Niculescu-Mizil, R. Caruana. *Predicting good probabilities with supervised learning*. Proceedings of the 22^{nd} International Conference on Machine Learning, 2005.

[21] E. Strumbelj, I. Kononenko, M. Robnik-Sikonja. *Explaining instance classifications with interactions of subsets of feature values*. Data & Knowledge Engineering, 68(10), 886–904, 2009.

[22] E. Strumbelj, I. Kononenko. *An efficient explanation of individual classifications using game theory*. Journal of Machine Learning Research, 11, 1–18, 2010.

[23] A. Keinan, B. Sandbank, C.C. Hilgetag, I. Meilijson, E. Ruppin. *Fair Attribution of Functional Contribution in Artificial and Biological Networks*. Neural Computation 16, 1887–1915, 2004.

[24] S. Cohen, G. Dror, E. Ruppin. *Feature Selection Via Coalitional Game Theory*. Neural Computation 19, 1939–1961, 2007.

[25] J. Castro, D. Gómez, J. Tejada. *Polynomial calculation of the Shapley Value based on sampling*. Computers & Operations Research 36, 1726–1730, 2009.

[26] L.S. Shapley. *A value for n-person games*. Contributions to the theory of games 2, 307–317, 1953.

[27] H. Peng, F. Long, C. Ding. *Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy*. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(8), 1226–1238, 2005.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten. *The WEKA Data Mining Software: An Update*. SIGKDD Explorations, Volume 11, Issue 1, 2009.

[29] Chih-Chung Chang, Chih-Jen Lin. *LIBSVM: a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology 2(27), 1–27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[30] R Development Core Team (2011). *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3–900051–07–0, URL http://www.R-project.org/.

[31] GNU Software. http://www.gnu.org/software/

[32] GNU Bash. http://www.gnu.org/software/bash/

[33] GNU Gawk. http://www.gnu.org/software/gawk/

[34] A. Frank, A. Asuncion (2010). *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]*. Irvine, CA: University of California, School of Information and Computer Science.

[35] Fundación CIEN (Centro de Investigación de Enfermedades Neurológicas). http://www.fundacioncien.es/

[36] Fundación Reina Sofía. http://www.fundacionreinasofia.es/

[37] Instituto de Ingeniería del Conocimiento. http://www.iic.uam.es/

Índice de tablas

Índice de figuras

Índice de algoritmos