Árboles de estimación estocástica de probabilidades: Newton Trees

Tesis de Máster

Fernando Martínez Plumed

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Septiembre 2010

Árboles de estimación estocástica de probabilidades: Newton Trees

Tesis de Máster

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

Dirigida por los Doctores

Cèsar Ferri Ramírez
María José Ramírez Quintana

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Septiembre 2010

A mis padres
A Ruth
A Ana

Agradecimientos

Un trabajo científico puede ser considerado, en un sentido materialista, simple conocimiento generado o descubierto. Pero afortunadamente he aprendido que siempre hay nombres a los cuales debo referirme ya que sin su aporte, directo o indirecto, este trabajo no habría sido posible.

En primer lugar quiero dar las gracias a mis tutores, Cèsar Ferri, José Hernández y María José Ramírez por brindarme la oportunidad de poder trabajar con ellos y por la gran cantidad de tiempo, experiencia y consejos entregados.

Dar las gracias a mis profesores del Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información por su dedicación y por todo lo que me han enseñado este último año.

Quiero agradecer también el buen ambiente de trabajo que se respira en el grupo ELP gracias a todos sus miembros, especialmente a mis compañeros de laboratorio.

Y por último dar las gracias a mi familia, a la cual dedico este trabajo, ya que sin su gran apoyo y dedicación constante, esto no habría sido posible.

Fernando Martínez Plumed
Valencia, Septiembre 2010.

Motivación

La información hoy en día es una materia prima muy valiosa, tanto para empresas u organizaciones como para simples usuarios, ya que para todos es importante obtener información de buena calidad y oportuna. Es por esto que en la sociedad de la información se destina una gran cantidad de recursos en adquirir, almacenar y procesar enormes volúmenes de datos.

Aunque la especie humana posee habilidades extremadamente sofisticadas para detectar patrones y descubrir tendencias, este crecimiento constante de datos (se estima que cada 20 meses se duplica la información en todo el mundo), el cual es almacenado en grandes bases de datos, ha excedido la habilidad del hombre de poder comprenderlos.

Una de las razones por las cuales las técnicas de Minería de Datos han tenido tanto éxito es la necesidad de transformar grandes cantidades de datos en información y conocimiento útil.

Cuando se trata de un problema de clasificación supervisada, una de las técnicas que mejores resultados brindan en la asignación de una clase a un objeto son los árboles de decisión. Algunos ejemplos de algoritmos ya clásicos de árboles de decisión son CLS (*Concept Learning System*), ID3, C4.5 o CART (*Classification And Regression Trees*). Sin embargo, a pesar de ser una técnica ampliamente utilizada y ser la base de numerosísimas variaciones (PETs, Fuzzy Trees,...), la mayoría de ellas siguen presentando algunas desventajas como la ausencia de una función global de tratamiento de las variables, la sensibilidad de las reglas de asignación a pequeñas perturbaciones de los datos, la utilización de sólo una parte de toda la información presente en el árbol o la perdida de compresibilidad en la representación gráfica de los mismos debido al intento de dar una solución a los dos primeros problemas.

Por esto surge la motivación de proponer una nueva metodología de generación de árboles de decisión que permita dar una solución óptima y correcta a los anteriores problemas y sea competitiva con respecto a los algoritmos ya existentes.

Resumen

Este trabajo presenta un nuevo método de inducción de árboles de decisión, los Newton Trees (o Árboles de Estimación Estocástica de Probabilidades). Se trata de una redefinición de los árboles de decisión tradicionales que aúna las ventajas de los árboles de estimación de probabilidades (PETs) y de los árboles basados en distancias:

1. Tratamiento homogéneo de cualquier tipo de datos (nominales, numéricos u ordinales) usando la noción de distancia de forma univariante. Para datos numéricos se puede usar el valor absoluto de la diferencia, y, en el caso de datos nominales u ordinales, la función identidad.

2. Un nuevo criterio de partición basado en el uso de prototipos, obtenidos a partir del conjunto de valores de los atributos, y no de centroides (como en otras técnicas), para el correcto tratamiento de datasets con valores continuos y discretos. Para los datasets con solo atributos continuos, se construye un clúster por atributo y clase y no un punto de corte entre cada pareja de valores como en los algoritmos clásicos. De esta forma, en cada paso, evaluamos m particiones en lugar de $n \times m$, siendo m el número de atributos y n el número de ejemplos.

3. Representación gráfica del árbol sencilla de interpretar.

La novedad principal introducida en esta metodología es que los Newton Trees se basan en un conocimiento estocástico de los árboles de decisión (utilizan la información de todas las ramas y no solo de una como en los árboles de decisión clásicos) que sigue el principio de la *atracción*, el cual relaciona la masa y la distancia a través de la Ley Cuadrática Inversa. Estos árboles tratan estocásticamente las instancias y las distribuyen por todo el árbol usando las probabilidades de cada rama A su vez, aunque los árboles son univariantes, sus particiones no son necesariamente paralelas a los ejes. Esto implica que las particiones pueden crear límites más expresivos que los obtenidos por los árboles de decisión clásicos, sin tener que ser multivariantes.

La estructura, la aplicación y la representación gráfica de los Newton Trees proporcionan una forma de hacer sus predicciones de forma estocástica

y preservando una de las características más deseables de los árboles de decisión, la inteligibilidad.

Además, este nuevo método es directamente extensible para el tratamiento de datos faltantes siguiendo una aproximación similar a aquella que ante un valor faltante se asume que el valor podría tomar cualquiera de los valores (no nulos) de ese atributo, pero utilizando la frecuencia de los valores para asignar una probabilidad a cada uno de ellos.

Este trabajo está estructurado como sigue:

1. En el capítulo 1 se realiza una breve introducción sobre la temática que trata esta tesis en la cual se definen conceptos básicos y definiciones que caracterizan la Minería de Datos y más concretamente los métodos supervisados de inducción de árboles de decisión.

2. El capítulo 2 describe la metodología en la que se basan los Newton Trees, realizando un pequeño estudio del estado del arte, introduciendo la notación necesaria y exponiendo las distintas variaciones y configuraciones posibles de estos árboles.

3. El capítulo 3 proporciona una evaluación experimental de los Newton Trees en sus distintas versiones.

4. El capítulo 4 presenta las conclusiones y el posible trabajo futuro.

Los resultados de los capítulos 2 y 3 se presentan en un artículo aceptado para su publicación en el *23th Australasian Joint Conference on Artificial Intelligence (LNCS)* de título *Newton Trees* y en el artículo *Tratamiento de valores faltantes en Newton Trees* presentado en el *V Simposio de Teoría y Aplicaciones de Minería de Datos*.

Índice

Índice de figuras

Índice de Tablas

Parte I

Resumen de la Investigación

Capítulo 1

Introducción a los Árboles de Decisión

Torturando a los datos.

Resumen: En este capítulo se introduce la Minería de Datos como método de extracción no trivial de información implícita, previamente desconocida y potencialmente útil, a partir de datos. Asimismo, también se incluye una descripción de los Árboles de Decisión como método supervisado no paramétrico de representación secuencial de condiciones para la toma de decisiones.

1.1. Minería de Datos y Descubrimiento de Conocimento

En los últimos años, ha existido un gran crecimiento en nuestras capacidades de generar y recolectar datos, debido básicamente al gran poder de procesamiento de las máquinas así como a su bajo costo de almacenamiento. Sin embargo, dentro de estas enormes masas de datos existe una gran cantidad de información oculta, de gran importancia estratégica, a la que no se puede acceder por las técnicas clásicas de recuperación de la información. El descubrimiento de esta información oculta es posible gracias a la Minería de Datos (*Data Mining*), que se caracteriza por el análisis de grandes cantidades de datos con el fin de descubrir relaciones insospechadas y resumir la información mediante nuevas técnicas de forma que pueda ser interpretable y útil a sus usuarios. Las relaciones y resúmenes obtenidos mediante técnicas de Minería de Datos son conocidos a menudo como modelos o patrones.

Ejemplos de dichos modelos pueden ser reglas, clústeres, grafos, estructuras en árbol y patrones recurrentes en series temporales. Así, el valor real de los datos reside en la información que se puede extraer de ellos, información que ayude a tomar decisiones o mejorar nuestra comprensión de los fenómenos que nos rodean.

La Minería de Datos es un término relativamente moderno que integra numerosas técnicas de análisis de datos y extracción de conocimiento. Este conocimiento puede ser en forma de relaciones, patrones o reglas inferidas de los datos o en forma de una descripción más concisa (es decir, un resumen). Estas relaciones o resúmenes constituyen el modelo de los datos. Aunque la Minería de Datos se basa en varias disciplinas, algunas de ellas más tradicionales (como la estadística), se distingue de ellas en la orientación más hacia el fin que hacia el medio. Y el fin lo merece: ser capaces de extraer patrones, de describir tendencias y regularidades, de predecir comportamientos y, en general, de sacar partido a la información computarizada que nos rodea hoy en día, generalmente heterogénea y en grandes cantidades, permite a los individuos y a las organizaciones comprender y modelar de una manera más eficiente y precisa el contexto en el que deben actuar y tomar decisiones. En la práctica, los modelos pueden ser de dos tipos: predictivos y descriptivos. Los modelos predictivos pretenden estimar valores futuros o desconocidos de variables de interés, que denominamos variables objetivo o dependientes, usando otras variables o campos de la base de datos, a las que nos referiremos como variables independientes o predictivas. Los modelos descriptivos, en cambio, identifican patrones que explican o resumen los datos, es decir, sirven para explorar las propiedades de los datos examinados, no para predecir nuevos datos. Algunas tareas de Minería de Datos que producen modelos predictivos son la clasificación y la regresión, y las que dan lugar a modelos descriptivos son el agrupamiento, las reglas de asociación y el análisis correlacional.

La Minería de Datos no es más que un paso esencial de un proceso más amplio conocido como Descubrimiento de conocimiento en bases de datos (del inglés, *Knowledge Discovery from Databases, KDD*). Este proceso consta de una secuencia iterativa de etapas o fases para la extracción de conocimiento: Análisis del problema, Preparación de Datos, Minería de Datos, Evaluación, Difusión y Uso de Modelos:

1. **Análisis de las necesidades y definición del problema**: Se establecen los objetivos de Minería de Datos. La decisión de implantar un programa de Minería de Datos y el diseño de un plan del mismo deben preceder a cualquiera de las fases. De hecho, establecer cuál es el contexto del problema, los objetivos del mismo y plasmarlos en objetivos de Minería de Datos, es previo a pararnos a pensar en recopilar y preparar los datos, realizar los modelos, evaluarlos y utilizarlos.

2. **Filtrado de datos**: En esta fase se realiza una selección y pre-procesado de los elementos “en bruto"que contienen una base de datos.

3. **Selección de variables**: Aun a pesar de la fase anterior, es habitual que en los bancos de información haya un exceso de datos que dificulte su manejo.

4. **Algoritmos de extracción de conocimiento**: Mediante una técnica de Minería de Datos se obtiene un modelo de conocimiento que representa patrones de comportamiento observados en los valores de las variables del problema o relaciones de asociación entre dichas variables. Generalmente cada técnica de minería aplicada a un modelo de conocimiento distinto precisará un filtrado y una selección de variables previa diferente. Es en la construcción del modelo donde vemos mejor el carácter iterativo del proceso de KDD, ya que será necesario explorar modelos alternativos hasta encontrar aquel que resulte más útil para resolver nuestro problema.

5. **Interpretación y evaluación**: Una vez obtenido el modelo de conocimiento se debe proceder a la validación del mismo, comprobando que las conclusiones obtenidas son válidas y satisfactorias. Medir la calidad de los patrones descubiertos por un algoritmo de Minería de Datos no es un problema trivial, ya que esta medida puede atañer a varios criterios, algunos de ellos bastante subjetivos. Idealmente, los patrones descubiertos deben tener tres cualidades: ser precisos, comprensibles (es decir, inteligibles) e interesantes (útiles y novedosos). Según las aplicaciones puede interesar mejorar algún criterio y sacrificar ligeramente otro. Para entrenar y probar un modelo se parten los datos en dos conjuntos: el conjunto de entrenamiento (training set) y el conjunto de prueba o de test (test set). Esta separación es necesaria para garantizar que la validación de la precisión del modelo es una medida independiente. Si no se usan conjuntos diferentes de entrenamiento y prueba, la precisión del modelo será sobreestimada, es decir, tendremos estimaciones muy optimistas.

6. **Fase de difusión, uso y monitorización**: Una vez construido y validado el modelo puede usarse es necesario su difusión, es decir que se distribuya y se comunique a los posibles usuarios, ya sea por cauces habituales dentro de la organización, reuniones, intranet, etc. El nuevo conocimiento extraído debe integrar el *know-how* de la organización. También es importante medir lo bien que el modelo evoluciona. Aun cuando el modelo funcione bien, debemos continuamente comprobar las prestaciones del mismo. Esto se debe principalmente a que los patrones pueden cambiar. Por lo tanto, el modelo deberá ser monitorizado, lo

que significa que de tiempo en tiempo el modelo tendrá que ser re-evaluado, reentrenado y posiblemente reconstruido completamente.

1.2. Taxonomía de los Métodos de Minería de Datos

Es posible distinguir entre dos tipos principales de paradigmas en la Minería de Datos: paradigma orientado a la verificación (el sistema comprueba la hipótesis del usuario) y paradigma orientado hacia el descubrimiento (el sistema encuentra nuevas normas y patrones de forma autónoma) (15). La figura 1.1 ilustra esta taxonomía. Cada tipo tiene su propia metodología.

Figura 1.1: Taxonomía de los Métodos de Minería de Datos

Los métodos orientados al descubrimiento, que identifican automáticamente patrones en los datos, implican tanto a los métodos de predicción como de descripción. Los métodos descriptivos se centran en la comprensión de la forma de los datos subyacentes, mientras que el objetivo de los métodos predictivos es la construcción de un modelo de comportamiento para predecir los valores de una o más variables y obtener nuevas muestras (no conocidas). Sin embargo, algunos métodos orientados a la predicción, también pueden ayudar a proporcionar una comprensión de los datos. La mayoría de las técnicas orientadas al descubrimiento se basan en aprendizaje inductivo (33), en donde se construye un modelo de manera explícita o implícita generalizando a partir de un número suficiente de ejemplos de entrenamiento. El supuesto

básico del enfoque inductivo es que el modelo de entrenamiento es aplicable a los futuros ejemplos no conocidos. Estrictamente hablando, cualquier forma de inferencia en la que las conclusiones no son derivadas deductivamente de las premisas, puede considerarse como inducción. Por otra parte, los métodos de verificación, evalúan una hipótesis propuesta por una fuente externa (como un experto, etc.). Estos métodos incluyen la mayoría de los métodos estadísticos tradicionales como la prueba de bondad de ajuste, el t-test de las medias y el análisis de la varianza. Estos métodos se asocian en menor medida con la Minería de Datos en comparación con sus colegas orientados al descubrimiento. Esto es debido a que la mayoría de los problemas de minería de datos se centran en la selección de una hipótesis (de un total de un conjunto de hipótesis) en lugar de realizar pruebas sobre una hipótesis ya conocida. El enfoque tradicional de los métodos estadísticos es, por lo general, la estimación de un modelo en contraposición con el principal objetivo de la minería de datos: la identificación de un modelo (26).

1.3. Metodos Supervisados

En el área del Aprendizaje Automático, los métodos de predicción se conocen comúnmente como aprendizaje supervisado. Este se opone al aprendizaje no supervisado en que este último tiene como objetivo la modelización de la distribución de las instancias en un espacio de entrada de n dimensiones. De acuerdo con (36), el término "aprendizaje no supervisado" se refiere a "técnicas de aprendizaje de agrupación de instancias sin un atributo dependiente especificado de antemano", es decir, el objetivo no es predecir nuevos datos, sino describir los existentes. Así, el término "aprendizaje no supervisado" cubre tareas como el agrupamiento, las reglas de asociación, las dependencias funcionales, etc. Sin embargo, los métodos supervisados tienen como objetivo intentar descubrir la relación entre los atributos de entrada (a veces denominados como variables independientes) y un atributo de salida (a veces conocido como variable dependiente). La relación descubierta se representa en una estructura conocida como Modelo tal y como hemos mencionado. Es útil distinguir entre dos modelos principales de aprendizaje supervisado: *Modelos de Clasificación* (clasificadores) y los *Modelos de Regresión*. Estos últimos mapean el espacio de entrada en un dominio de valor real. Por ejemplo, un modelo de regresión puede predecir la demanda de un determinado producto por sus características. Por otra parte, los clasificadores mapean el espacio de entrada en clases predefinidas. Por ejemplo, los clasificadores pueden ser utilizados para clasificar a los consumidores de hipotecas como *"buenos"* (la hipoteca será pagada en el tiempo determinado) y como *"malos"* (la hipoteca no será pagada o se pagara en un tiempo superior al acordado). Entre las muchas alternativas para la representación de los clasificadores, existen, por ejemplo, las máquinas de soporte vectorial, los árboles de decisión,

los estimadores probabilísticos, funciones algebraicas, etc. Este trabajo se centra principalmente en problemas de clasificación, que, junto con la regresión, es una de las aproximaciones mas estudiadas y de las que existen mas variantes.

1.4. Árboles de Decisión

En la minería de datos, los árboles de decisión (o clasificación) son modelos predictivos que pueden ser utilizados para representar, tanto modelos clasificadores como modelos de regresión. Por otra parte, en entornos operacionales, los arboles de decisión se atribuyen a un modelo jerárquico de decisiones y consecuencias. La/s persona/s encargada/s de la toma de decisiones emplea estos árboles de decisión para identificar la estrategia más apropiada (o con más probabilidad) para lograr su objetivo.

Cuando un árbol de decisión se utiliza para tareas de clasificación, se denomina árbol de clasificación; en cambio, cuando se utiliza para tareas de regresión se denomina árbol de regresión. Este trabajo se centra principalmente en los árboles de clasificación.

Los árboles de decisión o clasificación son uno de los métodos de aprendizaje inductivo supervisado no paramétrico más utilizado. Como forma de representación del conocimiento, los árboles de clasificación destacan por su sencillez. A pesar de que carecen de la expresividad de las redes semánticas o de la lógica de primer orden, su dominio de aplicación no está restringido a un ámbito concreto sino que pueden utilizarse en diversas áreas: diagnóstico médico, juegos, predicción meteorológica, control de calidad, etc.

Un árbol de clasificación es una forma de representar el conocimiento obtenido en el proceso de aprendizaje inductivo. Puede verse como un conjunto de condiciones organizadas en una estructura jerárquica en forma de árbol. Cada nodo interior contiene una pregunta sobre un atributo concreto (con un hijo por cada posible respuesta) y cada nodo hoja se refiere a una decisión (clasificación).

La clasificación de nuevas instancias se realiza en base a una serie de preguntas sobre los valores de sus atributos, empezado por el nodo raíz y siguiendo el camino determinado por las respuestas a las preguntas de los nodos internos, hasta llegar a un nodo hoja. La etiqueta asignada a esta hoja es la que se asignará a la instancia a clasificar.

La figura 1.2 muestra un ejemplo típico de árbol de decisión sobre la posibilidad de jugar a tenis dependiendo de las condiciones climatologicas del dia : cielo, humedad y viento (datos discretos)(32).

Es posible representar el árbol de decisión en forma de reglas de la lógica proposicional que se pueden aplicar para poder clasificar nuevas instancias que lleguen al sistema. En concreto, el árbol anterior corresponde a la hipótesis:

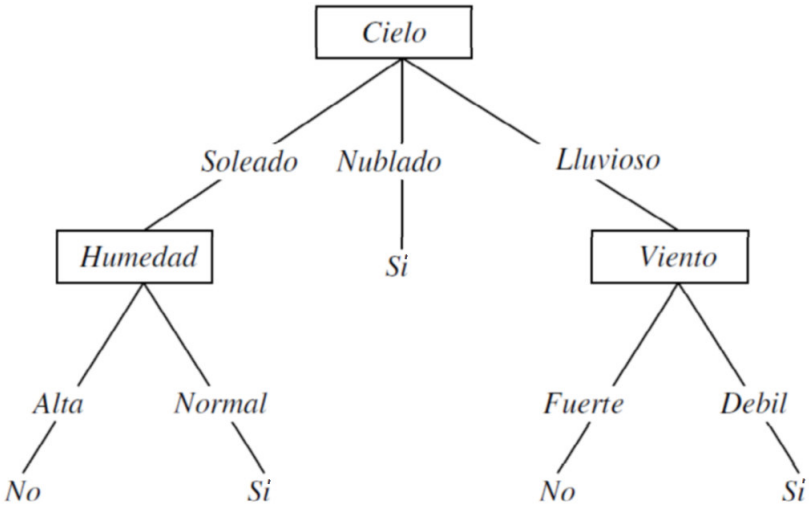

Figura 1.2: Ejemplo de árbol de decisión clásico

$$(Cielo = Soleado \wedge Humedad = Normal)$$
$$\vee (Cielo = Nublado)$$
$$\vee (Cielo = Lluvioso \wedge Viento = Debil)$$

Si a nuestro sistema nos llega la instancia ($Cielo = Soleado$, $Humedad = Alta$, $Viento = Fuerte$), hará cierta (segun las tablas de verdad) la hipotesisi anterior, por tanto, clasificaremos dicha instancia como S.

Los árboles de decisión son una técnica muy popular en la Minería de Datos. Algunas de sus principales ventajas son la simplicidad y transparencia: los árboles de decisión se explican por sí mismos, no es necesario ser un experto en Minería de Datos para ser capaz de seguir una determinada decisión del árbol. Otra de las ventajas es la comprensibilidad de la representación gráfica, ya que suelen representarse como estructuras jerárquicas lo que las hace más fáciles de interpretar que otras técnicas.

1.5. Inducción de los Árboles de Decisión

La construcción del árbol de decisión constituye la fase de aprendizaje. Debido al hecho de que la clasificación trata con clases o etiquetas disjuntas, un árbol de decisión asignará una única clase por ejemplo. Por ello, las particiones existentes en el árbol deberán ser también disjuntas. Esta propiedad

dio lugar al esquema básico de los primeros algoritmos de aprendizaje de árboles de decisión: el espacio de instancias se iba partiendo desde la raiz del arbol en sentido descendente, utilizando cada vez una partición, es decir, un conjunto de condiciones excluyentes y exhaustivas. Una vez seleccionada la partición, ésta no se podía cambiar, aunque más tarde se pensara que había sido una mala elección. Éste es uno de los aspectos clave en los sistemas de aprendizaje de árboles de decisión, el denominado criterio de partición.

El algoritmo recursivo básico de construcción de árboles de decisión es el siguiente:

- **Árbol inicial:** Árbol con un único nodo, sin etiquetar, al que asignamos como conjunto de ejemplos todo el conjunto de entrenamiento.
- **Bucle principal:**

  Consideramos el primer nodo, N, sin etiquetar.
  - Si los ejemplos asignados a N tienen todos la misma clasificación, etiquetamos N con esa clase.
  - En otro caso ...
    - Etiquetamos N con el *mejor* atributo A según el conjunto de ejemplos asignado.
    - Si el atributo es de tipo:
      - **Nominal:** Para cada uno de los k valores de A creamos una nueva arista descendente en el nodo N, y al final de cada una de esas nuevas aristas creamos un nuevo nodo sin etiquetar, $N_1, \ldots, N_k$.
      - **Numérico:** Es necesario seleccionar un valor umbral (*threshold*) T adecuado, el cual debe discretizar correctamente el atributo continuo A en dos intervalos $A_1 = [min(A), T[$ y $A_2 = [T, max(A)]$. Todo valor $A < T$ será asignado a la rama izquierda del árbol y todo $A \geq T$ será asignado a la derecha.
    - Separamos los ejemplos asignados al nodo N según el valor que tomen para el atributo A y creamos nuevos conjuntos de ejemplos para A_i asignados a cada nuevo nodo creado N_i.
- **Hasta** que todos los nodos estén etiquetados (criterio de parada).

Del anterior algoritmo cabe destacar tres puntos clave que se explicaran a continuación, uno es el criterio de selección del mejor atributo para realizar la partición, otro es la selección del umbral en el caso de los atributos numéricos y otro es el criterio de parada.

En lo que sigue, cuando el problema de clasificación sea binario, nos referiremos a ambas clases como clase positiva (que representaremos como $\oplus$) y clase negativa (representada como $\ominus$).

1.5.1. Particiones Posibles

Las particiones son un conjunto de condiciones exhaustivas y excluyentes. Lógicamente, cuantos más tipos de condiciones permitamos, más posibilidades tendremos de encontrar los patrones que hay detrás de los datos, es decir, más expresivos podrán ser los árboles de decisión generados. No obstante, cuantas más particiones elijamos, la complejidad del algoritmo será mayor. Los arboles de decisión tradicionales solo permiten un juego muy limitado de particiones (como veremos en secciones posteriores), es decir, la mayoría contienen un solo tipo de partición para los atributos nominales y un solo tipo de partición para los atributos numéricos:

- **Particiones nominales:** Si un atributo x_i es nominal y tiene posibles valores $\{v_1, v_2, \ldots, v_k\}$, sólo existirá una partición posible para dicho atributo y dicha partición será$(x_i = v_1, x_i = v_2, \ldots, x_i = v_k)$, es decir, una condición con la igualdad entre el atributo y cada posible valor.
- **Particiones numéricas:** Si un atributo x_i es numérico y continuo, puede haber tomado muchos valores diferentes en los ejemplos y puede tomar infinitos posibles valores en general. Por esta razón, se intentan obtener particiones que separen los ejemplos en intervalos. Para ello, las particiones numéricas admitidas son de la forma $(x \leq a, x_i > a)$ donde a es una constante numérica elegida entre un conjunto finito de constantes que discriminen los ejemplos vistos.

La expresividad resultante de las particiones anteriores se conoce como expresividad proposicional cuadricular. El termino proposicional se refiere a que son particiones que solo afectan a un atributo de un ejemplo a la vez, es decir, ni relacionan dos atributos del mismo ejemplo, ni dos atributos de distintos ejemplos. El termino cuadricular hace referencia al tipo de particiones que realizan, especialmente cuando atendemos a los atributos numéricos (véase la Figura 1.3) donde las mismas son paralelas a los ejes y dividen el espacio en áreas rectangulares que determinan los puntos del espacio de entrada que pertenecen a cada clase. Por lo tanto, cada región se define por sus límites de clase que se expresan mediante condiciones sobre el valor del atributo.

Aunque las particiones descritas anteriormente son bastante sencillas, permiten obtener árboles de decisión bastante precisos y muy comprensibles, ya que las condiciones $(x_i = v_1, x_i = v_2, \ldots, x_i = v_k)$ y $(x \leq a, x_i > a)$ se pueden ajustar a muchos patrones y son fácilmente interpretables por los seres humanos.

1.5.2. Criterio de partición

Como vemos en el anterior algoritmo, necesitamos algún tipo de criterio para seleccionar el *mejor* atributo que se usará para ramificar el árbol en cada

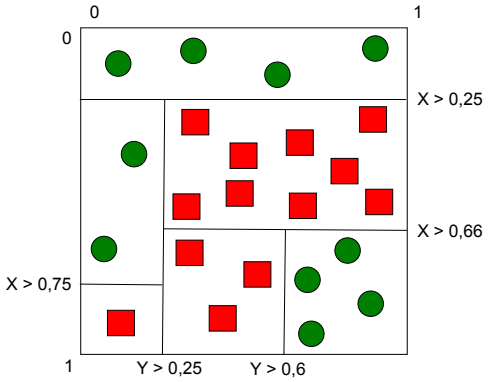

Figura 1.3: Clasificación cuadricular realizada por un árbol de decisión con dos atributos (x e y) numéricos.

nodo. Para ello se usan funciones heurísticas (como por ejemplo, Ganancia de información, GINI, Entropia, DKM)(39; 8; 27). Estos criterios de partición buscan la partición s con el menor $l(s)$, definido de la siguiente forma:

$$l(s) = \sum_{j=1}^{n} p_j \cdot f(p_j^1, p_j^2, \ldots, p_j^c)$$

donde n es el número de nodos hijos de la partición, p_j es la probabilidad de ´´caer" en el nodo j, p_j^i es la proporción de elementos de la clase i en el nodo j y C es el número total de clases. Bajo esta fórmula general, cada criterio de partición implementa una función f distinta, denominada función de impureza (mide la homogeneidad de los nodos), y, por tanto, la función $l(s)$ calcula la media ponderada (dependiendo de la cardinalidad de cada hijo) de la impureza de los hijos en una partición.

Algunos de los criterios de particion más utilizados se estudiaran a continuación.

1.5.2.1. Entropia

Sea $P = p_1, p_2, \ldots, p_n$ una distribución de probabilidad. Llamamos *entropía b-aria* de la distribución P a

$$H_b(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i log_b p_i$$

Donde la misma para $b = 2$

$$H_2(p, 1-p) = p log_2 \frac{1}{p} + (1-p) log_2 \frac{1}{1-p}$$

es muy común y nos referimos a ella como *la* función de entropía y se denota por $H(p)$, cuya gráfica es la siguiente:

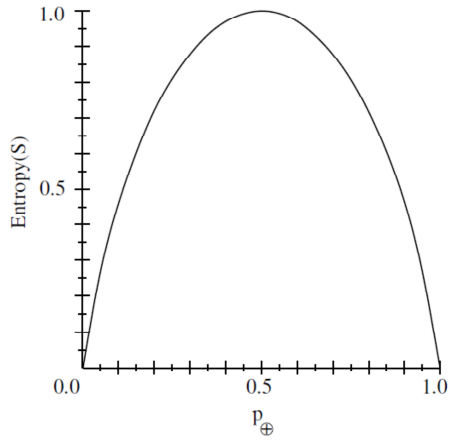

Figura 1.4: Gráfica de la función de entropía

Podemos ver a continuación un ejemplo general de cálculo de la entropía de una distribución, donde D es un conjunto de entrenamiento, P y N son, respectivamente, la cantidad de ejemplos positivos y negativos en D y T es el número total de ejemplos de D:

$$H_2(\tfrac{P}{T}, \tfrac{N}{T}) = \begin{cases} 0 & \text{si } N * P = 0 \\ -\frac{P}{T} log_2 \frac{P}{T} - \frac{N}{T} log_2 \frac{N}{T} & \text{si } N * P \neq 0 \end{cases}$$

1.5.2.2. Ganancia de Información

La $Ganancia(D, A)$ es la estimación de la reducción de la entropía en el conjunto de entrenamiento D si tomamos el atributo A para hacer la partición según sus valores.

$$Ganancia(S, A) \equiv Entropia(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropia(S_v)$$

donde $Entropia(S)$ es la funcioón de entropia vista anteriormente, $Valores(A)$ es el conjunto de valores del atributo A, y S_v es el subconjunto de S formado por aquellas instancias que en el atributo A tiene el valor v.

Siguiendo con el ejemplo de la construcción del árbol de la figura 1.2, supongamos que tenemos un conjunto S de entrenamiento con 14 ejemplos, 9 de ellos son ejemplos positivos (clase S) y 5 negativos (clase No) que denotaremos como ([9$\oplus$,5$\ominus$]), El atributo *Viento* puede tomar los valores *Débil* y *Fuerte*, y la distribución de ejemplos positivos y negativos según los valores de *Viento* son:

	$\oplus$	$\ominus$
Débil	6	2
Fuerte	3	3

La Ganancia de información que obtenemos si clasificamos los 14 ejemplos según el atributo *Viento* es:

$$Ganancia(S, Viento) = Entropia(S) - \sum_{v \in Valores(A)} \frac{S_v}{S} Entropia(S_v)$$

$$= Entropia(S) - \frac{8}{14} Entropia(S_{Débil}) - \frac{6}{14} Entropia(S_{Fuerte})$$

$$= 0,940 - \frac{8}{14} 0,811 - \frac{6}{14} 1,00$$

$$= 0,048$$

Si realizamos los mismo cálculos para el resto de atributos:

$$Ganancia(S, Cielo) = 0,246$$

$$Ganancia(S, Humedad) = 0,151$$

$$Ganancia(S, Temperatura) = 0,029$$

Vemos que el atributo con el que tenemos mayor Ganancia de información es *Cielo*, luego, seleccionamos éste como atributo de partición para el nodo raíz, creando para cada uno de sus posibles valores *(Soleado, Lluvioso, Lluvioso)* una rama, repitiendo este proceso en los nodos en los que los ejemplos no estén perfectamente clasificados. La Figura 1.5 muestra el árbol de decisión parcial obtenido tras este primer paso.

1.5.3. Selección de Umbral en Atributos Numéricos

Asumiendo que hemos seleccionado un atributo numérico A para un nodo que contiene un conjunto S de n ejemplos. Sea $d_1, \ldots, d_n$,los ejemplos de S ordenados de acuerdo al valor de su atributo, A tales que $a_i \leq a_{i+1}$. Cada par de datos adyacentes d_i, d_{i+1} sugiere un valor potencial de umbral $T = (a_i + a_{i+1})/2$ para crear un punto de corte y generar así la correspondiente

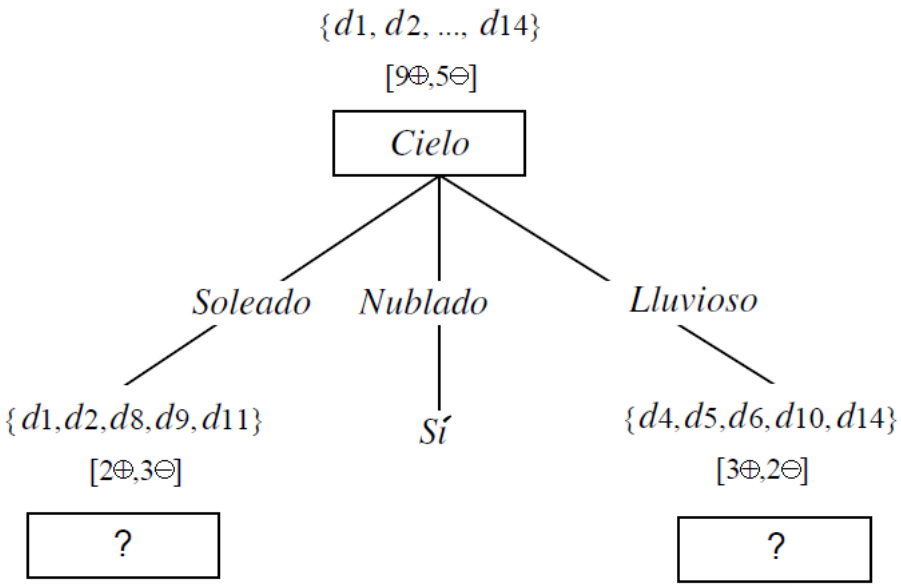

Figura 1.5: Árbol de decisión parcial

partición binaria para le atributo A. Se ha demostrado en la literatura (14) que solo los puntos limítrofes de clase, es decir, aquellos puntos definidos por datos pertenecientes a diferentes clases, pueden ser puntos de corte que obtengan el mayor valor de ganancia de información. Esto implica que si d_i y d_{i+1} pertenecen a la misma clase, un punto de corte entre ellos no conducirá a la partición con mayor valor de ganancia de información. Por lo tanto, podemos generar un pequeño conjunto de candidatos para ser puntos de corte a partir de los puntos limítrofes de clase. Sea c el número de clases, y sea $p(j, S)$ la proporción de ejemplos en S que pertenecen a la clase j. La incertidumbre residual en clasificación es expresada como la entropía de clase:

$$Entropia(S) = -\sum_{j=1}^{c} p(c_j, S) log(p(c_j, S)),$$

Después del particionamiento del conjunto de ejemplos S en dos subconjuntos S_1 y S_2 mediante un valor umbral T, el valor de entropía de clase es expresado mediante la media ponderada del valor de sus entropías de clase respectivas:

$$Entropia(A, T; S) = \frac{n_1}{n} E(S_1) + \frac{n_2}{n} E(S_2),$$

$$Entropia(S_1) = -\sum_{j=1}^{c} p(j, S_1) log(p(j, S_1)),$$

$$Entropia(S_2) = -\sum_{j=1}^{c} p(j, S_2) log(p(j, S_2)),$$

Donde $n_1 = |S_1|$, $n_2 = |S_2|$, $n = |S|$ y $p(j, S_1)$ y $p(j, S_2)$ son la proporción de ejemplos de la clase j en S_1 y S_2 respectivamente. El punto de corte cuya $Entropia(A, T; S)$ es mínimo entre todos los candidatos a punto de corte del atributo A será seleccionado para generar dos nodos sucesores. En cada nodo sucesor, la selección de atributo y la correspondiente discretizacion es llevada a a cabo con los ejemplos de las particiones. El proceso anterior se repite de forma recursiva hasta cumplir algún criterio de parada determinado.

1.5.4. Criterio de parada

El criterio de parada usado en el último punto del algoritmo establece las condiciones bajo las que se debe detener la construcción del árbol de decisión. Estas condiciones comprenden los siguientes 3 supuestos:

1. cuando todos los ejemplos que queden pertenezcan a la misma clase (se añade una hoja al árbol con la etiqueta de clase),

2. cuando no queden atributos con diferentes valores por los que ramificar (se añade una hoja etiquetada con la clase más frecuente en el nodo),

3. o cuando no nos queden más datos para clasificar.

Además pueden existir otros criterios adicionales que se pueden aplicar durante la construcción del arbol o una vez este ha sido ya construido, como por ejemlo la poda (ya sea mediante preproceso o postproceso). Podar un nodo de un árbol de decisión consiste en eliminar un subárbol anidado en ese nodo transformándolo en una hoja y asignándole la clasificación más común de los ejemplos de entrenamiento considerados en ese nodo.

Capítulo 2

Newton Trees

Lo que sabemos es una gota de agua; lo que ignoramos es el océano.

Sir Isaac Newton (1643-1727)

Resumen: En este capítulo describimos un nuevo método de aprendizaje de árboles de decisión estocásticos, denominados Newton Trees, basado en el símil de la atracción gravitatoria usando prototipos y distancia.

2.1. Variantes de los Árboles de Decisión

Como ya se ha comentado en el capítulo anterior, un árbol de decisión es una función $d : E \rightarrow 1..C$ donde E un conjunto de ejemplos no etiquetados. Para clasificar un nuevo ejemplo, se recorre el árbol desde la raíz hasta alcanzar una hoja de acuerdo a las condiciones del arbol y los valores del atributo del ejemplo a etiquetar y se le asigna la clase de las instancias en dicha hoja. A su vez, la construcción de los árboles de decisión se hace de forma recursiva y descendente (desde la raiz a las hojas), por lo que se emplea el acrónimo TDIDT (*Top-Down Induction on Decision Trees*) para referirse a la familia completa de algoritmos de este tipo.

La familia de algoritmos TDIDT abarca desde algoritmos ya clásicos de Inteligencia Artificial (IA) como CLS (*Concept Learning System*), ID3 (39), C4.5 (39) o CART (8)(Classification And Regression Trees) hasta algoritmos optimizados como SLIQ (1) o SPRINT (23), dos algoritmos desarrollados en el *IBM Almaden Research Center* que se usan en Data Mining.

A su vez, los algoritmos de construcción de árboles de decisión se pueden clasificar de acuerdo al tipo de datos con los que pueden tratar. Por ejemplo el ID3 sólo acepta datos de tipo nominal, algoritmos como CART y C4.5

también pueden manejar datos numéricos. A partir de éstos se han introducido numerosas variantes que permiten tratar con tipos más complejos como los datos estructurados de primer orden. Ejemplos de este tipo de algoritmos son FOIL (40), TILDE (7), MRDT (5), etc.

Sin embargo, a pesar que los árboles de decisión son una técnica de aprendizaje ampliamente usada en la práctica ya que con ella se obtienen buenos resultados en cuanto a la tasa de acierto en la clasificación (*accuracy*), los árboles de decisión presentan los siguientes desventajas:

- sólo pueden tratar atributos nominales o numéricos,
- tratan de forma diferente los atributos nominales y los numéricos ya que en estos últimos la partición siempre es binaria de la forma $valor \leq T$ y $valor > T$, siendo T un cierto umbral determinado según cierto criterio,(ver secciones 1.5.1 y 1.5.2),
- en problemas reales es habitual tener atributos numéricos, pero, tal y como hemos indicado en el punto anterior, la forma tan rígida en la que se decide el resultado del test (y por lo tanto de la clasificación) parece inapropiada para instancias cuyos valores para dicho atributo numérico son próximos a T,
- en presencia de ruido (lo cual es habitual en datos reales) la condición de seguir expandiendo el árbol hasta tener hojas puras es demasiado estricta y puede conducir a árboles grandes y complejos, por lo que se ha propuesto aplicar técnicas de poda que permiten no expandir un nodo aunque éste no sea puro,
- cuando la instancia a clasificar tiene valores desconocidos para uno o más atributos, si un nodo en el árbol hace un test sobre este atributo, no se puede determinar cuál es la rama adecuada.

Los últimos tres problemas sugieren que cada nuevo caso a clasificar, en lugar de corresponderle un único camino en el árbol, le corresponden un conjunto de caminos posibles conducentes a un conjunto de hojas que contienen posiblemente ejemplos de diferentes clases. Así surgen los árboles de decisión probabilísticos (*Probabilistic Decision Trees, PET*) (17; 35), los cuales difieren de los árboles de decisión tradicionales en que estiman en cada hoja una probabilidad de pertenencia a cada clase.

Recientemente, en (13) se ha propuesto un método alternativo para la construcción de árboles de decisión basados en distancias llamado DBDT (*Distance-Based Decision Trees*) con un único criterio de partición independientemente del tipo de los atributos. La idea básica es la siguiente: una vez seleccionado el atributo, se computa un prototipo por clase y se calculan las particiones en función de la distancia entre el valor de dicho atributo en la

instancia y en el prototipo. Por tanto, este método nos devuelve un árbol cuyos nodos en lugar de ser tests sobre los atributos son prototipos por atributo y clase. De esta forma, este algoritmo no sólo trata de forma uniforme tanto a los atributos nominales como a los numéricos sino que también puede aplicarse a atributos de cualquier tipo, siempre y cuando se haya definido una función de distancia sobre los valores del tipo.

Aunando las ventajas del DBDT y de los PET's, en este trabajo se presenta una extensión del DBDT para computar en los nodos del árbol probabilidades de pertenencia a cada clase. Hemos llamado a este nuevo método *Newton Trees*, se trata de un metodo de inducción de arboles de estimación estocástica de probabilidades. La principal diferencia con los PET's es que las probabilidades se calculan a partir de la distancia y se propagan desde las hojas hasta la raíz del árbol. Así, para clasificar una nueva instancia, se recorre el árbol desde la raíz y se van calculando para cada nodo la probabilidad por clase como la agregación de las probabilidades de sus hijos, y así sucesivamente.

En los próximos apartados se describiran más detalladamente estos 3 métodos.

2.2. Árboles Probabilísticos

Tradicionalmente, los árboles de decisión clásicos eran suficiente para la mayoría de problemas de clasificación; sin embargo, actualmente se requiere algo más de información como probabilidades, algún tipo de fiabilidad o datos numéricos sobre la calidad de cada clasificación. En otras palabras, no sólo queremos que el modelo prediga un valor de clase para cada ejemplo, sino que también que pueda dar una estimación de la fiabilidad de cada predicción.

Los clasificadores de este tipo suelen ser llamados *soft classifiers* y son muy útiles en diversos escenarios como en la combinación de clasificadores, el aprendizaje sensible al coste y en aplicaciones de seguridad crítica. El uso más general de un soft classifier es la de un estimador de probabilidades, es decir, un modelo que estima la probabilidad $p_i(e)$ de pertenencia a la clase $1 \leq i < c$ para cada ejemplo $e \in E$.

Un árbol de decisión entrenado puede ser fácilmente adaptado para ser un estimador de probabilidades mediante el uso de las frecuencias absolutas de clase de cada hoja del árbol. Por ejemplo, si un nodo tiene la siguientes frecuencias absolutas $n_1, n_2, \ldots, n_c$ (obtenidas del conjunto de entrenamiento), las probabilidades estimadas para ese nodo se puede derivar como $p_i = n_i / \sum n_i$. Cada nuevo ejemplo que caiga en una hoja tendra estas probabilidades de clase estimadas. Este tipo de árboles se denominan Probability Estimation Trees (PETs)(17; 35). A pesar de la sencilla conversión de un árbol de decisión tradicional en un PET, las probabilidades estimadas

son muy pobres con respecto a otros estimadores de probabilidad.

Algunos trabajos recientes han cambiado esta situación. Provost y Domingos (35) han mejorado la calidad de los PET reevaluando algunas técnicas clásicas en el aprendizaje de los árboles de decisión. En particular, descubrieron que el suavizado de la frecuencia en las probabilidades de las hojas, por ejemplo con la aplicación de la corrección de Laplace [1], mejora significativamente las estimaciones, especialmente cuando se usan para clasificación. De modo similar, en (17) se mejoraban las probabilidades usando una ponderación de todas las probabilidades calculadas a lo largo de cada rama del árbol, usando lo que se llamó *m-branch smoothing*.

2.3. Arboles de Decisión basados en Distancias

Una de los principales desventajas de la mayoría de métodos de árboles de decisión clásicos es la heterogeneidad en el manejo de los distintos tipos de datos (numéricos, nominales, estructurados,. . .). Una forma de solucionar esto es considerar diferentes tipos de partición dependiendo del tipo de datos a tratar (ésta es la forma en que trabaja el C4.5). Si extendemos esta filosofía hacia otros tipos de datos como listas, multiconjuntos, grafos, etc. se requerirá un gran número de distintos tipos de partición para tratar con todos los tipos de datos. Esto repercutirá en una menor flexibilidad y homogeneidad del algoritmo.

Para evitar este escenario se han investigado las ventajas de utilizar distancias y funciones de similitud para diseñar árboles de decisión más flexibles. En este contexto se desarrolló el algoritmo DBDT que permite tratar con cualquier tipo de datos en el cual se puedan definir distancias (virtualmente sobre cualquier tipo de datos se pueden definir distancias). Es importante recalcar que, como consecuencia de esto, este método retorna un árbol de prototipos de clase donde cada prototipo de clase representa un nodo en el árbol. Esto significa que un ejemplo caerá en un nodo dado dependiendo de la proximidad a un prototipo de clase. La figura 2.1 muestra un ejemplo de un árbol de decisión tradicional y el correspondiente árbol basado en distancias construido para una evidencia con un atributo numerico (x_1),otro nominal (x_2) y dos clases (YES y NO). En el caso del atributo numérico x_1 se computan dos prototipos (como se explicará más adelante) 0 y 6 y para el atributo nominal x_2 se computan tres prototipos YES, NO y $UNKNOWN$.

Para la construcción del arbol basado en distancias se han usado las siguientes funciones:

$$d_{num}(x, y) = |x - y|,\ d_{nom}(x, y) = 1 - \delta(x, y)$$

[1] Suavizado de la frecuencia en las probabilidades de las hojas con un $\epsilon > 0$

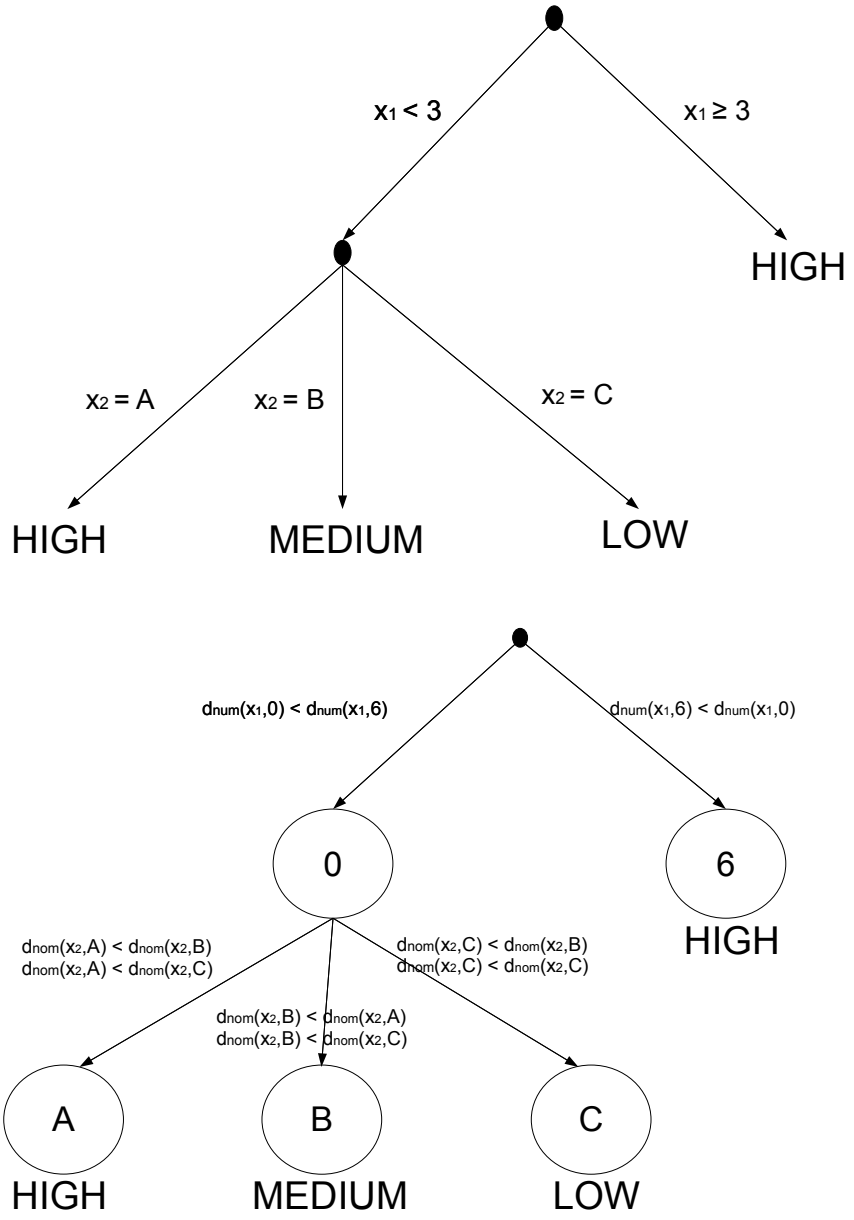

Figura 2.1: Comparación de un árbol de decisión tradicional (arriba) con un árbol de decisión basado en distancias (abajo)

y

$$\delta(x,y) = \begin{cases} 1 & \text{s } x \neq y \\ 0 & \text{en otro caso} \end{cases}$$

Básicamente el algoritmo DBDT se basa en la utilización de distancias de una forma similar a como lo hace el algoritmo *Centre Spliting* (42), el cual divide el espacio métrico en regiones cada una de las cuales está representada por un punto llamado centro y en la que cada ejemplo se asocia con su centro más cercano recalculándose el mismo hasta que se eliminan las zonas impuras. A diferencia del *Centre Spliting*, que trata los ejemplos como un todo, en el DBDT las particiones se realizan teniendo en cuenta únicamente los valores de un atributo, es decir, los prototipos o centroides se computan considerando sólo los valores de un atributo. El atributo será seleccionado usando una función heurística (Gain Ratio, GINI, index, etc.) como ya se ha explicado en el punto anterior. Finalmente esta aproximación asocia a cada atributo una función de distancia tal que la partición se lleva a cabo considerando las distancias y la distribución de clase del atributo seleccionado. Una descripción más detallada del algoritmo puede encontrarse en (13).

En resumen, como se ve en la Figura 2.1, la única diferencia entre los métodos de cálculo de arboles de decisión clásicos y el algoritmo DBDT es la forma de expresar las condiciones; en el primer caso éstas se expresan en términos de un test sobre el valor del atributo y el el segundo caso en términos de distancias. El tratamiento para un ejemplo no etiquetado es el mismo en un caso y en otro, la forma de calcular la clase sigue la manera tradicional, siguiendo las condiciones desde la raiz hasta una hoja. En la figura anterior, el ejemplo $x_1 = 2$, $x_2 = B$ es clasificado en ambos casos como NO

2.4. Árboles de Estimación Estocástica de Probabilidades: Newton Trees

En (31), se presenta una nueva técnica de inducción de Arboles de Estimación Estocástica de Probabilidades, denominados Newton Trees, que aúna las ventajas de los árboles probabilísticos y los árboles de decisión basados en distancias (13). En los Newton Trees, las particiones se construyen mediante el uso de prototipos (medioides obtenidos a partir de los distintos valores del atributo seleccionado para la partición) que funcionan como *atractores* de los ejemplos siguiendo una ley cuadratica inversa usando la distancia con el prototipo y su masa, similar a otros enfoques gravitatorios de aprendizaje automático (46)(10)(25)(20)(34). La diferencia con estos otros enfoques es que, en lugar de derivar un punto de corte, se obtiene una probabilidad sabiendo que la probabilidad y la atracción siguen una relación inversamente

proporcional, (6).

Algunas de las ventajas de utilizar este tipo de árboles son:

- Tratamiento homogéneo de cualquier tipo de datos (nominales, numéricos u ordinales) usando la noción de distancia de forma univariante. Para datos numéricos se puede usar el valor absoluto de la diferencia, y, en el caso de datos nominales u ordinales, la función identidad.
- Construcción, uso y representación del árbol basados en el principio de atracción. Las probabilidades se derivan también a partir de dicho principio.
- Uso de prototipos, obtenidos a partir del conjunto de valores de los atributos, y no de centroides, para el correcto tratamiento de datasets continuos y discretos. Para los datasets con valores continuos, solo se construye un grupo por atributo y clase y no un punto de corte entre cada pareja de valores adyacentes.
- Representación gráfica del árbol sencilla de interpretar.
- Los árboles son univariantes, pero sus particiones no son necesariamente paralelas a los ejes. Esto implica que las particiones pueden crear límites más expresivos que los obtenidos por los árboles de decisión clásicos, sin tener que ser multivariantes.
- Es posible convertir estos árboles estocásticos en arboles *crisp*[2], los cuales pueden ser descritos en forma de reglas de forma similar a los arboles de decisión tradicionales.

2.4.1. Notación

En esta sección se presentan algunas definiciones necesarias, a fin de establecer las bases para el resto del capitulo E denota el conjunto de todos los ejemplos posibles sin etiquetar representados como $e = \langle e_1, e_2, \ldots, e_m \rangle$ siendo m el número de atributos. Los nombres de los atributos se denotan por $\langle x_1, x_2, \ldots, x_m \rangle$. Un conjunto de datos etiquetados D es un conjunto de pares $\langle e, i \rangle$ donde $e \in E$ e $i \in C$, donde C es el conjunto de las clases. El número de clases, $|C|$, se denota por c.

Se define un estimador de la probabilidad como un conjunto c de funciones $p_{i \in C} : E \to R$ tal que $\forall i \in C, e \in E : 0 \leq p_i(e) \leq 1$ y $\forall e \in E : \sum p_i(e) = 1$. Los árboles de decisión están formados por nodos, particiones y condiciones. Una *condición* es cualquier función *Booleana* $g : E \to \{verdadero, falso\}$. Una *partición* o *división* es un conjunto de s condiciones $g_k : 1 \leq k \leq s$.

[2]Proporcionan una salida discreta

Dado un nodo ν, $Children(\nu)$ denota el conjunto de sus nodos hijos y $Parent(\nu)$ denota su nodo predecesor. El nodo especial donde $Parents(\nu) = \emptyset$ se llama la *raíz* del árbol.

Después de la etapa de entrenamiento de un arbol de decisión, los ejemplos se han distribuido entre todos los nodos en el árbol, el nodo raíz contiene todos los ejemplos y los nodos sucesores contienen un subconjunto de ejemplos que son coherentes con las condiciones de todos sus predecesores. Por lo tanto, cada nodo tiene un conjunto particular de frecuencias absolutas $n_1, n_2, \ldots, n_c$ por cada clase. La cardinalidad del nodo está dada por $\sum n_i$. Generalmente, en un árbol de clasificación las hojas se etiquetan con la clase de mayor frecuencia.

2.4.2. Trabajos Relacionados

Los árboles de estimación de probabilidad existentes proporcionan como salida una probabilidad, pero no son necesariamente de naturaleza probabilística. La primera cuestión es que suelen utilizar la filosofía ´´divide y vencerás" para la construcción del árbol, pero también para hacer las predicciones. Visto con un ejemplo, una secuencia de decisiones dará lugar a una hoja del árbol donde se devuelve un valor (una clase en los árboles de clasificación, un número en los árboles de regresión, una probabilidad en los PETs, etc.). El resto de la información del árbol se desprecia. Sin embargo, en la teoría de la decisión, esta visión nítida de las decisiones es difícil, ya que cada decisión puede tener una probabilidad asociada, y la probabilidad global debe ser calculada teniendo en cuenta toda la estructura del árbol. Este tipo de árboles son frecuentemente (pero no siempre) llamados árboles de decisión estocástica (por ejemplo, (22)). En el aprendizaje de árboles de decisión, el uso estocástico de un árbol de decisión es menos común que el uso de árboles individuales. *Option Trees* (9)(29) y *Shared multitrees* (18) son otras extensiones de árboles de decisión los cuales consideran distintas alternativas de particiones en cada punto del árbol, pero al final no es un solo el árbol que se aprende, sino varios, de forma similar a la aproximación conocida como conjuntos de clasificadores (*ensembles*) (12). Más estrechamente relacionado con la noción de un árbol de decisión estocástico es la agregación probabilística o bayesiana para decidir cómo una secuencia de decisiones ha de ser seguida o podada (por ejemplo (9), (17)). Sin embargo, hasta donde sabemos, sólo (30) utiliza toda la información de un árbol con el fin de obtener mejores estimaciones de probabilidad, aparte de algunos árboles de decisión difusos (por ejemplo, (44))

Una segunda cuestión es que el uso de todos los caminos en el árbol se puede hacer de tal manera que las probabilidades del árbol son independientes de la instancia que está siendo procesada. De hecho, éste ha sido el enfoque en (30), utilizando un parámetro de ad-hoc que se utiliza para determinar la probabilidad de cada hijo en una partición. Los enfoques más

recientes, (3) y (4) han hecho que la probabilidad dependa de la proximidad al punto de corte para el atributo, utilizando *Kernel Density Estimates*. Por ejemplo, dada una condición $X > 3$, se supone que la probabilidad de alcanzar la rama debe ser mayor que la máxima separación del punto de corte 3. Sin embargo, estos planteamientos todavía construyen el árbol de forma clásica, pudiendo descartar las cardinalidades (es decir, las masas) al utilizar el árbol para obtener las probabilidades. En otras palabras, un árbol puede construirse mediante un algoritmo clásico (como C4.5 (37) o CART (8)) y su interpretación probabilística o estocástica puede ser incoherente con la forma en que se ha construido el árbol de decisión. Por ejemplo, (4) utiliza una forma de estimar las probabilidades que es completamente diferente a la forma comprensible en que el árbol es interpretado por los seres humanos. De hecho, esto es lo que sucede con muchas de las extensiones multivariantes del aprendizaje de árboles de decisión. Éstos se limitan a preservar en las hojas la inteligibilidad y la esencia de un árbol de decisión. Por ejemplo, (28) utiliza *Naive Bayes* en cada hoja, pero no en la construcción o en los nodos internos. En (47), la filosofía *Naive Bayes* se propaga por el árbol entero, lo que hace difícil extraer conocimiento comprensible de los árboles, ya que se convierte en un *Naive Bayes* jerárquico en vez de un árbol de decisión.

Una tercera cuestión es cómo se tratan los diferentes tipos de datos. Muchos de los enfoques anteriores sólo manejan atributos numéricos (3; 4) o sólo tratan con atributos nominales. Cuando manejan ambos tipos, los árboles utilizan modos específicos para el tratamiento de atributos numéricos (con puntos de corte) y de atributos nominales (con igualdades), como C4.5 (37) o CART (8) los cuales tienen métodos muy refinados (y en ocasiones ad-hoc) para obtener los puntos de corte. Otros árboles de decisión clásicos, como QUEST (2) y CHAID (41) repiten este esquema: mientras que en la construcción del arbol para atributos ordinales o continuos se realizan ensayos F-ANOVA o pruebas de Levene, para los atributos numéricos se realizan pruebas de Chi-cuadrado. Incluso en el caso de los árboles de decisión difusos (que en ocasiones ofrecen una visión más integrada de los atributos nominales y numéricos) no está claro cómo estos se pueden aplicar a problemas en los que algunos atributos son de tipos tales como como intervalos, secuencias, series u otro tipo de datos estructurados.

Una vez vistos todos los enfoques relacionados, proponemos a contunuación un nuevo método de inducción de árboles el cual ha sido diseñado desde cero con los objetivos de ser estocástico en su naturaleza, general y flexible en la forma en la que trata los atributos, e inteligible.

2.4.3. Árboles de Estimación Estocástica de Probabilidades basados en Distancias

En esta sección definimos formalmente la técnica de aprendizaje de Árboles de Estimación Estocástica de Probabilidades conocidos como Newton

Trees.

2.4.3.1. Particiones Gravitacionales

En la creación de las distintas particiones, los árboles de decisión tradicionales suelen generar las condiciones que luego son evaluadas para ver qué tan bien se separan las clases. En lugar de eso, se propone definir un nodo/-cluster por clase y luego tratar de encontrar la caracterización de cada nodo en términos de un atributo a la vez (univariante).

Siguiendo esta idea, una primera aproximación es utilizar un Kernel de Estimacion de Densidad (43) con el fin de obtener una función de densidad de probabilidad (fdp) de los ejemplos pertenecientes a cada clase. Sin embargo, muchas de estas técnicas crean una fdp paramétrica o compuesta que hace que las particiones sean ininteligibles, aparte del riesgo de sobreajuste. Otro enfoque (relacionado) es obtener un prototipo para cada nodo y, a continuación, obtener una probabilidad de los prototipos. Para tratar adecuadamente los tipos de datos discretos, utilizamos un medioide (el elemento de cada cluster cuya distancia media al resto de elementos es mínima). Si generamos prototipos, una posibilidad para obtener probabilidades a partir de ellos es suponer una cierta distribución de probabilidad. Por ejemplo, si tenemos en cuenta una distribución normal para cada nodo con centro en el prototipo y con una desviación estándar igual a la media de las distancias de los elementos del nodo, tenemos un fdp. La figura 2.2 muestra el fdp usando una distribución gaussiana con centros 3 y 8, con desviaciones estándar de 1 y 3,5 (respectivamente) y con masas 20 y 100 (respectivamente). Esto puede convertirse en una probabilidad mediante una mera normalización, como se muestra en la figura 2.3.

El problema del enfoque anterior es que cuando las masas son demasiado dispares, una distribución puede cubrir la otra, originando una partición simple (e inútil) donde todos los elementos van a un prototipo. Uno de los criterios de evitar esto es dar una importancia adicional a la distancia, de tal manera que a una distancia de 0, la probabilidad es siempre 1. Una manera sencilla de hacer esto es emplear una ley cuadrática inversa que relacione ambos conceptos, como en la ley de Gravitación Universal: ´´La fuerza de atracción gravitacional entre dos masas puntuales es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de su distancia de separación. La fuerza siempre es atractiva y actúa a lo largo de la línea que los une". La ley cuadrática inversa se aplica generalmente cuando alguna fuerza, energía u otra cantidad conservada se irradia hacia el exterior de forma radial a partir de una fuente.

Siguiendo esta idea, se define la siguiente función de *atracción* entre un elemento e de masa m_e (se supone $m_e = 1$) y un prototipo π la masa m_π separados por una distancia $d(e, \pi) = d$:

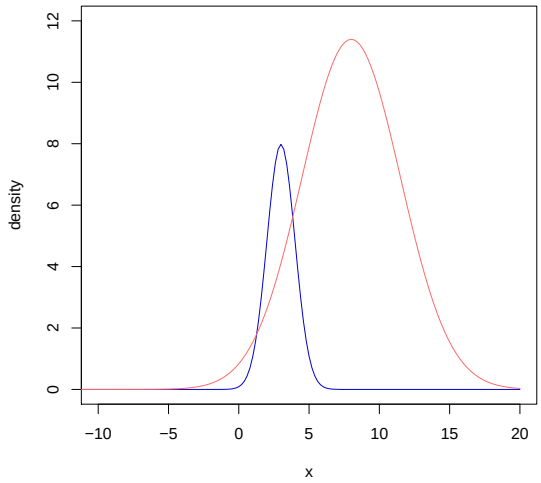

Figura 2.2: Dos distribuciones normales con centros 3 y 8, con desviaciones estándar 1 y 3,5 (respectivamente) y con masas 20 y 100 (respectivamente)

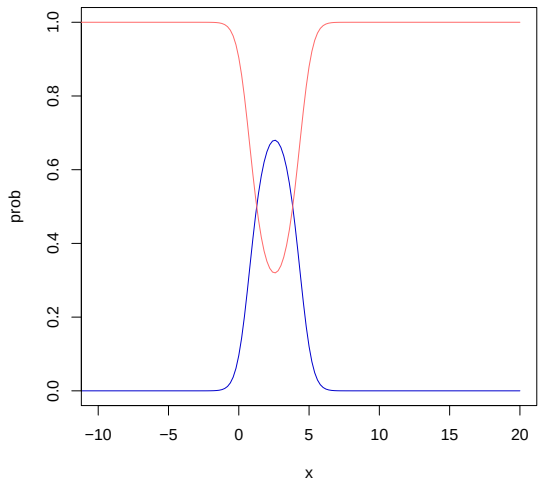

Figura 2.3: Probabilidades derivadas a partir de las Gaussianas de la Figura 2.2

$$atracción(e, \pi) = \frac{m_e m_\pi}{d(e, \pi)^2} = \frac{m_\pi}{d(e, \pi)^2}$$

Estamos interesados en la obtención de probabilidades teniendo en cuenta de esta función de atracción. La Figura 2.4 muestra la función de atracción con los mismos parámetros que en la figura 2.2 (cabe recalcar que la desviación estándar ya no se usa). De nuevo, esto se puede convertir en meras probabilidades por la normalización, como se muestra en la Figura 2.5.

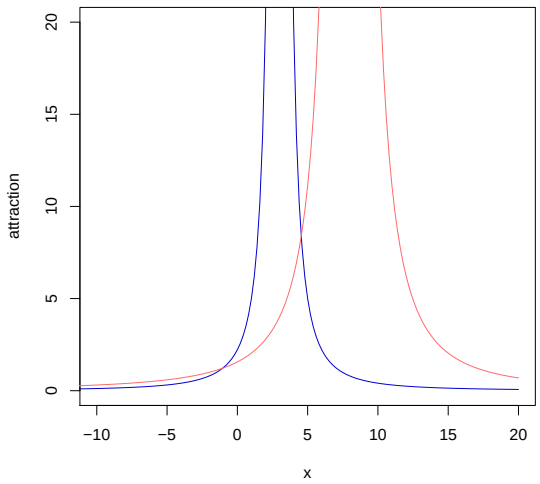

Figura 2.4: Dos distribuciones normales con centros 3 y 8 y con masas 20 y 100 (respectivamente)

Una propiedad interesante es que cuando la distancia tiende a infinito las probabilidades tienden a converger a la proporcion entre las masas. Por ejemplo, si tenemos dos centros en 3 y 8 (correspondientes a las dos clases), y 8 tiene una masa mucho mayor (como en el ejemplo de la figura 2.4), es fácil ver que la atracción a 8 será mayor que la atracción a 3 para un punto situado en -100. Esto significa que el uso de la atracción típicamente produce una relación no monótona entre la probabilidad y la posición.

Por supuesto, la idea de usar una ley de gravitación en el aprendizaje automatico no es nueva en absoluto, por ejemplo en clustering (46), (20), en la visualización (10) o la clasificación (34). De hecho, el mismo principio de la Ley cuadrática inversa está presente en algunas variantes del *Kernel Density Estimation*, varias técnicas de clasificación como kNN ponderada, donde el peso es un kernel que se define simplemente como la inversa de la distancia, o en algunos otros algoritmos de clusterin. Hasta donde sabemos, su uso para los árboles de decisión es nuevo.

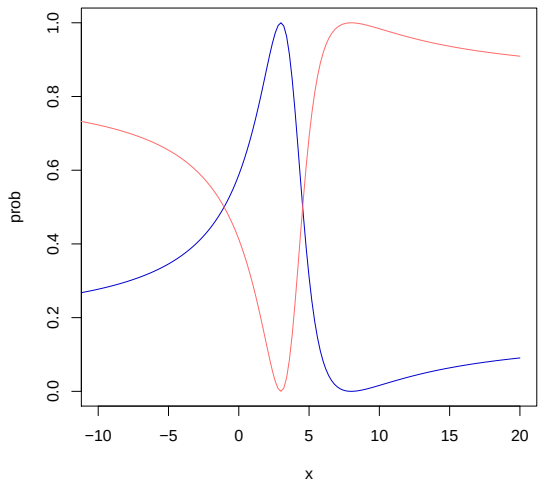

Figura 2.5: Probabilidades derivadas a partir de las Gaussianas de la Figura 2.4

2.4.3.2. Generación del Árbol

Este método está basado en el uso de prototipos y distancias para definir las particiones de una manera jerárquica, de forma semejante al método de particiones de centro (*centre splitting*). Básicamente, el método de particiones de centro consiste en dividir el espacio de entrada en diferentes regiones donde cada región es representada por un centro (que puede coincidir con un ejemplo existente o no). En cada iteración del proceso, se calcula un centro para cada clase diferente presente en el área. Posteriormente, todos los ejemplos se asocian a su centro más cercano. Este proceso se repite hasta que el área es pura. Una de las características especiales de este método es que los ejemplos se gestionan como un conjunto, lo cual se opone a la utilización de particiones univariadas. Esta apreciación nos lleva a proponer una estrategia de inferencia de árboles de decisión cuyas particiones se hacen teniendo en cuenta un atributo cada vez. De esta manera los prototipos se calculan teniendo en cuenta sólo los valores de un atributo, lo que nos permite unir la aproximación de las particiones de centro y las técnicas de aprendizaje de árboles de manera elegante.

La definición detallada del algoritmo puede encontrarse en el apéndice A. Aquí vamos a dar una visión más reducida de su funcionamiento. Básicamente, el algoritmo de generación del árbol trabaja de la siguiente forma: para cada atributo x_r, y para cada clase i, se calcula un prototipo $\pi_{r,i}$ como el valor de dicho atributo con la menor distancia media al resto de elementos de la clase. Una vez terminado el proceso, se selecciona el atributo que se us-

ará en la partición de acuerdo a algún criterio de partición (por ejemplo, *gain ratio* (37)). Posteriormente, la partición procede a asociar cada instancia al prototipo que ejerza sobre ella una mayor atracción, lo que produce clústeres impuros. Nótese que, durante el proceso de partición, aplicamos la función de atracción asumiendo que la masa del nodo es la unidad. Esto es debido al hecho de que la masa total del nodo no es conocida hasta que todas las instancias han sido asociadas a su prototipo correspondiente. Hay que recalcar que las distancias son computadas entre los valores de los atributos y no entre los ejemplos completos. Esta característica es crucial para la eficiencia.

Aunque el cálculo de las distancias es cuadrático en el número de casos, podemos reducirlo mediante el uso de una matriz de distancia para cada atributo (de tamaño de $n_r \times n_r$, donde n_r es el número de valores diferentes del atributo) generada antes de la ejecución del algoritmo. Para los atributos numéricos sólo podemos calcular una media si se utiliza como distancia la diferencia absoluta. Lo más importante es que si tenemos m atributos y n_r valores por atributo, sólo se construyen (y evalúan) $O(m)$ particiones y no $O(n_r \times m)$ (el orden típico de los algoritmos de aprendizaje de árboles de decisión puntos medios para los atributos continuos). Además, es importante tener en cuenta que las distancias se calculan entre los valores de los atributos y no entre los ejemplos (vistos como un todo). Esta característica también es crucial para la eficiencia.

2.4.3.3. Cálculo Estocástico de Probabilidad

En esta sección se va a describir cómo un Newton Tree es usado para estimar probabilidades de una forma estocástica. A partir de ahora , $\overrightarrow{p}(\nu, e) = \langle p_1(\nu, e), \ldots, p_c(\nu, e)\rangle$, denotará el vector de probabilidad del ejemplo e en el nodo ν, donde $p_i(\nu, e)$ denota la probabilidad de que e pertenezca a la clase i en el nodo ν. Con $\hat{p}(\nu, e)$ se denota la probabilidad de que e caiga en el nodo ν (viniendo de su progenitor), que se deriva de la atracción que ν ejerce sobre e, es decir

$$\hat{p}(\nu, e) = \frac{attraction(e, \nu)}{\sum_{\mu \in Children(Parent(\nu))} attraction(e, \mu)}$$

Dado un nuevo ejemplo e y un Newton Tree T, el objetivo es calcular el vector de probabilidad en la raíz de T, $\overrightarrow{p}(root, e)$. La idea básica es computar la probabilidad de caer en cada hoja, calcular el vector de probabilidad en cada hoja y finalmente propagar hacia arriba este vector para obtener en la raíz el vector total de probabilidad $\overrightarrow{p}(root, e)$. Los vectores de probabilidad de las hojas pueden ser obtenidos una vez se ha generado el árbol aplicando Laplace como en (35)(17). Para cada ejemplo, calculamos la probabilidad de escoger cada nodo hijo μ (a partir del nodo donde se encuentra ν) usando la atracción (i.e., $\hat{p}(\mu, e)$). Esta probabilidad se multiplica por el vector de

probabilidad del nodo hijo ($\overrightarrow{p}(\mu, e)$), tal y como se describe en la siguiente definición:

DEFINICIÓN **2.1 Estimación del Vector Estocástico de Probabilidad** *Dado un ejemplo e y un Newton Tree T, el vector de probabilidad $\overrightarrow{p}(root, e)$ en la raíz de T se estima aplicando*

$$\forall \nu \in T : \overrightarrow{p}(\nu, e) = \begin{cases} \sum_{\mu \in Children(\nu)} \hat{p}(\mu, e) \cdot \overrightarrow{p}(\mu, e) & si\ \nu\ no\ es\ Hoja \\ \langle Laplace(1, \nu), \ldots, Laplace(c, \nu) \rangle & si\ \nu\ es\ Hoja \end{cases}$$

donde $Laplace(j, \nu)$ es la corrección de Laplace de la frecuencia de los elementos de la clase j en el nodo ν.

Si la inteligibilidad es un requisito, el cálculo estocástico de probabilidades descrito puede parecer muy críptico para un uso general de estos árboles, pero si atendemos a la representación gráfica (ver Figura 2.6), ésta es altamente comprensible.

En la Figura 2.6 se muestra el Newton Tree generado para el dataset Hepatitis del repositorio UCI (19). Se observa que todas las particiones del árbol son binarias debido a que se trata de un problema con dos clases, DIE o LIVE. Las dos primeras particiones se han realizado sobre atributos numéricos ($PROTIME$ y $ALK\_PHOSPHHATE$) y las 2 últimas sobre dos atributos nominales (SEX y $FATIGUE$). Los nodos del árbol se representan como bolas de tamaño proporcional a la masa del nodo. Por ejemplo, el nodo izquierdo del primer nivel tiene una masa igual a 17, lo cual quiere decir que 17 instancias de entrenamiento han caído en dicho nodo. Las bolas también muestran, en diferentes tonalidades, la proporción de ejemplos de cada clase y el valor del prototipo en el medio de las mismas. También podemos ver las probabilidades suavizadas en las hojas (en las tablas debajo de cada hoja). Para facilitar la comprensión de cómo las distintas probabilidades son derivadas, la Figura 2.7 muestra las distintas probabilidades internas (vector y nodo) para la instancia $\langle PROTIME = 40;\ ALK\_PHOSPHATE = 120;\ SEX = FEMALE;\ FATIGUE = NO \rangle$, cuyo vector de probabilidad a nivel raíz es (0,3248, 0,6752), por lo que esta instancia se clasifica como de clase $LIVE$. Cabe recalcar cómo el árbol ha derivado las probabilidades en la última partición para el atributo $FATIGUE$, centrándose únicamente en las masas de los nodos puesto que la instancia poseía un valor faltante para dicho atributo.

Todos estos elementos gráficos que hemos incluido en la representación de los Newton Trees pueden ayudar a un usuario final a comprender la forma en que las probabilidades se estiman, por lo que los Newton Trees son menos críptico que otros métodos de árboles de estimación de probabilidad, especialmente los que son estocásticos.

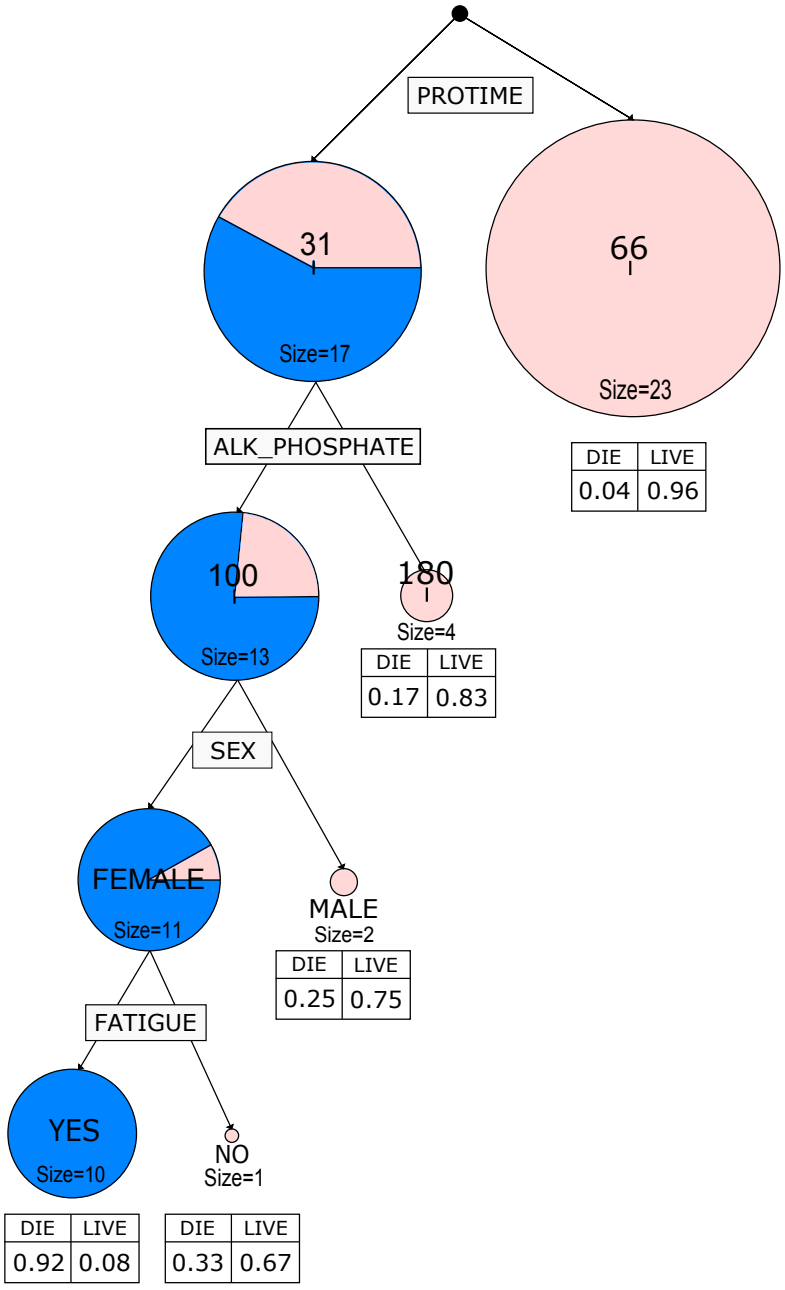

Figura 2.6: Newton Tree a partir del dataset Hepatitis del repositorio UCI

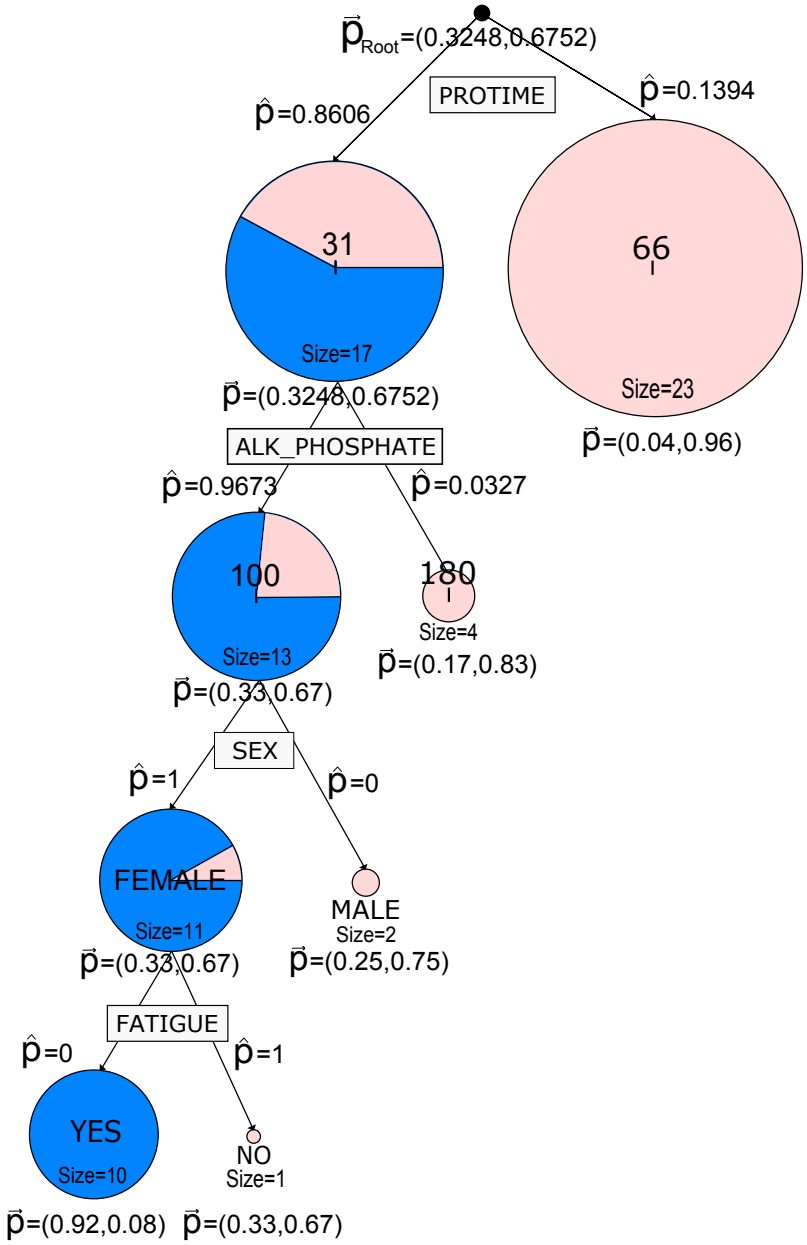

Figura 2.7: Los vectores de probabilidad en los nodos, las probabilidades de los nodos hijos, y el vector de probabilidad total para el ejemplo (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE=NO)

2.4.4. Newton Trees con Valores Faltantes

¿Qué pasa cuando parte de los datos están incompletos como ocurre generalmente con cualquier conjunto de datos de la vida real? Podemos tomar dos caminos posibles ante los datos incompletos: descartar una proporción importante de los datos por incompletos y declarar algunos casos como inclasificables, o adaptar los algoritmos para poder trabajar con atributos con valores faltantes. En la mayoría de los casos, la primera opción es inaceptable. A continuación vamos a ver las aproximaciones habituales en el ámbito de los árboles de decisión.

Quinlan (38) comparó experimentalmente la efectividad de diversas aproximaciones para el tratamiento de valores faltantes en los árboles de decisión y demostró que la combinación de todos los posibles resultados con un valor faltante en el ejemplo de test en la fase de clasificación proporciona una mejor precisión general que otros enfoques. En esta aproximación, cuando se tiene que clasificar un ejemplo e al que le falta el valor del atributo que se está utilizando en la partición, éste se propaga por todas las ramas de forma fragmentada, donde cada fragmento es proporcional al número de instancias de entrenamiento (con valores conocidos para el atributo de partición) que han correspondido con cada rama. Notese que cada fragmento representa en realidad una probabilidad. La fracción de e que se propaga por cada rama es proporcional . En el caso de que e tenga un valor conocido para el atributo de la partición, la propagación de dicho ejemplo se hará de forma habitual. Después de que todas las fracciones de le se han propagado, las distribuciones de clase en las hojas se combinan aritméticamente con las fracciones como coeficientes de ponderación. Finalmente, a e se le asigna la clase con mayor probabilidad. El algoritmo C4.5 adopta esta aproximación.

Podemos ver que esta manera de funcionar del algoritmo C4.5 para valores faltantes es justamente la manera que los Newton Trees tratan cualquier instancia, con la salvedad de que para atributos con valores faltantes habríamos de asumir que las distancias a los prototipos son iguales. Haciendo esta modificación e ignorando lo valores faltantes en el aprendizaje tenemos una adaptación sencilla e inmediata de los Newton Trees para trabajar con datasets con valores faltantes en sus atributos. Nuestro algoritmo omite los valores faltantes en los atributos de las instancias de entrenamiento en cuanto a la selección de los prototipos en la construcción del árbol ya que es imposible que maximicen ningún tipo de criterio de ganancia. En cambio, en el momento de la predicción, establecemos constante (igual a 1) el valor de la distancia (para cualquier tipo de datos) a cualquier prototipo para conseguir que sólo se tenga en cuenta la masa de cada prototipo en el cálculo de su correspondiente probabilidad mediante la formula anterior:

$$\hat{p}(\nu, e) = \frac{m_\nu}{\sum_{\mu \in Children(Parent(\nu))} m_\mu}$$

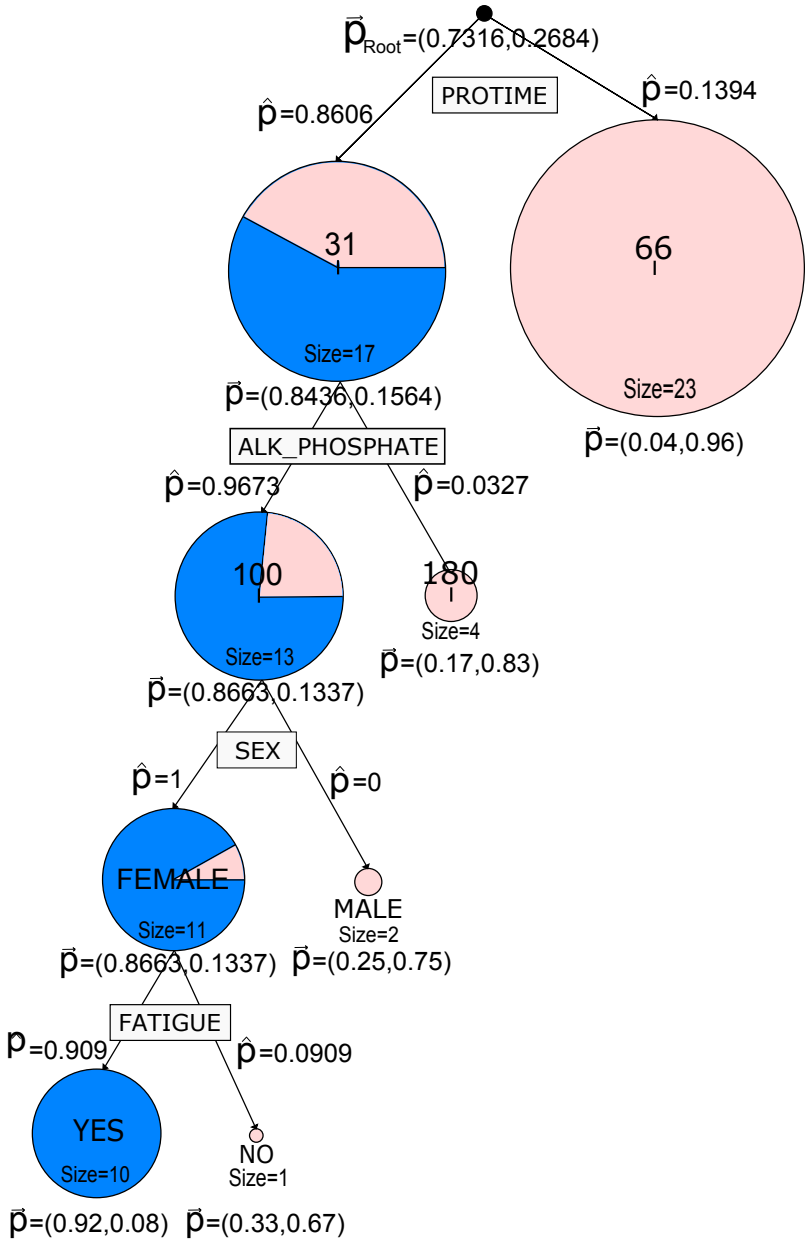

Figura 2.8: Los vectores de probabilidad en los nodos, las probabilidades de los nodos hijos, y el vector de probabilidad total para el ejemplo (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE=UNKNOWN)

Con el fin de clarificar el proceso de clasificación de una instancia con valores faltantes, la figura 2.8 muestra cómo el árbol ha derivado las probabilidades en la última partición para el atributo $FATIGUE$, centrándose únicamente en las masas de los nodos puesto que la instancia poseía un valor faltante para dicho atributo.

2.4.5. Expresividad y Versiones no Estocásticas

La construcción de los Newton Trees concuerda con el aprendizaje de los árboles de decisión clásicos en que ambos se basan en particiones univariantes, como ya se ha visto en secciones anteriores. En el caso de los algoritmos tradicionales, esto significa que las particiones son paralelas a los ejes y dividen el espacio en áreas rectangulares que determinan los puntos del espacio de entrada que pertenecen a cada clase. Por lo tanto, cada región se define por sus límites de clase que se expresan mediante condiciones sobre el valor del atributo de la forma $atributo = valor$, $atributo < valor$ o $atributo > valor$, dependiendo de si el atributo es nominal o numéricos. Sin embargo, las particiones generadas por los Newton Trees no son paralelas a los ejes. Esto se debe al hecho de que los límites de clase se definen estocásticamente y se utiliza la información de todo el árbol. La figura 2.10 (Izquierda) muestra los límites de clase generados por los Newton Trees para el dataset Hepatitis (considerando únicamente los dos niveles superiores). Los ejes representan los atributos utilizados para las dos primeras particiones (como hemos visto en la figura in Figure 2.6). La clase DIE se representa en azul (oscuro) y la clase $LIVE$ en color rosa (claro). Próximas a cada eje hay dos pequeñas bolas que representan los prototipos (del color de su clase mayoritaria), situadas en sus valores correspondientes. Esta capacidad de construir curvas límites puede explicar parte del aumento de rendimiento con respecto a los PETs tradicionales, más aún cuando se evalúa un número mucho menor de particiones por nodo.

Ya se ha dicho que la representación gráfica de los Newton Trees permite a los usuarios tener un control del árbol y explicar sus predicciones. Sin embargo, parece razonable ver si podemos utilizar los árboles de Newton de una manera no estocástica y obtener particiones *crisp* de ellos. La respuesta a esta pregunta es afirmativa. Vamos a mostrar que pueden ser transformados en árboles de decisión *crips* de una manera fácil y directa, por un simple post-procesado del árbol. Para atributos nominales y usando la distancia discreta la transformación es trivial:

DEFINICIÓN **2.2** *Sea x_i un atributo nominal, y $\pi_{i,k}, 1 \leq k \leq s$ los prototipos computados en un nodo. Entonces, la particion es definida de forma* crisp *como un conjunto s de condiciones $x_i = \pi_{i,k}$. Si existen más valores de atributos que prototipos, entonces escogemos el nodo más poblado y lo marcamos como* "all the rest".

Para atributos numéricos, la idea es calcular puntos de corte:

PROPOSICIÓN **2.1** *Dado un atributo numérico x_i, usando la diferencia absoluta como distancia y dados dos prototipos consecutivos $\pi_{i,j}$ y $\pi_{i,k}$, con $\pi_{i,j} < \pi_{i,k}$, los valores para el atributo numérico x_i definidos como*

$$t_1 = a + sqrt\left(\frac{a^2}{4} - \frac{b}{c}\right) \quad t_2 = a - sqrt\left(\frac{a^2}{4} - \frac{b}{c}\right)$$

donde $a = (m_{\pi_{i,k}} \cdot \pi_{i,j} - m_{\pi_{i,j}} \cdot \pi_{i,k})/c$, $b = (m_{\pi_{i,j}} \cdot \pi_{i,k}^2 - (m_{\pi_{i,k}} \cdot \pi_{i,j}^2)$ y $c = (m_{\pi_{i,j}} - m_{\pi_{i,k}})$, satisfacen que, si $t_1 < t_2$[3]*,*

$$\forall e : \begin{cases} si\ e_i < t_1 & entonces \quad e\ cae\ en\ el\ prototipo\ \pi_{i,k} \\ si\ t_1 < e_i \leq t_2 & entonces \quad e\ cae\ en\ el\ prototipo\ \pi_{i,j} \\ si\ t_2 \leq e_i & entonces \quad e\ cae\ en\ el\ prototipo\ \pi_{i,k} \end{cases}$$

Proof. Ya que los prototipos en los Newton Trees ejercen una fuerza de atracción sobre los ejemplos, un ejemplo dado e caerá en el prototipo con la probabilidad más alta. Como hemos visto en la sección 2.4.3.2, la densidad de probabilidad se deriva de las fuerzas de atracción. Dados dos prototipos y sus densidades de probabilidad, los puntos donde las densidades se corten entre sí determinará el intervalo en el que prevalece cada prototipo. Estos puntos se obtienen al igualar las dos funciones de atracción:

$$\frac{m_{\pi_{i,j}}}{(t - \pi_{i,j})^2} = \frac{m_{\pi_{i,k}}}{(t - \pi_{i,k})^2}$$

Resolviendo esta ecuación cuadrática para t tenemos dos soluciones: t_1 y t_2.

La figura 2.9 muestra los puntos de corte para las dos primeras particiones del dataset Hepatitis. Nótese que uno de los puntos de corte esté siempre entre los prototipos y el otro está a la izquierda o a la derecha de uno de estos. De acuerdo con la Proposición 2.1, la primera partición para este ejemplo puede ser expresada por las condiciones:

```
if (-183 < PROTIME) and (PROTIME <= 47.2) then fall into prototype 31 (...)
                                          else fall into prototype 66 (class=LIVE)
```

Ahora, las particiones *crisp* son paralelas a los ejes. La figura 2.10 (Centro) presenta las particiones correspondientes a las dos primeras divisiones del dataset Hepatitis utilizando dos puntos de corte. Podemos observar que el punto de corte situado fuera del espacio entre los prototipos normalmente limita las clases en una región muy escasa, mientras que el punto de corte entre los prototipos define el límite de la clase en una región mucho más densa. Esto sugiere que con sólo utilizar un punto de corte, como presenta la Figura 2.10 (Abajo), se podría obtener una transformación más simple pero aún competitiva.

[3] Si $t_2 < t_1$ la proposicion se mantiene trivialmente intercambiando ambos valores.

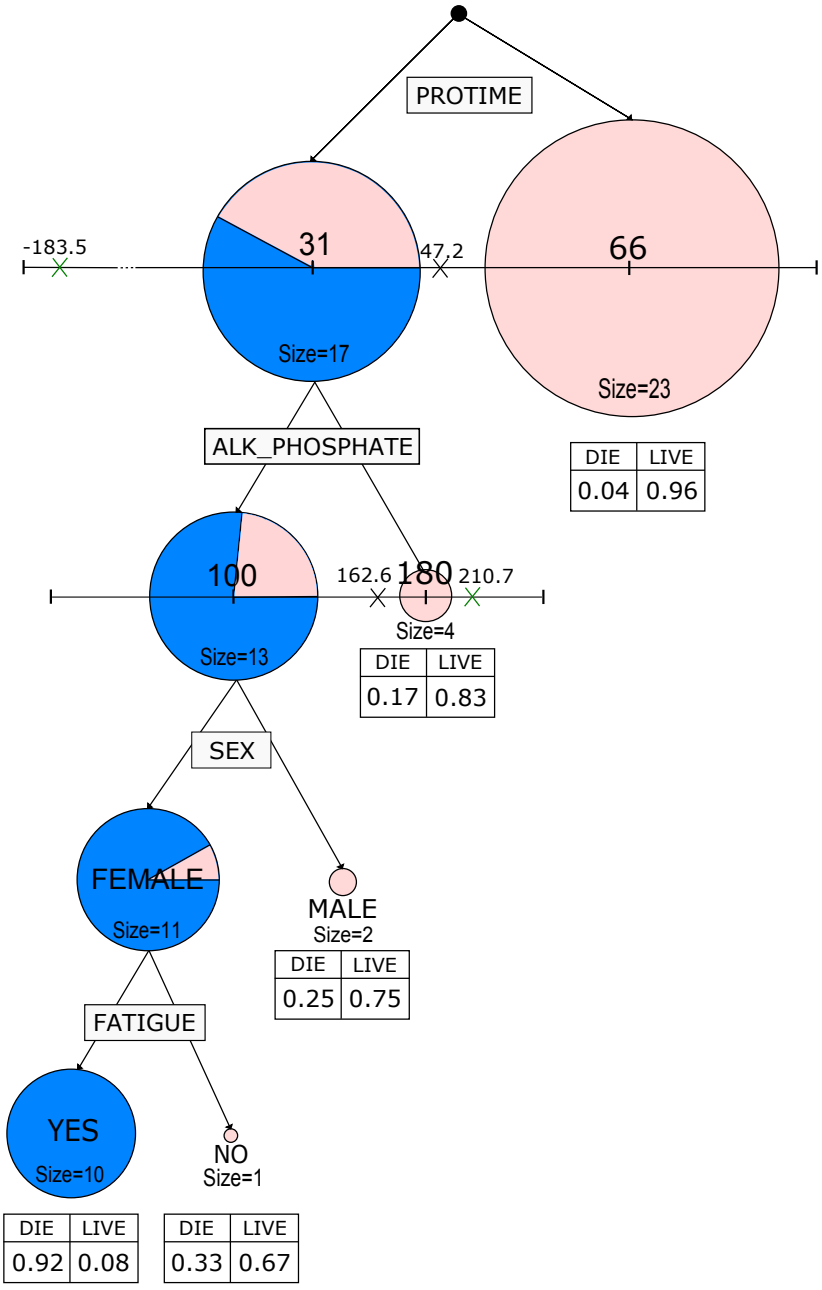

Figura 2.9: Newton Tree con puntos de corte a partir del dataset Hepatitis del repositorio UCI

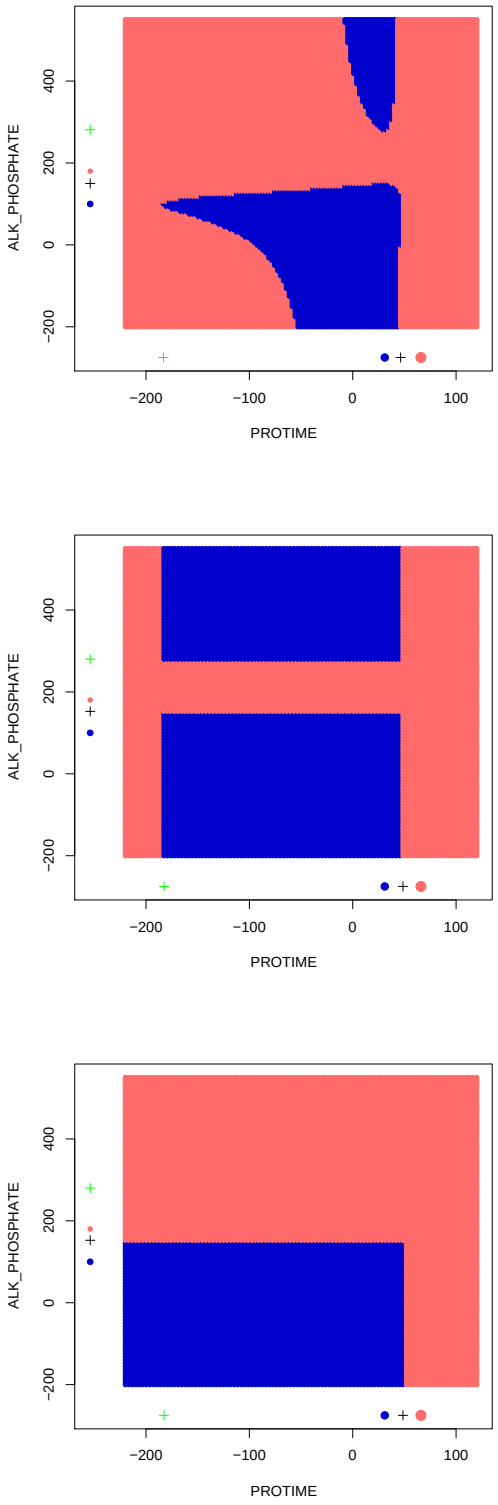

Figura 2.10: Límites de clase para el dataset: Newton Tree original (Arriba), Newton tree con dos puntos de corte *crisp* (Centro) con un punto de corte *crisp* (Abajo).

Capítulo 3

Experimentación

Resumen: En este capítulo mostramos la evaluación empírica de los Newton Trees (y sus distintas aproximaciones) en comparación con el algoritmo J48.

3.1. Experimentación

El objetivo de esta sección es comparar los Newton Trees y sus distintas adaptaciones (versiones no estocásticas y para el tratamiento de valores faltantes) con una implementación común de árboles de estimacion de probabilidades (PETs), es decir, árboles de decisión sin poda aplicada y con suavizado de Laplace en las hojas como se sugiere en (35)(17) . En particular, se ha optado por J48 (la variante de C4.5 implementada en Weka (45)). Se ha usado *Gain Ratio* como criterio de partición para los Newton Trees y J48.

La evaluación se ha llevado a cabo con 30 datasets del repositorio UCI (19) (véase la Tabla 3.1 donde se muestran sus características principales). Se han realizado una validación cruzada de 20 repeticiones por 5 pliegues, realizando un total de 100 ejecuciones para cada par de conjuntos de datos y método (con un resultado global de 3.000 ejecuciones). Como métricas de evaluación se han utilizado tres de las más importantes familias de métricas para la evaluación de clasificadores: *accuracy* como medida cualitativa del error, *AUC* (*Area Under the Curve* como medida de calidad de ranking (usando la versión Hand & Till's para multiclase (21)), y *MSE* (*Mean Squared Error* como medida de calidad de calibración y refinamiento.

DataSet	Classes	Nom.	Num.	Size	Majority Class	Minority Class
Anneal	6	32	6	898	(3)684	(4)
Autos	7	10	15	159	(0)48	(-3)0
autos_5c	5	10	15	156	(0)48	(3)13
Balance-Scale	3	0	4	625	(L),(R)288	(B)49
Breast-Cancer	2	9	0	277	(non-recurrence)196	(recurrence)81
chess-kr-vs-kp	2	36	0	3196	(won)1669	(nowin)1527
cmc	3	7	2	1473	(1)629	(2)333
Credit-a	2	9	6	653	(-)357	(+)296
Credit-g	2	13	7	1000	(good)700	(bad)300
Diabetes	2	0	9	768	(positive)500	(negative)268
Glass	7	0	9	214	(build wind non-float)76	(vehic wind non-float)0
Heart-statlog	2	0	13	270	(absent)150	(present)120
Hepatitis	2	14	5	80	(live)67	(die)13
Ionosphere	2	0	34	351	(g)225	(b)126
Iris	3	0	4	150	50	50
Monks 1	2	6	0	556	(0)278	(1)278
Monks 2	2	6	0	601	(0)395	(1)206
Monks 3	2	6	0	554	(1)288	(0)266
Mushrooms	2	22	0	5644	(e)3488	(p)2156
new-thyroid	3	0	5	215	(1)150	(3)30
pimaW	2	0	8	768	(0)500	(1)268
Sonar	2	0	60	208	(mine)111	(Rock)97
SoyBean	19	36	0	562	(brown-spot)92	(herbicide-injury)0
spectf_train	2	0	44	80	(0)40	(1)40
Tae	3	2	3	151	(3)52	(1)49
Tic-TacW	3	8	0	958	(1)626	(0)332
Vehicle3C	3	0	18	846	(saab_bus)435	(van)199
Vote	2	16	0	435	(democrat)267	(republican)168
Vowel	11	3	10	990	90	90
Wine	3	0	13	178	(2)71	(3) 48
Zoo	7	16	1	101	(mammal)41	(reptile)5

Tabla 3.1: Características de los datasets usados en los experimentos.

3.1.1. Experimentación con Newton Trees

La Tabla 3.2 muestra las medias de accuracy, AUC y MSE obtenidos por los dos algoritmos. En la parte inferior de la misma, también se presentan los valores promedio de todos los conjuntos de datos y separados por binarios/multiclase y nominales/numéricos/mixtos. Estos valores medios son sólo ilustrativos. Para analizar si las diferencias son significativas, se ha utilizado el test de *Wilcoxon signed-ranks* con un nivel de confianza de $\alpha = 0,05$ y $N = 30$ conjuntos de datos, como se sugiere en (11). Las diferencias significativas se indican en negrita. La Tabla 3.3 se centra en estas diferencias, mostrando una entrada $w/t/l$ para cada medida y un subconjunto de datasets, indicando que los Newton Trees ganan en w, empatan en t y pierden en l datasets, en comparación con el PET J48. De las tablas podemos concluir que los Newton Trees superan al J48 en las tres medidas (accuracy, AUC y MSE), y en los resultados promedios en cualquier tipo de dataset (multiclase / binarios, nominales / numérica / mixto). Las diferencias más fuertes se sitúan en los resultados obtenidos de AUC, que es la medida re-

comendada en la evaluación de PET (24).

3.1.2. Experimentación con versiones no estocásticas de los Newton Trees

Con el fin de analizar en qué medida decrece el rendimiento entre la versión original y las versiones *crisp* de los Newton Tress, se ha llevado a cabo una evaluación empírica de los tres enfoques para evaluar su rendimiento en: Newton Tress *crisp* con un punto de corte, con dos puntos de corte y con la versión estocástica original de los mismos. La tabla 3.4 resume los resultados promedios obtenidos para los datasets utilizados para realizar los experimentos. A fin de simplificar el proceso se ha limitado el número de hijos por nodo a 2. Vemos que las dos aproximaciones *crisp* (especialmente la versión con un solo punto de corte) es suficientemente competitiva con respecto a los Newton Trees originales. Sin embargo, la versión estocástica es mejor en términos de AUC y sobre todo en términos de MSE. Esto puede estar relacionado con el hecho de haber restringido el número de hijos por nodo, y forzar a las particiones nominales a tener un cajón de sastre (*all-the-rest*) como nodo por defecto.

Por otra parte, el rendimiento de los Newton Trees *crisp* con un solo punto de corte es tan buena como la versión con dos puntos de corte, por lo tanto, se recomienda su uso si el orden del árbol es elevado, para simplificar las reglas de las condiciones que representan las particiones.

3.1.3. Experimentación Newton Trees con Valores Faltantes

Al igual que ocurría en (31) donde los Newton Trees superaban claramente al J48 en los tres tipos de medidas utilizadas (Accuracy, AUC y MSE), encontrando las diferencias más notables en cuanto a valores de AUC, los resultados obtenidos al introducir datasets con valores faltantes han seguido la misma sintonía mejorando los resultados del J48 (ver Tabla 3.5 donde se muestran los resultados obtenidos en datasets con valores faltantes y limpios de ellos). Cabe destacar los resultados en AUC (medida recomendada para medir la evaluación de PETs (24)). A pesar de ello, debido al limitado número de datasets con valores faltantes utilizados (en este trabajo sólo hemos utilizado 7), es necesaria una experimentación más amplia y exhaustiva del problema para certificar estos buenos resultados.

Name	Classes	Att Type	Newton Trees			Unpruned Laplace J48		
			Acc.	AUC	MSE	Acc.	AUC	MSE
anneal	6	Mixed	97.5110	**0.8943**	0.0119	**98.7800**	0.8890	**0.0073**
autos_5c	5	Mixed.	**79.5060**	**0.9043**	0.0825	77.7130	0.8827	0.0840
balance-scale	3	Num.	**79.5520**	0.7962	0.1050	78.6880	**0.8199**	**0.0998**
breast-cancer	2	Nom.	**73.0110**	**0.6436**	**0.1929**	67.9360	0.6084	0.2233
chess-kr-vs-kp	2	Nom.	98.5050	0.9975	0.0135	**99.3050**	**0.9988**	**0.0064**
cmc	3	Mixed.	**50.1720**	**0.6739**	**0.2025**	49.1100	0.6658	0.2107
credit-a	2	Mixed.	**84.9310**	**0.9107**	**0.1118**	82.7960	0.8982	0.1256
credit-g	2	Mixed.	**70.3300**	**0.7202**	**0.1897**	68.2900	0.7016	0.2159
diabetes	2	Num.	71.8630	0.7801	**0.1798**	**72.8070**	0.7772	0.1877
glass	7	Num.	67.2940	0.7828	0.0901	67.0340	0.7895	0.0879
heart-statlog	2	Num.	**78.0740**	**0.8626**	**0.1490**	76.1850	0.8398	0.1753
hepatitis	2	Mixed.	**83.4370**	**0.7570**	**0.1143**	79.4370	0.6542	0.1498
ionosphere	2	Num.	88.9160	0.9235	0.0916	88.8460	0.9195	0.0917
iris	3	Num.	**94.7660**	**0.9938**	**0.0315**	94.0330	0.9710	0.0349
monks1W	2	Nom.	93.5230	**0.9899**	0.0606	92.7690	0.9761	**0.0519**
monks2W	2	Nom.	**85.8750**	**0.9378**	**0.1124**	61.3790	0.6456	0.2348
monks3W	2	Nom.	98.6730	**0.9926**	0.0166	98.6370	0.9909	**0.0135**
mushroom	2	Nom.	99.9910	0.9999	0.0193	100.0000	1.0000	**0.0001**
new-thyroid	3	Num.	92.6970	**0.9854**	**0.0438**	92.3480	0.9237	0.0454
pimaW	2	Num.	71.8630	0.7801	**0.1798**	**72.7750**	0.7772	0.1877
sonar	2	Num.	**77.5990**	**0.8499**	**0.1538**	73.3710	0.7888	0.2162
soybean	19	Nom.	89.2420	0.9771	0.0228	**91.2270**	0.9770	**0.0183**
spectf_train	2	Num.	67.3120	0.7301	0.2097	**71.7500**	0.7365	0.2196
tae	3	Mixed.	**58.7010**	**0.7398**	**0.1877**	54.1660	0.7078	0.1996
tic-tacW	3	Nom.	78.1110	0.8526	0.1426	**79.3990**	**0.8699**	**0.1393**
vehicle3c	3	Num.	72.1210	0.8441	0.1355	**73.0240**	**0.8807**	**0.1251**
vote	2	Nom.	94.5020	**0.9892**	0.0383	**95.1370**	0.9827	**0.0355**
vowel	11	Mixed.	75.3580	**0.9671**	0.0578	**79.5400**	0.9157	**0.0447**
wine	3	Num.	**94.3840**	**0.9905**	**0.0408**	92.2070	0.9544	0.0471
zoo	7	Mixed.	**94.9020**	**0.7243**	0.0252	93.1610	0.7147	**0.0234**
Mean (All)			82,0907	0,8664	0,1004	80,7283	0,8419	0,1101
Mean ($c = 2$)			83,6503	0,8665	0,1146	81,3388	0,8310	0,1334
Mean ($c > 2$)			80,3084	0,8662	0,0843	80,0307	0,8544	0,0834
Mean (Nominal)			90,1592	0,9311	0,0688	87,3099	0,8944	0,0803
Mean (Numerical)			79,7034	0,8599	0,1175	79,4223	0,8482	0,1265
Mean (Mixed)			77,2053	0,8102	0,1093	75,8881	0,7811	0,1179

Tabla 3.2: Comparación entre los Newton Trees y J48 sin poda y con suavizado de Laplace.

Netwon TreesUnpruned Laplace J48	Acc.	AUC	MSE
All	14/6/10	18/8/4	14/4/12
Nominal	2/3/4	5/2/2	2/0/7
Numerical	5/3/4	5/5/2	7/3/2
Mixed	7/0/2	9/0/0	5/1/3

Tabla 3.3: Resultados agregados usando tests estadisticos.

	NCrisp 1			NCrisp 2			Stochastic		
	ACC	**AUC**	**MSE**	**ACC**	**AUC**	**MSE**	**ACC**	**AUC**	**MSE**
Mean (All)	81.7344	0.8545	0.1728	81.2794	0.8468	0.1754	81.8084	0.8671	0.1008
Mean. ($C = 2$)	82.9629	0.8592	0.2523	82.8376	0.8567	0.2544	83.6503	0.8665	0.1146
Mean. ($C > 2$)	80.3304	0.8492	0.0819	79.4986	0.8354	0.0852	79.7033	0.8677	0.0851
Mean (Nom.)	90.0140	0.9266	0.2864	90.4361	0.9284	0.2862	89.8192	0.9311	0.0690
Mean (Num.)	79.0648	0.8498	0.1295	77.9973	0.8367	0.1352	79.7468	0.8618	0.1170
Mean (Mixt.)	77.0143	0.7888	0.1170	76.4988	0.7784	0.1183	76.5462	0.8100	0.1110

Tabla 3.4: NCrisp 1 = Newton Trees Crisp con un punto de corte, NCrisp 2 = Newton Trees Crisp con dos puntos de corte, Estocástico = Newton trees Estocásticos .

	Datasets sin valores faltantes						Datasets con valores faltantes					
Dataset	Newton Trees			J48			Newton Trees			J48		
	Acc.	**AUC**	**MSE**	**Acc.**	**AUC**	**MSE**	**Acc.**	**AUC**	**MSE**	**Acc.**	**AUC**	**MSE**
autos_5c	79,51	0,90	0,08	77,71	0,88	0,08	74,95	0,91	0,09	78,76	0,90	0,08
breast-cancer	73,01	0,64	0,19	67,94	0,61	0,22	71,61	0,63	0,20	67,09	0,60	0,22
credit-a	84,93	0,91	0,11	82,80	0,90	0,13	84,39	0,91	0,11	82,86	0,90	0,13
hepatitis	83,44	0,76	0,11	79,44	0,65	0,15	78,19	0,80	0,14	78,48	0,77	0,15
ionosphere	88,92	0,92	0,09	88,85	0,92	0,09	88,92	0,92	0,09	88,85	0,92	0,09
mushroom	99,99	1,00	0,02	100,00	1,00	0,00	99,55	1,00	0,01	100,00	1,00	0,00
soybean	89,24	0,98	0,02	91,23	0,98	0,02	89,93	0,94	0,02	89,45	0,93	0,02
Media (Todos)	85,58	0,87	0,09	83,99	0,85	0,10	83,93	0,87	0,10	83,64	0,86	0,10
Media (Faltantes)	86,06	0,85	0,11	83,80	0,82	0,12	84,53	0,85	0,11	83,45	0,84	0,12
Media (No Faltantes)	86,06	0,85	0,11	83,80	0,82	0,12	84,53	0,85	0,11	83,45	0,84	0,12

Tabla 3.5: Comparativa entre Newton Trees y J48 para Datasets sin/con valores faltantes

Capítulo 4

Conclusiones

4.1. Conclusiones

Este artículo ha presentado un novedoso método de aprendizaje de árboles de estimación de probabilidad basado en el cómputo de prototipos en las distintas particiones y en la aplicación de una Ley Cuadrática Inversa la cual utiliza la distancia con el prototipo y su masa a fin de obtener una fuerza de atracción que posteriormente se convierte en una probabilidad. Los árboles pueden ser representados gráficamente de manera que los patrones y su significado son inteligibles para el usuario y, a su vez, las predicciones pueden ser seguidas y explicadas, lo cual no es sencillo en los árboles de decisión que son usados de manera estocástica. A pesar de que los Newton Trees siguen siendo univariantes, se ha visto que tienen el poder de expresar y construir fronteras no paralelas a los ejes, por lo que su expresividad es superior que la de los árboles de decisión tradicionales.

Como la implementación de los Newton Trees es relativamente sencilla y la representación gráfica de los mismos es relativamente fácil de usar, es posible pensar que, en un futuro, podría aparecer en herramientas de aprendizaje automático y en suites de Minería de Datos.

El uso de prototipos (medioides) en lugar de centroides permite el uso de nuestros árboles para cualquier tipo de datos (continuos o discretos), siempre y cuando sea posible proporcionar una función de distancia para cada tipo de datos. En consecuencia, podemos aplicar los Newton Trees a tipos de datos estructurados, tales como secuencias, series, datos ordinales (que no requerirían numeración pero si una distancia adecuada), o incluso intervalos y textos. Aún más importante es que podemos usarlos con una mezcla de todos estos tipos de datos. Si las matrices de distancia son pre-procesadas

(una única vez por cada atributo antes de comenzar), el cómputo de los prototipos es mucho más eficiente que en los conocidos sistemas de partición los árboles de decisión tradicionales, ya que este método agrupa por clases y después calcula el medioide para cada grupo. En consecuencia, el número de particiones diferentes a evaluar en cada nodo es igual al número de atributos y no depende de puntos medios o del tamaño del conjunto de datos, lo cual contrasta dramáticamente con la evaluación exhaustiva de (casi) todos los puntos medios de los atributos numéricos en la mayoría árboles de decisión clásicos y PETs. Este factor de eficiencia es otro valor añadido para las aplicaciones de minería de datos que manejan grandes conjuntos de datos.

4.2. Trabajo Futuro

Existan muchas líneas de investigación a seguir. La primera de ellas es usar el valor de la masa (de los prototipos) al construir el árbol (y no sólo en la fase de evaluación) o el uso de todos los valores de atributo para calcular los grupos. Sin embargo, estas dos modificaciones implicarían un coste computacional extra y sólo podrían justificarse si existiera una mejora equitativa en los resultados. Otra línea diferente de trabajo futuro sería la de reconsiderar el uso de GainRatio para realizar las particiones. Tal y como se presentó en (16), está previsto incorporar como criterio de partición el AUC (el cual no ha sido utilizado con el objetivo de que las comparaciones con J48 fueran más significativas y así excluir otras causas de la mejora de resultados). Ya que toda la información del árbol es usada para clasificación, es necesario evaluar si utilizar el valor de AUC proporciona mejores resultados que GainRatio. Si este fuera el caso, se eliminaría el último vestigio de los árboles de decisión clásica en los Newton Trees.

Como trabajos futuros más ambiciosos, tenemos previsto la aplicación y extensión de los Newton Trees a la regresión y al clustering u otros métodos de aprendizaje, además de evolucionar dicho método para el permitir el tratamiento y clasificación de otros tipos de datos estructurados (como cadenas o listas) y de instancias multi-etiqueta.

Parte II

Publicaciones Asociadas a las Tesis

Capítulo 5

Publicaciones (texto completo)

5.1. Newton Trees

1. F. Martinez-Plumed, V. Estruch, C. Ferri,J. Hernández-Orallo, and M. J.Ramírez-Quintana. **Newton trees**. In Jiuyong Li and John Debenham, editors, *23rd Australasian Joint Conference on Artificial Intelligence, AI'10, Lecture Notes in Computer Science*, Adelaide, South Australia, 2010.

Newton Trees *

Fernando Martínez-Plumed, Vicent Estruch, Cèsar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana

DSIC, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain.
{fmartinez,vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Abstract. This paper presents Newton trees, a redefinition of probability estimation trees (PET) based on a stochastic understanding of decision trees that follows the principle of attraction (relating mass and distance through the Inverse Square Law). The structure, application and the graphical representation of Newton trees provide a way to make their stochastically driven predictions compatible with user's intelligibility, so preserving one of the most desirable features of decision trees, comprehensibility. Unlike almost all existing decision tree learning methods, which use different kinds of partitions depending on the attribute datatype, the construction of prototypes and the derivation of probabilities from distances are identical for every datatype (nominal and numerical, but also structured). We present a way of graphically representing the original stochastic probability estimation trees using a user-friendly gravitation simile.We include experiments showing that Newton trees outperform other PETs in probability estimation and accuracy.

Keywords: Probability Estimation Trees, Decision Trees, Distance Methods, Inverse Square Law, Stochastic Decision Trees.

1 Introduction

Decision tree learning [19] is one of the most popular (and powerful) techniques in machine learning and, very especially, in data mining. Two of the most important features of decision trees are their divide-and-conquer covering of the problem space and the use of decisions defined over univariate conditions (although multivariate variants exist). Decision tree learning has evolved through the introduction of datatype-specific condition schemes, dozens of splitting criteria, and many class assignment, pruning and stopping rules.

Probability Estimation Trees (PETs) [17][6], whose output is a probability rather than a crisp decision, are heirs of this technology, and are generally preferable over classical decision trees, whenever the goal is good rankings or good probability estimation. Initially, PETs were improved by using smoothing in the leaves [17] or through a pruning-smoothing [6]. The decision tree was

* This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN, under grant TIN 2007-68093-C02 and the Spanish project "Agreement Technologies" (Consolider Ingenio CSD2007-00022).

unaltered and the rules which were derived from it were consistent with its predictions. However, many other recent extensions of PETs use the decision tree as a skeleton upon which a complex decision making process takes place. The way the decision tree looks and the way it must be used to obtain the predictions are no longer easy to understand or even consistent.

In an effort of getting the most from decision tree learning for probability estimation, in this paper we present a new Stochastic Probability Estimation Tree learning technique. Splits are constructed by using attribute prototypes which work as attractors, following an inverse square law using the distance to the prototype and its mass, similar to other 'gravitational' approaches in machine learning [9][16]. We will present the details of Newton trees and we will show that they introduce a series of new features and important contributions, namely:

- We use the notion of distance in a univariate way as a general way of treating any kind of datatype (numerical, nominal, ordinal or structured).
- We construct the tree based on the principle of attraction and we derive the probabilities, use and represent the tree using the same principle.
- We handle numerical, nominal and ordinal attributes in the same way. We do not have to *type* attributes but just provide a distance for each datatype.
- We use medoids (prototypes from the set of attribute values) and not centroids, so properly handling both continuous and discrete datatypes. For continuous datatypes we only construct a cluster per attribute and class, and not a cutpoint between each pair of values. So, we reduce the number of partitions to evaluate (see Section 3.2).
- We provide a graphical representation of the trees to easily interpret them.
- We evaluate the trees using a qualitative measure of error (accuracy), a measure of ranking quality (AUC, *Area Under the ROC Curve*) and a measure of calibration and refinement quality (MSE, *Mean Squared Error*).

The paper is organised as follows. Section 2 introduces notation and basic terminology on decision tree learning and probability estimation trees, and also reviews some related work. Section 3 introduces Newton Trees, by first describing the attraction function and then explaining how trees are learned and used to obtain the probability estimations. It also introduces a user-friendly representation of Newton trees. Section 4 includes a set of experiments, which compare Newton Trees with a common PET (C4.5 without pruning and Laplace estimation). Finally, Section 5 presents the conclusions and the future work.

2 Notation and Previous Work

2.1 Notation

The set of all possible unlabelled examples E is composed of all the elements $e = \langle e_1, e_2, ..., e_m \rangle$ with m being the number of attributes. The attribute names are denoted by $\langle x_1, x_2, ..., x_m \rangle$. A labelled dataset D is a set of pairs $\langle e, i \rangle$ where

$e \in E$ and $i \in C$, where C is the set of classes. The number of classes, $|C|$, is denoted by c. We define a probability estimator as a set of c functions $p_{i \in C} : E \to \mathcal{R}$ such that $\forall i \in C, e \in E : 0 \leq p_i(e) \leq 1$ and $\forall e \in E : \sum p_{i \in C}(e) = 1$. Decision trees are formed of nodes, splits and conditions. A *condition* is any Boolean function $g : E \to \{true, false\}$. A *split* or *partition* is a set of s conditions $g_k : 1 \leq k \leq s$. A *decision tree* can be defined recursively as follows: (i) a node with no associated split is a decision tree, called a leaf; (ii) a node with an associated split $g_k : 1 \leq k \leq s$ and a set of s children t_k, such that each condition is associated with one and only one child, and each child t_k is a decision tree, is also a decision tree. Given a node ν, $Children(\nu)$ denotes the set of its children and $Parent(\nu)$ denotes its predecessor node. The special node where $Parent(\nu) = \emptyset$ is called the *root* of the tree. After the training stage, the examples will have been distributed among all the nodes in the tree, where the root node contains all the examples and downward nodes contain the subset of examples that are consistent with all its ancestors' conditions. Therefore, every node has particular absolute frequencies $n_1, n_2, ..., n_c$ for each class. The cardinality of the node is given by $\sum n_i$. A *decision tree classifier* is defined as a decision tree with an associated labelling of the leaves with classes. A PET is a decision tree which outputs a probability for each class.

2.2 Related Work

Existing Probability Estimation Trees output a probability but are not necessarily probabilistic in nature. A first issue is that they typically use a divide-and-conquer philosophy for constructing the tree but the same philosophy is used to make a prediction. Given an example, a sequence of decisions will lead to a leaf of the tree where a value is returned (a class in classification trees, a number in regression trees, a probability in PETs, etc.). The rest of the information of the tree is wasted (although there are exceptions [4, 6, 14]). In decision theory, though, this crisp view of decisions is awkward, since each decision can have an associated probability, and the overall probability must be computed by considering the whole structure of the tree. This kind of tree are frequently (but not always) called stochastic decision trees (e.g. [12]).

A second issue is that this use of all the paths in the tree can be made in such a way that the probabilities of the tree are independent to the instance which is being processed. In fact, this has been the approach in [14], by using an ad-hoc parameter which is used to determine the probability of each child in a partition. More recent approaches ([1], [2]) have made the probability depend on the proximity to the cut-point for the attribute, by using Kernel Density Estimates. In other words, a tree can be constructed by a classical algorithm (such as C4.5 [18] or CART [3]) and its probabilistic or stochastic interpretation can be inconsistent to the way the decision tree was constructed.

A third issue is how different datatypes are handled. Many of the previous approaches only deal with numerical attributes ([1], [2]) or only with nominal attributes. When handling both, the trees just preserve the very specific way of handling numerical attributes with cutpoints and nominal attributes with

equalities, as C4.5 [18] or CART [3]. Even in the case of fuzzy decision trees (which often provide a more integrated view of nominal and numerical attributes) it is unclear how decision trees can be applied to problems where some attributes are from other (structured) datatypes such as intervals, sequences or sets.

Having all the previous approaches to PETs, in this work we propose a new decision tree learning method which has been designed from scratch with the goals of being stochastic in nature, general and flexible in the way it handles data attributes, and intelligible.

3 Stochastic Distance-based Probability Estimation Trees

In this section we define our Stochastic Probability Estimation Tree learning technique which leads to Newton trees.

3.1 Gravitational Partitions

When constructing splits, decision trees typically generate conditions which are then evaluated to see how well they separate the classes. Instead of that, we propose to define a node/cluster per class and then try to find the characterisation of each node in terms of one attribute at a time (univariate).

Following this idea, one first approach is to use Kernel Density Estimation [21] in order to derive a probability density function (*pdf*), from the examples belonging to each class. However, many of these techniques will construct a parametrised or composite *pdf* that will make partitions unintelligible, apart from having the risk of overfitting. Another approach is to derive a prototype for each node, and then, to derive a probability from the prototypes. In order to treat discrete datatypes appropriately, we use a medoid (the element in each cluster such that its average distance to the rest is the lowest). If we generate prototypes, one possibility to derive probabilities from them is to assume some probability distribution. For instance, if we consider a normal distribution for each node with centre at the prototype and with standard deviation equal to the mean of distances of the elements of the node, we have a *pdf*. Figure 1 (left) shows the *pdf* using a Gaussian with centres 3 and 8, with standard deviations 1 and 3.5 (respectively) and masses 20 and 100 (respectively). This can be converted into conditional probabilities by mere normalisation, as shown in Figure 1 (right).

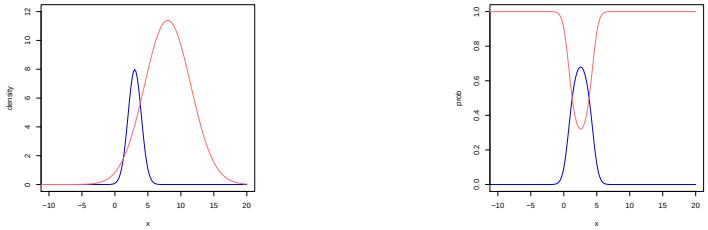

Fig. 1. (Left) Two normal distributions placed at centres 3 and 8, with standard deviations 1 and 3.5 (respectively) and masses 20 and 100 (respectively). (Right) The conditional probabilities derived from the Gaussians.

The problem of the previous approach is that when masses are too disparate, one distribution can cover the other, giving a plain partition where all the elements go to one prototype. One criterion to avoid this is to give extra importance to distance, so that at distance 0 the probability is always 1. A way to do this is to employ an inverse-square law such as in gravitation. Hence, we define the following *attraction* function between an element e of mass m_e (we will assume $m_e = 1$) and a prototype π of mass m_π separated by a distance $d(e, \pi) = d$:

$$attraction(e, \pi) = \frac{m_e m_\pi}{d(e, \pi)^2} = \frac{m_\pi}{d^2}$$

We are interested in deriving class probabilities by considering this attraction. Figure 2 shows the attraction (left) and the probability (right) with the same parameters as before (note that the standard deviation is no longer used).

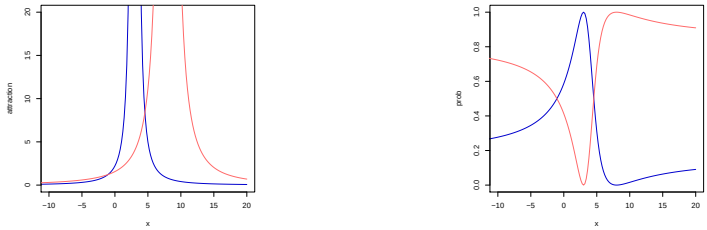

Fig. 2. (Left) Two gravitational centres at 3 and 8 with masses 20 and 100 (respectively). (Right) The probabilities derived from the gravitational centres.

An interesting property is that when the distance goes to infinity the probabilities tend to converge to the mass proportion. For instance, if we have two centres at 3 and 8, and 8 has much more mass (as in the previous example), it is easy to see that the attraction to 8 will be higher than the attraction to 3 for a point placed at -100.

Of course, the idea of using the gravitational law in machine learning is not new at all, for instance in clustering [9] or classification [16]. The same Inverse Square Law principle is presented in some variants of Kernel Density Estimation, several classification techniques such as weighted kNN, where the weight is a kernel which is simply defined as the inverse of the distance, or in some other clustering algorithms. To our knowledge, its use for decision trees is new.

3.2 Tree Generation

Centre splitting [20] is a machine learning method which consists in dividing the input space in different regions where each region is represented by a centre[1]. In every iteration, a centre is calculated for every different class which is presented in the area. Then, every example is associated to its nearest centre. This process is repeated until the area is pure. One of the special features of this method is

[1] The centre may match to an existing example or not

that examples are managed as a whole.This appreciation leads us to propose a decision tree inference strategy where centroids are computed by considering only the values of one attribute, which allows us to join centre splitting and decision tree learning techniques in an elegant way.

The detailed definition of the algorithm can be found in [15]. Here, we give a more sketchy description: for each attribute x_r and for each class i, a prototype $\pi_{r,i}$ is calculated as the attribute value with lowest mean distance to the elements of the class. Once this process is finished, the splitting attribute is selected according to one of the well-known splitting criteria (for instance, gain ratio [18]). Then, the split proceeds by associating every instance to its closest attribute prototype, which typically produces impure clusters[2]. Although the computation of distances is quadratic on the number of instances, we can reduce it by using a distance matrix per attribute (of size $n_r \times n_r$, where n_r is the number of different attribute values) prior to the algorithm execution. But, more importantly, if we have m attributes and n_r values per attribute, we only construct (and evaluate) $O(m)$ partitions and not $O(n_r \times m)$, the typical order for classical decision tree learning algorithms using midpoints for continuous attributes.

3.3 Stochastic Probability Calculation

Now, we illustrate how a Newton Tree is used to estimate probabilities in a stochastic way. In what follows, $\overrightarrow{p}(\nu, e) = \langle p_1(\nu, e), \ldots, p_c(\nu, e)\rangle$ denotes the probability vector of example e at node ν, where $p_i(\nu, e)$ denotes the probability that e belongs to class i at node ν. With $\hat{p}(\nu, e)$ we denote the probability that e falls into node ν (coming from its parent), which is derived from the attraction that ν exert over e, that is $\hat{p}(\nu, e) = \frac{attraction(e,\nu)}{\sum_{\mu \in Children(Parent(\nu))} attraction(e,\mu)}$.

Given a new example e and a Newton tree T, the objective is to calculate the probability vector at the root of T, $\overrightarrow{p}(root, e)$. Basically, the idea is to compute downwards the probability of falling in each leaf, calculate the leaf probability vector and then to propagate upwards the leaf probability vector to the root to obtain the total class probability vector $\overrightarrow{p}(root, e)$. The leaf probability vectors can be obtained once the tree T has been built by applying Laplace correction as has been shown in [17, 6]. For each example, we calculate the probability of choosing each child node μ if placed at the parent node ν using the attraction (i.e., $\hat{p}(\mu, e)$). This probability is multiplied by the probability vector of the child ($\overrightarrow{p}(\mu, e)$):

Definition 1. Stochastic Probability Vector Estimation
Given an example e and a Newton tree T, the probability vector $\overrightarrow{p}(root, e)$ at the root of T is estimated by applying

$$\forall \nu \in T : \overrightarrow{p}(\nu, e) = \begin{cases} \sum_{\mu \in Children(\nu)} \hat{p}(\mu, e) \cdot \overrightarrow{p}(\mu, e) & \text{if } \nu \text{ is not a leaf} \\ \langle Laplace(1, \nu), \ldots, Laplace(c, \nu)\rangle & \text{if } \nu \text{ is a leaf} \end{cases}$$

[2] Note that, during the splitting process, we apply the *attraction* function assuming that the mass is the unit. This is due to the fact that the total mass of a node is not known until all the instances have been associated to its prototype.

where $Laplace(j, \nu)$ *is the Laplace correction of the frequency of elements of class* j *in node* ν*.*

The stochastic calculation of the probabilities seen above may seem too cryptic for a general use of these trees if intelligibility is a requirement. In order to address this issue, we show a graphical representation of Newton trees, which may help users understand how the stochastic probability assignment is made, and to get insight from the tree.

Figure 3 (left) shows this user-friendly representation of a Newton Tree for the Hepatitis dataset from the UCI repository [8]. Note that all partitions are binary because this is a two-class problem, namely DIE and $LIVE$. The two first splits are made over the numerical attributes $PROTIME$ and $ALK\_PHOSPHATE$, respectively, and the other two splits are made over the nominal attributes SEX and $FATIGUE$. The nodes are represented as balls of a size which is proportional to the node mass (for instance, the node with a mass of 17 represents that 17 training examples fall into it). The ball also shows the proportion of examples of each class in different colours. Additionally, the value for the attribute prototype is shown in the middle of each ball. Finally, the smoothed probabilities per class at the leaves are also provided (in the figure, as a small table below each leaf). In order to ease the understanding on how probabilities are derived, Figure 3 (right) shows the internal probabilities (vectors and node probabilities) and the top vector probability for example ($PROTIME = 40; ALK\_PHOSPHATE = 120; SEX = FEMALE; FATIGUE = UNKNOWN$), which is $(0.7316, 0.2684)$, a relatively clear DIE case. All these graphical elements in the Newton Trees representation may help users understand the way in that probabilities are estimated, making Newton trees less cryptic than other PET methods.

4 Experiments

The aim of this section is to compare Newton trees with a common implementation of Probability Estimation Trees, namely unpruned decision trees with Laplace smoothing in the leaves as suggested by [17][6]. In particular, we chose J48 (the variant of C45.) implemented in Weka [10]. We used *Gain ratio* as splitting criterion for Newton trees and J48. The evaluation was performed over 30 datasets from the UCI repository [8], from which we removed instances with missing values (see [15] for their characteristics). We set up a 20×5-fold cross validation, making a total of 100 learning runs for each pair of dataset and method (3,000 overall). As evaluation metrics we used ([7]): accuracy, as a qualitative measure of error, AUC (*Area Under the Curve*) as a measure of ranking quality, (using Hand & Till's multiclass version [11]) and MSE (*Mean Squared Error*) as a measure of calibration and refinement quality.

Table 1 shows the average accuracy, AUC and MSE obtained by the two algorithms. At the bottom, we also show the mean values for all the datasets. These means are just illustrative. To analyse whether the differences are significant, we used the Wilcoxon signed-ranks test with a confidence level of α

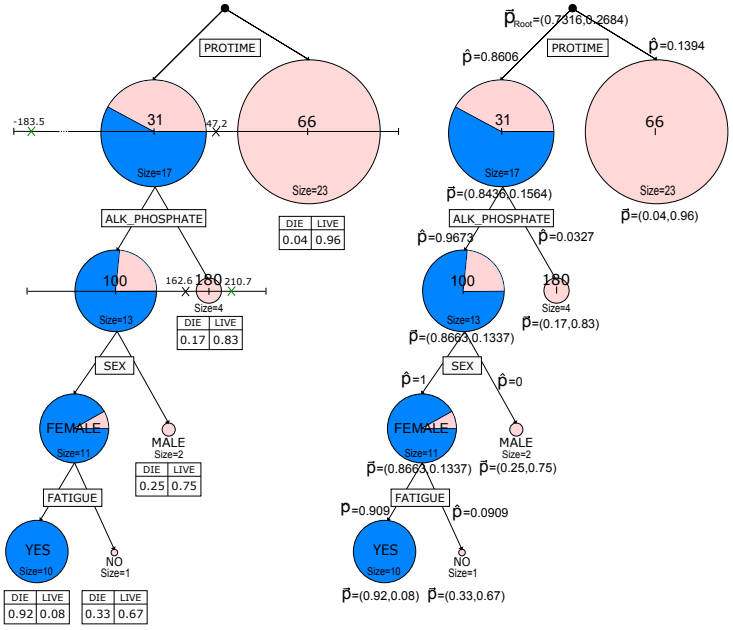

Fig. 3. (Left) Newton Tree for the hepatitis dataset. (Right) The node probability vectors, children probabilities and global probability vector for example (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE= UNKNOWN)

$= 0.05$ and $N = 30$ data sets, as suggested in [5]. Significant differences are shown in bold. Finally, in Table 2 we focus on these differences, showing an entry $w/t/l$ for each measure and dataset subset, which indicates that Newton trees win in w, tie in t, and lose in l datasets, compared to the J48 PETs. From the tables, we see that Newton trees outperform J48 PETs in the three measures (Accuracy, AUC and MSE), and with the means in Table 1, in any selection depending on the type of dataset (multiclass/binary, nominal/numerical/mixed). The strongest differences are found in AUC, which is the recommended measure when evaluating PETs ([13]). If we look at the significance results in Table 2, we have a similar picture. The exception is the result for nominal datasets. While AUC is still much better, the results in MSE are worse (and as a result so is accuracy). This indicates a bad calibration of the results for datasets with only nominal partitions, which might be caused by the way discrete distances affect on the attraction measure, although more research should be done to clarify this (since there are only 7 datasets in this subset).

5 Conclusions and Future Work

This paper has presented a novel probability estimation tree learning method which is based on computing prototypes and applying an Inverse Square Law

Name	Classes	Att Type	Newton Trees			Unpruned Laplace J48		
			Acc.	AUC	MSE	Acc.	AUC	MSE
anneal	6	Mixed	97.5110	**0.8943**	0.0119	**98.7800**	0.8890	**0.0073**
autos_5c	5	Mixed.	**79.5060**	**0.9043**	0.0825	77.7130	0.8827	0.0840
balance-scale	3	Num.	**79.5520**	0.7962	0.1050	78.6880	**0.8199**	**0.0998**
breast-cancer	2	Nom.	**73.0110**	**0.6436**	**0.1929**	67.9360	0.6084	0.2233
chess-kr-vs-kp	2	Nom.	98.5050	0.9975	0.0135	**99.3050**	**0.9988**	**0.0064**
cmc	3	Mixed.	**50.1720**	**0.6739**	**0.2025**	49.1100	0.6658	0.2107
credit-a	2	Mixed.	**84.9310**	**0.9107**	**0.1118**	82.7960	0.8982	0.1256
credit-g	2	Mixed.	**70.3300**	**0.7202**	**0.1897**	68.2900	0.7016	0.2159
diabetes	2	Num.	71.8630	0.7801	**0.1798**	**72.8070**	0.7772	0.1877
glass	7	Num.	67.2940	0.7828	0.0901	67.0340	0.7895	0.0879
heart-statlog	2	Num.	**78.0740**	**0.8626**	**0.1490**	76.1850	0.8398	0.1753
hepatitis	2	Mixed.	**83.4370**	**0.7570**	**0.1143**	79.4370	0.6542	0.1498
ionosphere	2	Num.	88.9160	0.9235	0.0916	88.8460	0.9195	0.0917
iris	3	Num.	**94.7660**	**0.9938**	**0.0315**	94.0330	0.9710	0.0349
monks1W	2	Nom.	93.5230	**0.9899**	0.0606	92.7690	0.9761	**0.0519**
monks2W	2	Nom.	**85.8750**	**0.9378**	**0.1124**	61.3790	0.6456	0.2348
monks3W	2	Nom.	98.6730	**0.9926**	0.0166	98.6370	0.9909	**0.0135**
mushroom	2	Nom.	99.9910	0.9999	0.0193	100.0000	1.0000	**0.0001**
new-thyroid	3	Num.	92.6970	**0.9854**	**0.0438**	92.3480	0.9237	0.0454
pimaW	2	Num.	71.8630	0.7801	**0.1798**	**72.7750**	0.7772	0.1877
sonar	2	Num.	**77.5990**	**0.8499**	**0.1538**	73.3710	0.7888	0.2162
soybean	19	Nom.	89.2420	0.9771	0.0228	**91.2270**	0.9770	**0.0183**
spectf_train	2	Num.	67.3120	0.7301	0.2097	**71.7500**	0.7365	0.2196
tae	3	Mixed.	**58.7010**	**0.7398**	**0.1877**	54.1660	0.7078	0.1996
tic-tacW	3	Nom.	78.1110	0.8526	0.1426	**79.3990**	**0.8699**	**0.1393**
vehicle3c	3	Num.	72.1210	0.8441	0.1355	**73.0240**	**0.8807**	**0.1251**
vote	2	Nom.	94.5020	**0.9892**	0.0383	**95.1370**	0.9827	**0.0355**
vowel	11	Mixed.	75.3580	**0.9671**	0.0578	**79.5400**	0.9157	**0.0447**
wine	3	Num.	**94.3840**	**0.9905**	**0.0408**	92.2070	0.9544	0.0471
zoo	7	Mixed.	**94.9020**	**0.7243**	0.0252	93.1610	0.7147	**0.0234**
Mean (All)			82,0907	0,8664	0,1004	80,7283	0,8419	0,1101
Mean ($c = 2$)			83,6503	0,8665	0,1146	81,3388	0,8310	0,1334
Mean ($c > 2$)			80,3084	0,8662	0,0843	80,0307	0,8544	0,0834
Mean (Nominal)			90,1592	0,9311	0,0688	87,3099	0,8944	0,0803
Mean (Numerical)			79,7034	0,8599	0,1175	79,4223	0,8482	0,1265
Mean (Mixed)			77,2053	0,8102	0,1093	75,8881	0,7811	0,1179

Table 1. Comparison between Newton trees and unpruned J48 with Laplace correction.

Unpruned Laplace J48 / Netwon Trees	Acc.	AUC	MSE
All	14/6/10	18/8/4	14/4/12
Nominal	2/3/4	5/2/2	2/0/7
Numerical	5/3/4	5/5/2	7/3/2
Mixed	7/0/2	9/0/0	5/1/3

Table 2. Aggregated results using the statistical tests

that uses the distance to the prototype and its mass, in order to derive an attraction force which is then converted into a probability. The trees can be graphically represented in such a way that their meaning and patterns can be understood. The use of prototypes (mediods) instead of centroids allows for the use of our trees for any kind of datatype (continuous or discrete), as long as we provide a distance function for each datatype. Consequently, we can apply our trees to structured datatypes, such as sequences, sets, ordinal data, intervals or even images and texts. More importantly, we can use the tree with a mixture of all these datatypes. If distance matrices are preprocessed (only once for each attribute before start), the computation of the prototypes is much more efficient than the split population schemes in traditional decision trees, since we group by classes and then compute the mediod of each cluster. Consequently, the number

of different splits to evaluate at each node is equal to the number of attributes and does not depend on midpoints or the size of the dataset.

There are many research lines to pursue. One is to use the mass also when constructing the tree or using all the attribute values as possible clusters. However, these two modifications would entail extra computational cost and could only be justified if there is a significant improvement in the results.

References

1. I. Alvarez and S. Bernard. Ranking cases with decision trees: a geometric method that preserves intelligibility. In *IJCAI*, pages 635–640, 2005.
2. I. Alvarez, S. Bernard, and G. Deffuant. Keep the decision tree and estimate the class probabilities using its decision boundary. In *IJCAI*, pages 654–659, 2007.
3. Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.
4. W. Buntine. Learning classification trees. *Stats. and Computing*, 2(2):63–73, 1992.
5. J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.
6. C. Ferri, P. Flach, and J. Hernandez-Orallo. Improving the auc of probabilistic estimation trees. In *Proc. ECML*, volume 2837 of *LNCS*, pages 121–132, 2003.
7. C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recogn. Lett.*, 30(1):27–38, 2009.
8. A. Frank and A. Asuncion. UCI machine learning repository, 2010.
9. J. Gomez, D. Dasgupta, and O. Nasraoui. A new gravitational clustering algorithm. In *Int. Conf. on Data Mining*, page 83. Society for Industrial & Applied, 2003.
10. M. Hall, Frank E.and G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software. *SIGKDD Explorations*, 11(1):10–18, 2009.
11. D.J. Hand and R.J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.
12. R.F. Hespos and P.A. Strassmann. Stochastic decision trees for the analysis of investment decisions. *Management Science*, 11(10):244–259, 1965.
13. Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms - appendices. *IEEE Trans. Knowl. Data Eng.*, 17(3), 2005.
14. C.X. Ling and R.J. Yan. Decision tree with better ranking. In *International Conference on Machine Learning*, volume 20-2, page 480, 2003.
15. F. Martinez-Plumed, V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Newton trees. extended report. Technical report, DSIC, UPV, http://www.dsic.upv.es/∼flip/NewtonTR.pdf, 2010.
16. L. Peng, B. Yang, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.
17. Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.
18. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
19. Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2008.
20. C.J. Thornton. *Truth from trash: how learning makes sense*. The MIT Press, 2000.
21. Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993.

5.2. Valores Faltantes en Newton Trees

2. F. Martinez-Plumed, V. Estruch, C. Ferri,J. Hernández-Orallo, and M. J.Ramírez-Quintana. **Valores Faltantes en Newton trees** . *V Simposio de Teoría y Aplicaciones de Minería de Datos, TAMIDA'10*, Valencia, Spain, 2010.

Valores Faltantes en Newton Trees

Fernando Martínez-Plumed
Vicent Estruch
Cèsar Ferri
José Hernández-Orallo
María José Ramírez-Quintana

DSIC, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain.

{fmartinez,vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Resumen

El tratamiento de atributos con valores faltantes es un aspecto importante dentro del aprendizaje automático y la minería de datos, ya que en la mayoría de aplicaciones reales la existencia de atributos faltantes es muy común. La existencia de éstos, tanto en los datasets de entrenamiento como de test, afecta notablemente al porcentaje de clasificaciones correctas. Una aproximación frecuente seguida en estos casos es considerar que estos valores podrían tomar, probabilísticamente, cualquiera de los valores existentes para ese atributo. Recientemente, hemos introducido un nuevo método de aprendizaje de árboles de decisión estocásticos, basado en el simil de atracción gravitatoria usando centros y distancias, y que denominamos *Newton Trees*. Estos árboles tratan estocásticamente las instancias y las distribuyen por todo el árbol usando las probabilidades de cada rama. Por tanto, nuestro método es directamente extensible al tratamiento de datos faltantes, asumiendo una distancia igual a todos los centros y distribuyendo la instancia por todos los hijos de un nodo. Los resultados obtenidos en la experimentación han sido comparados con los obtenidos por el algoritmo C4.5. Los resultados han seguido la línea de los obtenidos con los Newton Trees sin valores faltantes, siendo más competitivos que los obtenidos por el C4.5 aun utilizando la misma estrategia para el tratamiento de este tipo de datos.

1. Introducción

Los árboles de decisión son una de las técnicas más populares y potentes dentro del aprendizaje automático y más concretamente, en el campo de la minería de datos. Tradicionalmente, los algoritmos de aprendizaje de árboles de decisión se basan en la idea de partir el conjunto de entrenamiento en conjuntos cada vez más pequeños hasta que el conjunto es puro, es decir, sólo contiene ejemplos de una clase. Los nodos internos del árbol corresponden a condiciones sobre un atributo elegido de acuerdo a una cierta heurística. Para clasificar un nuevo ejemplo, se recorre el árbol desde la raíz hasta alcanzar una hoja siguiendo las condiciones que cumple el ejemplo y se le asigna la clase de las instancias en dicha hoja. Ejemplos de algoritmos que funcionan así son ID3 [10], C4.5 [10] o C5.0 [10].

Aunque se trata de una técnica de aprendizaje ampliamente usada en la práctica, ya que con ella se obtienen buenos resultados en cuanto a la tasa de acierto en la clasificación (*accuracy*) mientras se producen modelos comprensibles, no fueron diseñados para mejorar otras medidas más enfocadas a evaluar su calidad como estimadores de probabilidad. En la última década, los árboles de estimación de probabilidad se han constituido como nuevas variantes de árboles de decisión que permiten mejorar los resultados en términos del Área bajo la Curva ROC (AUC, del inglés, *Area Under the Curve*) y MSE (del inglés, *Mean*

Squared Error). En la mayoría de casos, estos árboles (conocidos como *Probabilistic Decision Trees, PET* [3, 7]) no han modificado sensiblemente la manera de construir los árboles, usando todavía criterios, particiones y métodos de poda diseñados para otras tareas o medidas. De hecho, aunque los PETs retornan probabilidades, no son probabilísticos en términos de cómo el árbol se usa y si los ejemplos descienden por una rama o varias ramas a la vez.

En [6], se presenta una nueva técnica de inducción de Arboles de Estimación Estocástica de Probabilidades, denominados Newton Trees, que aúna las ventajas de los árboles probabilísticos y los árboles de decisión basados en distancias [2]. Algunas de las ventajas de utilizar este tipo de árboles son:

- Tratamiento homogéneo de cualquier tipo de datos (nominales, numéricos u ordinales) usando la noción de distancia de forma univariante. Para datos numéricos se puede usar el valor absoluto de la diferencia, y, en el caso de datos nominales u ordinales, la función identidad.
- Construcción, uso y representación del árbol basados en el principio de atracción. Las probabilidades se derivan también a partir de dicho principio.
- Uso de prototipos, obtenidos a partir del conjunto de valores de los atributos, y no de centroides, para el correcto tratamiento de datasets continuos y discretos. Para los datasets con valores continuos, solo se construye un clúster por atributo y clase y no un punto de corte entre cada pareja de valores. Por lo tanto, tenemos un orden de (m) en lugar de $(n \cdot m)$ particiones para evaluar, donde m es el número de atributos y n el número de ejemplos.
- Representación gráfica del árbol sencilla de interpretar.
- Los árboles son univariantes, pero sus particiones no son necesariamente paralelas a los ejes. Esto implica que las particiones pueden crear límites más expresivos que los obtenidos por los árboles de decisión clásicos, sin tener que ser multivariantes.

Por otra parte, el tratamiento de atributos con valores faltantes en minería de datos es tanto o más importante cuanto más real es la aplicación de la herramienta o mayor es el volumen de datos, pues la integración de datos conlleva frecuentemente que existan datos faltantes para un número significativo de atributos e instancias. Generalmente, la ausencia de información tiene un tratamiento diferente según la técnica de aprendizaje empleada. En caso de técnicas basadas en reglas o condiciones, supone implícitamente considerar una tercera opción (algunos árboles de decisión contienen una hoja que captura los atributos con valor nulo) o considerar que la ausencia de información implica que cualquiera de los valores posibles tiene igual plausibilidad o probabilidad, teniendo (o no) en cuenta la frecuencia de los valores para ese atributo.

Los Newton Trees tratan estocásticamente las instancias y las distribuyen por todo el árbol usando las probabilidades de cada rama. Por tanto, nuestro método es directamente extensible al tratamiento de datos faltantes, asumiendo una distancia igual a todos los centros y distribuyendo la instancia por todos los hijos de un nodo. Pero para ello usamos la cardinalidad (con nuestra terminología, la masa) de cada nodo. Por tanto, nuestra aproximación es similar a aquella que ante un valor faltante, se asume que el valor podría tomar cualquiera de los valores (no nulos) de ese atributo, utilizando la frecuencia de los valores para asignar una probabilidad.

Este documento está organizado como sigue. En la próxima sección introducimos brevemente los Newton Trees, primero describiendo la función de atracción y posteriormente mostrando cómo se construye el árbol y cómo se usa para estimar las probabilidades. En la sección 3 describimos la adaptación de los Newton Trees para que puedan manejar datasets con valores faltantes y se presenta una sencilla representación gráfica de estos árboles para mostrar cómo se clasifican instancias con este tipo de valores. La sección 4 incluye un conjunto de experimentos donde se comparan

los Newton Trees con la versión del algoritmo C4.5 implementada en WEKA (J48 sin poda y con suavizado de Laplace en las hojas). Finalmente, la sección 5 cierra este artículo con las conclusiones y las ideas sobre trabajo futuro.

2. Newton Trees

En esta sección vamos a definir nuestra técnica de inducción de Árboles de Estimación Estocástica de Probabilidades, los Newton Trees.

2.1. Particiones Gravitacionales

Cuando se generan las particiones, los árboles de decisión tradicionales forman condiciones (de la forma *atributo=valor* si el atributo es nominal o *atributo* $<$ *valor* si es numérico) que son evaluadas para ver cuan bien éstas separan las clases. En lugar de esto, proponemos definir un nodo/clúster por clase e intentar buscar una caracterización para cada nodo en términos de un atributo cada vez (univariante). Con esta idea en mente, la aproximación que hemos seguido ha sido la de obtener un prototipo para cada nodo y, a partir de éstos, derivar una probabilidad. Con el fin de tratar apropiadamente los conjuntos de datos con valores discretos, estos prototipos son "medioides"(el elemento de cada clúster cuya distancia media al resto de elementos es mínima). Una vez se han generado los prototipos, sus probabilidades asociadas pueden ser calculadas empleando no simplemente la distancia, sino una ley inversamente proporcional al cuadrado de la distancia, similar a la Ley de Gravitación Universal. Usando esto, hemos definido la siguiente función de atraction (*attraction*) entre un elemento e de masa m_e (asumimos $m_e = 1$) y un prototipo π de masa m_π separados por una distancia $d(e, \pi) = d$:

$$attraction(e, \pi) = \frac{m_e m_\pi}{d(e, \pi)^2} = \frac{m_\pi}{d^2}$$

La figura 1 muestra la atracción (izquierda) y la probabilidad (derecha) obtenidas mediante la fórmula de la atracción para dos centros con valores 3 y 8, cuyas masas son 20 y 100

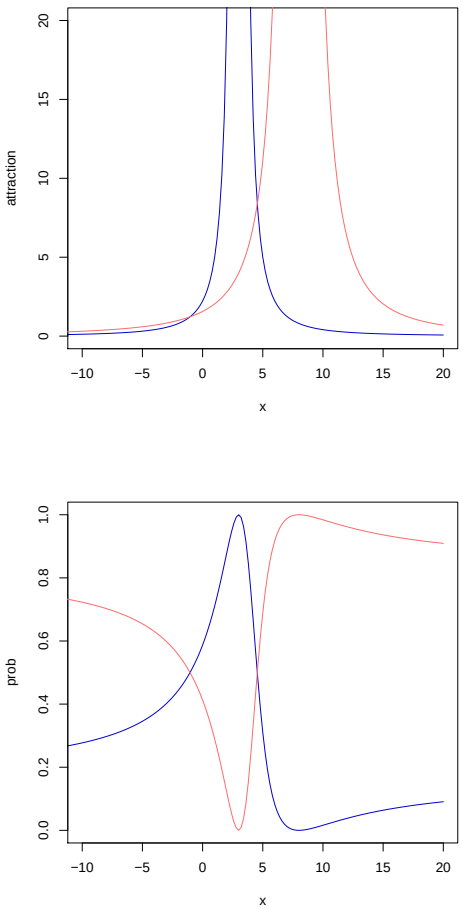

Figura 1: Dos centros gravitacionales en 3 y 8 con masas 20 y 100 (respectivamente)(arriba). Probabilidades derivadas a partir de los centros gravitacionales (abajo).

respectivamente. Una propiedad interesante es que cuando la distancia tiende a infinito, la probabilidad tiende a la proporción de la masa. Por ejemplo, si tenemos dos centros en 3 y 8, y 8 tiene mucha mas masa (como en el ejemplo anterior), parece más lógico esperar que la atracción hacia 8 sea mayor que la atracción hacia 3 para un punto situado en -100.

2.2. Generación del Árbol

La definición detallada del algoritmo puede encontrarse en [6]. Aquí vamos a dar una visión

más reducida de su funcionamiento. Básicamente, el algoritmo de generación del árbol trabaja de la siguiente forma: para cada atributo x_r, y para cada clase i, se calcula un prototipo $\pi_{r,i}$ como el valor de dicho atributo con la menor distancia media al resto de elementos de la clase. Una vez terminado el proceso, se selecciona el atributo que se usará en la partición de acuerdo a algún criterio de partición (por ejemplo, *gain ratio* [8]). Posteriormente, la partición procede a asociar cada instancia al prototipo que ejerza sobre ella una mayor atracción, lo que produce clústeres impuros. Nótese que, durante el proceso de partición, aplicamos la función de atracción asumiendo que la masa del nodo es la unidad. Esto es debido al hecho de que la masa total del nodo no es conocida hasta que todas las instancias han sido asociadas a su prototipo correspondiente. Hay que recalcar que las distancias son computadas entre los valores de los atributos y no entre los ejemplos completos. Esta característica es crucial para la eficiencia.

2.3. Cálculo Estocástico de la Probabilidad

Una vez obtenido el árbol, éste se usa para estimar las probabilidades de nuevas instancias de una forma estocástica.

Dado un nodo ν, $Child(\nu)$ denota el conjunto de sus nodos hijo y $Parent(\nu)$ denota su nodo predecesor. El nodo especial donde $Parent(\nu) = \emptyset$ es la raíz del árbol y lo denominaremos *root*. $\overrightarrow{p}(\nu, e) = \langle p_1(\nu, e), \ldots, p_c(\nu, e)\rangle$ denota el vector de probabilidad de un ejemplo e en un nodo ν. Con $\hat{p}(\nu, e)$ expresamos la probabilidad de que e caiga en un nodo ν (viniendo de su progenitor), y es derivada a partir de la atracción que ejerce ν sobre e, es decir:

$$\hat{p}(\nu, e) = \frac{attraction(e, \nu)}{\sum_{\mu \in Child(Parent(\nu))} attraction(e, \mu)}$$

Dado un nuevo ejemplo e y un Newton Tree T, el objetivo es calcular el vector de probabilidad en la raíz de T, $\overrightarrow{p}(root, e)$. La idea básica es computar, desde la raíz a las hojas, la probabilidad de caer en cada hoja, calcular el vector de probabilidad en cada hoja y finalmente propagar hacia arriba este vector para obtener en la raíz el vector total de probabilidad $\overrightarrow{p}(root, e)$. Los vectores de probabilidad de las hojas pueden ser obtenidos una vez se ha generado el árbol aplicando Laplace como en [7, 3]. Para cada ejemplo, calculamos la probabilidad de escoger cada nodo hijo μ (a partir del nodo donde se encuentra ν) usando la atracción (i.e., $\hat{p}(\mu, e)$). Esta probabilidad se multiplica por el vector de probabilidad del nodo hijo ($\overrightarrow{p}(\mu, e)$), tal y como se describe en la siguiente definición:

Definición 1 Estimación del Vector Estocástico de Probabilidad
Dado un ejemplo e y un Newton Tree T, el vector de probabilidad $\overrightarrow{p}(root, e)$ en la raíz T se estima aplicando

$$\forall \nu \in T : \overrightarrow{p}(\nu, e) = \begin{cases} \sum_{\mu \in Child(\nu)} \hat{p}(\mu, e) \cdot \overrightarrow{p}(\mu, e) & si\ \nu \neq Hoja \\ \langle Lap(1, \nu), \ldots, Lap(c, \nu)\rangle & si\ \nu = Hoja \end{cases}$$

donde $Lap(j, \nu)$ es la corrección de Laplace de la frecuencia de los elementos de la clase j en el nodo ν.

Si la inteligibilidad es un requisito, el cálculo estocástico de probabilidades descrito puede parecer muy críptico para un uso general de estos árboles, pero si atendemos a la representación gráfica (ver figura 2), ésta es altamente comprensible.

3. Newton Trees con Valores Faltantes

¿Qué pasa cuando parte de los datos están incompletos como ocurre generalmente con cualquier conjunto de datos de la vida real? Podemos tomar dos caminos posibles ante los datos incompletos: descartar una proporción importante de los datos por incompletos y declarar algunos casos como inclasificables, o adaptar los algoritmos para poder trabajar con atributos con valores faltantes. En la mayoría de los casos, la primera opción es inaceptable.

A continuación vamos a ver las aproximaciones habituales en el área de los árboles de decisión.

Quinlan [9] comparó experimentalmente la efectividad de diversas aproximaciones para el tratamiento de valores faltantes en los árboles de decisión y demostró que la combinación de todos los posibles resultados con un valor faltante en el ejemplo de test en la fase de clasificación proporciona una mejor precisión general que otros enfoques. En esta aproximación, cuando se tiene que clasificar un ejemplo e, éste se propaga por todas las ramas de forma proporcional si en e falta el valor del atributo que se está utilizando en la partición. La fracción de e que se propaga por cada rama es proporcional al número de instancias de entrenamiento (con valores conocidos para el atributo de partición) que han correspondido con cada rama. En el caso de que e tenga un valor conocido para el atributo de la partición, la propagación de dicho ejemplo se hará de forma habitual. Después de que todas las fracciones de le se han propagado, las distribuciones de clase en las hojas se combinan aritméticamente con las fracciones como coeficientes de ponderación. Finalmente, a e se le asigna la clase con mayor probabilidad. El algoritmo C4.5 adopta esta aproximación.

Podemos ver que esta manera de funcionar del algoritmo C4.5 para valores faltantes es justamente la manera que los Newton Trees tratan cualquier instancia, con la salvedad de que para atributos con valores faltantes habríamos de asumir que las distancias a los prototipos son iguales. Haciendo esta modificación e ignorando lo valores faltantes en el aprendizaje tenemos una adaptación sencilla e inmediata de los Newton Trees para trabajar con datasets con valores faltantes en sus atributos. Nuestro algoritmo omite los valores faltantes en los atributos de las instancias de entrenamiento en cuanto a la selección de los prototipos en la construcción del árbol ya que es imposible que maximicen ningún tipo de criterio de ganancia. En cambio, en el momento de la predicción, establecemos constante (igual a 1) el valor de la distancia (para cualquier tipo de datos) a cualquier prototipo para conseguir que solo se tenga en cuenta la masa de cada prototipo en el cálculo de su correspondiente probabilidad mediante la formula anterior:

$$\hat{p}(\nu, e) = \frac{m_\nu}{\sum_{\mu \in Child(Parent(\nu))} m_\mu}$$

Con el fin de clarificar el proceso de clasificación de una instancia con valores faltantes, la figura 2 muestra la representación gráfica de un *Newton Tree* obtenido para el dataset Hepatitis del repositorio UCI [1]. Se observa que todas las particiones del árbol son binarias debido a que se trata de un problema con dos clases, DIE o LIVE. Las dos primeras particiones se han realizado sobre atributos numéricos ($PROTIME$ y $ALK\_PHOSPHHATE$) y las 2 últimas sobre dos atributos nominales (SEX y $FATIGUE$). Los nodos del árbol se representan como bolas de tamaño proporcional a la masa del nodo. Por ejemplo, el nodo izquierdo del primer nivel tiene una masa igual a 17, lo cual quiere decir que 17 instancias de entrenamiento han caído en dicho nodo. Las bolas también muestran, en diferentes colores, la proporción de ejemplos de cada clase y el valor del prototipo en el medio de las mismas. También podemos ver las probabilidades suavizadas en las hojas (en las tablas debajo de cada hoja de la figura de la izquierda). Para facilitar la comprensión de cómo las distintas probabilidades son derivadas, la figura de la derecha muestra las distintas probabilidades internas (vector y nodo) para la instancia $\langle PROTIME = 40; ALK\_PHOSPHATE = 120; SEX = FEMALE; FATIGUE = "UNKNOWN"\rangle$ (nótese el valor faltante para el atributo $FATIGUE$), cuyo vector de probabilidad a nivel raíz es $(0{,}7316, 0{,}2684)$, por lo que esta instancia se clasifica como de clase DIE. Cabe recalcar cómo el árbol ha derivado las probabilidades en la última partición para el atributo $FATIGUE$, centrándose únicamente en las masas de los nodos puesto que la instancia poseía un valor faltante para dicho atributo.

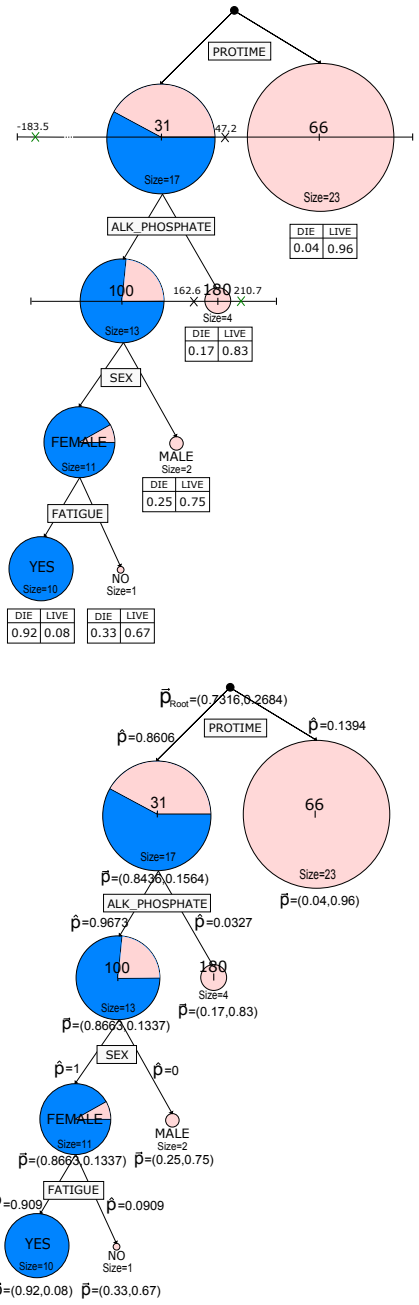

Figura 2: (Arriba) Newton Tree para el dataset hepatitis. (Abajo) Vectores de probabilidad, probabilidades en los nodos y vector de probabilidad total para el ejemplo (PROTIME=40, ALK_PH=120, SEX=FEMALE, FATIGUE=ÜNKNOWN")

#	Dataset	Valores
1	anneal	NO
2	autos_5c	SI
3	balance-scale	NO
4	breast-cancer	SI
5	chess-kr-vs-kp	NO
6	cmc	NO
7	credit-a	SI
8	credit-g	NO
9	diabetes	NO
10	glass	NO
11	heart-statlog	NO
12	hepatitis	SI
13	ionosphere	SI
14	iris	NO
15	monks1W	NO
16	monks2W	NO
17	monks3W	NO
18	mushroom	SI
19	new-thyroid	NO
20	pimaW	NO
21	sonar	NO
22	soybean	SI
23	spectf_train	NO
24	tae	NO
25	tic-tacW	NO
26	vehicle3c	NO
27	vote	NO
28	vowel	NO
29	wine	NO
30	zoo	NO

Cuadro 1: Datasets usados en la experimentación

4. Experimentación

El objetivo de esta sección es comparar los Newton Trees con el algoritmo J48 (sin poda y con suavizado de Laplace en las hojas) implementado en WEKA. Hemos usado *Gain Ratio* como criterio de partición para ambos algoritmos. La evaluación ha sido llevada a cabo mediante 30 datasets originales obtenidos del repositorio UCI[1] (Tabla 1). Se han realizado 20 repeticiones con 5 pliegues de validación cruzada para cada dataset, realizando un total de 100 iteraciones para cada dataset y método (3000 en total). Como métricas de evaluación se han usado tres de las más importantes familias de medidas para evaluar clasificadores (tal y como se explica en [4]): *accuracy* como medida de error cualitativa, *AUC (Area under the Curve)* como medida de la calidad del ránking, y *MSE (Mean Squared Error)* como medida de calidad de calibración y refinamiento. La tabla 2 muestra las medias aritméticas de accuracy, AUC y MSE obtenida por los dos

#	Newton Trees			J48		
	Acc.	AUC	MSE	Acc.	AUC	MSE
1	97,3490	0,5956	0,0107	98,7030	0,5925	0,0065
2	74,9530	0,9145	0,0901	78,7640	0,8965	0,0809
3	79,5520	0,7962	0,1050	78,6880	0,8199	0,0998
4	71,6070	0,6281	0,1992	67,0870	0,6046	0,2242
5	98,5050	0,9975	0,0135	99,3050	0,9988	0,0064
6	50,1720	0,6739	0,2025	49,1100	0,6658	0,2107
7	84,3910	0,9128	0,1132	82,8550	0,9000	0,1256
8	70,3300	0,7202	0,1897	68,2900	0,7016	0,2159
9	71,8630	0,7801	0,1798	72,8070	0,7772	0,1877
10	67,2940	0,7828	0,0901	67,0340	0,7895	0,0879
11	78,0740	0,8626	0,1490	76,1850	0,8398	0,1753
12	78,1930	0,7991	0,1404	78,4830	0,7659	0,1540
13	88,9160	0,9235	0,0916	88,8460	0,9195	0,0917
14	94,7660	0,9938	0,0315	94,0330	0,9710	0,0349
15	93,5230	0,9899	0,0606	92,7690	0,9761	0,0519
16	85,8750	0,9378	0,1124	61,3790	0,6456	0,2348
17	98,6730	0,9926	0,0166	98,6370	0,9909	0,0135
18	99,5500	0,9999	0,0114	100,000	1,0000	0,0010
19	92,6970	0,9854	0,0438	92,3480	0,9237	0,0454
20	71,8630	0,7801	0,1798	72,7750	0,7772	0,1877
21	77,5990	0,8499	0,1538	73,3710	0,7888	0,2162
22	89,9260	0,9381	0,0211	89,4510	0,9324	0,0241
23	67,3120	0,7301	0,2097	71,7500	0,7365	0,2196
24	58,7010	0,7398	0,1877	54,1660	0,7078	0,1996
25	78,1110	0,8526	0,1426	79,3990	0,8699	0,1393
26	72,1210	0,8441	0,1355	73,0240	0,8807	0,1251
27	94,6660	0,9847	0,0398	95,1370	0,9827	0,0355
28	75,3580	0,9671	0,0578	79,5400	0,9157	0,0447
29	94,3840	0,9905	0,0408	92,2070	0,9544	0,0471
30	94,9020	0,7243	0,0252	93,1610	0,7147	0,0234
Media (Todos)	81,7075	0,8563	0,1015	80,6435	0,8347	0,1103
Media (Faltantes)	83,9337	0,8737	0,0953	83,6409	0,8598	0,1002
Media (No Faltantes)	81,0300	0,8509	0,1034	79,7312	0,8270	0,1134

Cuadro 2: Comparativa entre Newton Trees y J48 sin poda y con suavizado de Laplace

Dataset	Datasets sin valores Faltantes					
	Newton Trees			J48		
	Acc.	AUC	MSE	Acc.	AUC	MSE
autos_5c	79,51	0,9	0,08	77,71	0,88	0,08
breast-cancer	73,01	0,64	0,19	67,94	0,61	0,22
credit-a	84,93	0,91	0,11	82,8	0,9	0,13
hepatitis	83,44	0,76	0,11	79,44	0,65	0,15
ionosphere	88,92	0,92	0,09	88,85	0,92	0,09
mushroom	99,99	1	0,02	100	1	0
soybean	89,24	0,98	0,02	91,23	0,98	0,02
Media (Todos)	85,58	0,87	0,09	83,99	0,85	0,1
Media ($c = 2$)	86,06	0,85	0,11	83,8	0,82	0,12
Media ($c \geq 2$)	84,37	0,94	0,05	84,47	0,93	0,05

Cuadro 3: Comparativa entre Newton Trees y J48 para Datasets sin valores faltantes

algoritmos para cada dataset. En la parte final de la tabla se muestran los valores medios para todos los datasets y para únicamente los dataset con valores faltantes. Estas medidas son bastante ilustrativas por sí mismas. Al igual que ocurría en [6] donde los Newton Trees superaban claramente al J48 en los tres tipos de medidas utilizadas (Accuracy, AUC y

Dataset	Datasets con valores Faltantes					
	Newton Trees			J48		
	Acc.	AUC	MSE	Acc.	AUC	MSE
autos_5c	74,95	0,91	0,09	78,76	0,89	0,08
breast-cancer	71,6	0,63	0,19	67,09	0,6	0,22
credit-a	84,4	0,91	0,11	82,85	0,9	0,13
hepatitis	78,2	0,8	0,14	78,48	0,77	0,15
ionosphere	88,92	0,92	0,09	88,85	0,91	0,09
mushroom	99,6	0,99	0,01	100	1	0
soybean	89,93	0,94	0,02	89,45	0,93	0,02
Media (Todos)	83,93	0,87	0,09	83,64	0,86	0,1
Media ($c = 2$)	84,53	0,85	0,11	83,45	0,84	0,12
Media ($c \geq 2$)	82,44	0,93	0,05	84,11	0,91	0,05

Cuadro 4: Comparativa entre Newton Trees y J48 para Datasets con valores faltantes

MSE), encontrando las diferencias más notables en cuanto a valores de AUC, los resultados obtenidos al introducir datasets con valores faltantes han seguido la misma sintonía mejorando los resultados del J48. Cabe destacar los resultados en AUC (medida recomendada para medir la evaluación de PETs [5]). A pesar de ello, debido al limitado número de datasets con valores faltantes utilizados (en este trabajo sólo hemos utilizado 7), es necesaria una experimentación más amplia y exhaustiva del problema para certificar estos buenos resultados.

5. Conclusiones y Trabajo Futuro

Es este trabajo hemos buscado analizar el comportamiento de nuestro algoritmo de clasificación más reciente, los *Newton Trees*, ante conjuntos de datos con valores faltantes obtenidos del repositorio UCI, al constatar que los Newton Trees podrían tratar todas las instancias uniformemente (sean con valores faltantes o no) de la misma manera que el C4.5 trata los valores faltantes, es decir, estocásticamente.

La aproximación usada para el tratamiento de este tipo de datos ha sido por tanto similar a la usada por el algoritmo con el que hemos comparado, C4.5. En esta aproximación, los valores faltantes en los atributos son totalmente ignorados en el proceso de construcción del árbol en cuanto a la selección de prototipos debido a la imposibilidad de maximizar el valor de ningún criterio ganancia (en nuestro caso, *Gain Ratio*). Sin embargo, estos valores si son tenidos en cuenta en la clasificación

estableciendo constante la distancia (tanto si son nominales como numéricos) a los prototipos. Esto hace que la función de atracción se convierta en una función que exclusivamente pondera la masa (es decir la cardinalidad) de los nodos sucesores para establecer, de forma proporcional, las respectivas probabilidades.

Esta caracterización ha proporcionado los resultados esperados (en concordancia con trabajos anteriores con los Newton Trees) cumpliendo nuestras expectativas de homogeneidad (que no trate los valores faltantes de manera muy diferente al resto), transparencia (que este tratamiento sea inteligibile), eficiencia (con un coste bajo) y eficacia (con resultados equiparables a los que los Newton Trees han demostrado para datasets sin valores faltantes).

Referencias

[1] C. Blake, E. Keogh, and C. Merz. UCI repository of machine learning databases, 1998. (http://www.ics.uci.edu/mlearn/MLRepository.html).

[2] V. Estruch. Bridging the gap between distance and generalisation: Symbolic learning in metric spaces. PhD Thesis, DSIC-UPV `http://www.dsic.upv.es/$\sim$vestruch/thesis.pdf`, 2008.

[3] C. Ferri, P. Flach, and J. Hernandez-Orallo. Improving the auc of probabilistic estimation trees. In *Proc. ECML*, volume 2837 of *LNCS*, pages 121–132, 2003.

[4] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recogn. Lett.*, 30(1):27–38, 2009.

[5] Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms - appendices. *IEEE Trans. Knowl. Data Eng.*, 17(3), 2005.

[6] F. Martinez-Plumed, V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Newton trees. extended report. Technical report, DSIC, UPV, http://www.dsic.upv.es/$\sim$flip/NewtonTR.pdf, 2010.

[7] Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[8] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[9] J. Ross Quinlan. Unknown attribute values in induction. In *ML*, pages 164–168, 1989.

[10] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA, 1993.

Parte III

Apéndices

Apéndice A

Pseudocódigo

A.1. Pseudocódigo

Antes de comenzar a describir los diferentes algoritmos se requiere un poco de notación. $C = (c_1, \ldots, c_k)$ es un conjunto de k etiquetas de clase. Un ejemplo (etiquetado) de entrenamiento $e = (x, y)$ se representa mediante una tupla de valores de atributos $x = (x1, \ldots, x_n)$ y una etiqueta de clase $y \in C$. Un ejemplo no etiquetado se representa por sus atributos, es decir $e = (x1, \ldots, xn)$. $Attr_x j$ devuelve el $j - ésimo$ atributo de ejemplo e. Del mismo modo, $Class(e)$ devuelve la clase de ejemplo e. Son necesarias, además, dos funciones para reducir la complejidad en el cálculo de distancias: $Values(S, X_j)$ que devuelve el número de valores diferentes para el atributo x_j y $Card(v, S, X_j)$ devuelve el número de ocurrencias de un valor v en el atributo $x_j$en la muestra S, es decir, $|\{e\ inS : Attr_{x_j} = v\}|$. La función $distance(x, y)$ calcula la distancia entre los valores de los atributos x e y, los cuales deben ser del mismo tipo. La funcion $attraction(e, \pi)$ define la atracción entre una instancia e y un prototipo de π, como se explica en la seccion 2.4.3.1. Por último tenemos dos funciones que trabajan con los nodos del árbol de decisión: $Children(\nu)$ devuelve un conjunto de todos los sucesores del nodo ν y $Parent(\nu)$ devuelve el nodo predecesor del nodo ν .

El algoritmo 1 es el procedimiento principal de nuestro método. Las entradas son un conjunto de datos de entrenamiento de la forma $(x_1, \cdots, x_n)$, $n \geq 1$, un parámetro m que limita el número máximo de nodos sucesores por partición (si se establece $m = 2$ se generan particiones binarias) y un espacio métrico ms donde se definen las distancias ente los distintos atributos del conjunto de ejemplos.

Por lo tanto, $TreeGeneration$ es un algoritmo típico de aprendizaje de árboles de decisión que, en este caso, determina, para cada atributo, una lista ordenada de prototipos que dará lugar a un grupo de hijos para cada nodo. La diferencia principal con los algoritmos de inferencia clásicos de árboles de decisión depende de cuatro funciones: $Compute\_Prototypes$, $Attracts$,

ProbTree y *ProbClass*.

Algorithm 1 $TreeGeneration(S, m, ms)$

Require: S es un conjunto de ejemplos con la forma: $(x_1, \cdots, x_n)$,$n \geq 1$, m es el número máximo de hijos por nodo, ms es el espacio métrico.

```
Heuristics ← 0
ProtList ← 0
if S = 0 then
    return
end if
for all attribute x_j do
    Heuristics_{x_j} ← Optimality(S, x_j, ms) //Gain ratio, GINI, etc.
end for
m_{x_j} ← Argmax_{x_j}(Heuristics)
if Heuristics[m_{x_j}] = 0 then
    return Leaf
else
    ProtList ← ComputePrototypes(m_{x_j}, S, m, ms, C)
    for i = 1 to length(ProtList) do
        TreeGeneration(ProtList[i], m, ms)
    end for
end if
```

La función *Attracts* (ver algoritmo 2) determina que prototipo se le asigna a un nuevo ejemplo. Este algoritmo puede ser implementado de múltiples formas (por ejemplo, teniendo en cuenta la densidad o no), pero hemos optado por la opcion mas simple y efectiva: devolver el prototipo más cercano a la muestra. En caso de empate, se devuelve uno prototipo al azar (de entre los empatados) .

Algorithm 2 $Attracts(e, ProtList, x_j)$

Require: e un ejemplo, $ProtList$ una lista ordenada de prototipos and x_j un atributo seleccionado.

Ensure: Índice del prototipo que atrae e.

```
for i = 1 to lenght(ProtList) do
    v ← attr_{x_j}(e)
    if ∀k ≥ i, distance(attr_{x_j}(ProtList[i], v) ≤ distance(attr_{x_j}(ProtList[k]), v) then
        return i
    end if
end for
```

La función $Compute\_Prototypes$ (ver algoritmo 3) es el más importante. Esta función también puede implementarse de múltiples maneras. La imple-

mentación realizada selecciona el mejor prototipo (de forma que el prototipo seleccionado (valor del atributo) será aquel que minimice las distancias con el resto de elementos del atributo), se elimina su clase y el valor del atributo y se busca el suiguiente mejor prototipo para una clase y valor diferente para el atributo, y así sucesivamente hasta que se alcanza el límite m (establecido por el usuario) o no quedan más clases o valores de atributo. Una vez que la lista de los prototipos se ha calculado, cada ejemplo se asocia con el prototipo que le "atrae".

Algorithm 3 $Compute\_Prototypes(x_j, S, m, ms, C)$

Require: x_j es un atributo, S es el dataset, m es el máximo numero de hijos del prototipo, ms es el espacio métrico, C es el conjunto de clases.
Ensure: Lista ordenada multidimensional de prototipos.

for all class $c \in C$ **do**
  $S_c \leftarrow \{e \in S : class(e) = c\}$
  if $S_c \neq 0$ **then**
    $V_c \leftarrow Values(x_j, S_c)$
    for all element $v \in V_c$ **do**
      $MeanDistance_c[v] \leftarrow \frac{\sum_{i \in V_c} distance(v,i) * Card(i, S_c, x_j)}{|S_c|}$
    end for
  end if
end for
$UV \leftarrow 0$
$ProtList \leftarrow 0$
$RC \leftarrow C$
$Values \leftarrow Values(x_j, S)$
$Prots \leftarrow min(|C|, m, Values)$
for $k = 1$ to $Prots$ **do**
  $BestProt \leftarrow Argmin_e\{MeanDistance_c[Attr_{x_j}(e)]\}_{c \in RC, e \in S, Attr_{x_j}(e) \notin UV}$
  $RC \leftarrow RC - Class(BestProt)$
  $ProtList \leftarrow append(ProtList, BestProt)$
  $UV \leftarrow UV \cup Attr_{x_j}$(BestProt)
end for
for $i = 1$ to $length(ProtList)$ **do**
  $\hat{S}_i \leftarrow \{e \in S : i = Attracts(e, ProtList, x_j)\}$
  $ProtList_i \leftarrow ProtList_i \cup \hat{S}_i$
end for
return ProtList

Hay que tener en cuenta que el procedimiento anterior es independiente del tipo de los atributos (todos los atributos se manejan de una forma similar). De hecho, se puede aplicar a cualquier tipo de atributo y no sólo a

atributos nominales o numéricos como sucede en los algoritmos de aprendizaje clásicos de árboles de decisión (en los cuales las particiones de los datos numéricos se realizan de forma muy diferente a las de los nominales). Por ejemplo, c4.5, evalúa todos los puntos de corte intermedios obtenidos a partir de los distintos valores del atributo seleccionado para la partición en el conjunto de datos con el fin de seleccionar la mejor partición para un atributo numérico. Análogamente, en el caso de los atributos nominales, C4.5 evalúa una partición para cada posible valor del atributo. Por lo tanto, esto hace que los criterios de partición utilizados en los Newton Trees sean más eficiente que los utilizados en los algoritmos de aprendizaje clásicos de árboles de decisión. Además, es importante tener en cuenta que las distancias se calculan entre los valores de los atributos y no entre ejemplos. Esta característica es fundamental para la eficiencia.

Hasta este punto, las funciones descritas se utilizan para la generación de nuestro sistema de aprendizaje. A continuación se van a describir las dos últimas funciones que componen nuestro método y que son las que proporcionan la nueva funcionalidad en el proceso de clasificación para las instancias. La primera de ellas, *ProbTree* (Algoritmo 4), es la responsable de calcular, para cada ejemplo de test utilizado para probar el sistema, las probabilidades asociadas a cada nodo de árbol. Como se explica en este trabajo, usamos los Newton Trees para estimar las probabilidades de una manera estocástica. Con $\hat{p}(\nu, e)$ se denota la probabilidad de que e caiga en un nodo nu (procedente de sus padres), que se deriva de la función *attraction*.

Por último, *ProbClass* (Algoritmo 5) devuelve un vector de estimación estocástica de probabilidad que se obtiene al recorrer el Newton Tree generado (con las probabilidades asociadas a cada nodo calculadas con la función anterior), de las hojas a la raíz.

Algorithm 4 $ProbTree(e, Children(\nu))$

Require: e un ejemplo, $Children(\nu)$ un conjunto de nodos sucesores (Prototipos).

1: **if** $|Children(v)| = 0$ **then**
2: **return**
3: **else**
4: **for all** $k \in Children(\nu)$ **do**
5: $\hat{p}(k, e) \leftarrow \frac{attraction(e,k)}{\sum_{\mu \in Children(Parent(k))} attraction(e,\mu)}$
6: $ProbTree(e, (Children(k))$
7: **end for**
8: **end if**

Algorithm 5 $ProbClass(e, Children(\nu))$

Require: $Children(\nu)$ un conjunto de nodos sucesores (Prototipos).
Ensure: un vector de probabilidades de clase en la raíz del árbol.

1: **if** $|Children(\nu)| = 0$ **then**
2: $\quad \overrightarrow{p}(\nu, e) \leftarrow \langle Laplace(1, \nu), ..., Laplace(c, \nu) \rangle \cdot \hat{p}(\nu, e)$
3: $\quad$ **return** $\overrightarrow{p}(\nu, e)$
4: **else**
5: $\quad$ **for all** $k \in Children(\nu)$ **do**
6: $\quad\quad \overrightarrow{p}(\nu, e) \leftarrow \overrightarrow{p}(v, e) + ProbClass(e, Children(k))$
7: $\quad$ **end for**
8: **end if**
9: **return** $\overrightarrow{p}(\nu, e) \cdot \hat{p}(\nu, e)$

Bibliografía

[1] Manish Mehta 0002, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 1996.

[2] Rakesh Agrawal, Manish Mehta 0002, John C. Shafer, Ramakrishnan Srikant, Andreas Arning, and Toni Bollinger. The quest data mining system. In *KDD*, pages 244–249, 1996.

[3] I. Alvarez and S. Bernard. Ranking cases with decision trees: a geometric method that preserves intelligibility. In *IJCAI*, pages 635–640, 2005.

[4] I. Alvarez, S. Bernard, and G. Deffuant. Keep the decision tree and estimate the class probabilities using its decision boundary. In *IJCAI*, pages 654–659, 2007.

[5] Anna Atramentov, Hector Leiva, and Vasant Honavar. A multi-relational decision tree learning algorithm - implementation and experiments. In TamÃ¡s HorvÃ¡th, editor, *ILP*, volume 2835 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2003.

[6] Adi Ben-Israel and Cem Iyigun. Probabilistic d-clustering. *J. Classification*, 25(1):5–26, 2008.

[7] Hendrik Blockeel. Top-down induction of first order logical decision trees. *AI Commun.*, 12(1-2):119–120, 1999.

[8] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[9] W. Buntine. Learning classification trees. *Stats. and Computing*, 2(2):63–73, 1992.

[10] Ian Davidson. Visualizing clustering results. In *SIAM Conf. on Data Mining*, 2002.

[11] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.

[12] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, 2000.

[13] V. Estruch. Bridging the gap between distance and generalisation: Symbolic learning in metric spaces. PhD Thesis, DSIC-UPV `http://www.dsic.upv.es/$\sim$vestruch/thesis.pdf`, 2008.

[14] Usama M. Fayyad and Keki B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[15] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[16] C. Ferri, P. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the roc curve. In *Int. Conf. on Machine Learning*, pages 139–146, 2002.

[17] C. Ferri, P. Flach, and J. Hernandez-Orallo. Improving the auc of probabilistic estimation trees. In *Proc. ECML*, volume 2837 of *LNCS*, pages 121–132, 2003.

[18] C. Ferri, J. Hernández-Orallo, and M. Ramírez-Quintana. From ensemble methods to comprehensible models. In *Discovery Science*, pages 223–234. Springer, 2002.

[19] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[20] J. Gomez, D. Dasgupta, and O. Nasraoui. A new gravitational clustering algorithm. In *Int. Conf. on Data Mining*, page 83. Society for Industrial & Applied, 2003.

[21] D.J. Hand and R.J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.

[22] R.F. Hespos and P.A. Strassmann. Stochastic decision trees for the analysis of investment decisions. *Management Science*, 11(10):244–259, 1965.

[23] Jon Hill, Matthew Hambley, Thorsten Forster, Muriel Mewissen, Terence M. Sloan, Florian Scharinger, Arthur S. Trew, and Peter Ghazal. Sprint: A new parallel framework for r. *BMC Bioinformatics*, 9, 2008.

[24] Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms - appendices. *IEEE Trans. Knowl. Data Eng.*, 17(3), 2005.

[25] M. Indulska and ME Orlowska. Gravity based spatial clustering. In *Proc. Int. Sym. on Advances in geographic information systems*, page 130, 2002.

[26] John F. Elder IV and Daryl Pregibon. A statistical perspective on knowledge discovery in databases. In *Advances in Knowledge Discovery and Data Mining*, pages 83–113. 1996.

[27] M.J. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *STOC*, pages 459–468, 1996.

[28] R. Kohavi. Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid. In *Int. Conf. on Knowledge Discovery and Data Mining*, volume 7, 1996.

[29] R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Int. Conf. on Machine Learning, ICML*, pages 161–169, 1997.

[30] C.X. Ling and R.J. Yan. Decision tree with better ranking. In *International Conference on Machine Learning*, volume 20-2, page 480, 2003.

[31] F. Martinez-Plumed, V. Estruch, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Newton trees. extended report. Technical report, DSIC, UPV, http://www.dsic.upv.es/∼flip/NewtonTR.pdf, 2010.

[32] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[33] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[34] L. Peng, B. Yang, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.

[35] Foster J. Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[36] Foster J. Provost and Ron Kohavi. Guest editorsíntroduction: On applied research in machine learning. *Machine Learning*, 30(2-3):127–132, 1998.

[37] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[38] J. Ross Quinlan. Unknown attribute values in induction. In *ML*, pages 164–168, 1989.

[39] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA, 1993.

[40] J. Ross Quinlan and R. Mike Cameron-Jones. Induction of logic programs FOIL and related systems. *New Generation Computing*, 13(3 and 4):287–312, 1995.

[41] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2008.

[42] C. Thorton. Book reviews. truth from trash: How learning makes sense. *Pattern Anal. Appl.*, 6(1):88–89, 2003.

[43] Berwin A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*, 1993.

[44] M. Umano, H. Okamoto, I. Hatono, H. Tamura, F. Kawachi, S. Umedzu, and J. Kinoshita. Fuzzy decision trees by fuzzy ID 3 algorithm and its application to diagnosis systems. In *3rd IEEE Conf. on Fuzzy Systems*, pages 2113–2118, 1994.

[45] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Pub, 2005.

[46] WE Wright. Gravitational clustering. *Pattern Recognition*, 9(3):151–166, 1977.

[47] H. Zhang and J. Su. Learning probabilistic decision trees for AUC. *Pattern Recognition Letters*, 27(8):892–899, 2006.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes