ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

LIBRERÍA EN R PARA LA GENERACIÓN HEURÍSTICA DE ÁRBOLES DE DECISIÓN

R-BASED LIBRARY FOR HEURISTIC GENERATION OF DECISION TREES

Realizado por
Marcos Vergara Gómez
Tutorizado por
Eduardo Guzmán de los Riscos
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre 2016

Fecha defensa:
El Secretario del Tribunal

AGRADECIMIENTOS

En primer lugar, quiero darle las gracias más sinceras a mi director de Trabajo de Fin de Grado, Eduardo, por haberme dado la oportunidad para realizar este trabajo y poder aprender de él, pero sobre todo agradecer toda su ayuda y el magnífico trato recibido.

Gracias a todos mis amigos, compañeros y a todas aquellas personas que en algún momento me hayan apoyado y hayan estado dispuestas a ayudarme.

Gracias a mi tía Juana y a mis abuelos, María y José, que en paz descanse, por todo el apoyo diario desde pequeño y por haber dado una gran parte de su vida por hacerme mejor persona.

Gracias a mi hermano Eloy, por estar apoyándome en los buenos y malos momentos.

Gracias a mi padre, por aquellos días en que me obligabas a machacar y machacar la lección de pequeño, parece que por fin empieza a verse la recompensa.

Gracias a mi madre, por intentar ayudarme siempre en todo y haberme enseñado que trabajando, luchando y confiando en uno mismo, todo se acaba consiguiendo.

Resumen: Se ha implementado una API en R para la generación de árboles de decisión. Se ha trabajado con un fichero de datos, con una serie de síntomas que han provocado una enfermedad, que ha sido depurado posteriormente. Hemos implementado una función en la que, a partir del fichero de datos y un conjunto de síntomas ordenados a nuestra elección, se crea el árbol de decisión que en cada elección escogerá el síntoma indicado anteriormente.

El siguiente paso ha sido crear dos funciones de predicción que nos sirvan para comparar los resultados reales con los resultados obtenidos por el árbol de decisión. Tras construir el árbol y las funciones de predicción, crearemos una función que unificará ambas tareas.

Este método será la validación cruzada que nos permitirá obtener mejores resultados. Finalmente, se ha realizado una comparación de los resultados obtenidos con nuestro método y los resultados extraídos por los algoritmos existentes hasta la actualidad. La implementación de todas las funciones se ha llevado a cabo con el programa estadístico R, mientras que para la comparación de resultados se ha utilizado la herramienta de minería de datos Weka.

Palabras clave: Árbol de decisión, R, Weka, Clasificación, Predicción, Validación Cruzada

Abstract: An API has been implemented in R to generate decision trees. It has worked with a data file, with a series of symptoms that have caused a disease, which has been refined later. We have implemented a function in which from the data file and a set of ordered symptoms of our choice, a decision tree that in every choice will pick the symptom previously mentioned is created.

The next step was to create two prediction functions, which help us to compare actual results with results obtained by the decision tree. After building the tree and prediction functions, we will create a function that will unify both tasks.

This method will be the cross-validation that will enable us to obtain better results. Finally, it has been made a comparison between the results obtained with our method and the results extracted from existing algorithms until now. The implementation of all functions has been carried out with the statistical program R, while for the comparison of results has been used the data mining tool Weka.

Keywords: Decision Tree, R, Weka, Classification, Prediction, Cross-validation

Índice

1. INTRODUCCIÓN

1.1.CONTEXTO

El concepto de valor en sanidad identifica la mejora de la calidad de vida del paciente a través de determinados atributos, como son la prolongación de la vida, la comodidad en la asistencia, la rapidez en la atención, la seguridad en la asistencia, la asistencia menos cruenta, la mejora de la eficiencia o la reducción de costes.

Para permitir que la sanidad española sea de las mejores del mundo y hacer que nuestra sanidad sea sostenible en el tiempo y mejore la calidad de vida de los pacientes, se hace imprescindible renovar el sistema y, en este punto, las inversiones en tecnologías de diagnóstico precoz y tecnologías de la información juegan un papel esencial.

La interacción de estas tecnologías, cada vez más avanzadas, impacta no sólo en la detección temprana de enfermedades, sino también en la mejora de los procesos asistenciales, en los mecanismos de comunicación entre agentes sanitarios y en la agilización de los procesos internos del sistema. Se trata de distribuir mejor los recursos para tener una medicina más personalizada y más eficiente.

Está demostrado que las innovaciones en tecnología sanitaria ahorran tiempo a paciente y clínico, lo que redunda en un ahorro de costes; mejoran el acceso al sistema sanitario, la efectividad del diagnóstico, el control de la enfermedad o el tratamiento. Todo ello beneficia la toma de decisiones, la calidad del servicio y, lo que es más importante, la vida de los pacientes y su entorno. Esto último vuelve a suponer ahorros de costes. Por todas estas razones, no hay duda de que la inversión en tecnología debería convertirse en una pieza clave del sistema sanitario futuro.

Debido a todo esto, en este TFG (trabajo de fin de grado) queremos centrarnos en la efectividad del diagnóstico, realizando un programa mediante el cual obtendremos un árbol de decisión que nos indicará cual será la decisión más correcta a tomar para cada paciente y qué diagnóstico realizar en cada caso.

Este proyecto nos servirá para validar el trabajo realizado de forma paralela en el marco de una tesis doctoral. Utilizaremos el lenguaje de programación R para implementar una librería, en la cual crearemos un árbol de decisión, validaremos los resultados obtenidos para probar la eficiencia del árbol. También utilizaremos un programa para minería de datos como es Weka, para comparar los resultados obtenidos con nuestro árbol con los de otros métodos de obtención de árboles de decisión.

1.2. OBJETIVOS

Los principales objetivos de este trabajo de fin de grado son:

1. Obtener una visión general de los algoritmos existentes en la actualidad sobre los árboles de decisión y estudiar si existen librerías en R o en Weka que los implementen.
2. Implementar una API en R, que contenga una serie de funciones para crear árboles de decisión de una manera no tan rígida como en los algoritmos existentes hoy día.
3. Implementar un árbol de decisión, a partir de unos datos que le pasemos y unas variables dadas, que nos dirán la decisión a tomar en el árbol en cada momento.
4. Realizar una función de predicción, mediante la cual al pasar un árbol de decisión, esta función diga si el valor obtenido por el árbol coincide con el valor real que tenemos en el conjunto de datos.
5. Implementar el método de validación cruzada para este tipo de árbol y esta función de predicción que hemos construido.
6. Realizar una comparación entre los métodos existentes y el método creado.

1.3. ESTRUCTURA DE LA MEMORIA

Tras haber visto una pequeña introducción y definir los objetivos principales, la memoria estará estructurada en la consecución de estos objetivos que hemos planteado. En cada objetivo, realizaremos una breve introducción, explicaremos detalladamente el funcionamiento de las funciones creadas y cómo fue su desarrollo. Finalmente, pondremos un ejemplo con cada función y comentaremos los resultados obtenidos. Una parte importante de la memoria, será la comparación con otros métodos existentes, una comparación de los resultados obtenidos y ver si lo que hemos realizado es eficiente.

1.4. TECNOLOGÍAS UTILIZADAS

1.4.1. R

R es un lenguaje y entorno de programación para análisis estadístico y gráfico. Es un proyecto de software libre y posiblemente sea el lenguaje más utilizado en la investigación estadística. R forma parte del sistema GNU y se distribuye bajo la licencia GNU GPL. Fue desarrollo en sus inicios por Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland en 1993.

R presenta una amplia variedad de herramientas estadísticas (modelos lineales y no lineales, tests estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento, etc.) y gráficas. Al tratarse de un lenguaje de programación, permite que sus usuarios puedan definir sus propias funciones. Entre otras de sus funciones destacamos que puede integrarse con distintas bases de datos, que presenta una gran capacidad gráfica, con la que nos permite generar gráficos de alta calidad y la posibilidad de usarla como herramienta para el cálculo numérico.

La figura 1 representa la consola de R:

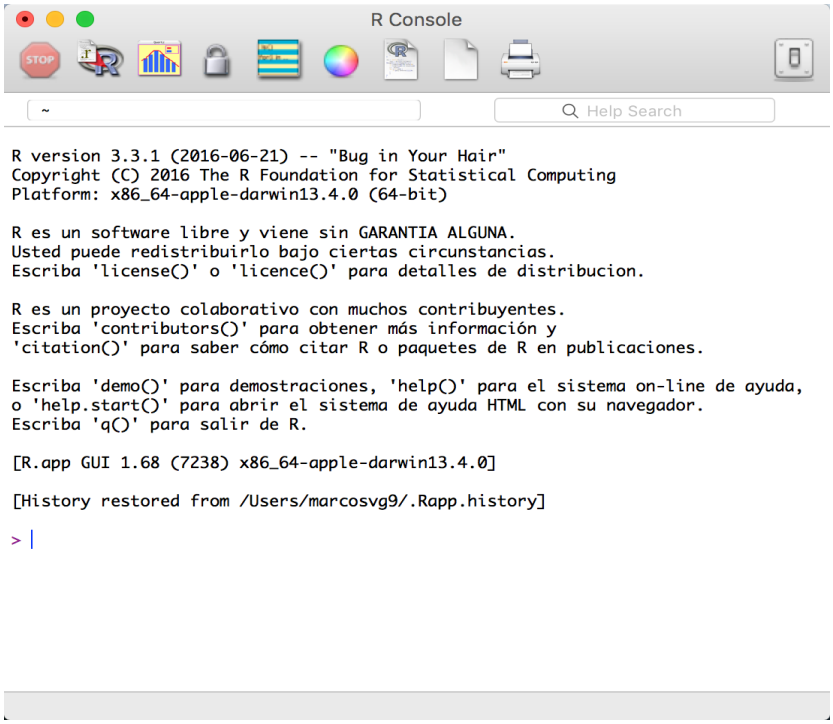


Figura 1: Consola de R

Pero nosotros vamos a usar RStudio, que es más completo que la simple consola de R. RStudio es un entorno de desarrollo integrado para R. Incluye una consola, editor de sintaxis que apoya la ejecución de código, y también tiene herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

RStudio está disponible para Windows, Mac y Linux, y tiene la misión de proporcionar el entorno informático estadístico R. Permite un análisis y desarrollo para que cualquiera pueda analizar los datos con R.

En la figura 2, veremos el aspecto de RStudio:

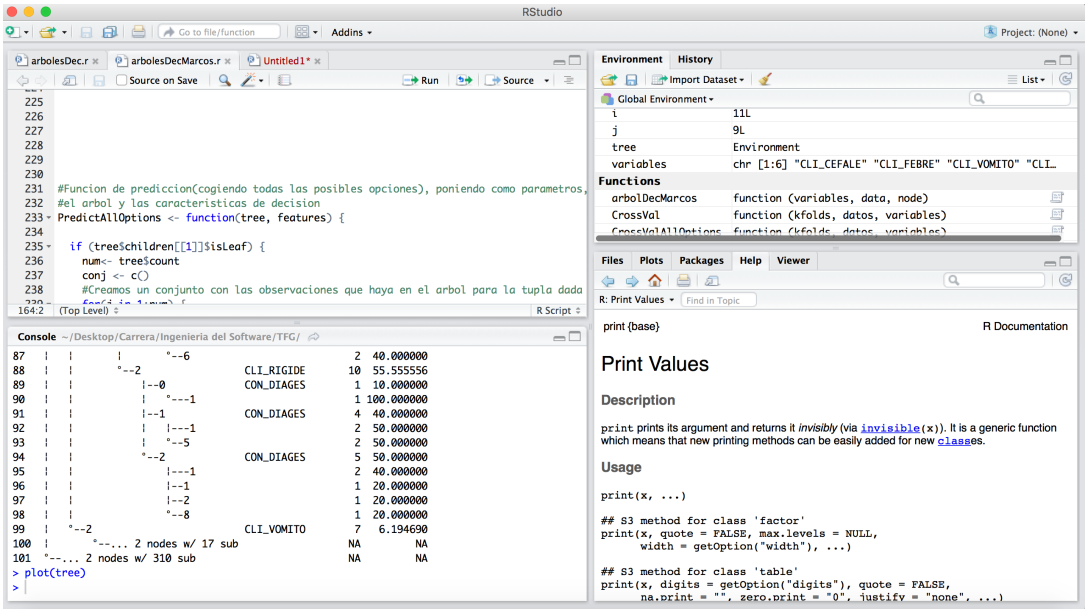


Figura 2: Aspecto de RStudio

1.4.2. WEKA

Weka es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato (Nueva Zelanda). El paquete presenta una colección de herramientas de visualización y algoritmos para el análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades.

Las principales ventajas de Weka son: estar disponible de forma libre bajo la licencia pública general GNU, ser portable ya que está implementado en Java y puede correr prácticamente en cualquier plataforma, tener una amplia colección de técnicas para el preprocesamiento de datos y el modelado, tener una buena interfaz gráfica de usuario.

Finalmente, nos gustaría comentar que se ha desarrollado una interfaz llamada RWeka, que sirve para interactuar R con Weka, la cual permite leer y escribir ficheros en el formato arff y enriquecer R con los algoritmos de minería de datos existentes en la plataforma Weka.

1.5. METODOLOGÍA

La metodología seleccionada para la realización de este TFG se denomina prototipado simple, que consiste en planear objetivos parciales a través de las distintas fases del proyecto.

Un prototipo es una versión previa del sistema, y podremos ir añadiendo posteriormente las distintas funcionalidades con lo que tendremos un método más fácil a la hora de trabajar y poder ir corrigiendo errores.

1.6. FASES DEL PROYECTO

Las distintas etapas en las que se divide el Trabajo de Fin de Grado son:

- Obtención de los datos de un estudio realizado sobre una muestra de pacientes, de los cuales tendremos valores sobre una serie de síntomas y posibles diagnósticos.
- Preprocesamiento del fichero de datos sobre la muestra de pacientes.
- Estudio sobre las diferentes técnicas existentes hasta la actualidad de árboles de decisión y otras técnicas de clasificación que nos resulten de interés.
- Implementación de un algoritmo que cree un árbol de decisión.
- Creación de un algoritmo para validar los resultados obtenidos con el árbol de decisión creado, usando la técnica de validación cruzada o cross-validation.
- Comparación de resultados con los obtenidos con otros métodos existentes de árboles de decisión.
- Elaboración de la memoria del Trabajo de Fin de Grado (aunque es una tarea para el final del proyecto la iremos realizando en paralelo junto al resto de tareas).

2. DEFINICIÓN Y PREPROCESAMIENTO DE LOS DATOS

2.1 DEFINICIÓN DE LOS DATOS A UTILIZAR

Durante todo el trabajo a realizar, vamos a trabajar con un fichero de datos de tipo .xls que estará compuesto por 1000 filas y 12 columnas. Las 1000 filas estarán referidas a las diferentes observaciones de los individuos de los diversos síntomas que puedan o no tener. Mientras que las 12 columnas nos indicarán los síntomas que podrán tener los individuos.

Los datos estarán referidos a casos de meningitis, ocurridos en Bahía (Brasil), durante los últimos 10 años, proporcionados por DATASUS.

La meningitis es una enfermedad infecciosa con alta tasa de mortalidad, especialmente en los países menos desarrollados. DATASUS, el departamento de Informática del sistema de salud pública brasileño (SUS), almacena y pone a disposición de profesionales, investigadores y de administradores de salud pública una enorme cantidad de datos sobre la salud, que es con frecuencia infrautilizado. Un estudio retrospectivo de estos datos para trazar un modelo de comportamiento de la enfermedad en una particular región y la simulación de posibles escenarios podían ser bastante útiles en la toma de decisiones para reducir la incidencia y la mortalidad de esta enfermedad.

Estos síntomas que pueden tener los individuos son:

1. CLI_CEFALE: Cefalea
2. CLI_FEBRE: Fiebre
3. CLI_VOMITO: Vómito
4. CLI_CONVUL: Convulsiones
5. CLI_RIGIDE: Rigidez
6. CLI_KERNIG: Kernig
7. CLI_ABAULA: Abultamiento
8. CLI_COMA: Coma
9. CLI_PETEQU: Petequias

Asimismo, el fichero contiene la siguiente información adicional

- CLASSI_FIN: Clasificación final
- CON_DIAGES: Diagnóstico

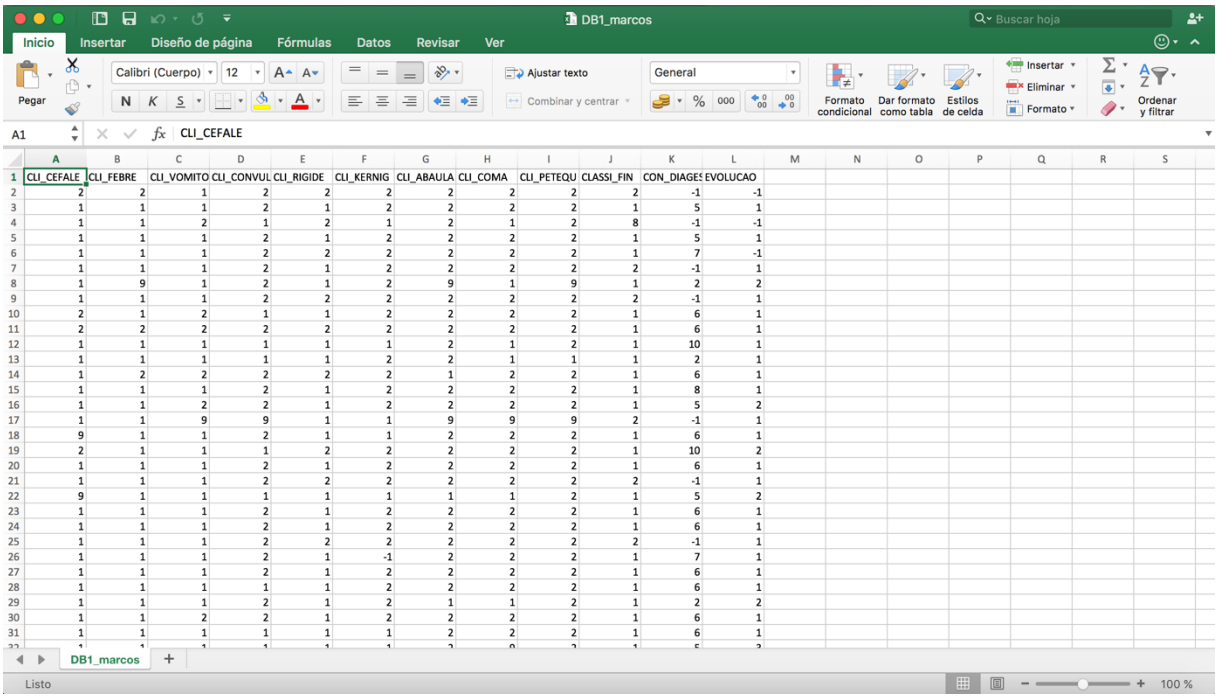
- EVOLUCAO: Evolución

Los primeros 9 síntomas pueden tomar los valores -1 (valor inválido), 1 (tiene el síntoma), 2 (no tiene el síntoma), 9 (valor inválido), donde como veremos posteriormente fusionaremos los valores -1 y 9 en el valor 0, para reducir la complejidad.

La variable “CLASSI_FIN” indica si el paciente tiene o no meningitis; “CON_DIAGES”, indica el tipo de meningitis, toma los valores -1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Si presenta el valor -1 indica que el dato no está. Los valores 1, 2, 3 indican que el individuo presenta meningitis meningocócica. El valor 4 significa que el individuo tiene meningitis tuberculosa. El valor 6 significa que no está especificado. El resto de valores indica que el individuo presenta otros tipos de meningitis.

La variable “EVOLUCAO” nos indica la evolución del individuo y toma los valores -1 (vacío), 1 (curado), 2 (muerto por meningitis), 3 (muerto por otras causas), 9 (no se sabe).

En la figura 3, veremos una vista previa del fichero de datos.



CLI_CEFALE	CLI_FEBRE	CLI_VOMITO	CLI_CONVUL	CLI_RIGIDE	CLI_KERNIG	CLI_ABAULA	CLI_COMA	CLI_PETEQU	CLASSI_FIN	CON_DIAGES	EVOLUCAO
2	2	1	2	2	2	2	2	2	2	-1	-1
1	1	1	2	1	2	2	2	2	1	5	1
1	1	2	1	2	1	2	1	2	8	-1	-1
1	1	1	2	1	2	2	2	2	1	5	1
1	1	1	2	2	2	2	2	2	1	7	-1
1	1	1	2	1	2	2	2	2	2	-1	1
1	9	1	2	1	2	9	1	9	1	2	2
1	1	1	2	2	2	2	2	2	2	-1	1
2	1	2	1	1	2	2	2	2	1	6	1
2	2	2	2	2	2	2	2	2	1	6	1
1	1	1	1	1	1	2	1	2	1	10	1
1	1	1	1	1	2	2	1	1	1	2	1
1	2	2	2	2	2	1	2	2	1	6	1
1	1	1	2	1	2	2	2	2	1	8	1
1	1	2	2	1	2	2	2	2	1	5	2
1	1	9	9	1	1	9	9	9	2	-1	1
9	1	1	2	1	1	2	2	2	1	6	1
2	1	1	1	2	2	2	2	2	1	10	2
1	1	1	2	1	2	2	2	2	1	6	1
1	1	1	2	2	2	2	2	2	2	-1	1
9	1	1	1	1	1	1	1	2	1	5	2
1	1	1	2	1	2	2	2	2	1	6	1
1	1	1	2	1	2	2	2	2	1	6	1
1	1	1	2	2	2	2	2	2	2	-1	1
1	1	1	2	1	-1	2	2	2	1	7	1
1	1	1	2	1	2	2	2	2	1	6	1
1	1	1	1	1	2	2	2	2	1	6	1
1	1	1	2	1	2	1	1	2	1	2	2
1	1	2	2	1	2	2	2	2	1	6	1
1	1	1	1	1	1	2	2	2	1	6	1

Figura 3: Vista previa del fichero de datos

2.2 PREPROCESAMIENTO DE LOS DATOS

Una vez que se tienen los datos que vamos a utilizar, realizaremos un par de cambios para menor complejidad y poder acercarnos a los resultados que deseamos obtener.

En primer lugar, eliminaremos la variable “EVOLUCAO”, puesto que el objetivo es, en la medida de lo posible, diagnosticar si un paciente tiene meningitis y el tipo de meningitis.

En segundo lugar, para los primeros 9 síntomas, en los individuos que tengan el valor -1 o 9, lo convertiremos en 0. Con esta medida, estas 9 variables tendrán un valor posible menos y nos ayudará posteriormente a reducir el tamaño del árbol de decisión.(Decir lo que significaban -1 y 9).

Finalmente, la última medida que vamos a tomar, es convertir todos los valores de la tabla, de Número a Factor, debido a que nos ayudará en la construcción del árbol de decisión. El principal motivo de esta medida es que si dejamos la tabla con valores de tipo Número, las divisiones se realizarán con números reales, por ejemplo, CLI_CEFALE > 1.94, y no es lo que queremos conseguir en la construcción del árbol.

3.ESTADO DEL ARTE

3.1 INTRODUCCIÓN

El objetivo de este capítulo es presentar aquellas tecnologías y trabajos relacionados con el presente proyecto. La finalidad es facilitar al lector la comprensión de los siguientes capítulos y esclarecer todos los conceptos de forma que pueda valorar adecuadamente las contribuciones del proyecto.

Concretamente, vamos a hablar sobre los árboles de decisión y comentar los diferentes algoritmos que han sido creados hasta la fecha. Finalmente, hablaremos sobre algunas de las librerías sobre árboles de decisión creadas hasta la actualidad.

3.2 CONCEPTO DE ÁRBOL DE DECISIÓN

Los árboles de decisión se pueden aplicar prácticamente a todo. Los sistemas de aprendizaje basados en árboles de decisión son posiblemente el método más fácil de utilizar y de entender. Un árbol de decisión es un conjunto de condiciones organizadas en una estructura jerárquica, de tal manera que la decisión final a tomar se puede determinar siguiendo las condiciones que se cumplen desde la raíz del árbol hasta alguna de sus hojas. Los árboles de decisión se utilizan desde hace siglos, y son especialmente apropiados para expresar procedimientos médicos (como utilizaremos durante este trabajo), legales, comerciales, estratégicos, matemáticos, lógicos, etc.

En la figura 4, vemos dos ejemplos simples de árboles de decisión:

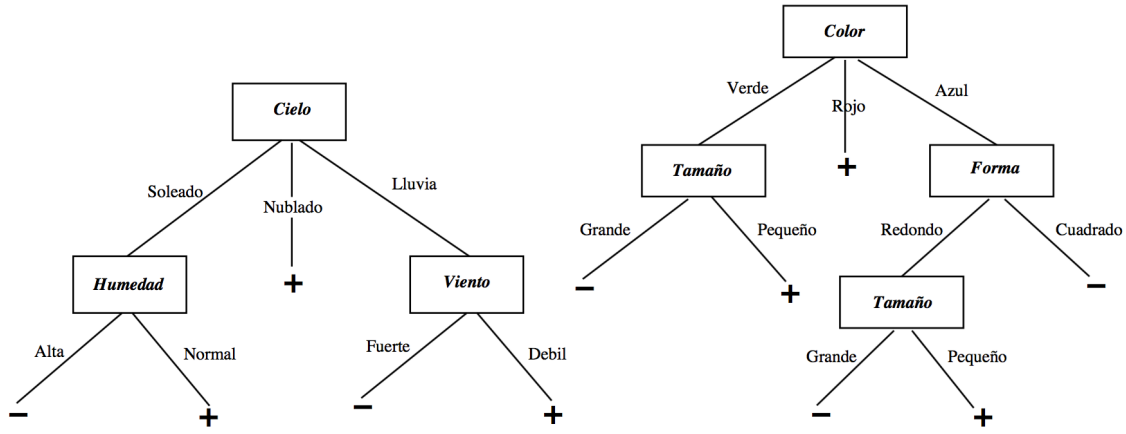


Figura 4: Ejemplos de árboles de decisión

Una de las ventajas más importantes de los árboles de decisión es que, en su forma más general, las opciones posibles a partir de una determinada condición son excluyentes. Esto permite analizar una situación y, siguiendo el árbol de decisión apropiadamente, llegar a una sola acción a decisión a tomar. Estos algoritmos se llaman algoritmos de partición o algoritmos de “divide y vencerás”.

Otra característica importante de los primeros algoritmos de aprendizaje de árboles de decisión es que una vez que elijamos la partición, dicha partición no se podrá cambiar, aunque más tarde se pueda llegar a pensar que ha sido una mala elección. Por lo tanto, uno de los aspectos más importantes en los sistemas de aprendizaje de árboles de decisión es el llamado criterio de partición, ya que una mala elección de la partición(sobre todo, en las partes superiores del árbol) generará un peor árbol.

En la figura 5, vemos en pseudolenguaje el algoritmo básico de los árboles de decisión:

```
ALGORITMO Partición(N:nodo, E:cjto. de ejemplos)
  SI todos los ejemplos E son de la misma clase c ENTONCES
     Asignar la clase c al nodo N.
     SALIR;  // Esta rama es pura, ya no hay que seguir partiendo. N es hoja.
  SI NO:
    Particiones := generar posibles particiones.
    MejorPartición:= seleccionar la mejor partición según el criterio de partición.
    PARA CADA condición i de la partición elegida.
       Añadir un nodo hijo i a N y asignar los ejemplos consistentes a cada hijo (Ei).
       Partición(i, Ei).      // Realizar el mismo procedimiento global con cada hijo.
    FIN PARA
  FIN SI
FIN ALGORITMO

Para clasificar un cjto de ejemplos E, se invoca con la llamada Partición(R,E),
donde R es un nodo raíz de un árbol por empezar.
```

Figura 5: Algoritmo de árbol de decisión en pseudolenguaje

Lo que hace el algoritmo es simplemente, ir construyendo el árbol (desde el árbol que sólo contiene la raíz) añadiendo particiones y los hijos resultantes de cada partición. Lógicamente, en cada partición, los ejemplos se van dividiendo entre los hijos. Finalmente, se llega a la situación en la que todos los ejemplos que caen en los nodos inferiores son de la misma clase y esa rama ya no sigue creciendo. La única condición que hay que exigir es que las particiones al menos separen ejemplos en distintos hijos.

Como acabamos de explicar, los dos puntos más importantes para que el algoritmo funcione bien son los siguientes:

- Particiones a considerar: un conjunto de condiciones exhaustivas y excluyentes. Cuantas más particiones permitamos más expresivos podrán ser los árboles de decisión generados y, probablemente, más precisos. No obstante, cuantas más particiones elijamos, la complejidad del algoritmo será mayor. Por tanto, debemos encontrar un buen compromiso entre expresividad y eficiencia. Debido a esto, la mayoría de algoritmos de aprendizaje de árboles de decisión sólo permiten un juego muy limitado de particiones. Por ejemplo, el C4.5 contiene un solo tipo de partición para los atributos nominales y un solo tipo de partición para los atributos numéricos.

- Criterio de selección de particiones: incluso con sólo los dos tipos de particiones sencillas vistas, el número de particiones posibles en cada caso puede dispararse(si existen n atributos y m valores posibles para cada atributo, el número de

particiones posibles es de n·m). Los algoritmos clásicos de aprendizaje de decisión son voraces, en el sentido de que una vez elegida la partición se continúa hacia abajo la construcción del árbol y no vuelven a plantearse las particiones ya construidas. Por tanto, se debe buscar un criterio que permita realizar una buena elección de la partición que parece más prometedora y que esto se haga sin demasiado esfuerzo computacional.

Esto es lo que diferencia fundamentalmente los distintos algoritmos de árboles de decisión hasta la actualidad, como ID3, C4.5, etc.

3.3 ALGORITMOS EXISTENTES SOBRE ÁRBOLES DE DECISIÓN Y MÉTODOS BAYESIANOS

Antes de comenzar a explicar los distintos algoritmos sobre árboles de decisión, conviene recordar una serie de conceptos estadísticos que nos serán de utilidad.

Dos variables son independientes si se cumple que:

$$P(x, y) = P(x)P(y)$$

Probabilidad condicionada: Sea (Ω, α, P) un espacio probabilístico y A un suceso de α de probabilidad no nula, P(A) > 0. Si B es un suceso cualquiera de α, se llama probabilidad de B condicionada a A, al número P(B | A) definido por:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

que mide la ocurrencia del suceso B si se sabe que ha ocurrido el suceso A.

El Teorema de Bayes nos dice: Sea (Ω, α, P) un espacio probabilístico, y sea A_i, para i=1,2,…,n, una partición de Ω formada por una colección finita de sucesos de probabilidad no nula. Sea B un suceso tal que P(B) ≠ 0. Entonces:

$$P(A_j|B) = \frac{P(A_j)P(B|A_j)}{\sum_{i=0}^{n} P(A_i)P(B|A_i)}$$

La entropía se calcula con la siguiente fórmula:

$$H(x) = -\sum_{i=1}^{n} P(X = x_i) \cdot log\ P(X = x_i)$$

3.3.1 ID3

ID3 es un algoritmo de aprendizaje que pretende modelar los datos mediante un árbol de decisión. Fue creado por J.R.Quinlan en 1986. Se encuentra englobado dentro del llamado aprendizaje inductivo y supervisado. En este árbol, los nodos intermedios son atributos de los ejemplos de la tabla de datos, las ramas representan valores de dichos atributos y los nodos finales son los valores de la clase, que en nuestro caso nos da el diagnóstico, como ya vimos al hablar de los árboles de decisión binarios. Tanto éste como el algoritmo que hablaremos a continuación, el C4.5, pertenecen a la familia de métodos de inducción TDIDT(Top Down Induction Trees). Ambos generan árboles y reglas de decisión a partir de datos preclasificados. Para construir los árboles usamos el método de divide y vencerás.

El ID3 construye árboles de decisión a partir de un conjunto de ejemplos. Estos ejemplos o tuplas están constituidos por un conjunto de atributos y un clasificador. Los dominios de los atributos y de las clases deben ser discretos, algo que se cumple en nuestro conjunto de datos.

Para elegir qué atributos y en qué orden aparecen en el árbol se utiliza una función de evaluación: minimización de la entropía. También es posible escribirlo en forma de reglas IF-THEN.

La principal aplicación de este algoritmo es para los problemas de decisión. Su empleo se centra en los problemas de clasificación (diagnóstico de enfermedades dados unos síntomas, problemas de malfuncionamiento de equipos, concesión de préstamos, clasificación de un ser en una especie animal dadas unas características del mismo, etc.).

La función de salida presentará valores discretos. Los datos de aprendizaje pueden contener errores, valores nulos en algún atributo para algún ejemplo. Nos saldrá un mejor árbol cuando los atributos tengan un dominio de valores reducido. Una vez aprendido el árbol, lo podremos emplear para clasificar nuevos casos. Las principales ventajas son que tiene buenos resultados en un amplio rango de aplicaciones y que presenta una precisión del resultado alta. Las desventajas más importantes son el hecho de que los atributos sean discretos y que a veces los árboles son demasiado frondosos, lo cual nos hace difícil su interpretación. Esta última desventaja, como veremos adelante, la tendremos en nuestro problema.

La selección de los atributos se realizará mediante el principio de ganancia de la información. La ganancia de información es una propiedad estadística que nos sirve para medir cómo clasifica un atributo a los ejemplos. Se elige como nodo del árbol aquel que tenga una mayor ganancia de información. Posteriormente, expando sus ramas y sigo igual, pero considerando la nueva partición formada por el subconjunto de ejemplos que tienen ese valor para el atributo elegido.

Llamaremos S, al conjunto de ejemplos clasificados en C clases, A, al atributo de los ejemplos y S_v, a los ejemplos que en el atributo A tiene el valor v.

Utilizaremos la siguiente fórmula:

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

La estructura general del algoritmo ID3 sería la que vemos en la figura 6:

```
Algoritmo ID3(Ejemplos, atributo_salida, Atributos)
      Ejemplos: Ejemplos de aprendizaje
      Atributos_salida: Atributo a predecir por el árbol
      Atributos: Lista de Atributos
      (1) Crear una raíz para el árbol
      (2) Si todos los ejemplos son positivos Return(raíz, +)
      (3) Si todos los ejemplos son negativos Return(raíz, -)
      (4) Si Atributos=∅ Return(raíz, l) (l se trata del máximo
valor común de Atributo_salida en Ejemplos)
      Si ninguna de las anteriores condiciones se cumple entonces
            (1) Seleccionar el atributo A con mayor
Ganancia(Ejemplos, A)
            (2) El atributo de decisión para raíz es A
            (3) Para cada posible valor v_i de A
                  (3.1) Añadir una rama a raíz con el test A= v_i
                  (3.2) Ejemplos_v_i es el subconunto de Ejemplos con
valor v_i para A
                  (3.3) Si Ejemplos_v_i=∅
                        Entonces añadir un nodo(n,l) a partir de la
rama creada (l se trata del máximo valor común de Atributo_salida en
Ejemplos)
                        Sino añadir a la rama creada el subárbol
ID3(Ejemplos_ v_i, Atributo_salida, Atributos-{A})
      End
```

```
            Return raiz
```

Figura 6: Fragmento de pseudolenguaje del algoritmo ID3

El algoritmo ID3 presenta algunos inconvenientes, para los cuales se han realizado mejoras de este algoritmo. La solución a muchos de estos inconvenientes la incorpora el algoritmo C4.5, que constituye una extensión de este algoritmo ID3.

Estas mejoras al algoritmo ID3 son:

- Manejo de atributos de gran número de valores.
- Manejo de atributos con valores continuos.
- Manejo de valores desconocidos en algunos de los atributos.
- El coste de conocer el valor de un atributo no es constante.
- El momento de cuando se debe parar de subdividir el árbol: poda o sobreajuste

Este algoritmo se encuentra implementado en Weka, y tiene el nombre de ID3.

3.3.2 C4.5

C4.5 es un algoritmo usado para generar un árbol de decisión desarrollado por Ross Quinlan. Este algoritmo se trata de una modificación propuesta para mejorar el algoritmo ID3. Los árboles de decisión generados por este algoritmo pueden ser usados para clasificación, y debido a esto, por norma general nos referiremos a él como clasificador estadístico.

C4.5 construye los árboles de decisión desde un conjunto de datos de entrenamiento de la misma forma que lo hace ID3, usando el concepto de entropía de información. Los datos de entrenamiento se tratan de un grupo de ejemplos $S= s_1, s_2, \ldots$ ya clasificados. Cada ejemplo $s_i = x_1, x_2, \ldots$ representa los atributos del ejemplo. Los datos de entrenamiento son aumentados con un vector $C= c_1, c_2, \ldots$, donde $c_1, c_2,\ldots$ representan la clase a la que pertenece cada muestra, y será el valor que comparemos con la variable explicada obtenida en el árbol de decisión.

En cada nodo del árbol, C4.5 elige un atributo de los datos que más eficazmente dividen el conjunto de datos en subconjuntos enriquecidos en una clase u otra. El criterio empleado es el de seleccionar el atributo que presente un mayor ratio de ganancia de información. El algoritmo divide recursivamente en sublistas más pequeñas.

El ratio de ganancia se define por:

$$ratio(S,A) = \frac{ganancia(S,A)}{infopart(S,A)}$$

donde

$$infopart(S,A) = -\sum_{i=1}^{n} \frac{|E_i|}{E} \cdot log \frac{|E_i|}{E}$$

donde S es el conjunto de datos y A es un atributo de la tabla del conjunto de datos.

Ahora a comentar brevemente lo que se realiza para conseguir las mejoras al algoritmo ID3.

Para el manejo de atributos con valores continuos, se ordenan los valores del atributo y se especifica la clase a la que pertenecen. El objetivo es definir atributos discretos dividiendo el atributo continuo en intervalos, y dando valores booleanos. La división del intervalo se realiza basándose en un punto umbral, dividiéndolo así en 2 intervalos. Otra posibilidad será dividir en tantos subintervalos como puntos de corte tengamos.

Para el manejo de valores desconocidos en algunos de los atributos, tenemos varias posibles soluciones. Una de ellas será estimar el valor desconocido como el valor mayoritario que aparece en el resto de ejemplo. Otra solución es asignar a cada posible valor una probabilidad (frecuencia) de acuerdo con el resto de ejemplos. Posteriormente, repartiremos el ejemplo en cada uno de sus valores de acuerdo con la probabilidad y hacer el cálculo de la ganancia. En el caso de la clasificación, los casos con valores desconocidos se clasifican de acuerdo con la mayor probabilidad que proporcione el árbol.

Para el manejo de atributos con costes diferentes, podemos preferir entonces atributos con menos costes asociados (tendencia a favor de atributos con bajo coste). Atribuimos entonces a cada atributo un coste, coste(A), y calcularemos la ganancia como G=Ganancia/coste. También es posible emplear otras medidas como (Ganancia^2 /coste) o (2^{Ganancia} – 1 / (Coste(A)+1)^W), donde w$\in$[0,1] es la importancia relativa del coste respecto a la ganancia de información. Esto sucede en ámbitos como la medicina, en los que elegir un atributo u otro puede depender del coste asociado(escáner, biopsia, análisis de sangre, …).

Ahora vamos a hablar sobre el momento de parar de subdividir el árbol. A medida que añadimos niveles al árbol de decisión, las hipótesis se refinan tanto que describen muy bien los ejemplos utilizados en el aprendizaje, pero presenta el problema de que el error de clasificación puede aumentar al evaluar ejemplos. Esto es lo que se conoce como sobreajuste (overfitting).

Se dice que una hipótesis h se sobreajusta al conjunto de entrenamiento si existe alguna otra hipótesis h' tal que el error de h es menor que el de h' sobre el

conjunto de entrenamiento, pero es mayor sobre la distribución completa de los ejemplos, los de entrenamiento más los de test. En resumen, clasifica muy bien los datos de entrenamiento pero luego no sabe generalizar al conjunto de test. Esto es debido a que aprende hasta el ruido del conjunto de entrenamiento, adaptándose a las regularidades del conjunto de entrenamiento.

Para evitar el sobreajuste, existen varias técnicas de poda: técnicas de pre-poda y técnicas de post-poda. Las técnicas de pre-poda tratan de detener el crecimiento del árbol antes de que éste llegue a adaptarse perfectamente al conjunto de entrenamiento. Las técnicas de post-poda permiten que el árbol se sobreajuste a los datos y luego se efectúa sobre él una poda.

Estas técnicas tratan de compensar la falta de backtracking del proceso de inducción. En la práctica, la más utilizada es la técnica de post-poda, porque es difícil estimar con precisión cuando se debe detener el crecimiento del árbol.

En el algoritmo C4.5, la técnica utilizada es la de post-poda. Se realizan los siguientes pasos:

- Construir el árbol de decisión a partir del conjunto de entrenamiento.
- Convertir dicho árbol en un conjunto equivalente de reglas.
- Podar (generalizar) cada regla eliminando cualquier predicado del antecedente que conlleve un aumento estimado del rendimiento sobre el conjunto de test o incluso sobre el conjunto de entrenamiento,
- Ordenar las reglas eliminadas por su rendimiento estimado.

La implementación de este algoritmo la tenemos en lenguaje Java dentro de la herramienta Weka. Tiene el nombre de J48.

3.3.3 NAÏVE BAYES

Para hablar del algoritmo Naïve Bayes, vamos a realizar una pequeña introducción sobre los clasificadores bayesianos. Las redes bayesianas, junto con los árboles de decisión y las redes neuronales artificiales, han sido los tres métodos más empleados en aprendizaje automático durante los últimos años en tareas de clasificación. Es un método importante no sólo porque ofrece un análisis cualitativo de los atributos y valores que pueden intervenir en el problema, sino porque da cuenta también de la importancia cuantitativa de esos atributos. En relación al aspecto cualitativo podemos representar cómo se relacionan esos atributos ya sea en una forma causal, o únicamente señalando la correlación que existe entre esas variables. Pero el aspecto novedoso de los métodos bayesianos, es el hecho de que da una medida probabilística de la importancia de esas variables en el problema. Este hecho es una de las diferencias fundamentales que ofrecen las redes bayesianas con

respecto a otros métodos como los árboles de decisión, que no dan una medida cuantitativa de esa clasificación.

Entre las características que poseen los métodos bayesianos en tareas de aprendizaje se pueden resaltar las siguientes:

- Cada ejemplo observado va a modificar la probabilidad de que la hipótesis formulada sea cierta. Por ejemplo, una hipótesis que no concuerda con un conjunto de ejemplos no es desechada totalmente sino que lo que harán será disminuir esa probabilidad estimada para la hipótesis
- Estos métodos son robustos al posible ruido presente en los datos de entrenamiento y a la posibilidad de tener datos incompletos o erróneos de entrenamiento.
- Los métodos bayesianos permiten tener en cuenta en la predicción de la hipótesis el conocimiento a priori o conocimiento del dominio en forma de probabilidades. El problema puede surgir cuando no tengamos datos suficientes.

Cualquier sistema de clasificación se basa en lo siguiente: dado un conjunto de datos (dividido en entrenamiento y test) representados por una serie de atributos y un valor, el problema consiste en encontrar una función, que será la hipótesis, que clasifique dichos ejemplos.

Este método está basado en el Teorema de Bayes, que hemos enunciado anteriormente. La idea de usar el teorema de Bayes en cualquier problema de aprendizaje automático (sobre todo, en los de clasificación) es que podemos estimar las probabilidades a posteriori de cualquier hipótesis consistente con el conjunto de entrenamiento para de esta forma escoger la hipótesis más probable. Para realizar la estimación de estas probabilidades se han propuesto numerosos algoritmos, en los que destaca el algoritmo Naïve Bayes.

Veamos más a fondo ahora el clasificador Naïve Bayes. Dado un ejemplo x representado por k valores, el clasificador Naïve Bayes se basa en encontrar la hipótesis más probable que describa a ese ejemplo. Si la descripción de ese ejemplo viene dada por los valores (a_1, a_2, …, a_n), la hipótesis más probable va a ser aquella que cumpla:

$$v_{MAP} = argmax_{v_j \epsilon V} P(v_j | a_1, a_2, \dots, a_n)$$

es decir, la probabilidad de que conocidos los valores que describen a ese ejemplo, éste pertenezca a la clase v_j (donde v_j es el valor de la función de clasificación en el conjunto finito V). Por el teorema de Bayes:

$$v_{MAP} = argmax_{v_j \epsilon V} \frac{P(a_1, a_2, \dots, a_n | v_j) P(v_j)}{P(a_1, a_2, \dots, a_n)} = argmax_{v_j \epsilon V} P(a_1, a_2, \dots, a_n | v_j) P(v_j)$$

La forma de estimar $P(v_j)$ puede ser contando las veces que aparece el ejemplo vj en el conjunto de entrenamiento y dividiéndolo por el número total de ejemplos del conjunto. Para estimar el término $P(a_1, a_2, \dots, a_n | v_j)$ debo recorrer todos los datos de entrenamiento y obtendremos esta probabilidad que será el número de veces en que para cada categoría aparecen los valores del ejemplo x.

Este cálculo resulta muy complejo cuando el número de ejemplos es muy elevado, con lo que se hace necesario simplificar la explicar. Por esto, se recurre a la hipótesis de independencia condicional con el objeto de poder factorizar la probabilidad. Esta hipótesis nos dice que los valores a_j que describen un atributo de un ejemplo cualquiera x son independientes entre sí conocido el valor de la categoría a la que pertenecen. Se calculará de la siguiente forma:

$$P(a_1, a_2, \dots, a_n | v_j) = \prod_i P(a_i | v_j)$$

Cabe reseñar que este clasificador se puede aplicar cuando se dispone de conjunto de entrenamiento de tamaño medio o grande, y cuando los atributos que describen a los ejemplos son independientes entre sí con respecto al concepto que se pretende aprender. Esto último se debe a lo que hemos comentado antes sobre la independencia.

El algoritmo tendrá la estructura que vemos en la figura 7:

```
Aprendizaje_Bayesiano_Naive(ejemplos)
  Para cada posible valor del resultado v_j
    Obtener estimación P'(v_j) de la probabilidad P(v_j)
      Para cada valor a_i de cada atributo a
        Obtener una estimación P'(a_i | v_j) de la
   probabilidad P'(a_i | v_j)

Clasificar_instancia(x)
  devolver v_nb = arg max_{v_j ∈ V} P(v_j) ∏_i P(a_i | v_j)
```

Figura 7: Pseudolenguaje de estructura del algoritmo Naive Bayes

Este método se encuentra representado en la herramienta Weka llamado de la forma NaiveBayes.

También cabe destacar que existe otro concepto dentro de los métodos bayesianos, que son las redes bayesianas. Una red bayesiana consta de dos partes. Una cualitativa que está formada por un grafo dirigido acíclico. En este grafo, tenemos un nodo por cada variable del problema y un conjunto de enlaces dirigidos sin crear

ciclos dirigidos. La otra parte es cuantitativa, donde una serie de probabilidades condicionadas determinan una única distribución de probabilidad conjunta. En el grafo dirigido acíclico, se tiene que un nodo puede tener varios padres.

Para representar la red bayesiana Naive, que será la representación gráfica de Naive Bayes, se tendrá un árbol de profundidad 1, donde el nodo raíz, será la probabilidad a priori (clase) y los nodos hijos serán las probabilidades a posteriori (la clase dado un atributo).

Un ejemplo de red bayesiana Naive será de la forma que vemos en la figura 8:

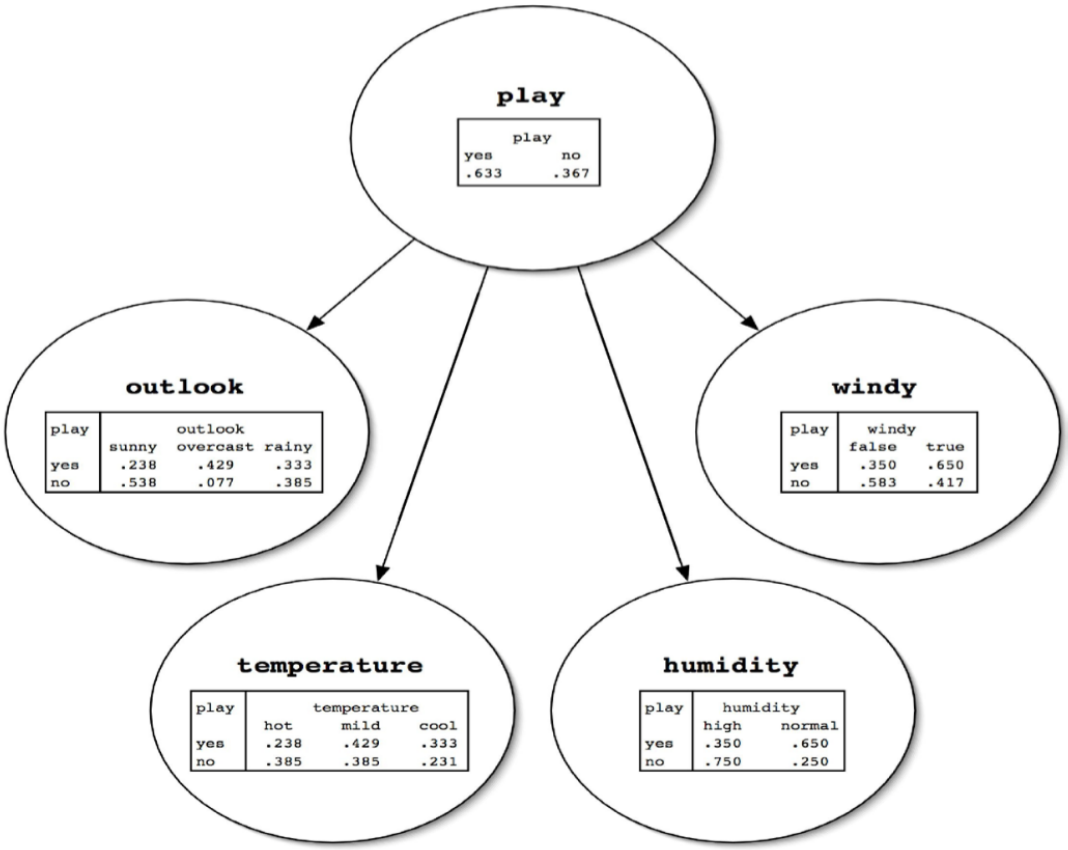


Figura 8: Red Bayesiana Naive

y un ejemplo de red bayesiana, sin ser Naive, tiene la que vemos en la figura 9:

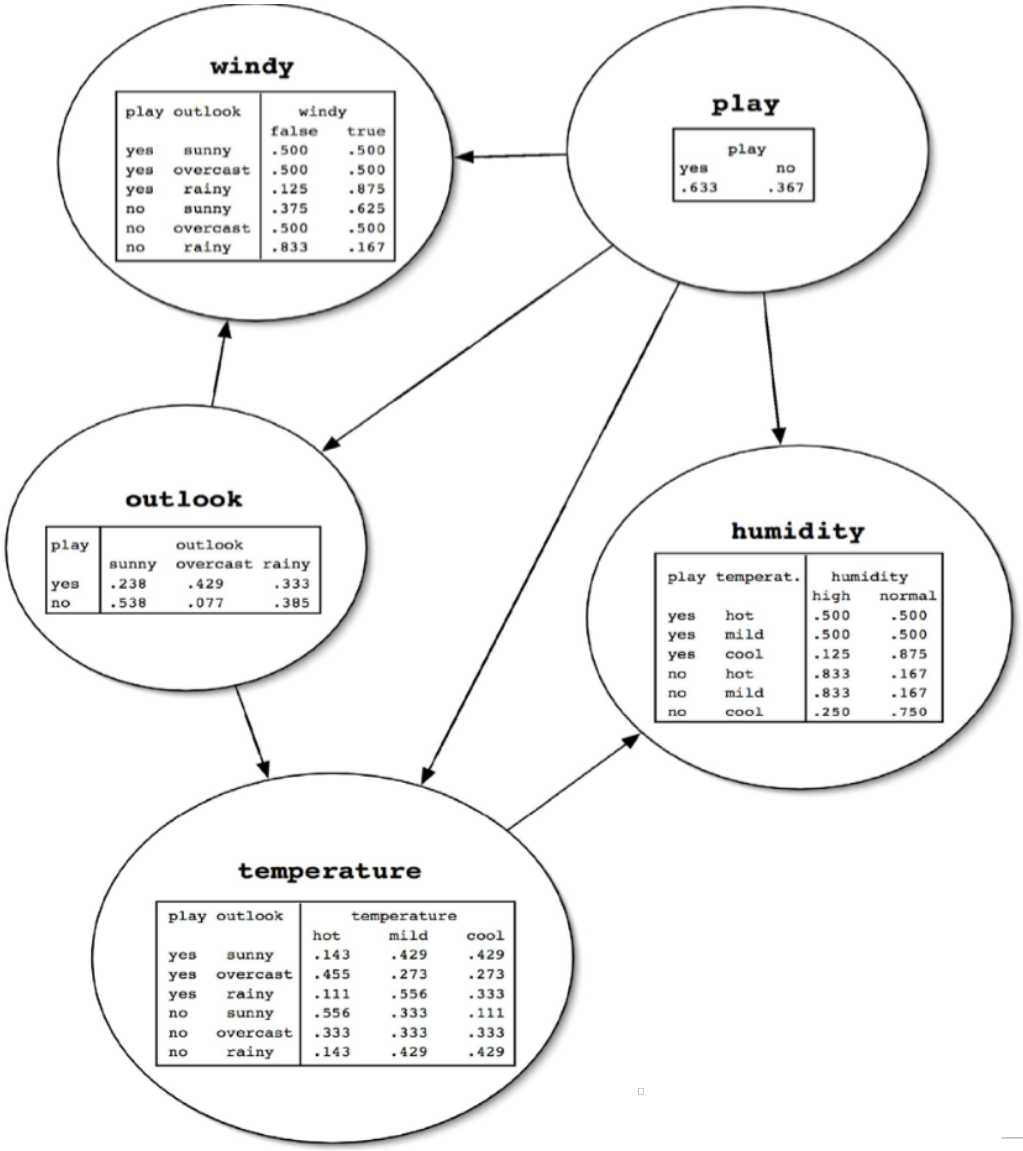


Figura 9: Red Bayesiana

Las redes bayesianas están implementadas en Weka mediante el método BayesNet, que las utilizaremos posteriormente para realizar la comparación entre métodos.

3.4 LIBRERÍAS EN R SOBRE ÁRBOLES DE DECISIÓN

En este apartado, vamos a hablar acerca de las distintas librerías que existen sobre árboles de decisión.

En primer lugar, vamos a comentar la librería data.tree. Esta librería nos sirve para crear estructuras de árbol de datos jerárquicos y recorrer el árbol en varios

niveles. Esta librería es útil para árboles de decisión, aprendizaje automático, finanzas y otras muchas más aplicaciones. Entre sus funciones más destacadas tenemos Sort, Prune, Aggregate, Cumulate y print, que nos servirán para ordenar, podar o imprimir el árbol, entre otras. También presenta varias funciones de construcción y conversión. Esta librería se diferencia de las que vamos a comentar posteriormente en que no presenta un método para construir un árbol de decisión con un algoritmo ya construido, sino que es útil para crearlo desde cero. Este es uno de los motivos que decidirán a usar esta librería en el siguiente apartado.

El paquete RPart fue descrito por Brieman, Friedman, Olshen y Stone, y se utiliza para árboles de clasificación y regresión(CART). Posiblemente, se trata del método más utilizado en R para árboles de decisión. Para ajustar árboles CART utilizaremos la siguiente expresión:

`rpart(y ~ x1 + x2 + … + xp, datos)`

donde y es la variable respuesta y x_1, x_2, …, x_p son las variables predictoras.

Si y es discreta la función ajusta un árbol de clasificación y si es continua un árbol de regresión. Como funciones dentro de esta librería vamos a destacar: predict, para realizar la predicción de un objeto de tipo RPart; print, imprime un objeto RPart; plotcp, da una representación visual de la cross-validation en un objeto RPart.

Veamos un ejemplo con nuestro conjunto de datos. Pondríamos las siguientes instrucciones como se ve en la figura 10:

```
      ModeloArbol<-rpart(datos$CON_DIAGES ~
.,data=datos,parms=list(split="information"))

      > ModeloArbol
```

Figura 10: Entrada para crear un árbol de decisión con RPart

y nos devolvería el siguiente árbol como vemos en la figura 11. Cada línea representará un nodo y presenta varias datos como el número de nodo (node), el atributo que realiza la división (split), el número de elementos con estos valores (n), el número de errores cometidos (loss), la respuesta predicha en dicho nodo (yval), un vector con las probabilidades de cada valor de decisión (yprob). Si al final de una de las líneas tenemos un *, esto nos indicará que el nodo es terminal.

```
      n= 1000


      node), split, n, loss, yval, (yprob)
            * denotes terminal node
```

```
       1) root 1000 680 -1 (0.32 0.027 0.058 0.011 0.007 0.11 0.21 0.19
0.02 0.004 0.036)

         2) CLASSI_FIN=-1,2,8 332  13 -1 (0.96 0.003 0.006 0.003 0 0.003
0.018 0 0.003 0 0.003) *

         3) CLASSI_FIN=1 668 461 6 (0.0015 0.039 0.084 0.015 0.01 0.17 0.31
0.29 0.028 0.006 0.052)

           6) CLI_PETEQU=1 57  36 1 (0 0.37 0.18 0.12 0 0.053 0.18 0.053
0.018 0 0.035) *

           7) CLI_PETEQU=0,2 611 414 6 (0.0016 0.0082 0.075 0.0049 0.011
0.18 0.32 0.31 0.029 0.0065 0.054)

            14) CLI_COMA=0,1 100  73 6 (0.01 0.04 0.19 0.01 0.03 0.22 0.27
0.09 0.02 0.01 0.11) *

            15) CLI_COMA=2 511 332 7 (0 0.002 0.053 0.0039 0.0078 0.17 0.33
0.35 0.031 0.0059 0.043)

              30) CLI_RIGIDE=1 294 179 6 (0 0.0034 0.071 0.0068 0.014 0.18
0.39 0.27 0.027 0.0034 0.041) *

              31) CLI_RIGIDE=0,2 217 116 7 (0 0 0.028 0 0 0.16 0.25 0.47
0.037 0.0092 0.046)

                62) CLI_CEFALE=0 9   1 6 (0 0 0.11 0 0 0 0.89 0 0 0 0) *

                63) CLI_CEFALE=1,2 208 107 7 (0 0 0.024 0 0 0.17 0.23 0.49
0.038 0.0096 0.048) *
```

Figura 11: Salida de árbol de decisión con RPart

También existe el paquete party que nos ofrece árboles de regresión no paramétricos para variables nominales, ordinales, numéricas, censuradas y respuestas multivariantes. Estos serán árboles basados en inferencia condicional, que será lo que diferencia este paquete del resto. En este paquete, podemos crear el árbol de clasificación o regresión, mediante la siguiente fórmula:

`ctree (formula= y ~ x1 + x2 + … + xp, datos)`

El tipo de árbol creado dependerá del tipo de la variable de salida(nominal, ordinal, numérico, etc.). El crecimiento del árbol estará basado con reglas estadísticas de parada, y podando cuando sea necesario.

El paquete tree es otro paquete para la clasificación y regresión de árboles. La sentencia para crear el árbol sigue la misma estructura que en los paquetes anteriores:

`tree (y ~ x1 + x2 + … + xp, datos)`

donde y es la variable respuesta y x_1, x_2, …, x_p son las variables predictoras.

Este paquete tiene una serie de funciones, entre las que destacamos las que se dedican a la predicción, como Predict, o las que nos muestran el árbol, como Plot o Print.

A continuación, vamos a ver un ejemplo de esta librería con los datos que vamos a usar durante todo el trabajo. En la figura 12, vemos la entrada para crear un árbol con la librería tree:

```
ModeloTree <- tree(datos$CON_DIAGES ~ ., datos)

> ModeloTree
```

Figura 12: Entrada para crear un árbol de decisión con la librería tree

y nos devolverá lo que vemos en la figura 13. Cada línea representará un nodo y presenta varias datos como el número de nodo (node), el atributo que realiza la división (split), el número de elementos con estos valores (n), la desviación (deviance), la respuesta predicha en dicho nodo (yval), un vector con las probabilidades de cada valor de decisión (yprob). Si al final de una de las líneas tenemos un *, esto nos indicará que el nodo es terminal.

```
node), split, n, deviance, yval, (yprob)

      * denotes terminal node


 1) root 1000 3647.00 -1 ( 0.320000 0.027000 0.058000 0.011000
0.007000 0.113000 0.213000 0.191000 0.020000 0.004000 0.036000 )

   2) CLASSI_FIN: -1,2,8 332  152.10 -1 ( 0.960843 0.003012 0.006024
0.003012 0.000000 0.003012 0.018072 0.000000 0.003012 0.000000 0.003012 )

     4) CLASSI_FIN: -1,8 46   97.19 -1 ( 0.717391 0.021739 0.043478
0.021739 0.000000 0.021739 0.130435 0.000000 0.021739 0.000000 0.021739 )
*

     5) CLASSI_FIN: 2 286    0.00 -1 ( 1.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 )
*

   3) CLASSI_FIN: 1 668 2353.00 6 ( 0.001497 0.038922 0.083832
0.014970 0.010479 0.167665 0.309880 0.285928 0.028443 0.005988 0.052395 )

     6) CLI_PETEQU: 0,2 611 2018.00 6 ( 0.001637 0.008183 0.075286
0.004910 0.011457 0.178396 0.322422 0.307692 0.029460 0.006547 0.054010 )
```

```
            12) CLI_COMA: 0,1 100  382.40 6 ( 0.010000 0.040000 0.190000
0.010000 0.030000 0.220000 0.270000 0.090000 0.020000 0.010000 0.110000 )
*

            13) CLI_COMA: 2 511 1570.00 7 ( 0.000000 0.001957 0.052838
0.003914 0.007828 0.170254 0.332681 0.350294 0.031311 0.005871 0.043053 )
*

           7) CLI_PETEQU: 1 57  197.70 1 ( 0.000000 0.368421 0.175439
0.122807 0.000000 0.052632 0.175439 0.052632 0.017544 0.000000 0.035088 )
*
```

Figura 13: Salida de un árbol de decisión con la librería tree

4. ALGORITMO DE CREACIÓN DEL ÁRBOL DE DECISIÓN

4.1 INTRODUCCIÓN Y MOTIVACIÓN

En este apartado, vamos a hablar de uno de los principales objetivos de este trabajo de fin de grado, que va a ser la construcción de un algoritmo para crear un árbol de decisión, con ciertas características que vamos a comentar a continuación.

4.2 UTILIZACIÓN DE LA LIBRERÍA DATA.TREE

Después de buscar sobre la utilización acerca de las librerías existentes sobre árboles de decisión como RPart, Tree, Party, Partykit, MVPart, Data.Tree, hemos decidido que para crear un árbol desde cero como pretendemos realizar, vamos a usar la librería data.tree.

Para poder utilizar esta librería, tenemos que instalarla en R y cargarla usando lo siguiente como veremos en la figura 14:

```
install.packages("data.tree")

library(data.tree)
```

Figura 14: Instalación y carga del paquete data.tree

4.3 ALGORITMO PARA CREAR EL ÁRBOL DE DECISIÓN

Dado un conjunto de datos, que tendremos en una tabla descrita como explicamos en el apartado 2 del trabajo, y un conjunto de variables que nos permitirán decidir en cada momento que decisión tomar, será posible construir el árbol. Respecto a las variables, tendremos n variables, de las cuales n-1 serán explicativas y una será explicada. Esta variable explicada siempre la situaremos en última posición del conjunto, debido a que nos indicará la decisión a tomar.

Para crear el árbol, partiremos del conjunto de datos y si tenemos en la primera variable del conjunto Variables, por ejemplo, CLI_CEFALE, que presenta valores 0,1,2, entonces dividiremos el conjunto en 3 diferentes, los que tienen CLI_CEFALE=0, CLI_CEFALE=1 y CLI_CEFALE=2, y en el árbol crearemos un nodo con cada opción. Una vez aquí, estudiaremos cada uno de los 3 conjuntos por

separado y seguiremos así recursivamente hasta llegar a la última variable del conjunto Variables.

Tenemos que destacar que a diferencia de otros algoritmos de creación de árboles de decisión, en nuestro algoritmo no existe ningún criterio de parada y es por esto por lo que nos sale un árbol tan grande.

Esta función tendrá atributos de entrada, que serán las siguientes: variables, que son las variables de creación para el árbol de decisión, data, que se refiere al conjunto de datos, donde tendremos a todos los pacientes con sus síntomas, y node, que será la raíz del árbol de decisión.

Cuando queramos imprimir el árbol, utilizaremos la función Print de la librería data.tree. En la figura 15, veremos la instrucción para mostrar un árbol:

```
print(tree, "feature", "obsCount", "porcentaje")
```

Figura 15: Función para imprimir un árbol

que nos mostrará la estructura del árbol con su raíz, sus nodos y sus hijos. A la derecha, nos indicará el nombre de la variable del nodo a la que se refiera, el número de observaciones y el porcentaje de las observaciones de ese tipo con respecto al total de observaciones del nodo padre.

En la figura 16, veremos el código para crear el árbol:

```
#Creamos el arbol de decision
#En variables, introduciremos las variables para crear el arbol, la última
#que pongamos sera la que nos diga el diagnostico
arbolDecMarcos <- function(variables, data, node) {
  long <- length(variables)
  node$obsCount <- nrow(data)
  node$feature <- variables[1]
    #Estudiamos primero el caso en que el numero de variables de los datos
    #sea mayor que 0
    if(length(names(data))>0) {
       #Dividimos en subconjuntos que tengan por ejemplo CLI_CEFALE=1
       childObs    <-    split(data[,!(names(data)    %in%    variables[1])],
data[,variables[1]], drop=TRUE)
       for(i in 1:length(childObs)) {
```

```
      #Construimos un hijo que tenga el nombre del valor de la
      #caracteristica
      child <- node$AddChild(names(childObs)[i])
      #Llama al algoritmo recursivamente en el hijo y el subconjunto
      varNueva <- variables[-1]
      if(length(varNueva)!=0) {
        datosNew <- childObs[[i]]
        child$porcentaje <- (nrow(childObs[[i]])*100)/nrow(data)
        arbolDecMarcos(varNueva, datosNew, child)
      }
      else {
        child$obsCount <- nrow(childObs[[i]])
        child$porcentaje <- (nrow(childObs[[i]])*100)/nrow(data)
      }
    }
  }
  #Estudiamos primero el caso en que el número de variables de los datos
  #sea 0. Diferenciamos porque el tipo de datos aqui no sera una tabla,
  #sino un vector
  else {
    node$obsCount <- length(data)
    node$porcentaje <-((length(data))*100) /(node$parent$obsCount)
    childObs <- sort(unique(data))
    for(i in 1:length(childObs)) {
      child <-node$AddChild(childObs[i])
      cont <- 0
      for(j in 1:length(data)) {
        if (childObs[i]== data[j]) {
          cont <- cont +1
        }
      }
```

```
        child$obsCount <- cont
        child$porcentaje <- ((cont)*100)/length(data)
    }
  }
}
```

Figura 16: Función para crear el árbol de decisión

4.4 EJEMPLO

Veamos un ejemplo de uso para el algoritmo de creación del árbol de decisión. Vamos a trabajar con los datos que estamos utilizando durante todo el proyecto, pondremos 3 variables(2 explicativas y 1 explicada), junto con un nodo inicial que llamaremos “Enfermedad”. Vamos a poner únicamente 3 variables para poder ver el árbol completo, ya si ponemos un número elevado de variables no es posible poder ver en pantalla el árbol completo(usando la función print).

En la figura 17, veremos el código que introduciremos:

```
variables <- c("CLI_CEFALE", "CLI_FEBRE", "CON_DIAGES")
tree <- Node$new("enfermedad")
arbolDecMarcos(variables, datos, tree)
print(tree, "feature", "obsCount", "porcentaje")
```

Figura 17: Instrucciones para crear árbol con nuestros datos con 3 variables

En la figura 18, vemos el árbol que nos devuelve, en el cual tenemos varias columnas que nos indican el nivel en que nos encontramos en el árbol que vendrá indicado con un valor, la decisión seleccionada, el número de observaciones y el porcentaje de observaciones de este nodo con respecto a los nodos que se encuentran en su nivel.

```
    levelName    feature       obsCount porcentaje
1  enfermedad     CLI_CEFALE     1000         NA
2   ¦--0           CLI_FEBRE      113 11.3000000
3   ¦   ¦--0      CON_DIAGES       37 32.7433628
4   ¦   ¦   ¦---1                  26 70.2702703
```

```
5   ¦   ¦   ¦--5                    2  5.4054054
6   ¦   ¦   ¦--6                    4 10.8108108
7   ¦   ¦   ¦--7                    3  8.1081081
8   ¦   ¦   ¦--8                    1  2.7027027
9   ¦   ¦   °--10                   1  2.7027027
10  ¦   ¦--1      CON_DIAGES       69 61.0619469
11  ¦   ¦   ¦---1                  28 40.5797101
12  ¦   ¦   ¦--1                    3  4.3478261
13  ¦   ¦   ¦--2                    3  4.3478261
14  ¦   ¦   ¦--3                    1  1.4492754
15  ¦   ¦   ¦--5                    8 11.5942029
16  ¦   ¦   ¦--6                   20 28.9855072
17  ¦   ¦   ¦--7                    1  1.4492754
18  ¦   ¦   ¦--8                    2  2.8985507
19  ¦   ¦   ¦--9                    1  1.4492754
20  ¦   ¦   °--10                   2  2.8985507
21  ¦   °--2      CON_DIAGES        7  6.1946903
22  ¦       ¦---1                   4 57.1428571
23  ¦       ¦--1                    1 14.2857143
24  ¦       ¦--5                    1 14.2857143
25  ¦       °--6                    1 14.2857143
26  ¦--1           CLI_FEBRE      719 71.9000000
27  ¦   ¦--0      CON_DIAGES        8  1.1126565
28  ¦   ¦   ¦---1                  3 37.5000000
29  ¦   ¦   ¦--2                    2 25.0000000
30  ¦   ¦   ¦--5                    2 25.0000000
31  ¦   ¦   °--6                    1 12.5000000
32  ¦   ¦--1      CON_DIAGES      619 86.0917942
33  ¦   ¦   ¦---1                 156 25.2019386
34  ¦   ¦   ¦--1                  12  1.9386107
35  ¦   ¦   ¦--2                  39  6.3004847
36  ¦   ¦   ¦--3                   6  0.9693053
```

```
37  ¦   ¦   ¦--4                    3  0.4846527
38  ¦   ¦   ¦--5                   76 12.2778675
39  ¦   ¦   ¦--6                  144 23.2633279
40  ¦   ¦   ¦--7                  143 23.1017771
41  ¦   ¦   ¦--8                   14  2.2617124
42  ¦   ¦   ¦--9                    2  0.3231018
43  ¦   ¦   °--10                  24  3.8772213
44  ¦   °--2      CON_DIAGES        92 12.7955494
45  ¦       ¦---1                   29 31.5217391
46  ¦       ¦--1                    2  2.1739130
47  ¦       ¦--2                    6  6.5217391
48  ¦       ¦--3                    1  1.0869565
49  ¦       ¦--4                    1  1.0869565
50  ¦       ¦--5                    6  6.5217391
51  ¦       ¦--6                   20 21.7391304
52  ¦       ¦--7                   24 26.0869565
53  ¦       ¦--8                    2  2.1739130
54  ¦       °--10                   1  1.0869565
55  °--2           CLI_FEBRE      168 16.8000000
56      ¦--1      CON_DIAGES      120 71.4285714
57      ¦   ¦---1                  45 37.5000000
58      ¦   ¦--1                   8  6.6666667
59      ¦   ¦--2                   7  5.8333333
60      ¦   ¦--3                   3  2.5000000
61      ¦   ¦--4                   3  2.5000000
62      ¦   ¦--5                  14 11.6666667
63      ¦   ¦--6                  15 12.5000000
64      ¦   ¦--7                  18 15.0000000
65      ¦   ¦--9                   1  0.8333333
66      ¦   °--10                  6  5.0000000
67      °--2      CON_DIAGES       48 28.5714286
68          ¦---1                  29 60.4166667
```

```
69          ¦--1                    1  2.0833333
70          ¦--2                    1  2.0833333
71          ¦--5                    4  8.3333333
72          ¦--6                    8 16.6666667
73          ¦--7                    2  4.1666667
74          ¦--8                    1  2.0833333
75          °--10                   2  4.1666667
```

Figura 18: Árbol de decisión con 3 variables

Mientras que si utilizamos un número más elevado de variables (por ejemplo con 6, pero desde 4 ocurriría lo mismo), quedaría como sigue. El código que introduciremos será el que vemos en la figura 19:

```
variables <- c("CLI_CEFALE", "CLI_FEBRE", "CLI_VOMITO", "CLI_CONVUL",
+                "CLI_RIGIDE", "CON_DIAGES")
tree <- Node$new("enfermedad")
arbolDecMarcos(variables, datos, tree)
print(tree, "feature", "obsCount", "porcentaje")
```

Figura 19: Instrucciones para crear árbol con nuestros datos con 6 variables

Y nos devolverá el árbol de la figura 20:

```
                          levelName    feature obsCount porcentaje
1   enfermedad                          CLI_CEFALE     1000         NA
2    ¦--0                                CLI_FEBRE      113  11.300000
3    ¦   ¦--0                           CLI_VOMITO       37  32.743363
4    ¦   ¦   ¦--0                       CLI_CONVUL       33  89.189189
5    ¦   ¦   ¦   ¦--0                   CLI_RIGIDE       31  93.939394
6    ¦   ¦   ¦   ¦   °--0               CON_DIAGES       31 100.000000
7    ¦   ¦   ¦   ¦       ¦---1                        22  70.967742
8    ¦   ¦   ¦   ¦       ¦--5                          1   3.225806
9    ¦   ¦   ¦   ¦       ¦--6                          3   9.677419
10   ¦   ¦   ¦   ¦       ¦--7                          3   9.677419
11   ¦   ¦   ¦   ¦       ¦--8                          1   3.225806
12   ¦   ¦   ¦   ¦       °--10                         1   3.225806
```

13	¦	¦	¦	°--1	CLI_RIGIDE	2	6.060606
14	¦	¦	¦	¦--0	CON_DIAGES	1	50.000000
15	¦	¦	¦	¦ °---1		1	100.000000
16	¦	¦	¦	°--1	CON_DIAGES	1	50.000000
17	¦	¦	¦	°---1		1	100.000000
18	¦	¦	¦--1		CLI_CONVUL	1	2.702703
19	¦	¦	¦	°--0	CLI_RIGIDE	1	100.000000
20	¦	¦	¦	°--0	CON_DIAGES	1	100.000000
21	¦	¦	¦	°---1		1	100.000000
22	¦	¦	°--2		CLI_CONVUL	3	8.108108
23	¦	¦	¦--0		CLI_RIGIDE	1	33.333333
24	¦	¦	¦	°--0	CON_DIAGES	1	100.000000
25	¦	¦	¦	°--5		1	100.000000
26	¦	¦	°--1		CLI_RIGIDE	2	66.666667
27	¦	¦	°--2		CON_DIAGES	2	100.000000
28	¦	¦	¦---1			1	50.000000
29	¦	¦	°--6			1	50.000000
30	¦	¦--1			CLI_VOMITO	69	61.061947
31	¦	¦	¦--0		CLI_CONVUL	6	8.695652
32	¦	¦	¦	¦--0	CLI_RIGIDE	1	16.666667
33	¦	¦	¦	¦ °--1	CON_DIAGES	1	100.000000
34	¦	¦	¦	¦ °---1		1	100.000000
35	¦	¦	¦	¦--1	CLI_RIGIDE	4	66.666667
36	¦	¦	¦	¦ °--0	CON_DIAGES	4	100.000000
37	¦	¦	¦	¦ ¦---1		2	50.000000
38	¦	¦	¦	¦ ¦--5		1	25.000000
39	¦	¦	¦	¦ °--6		1	25.000000
40	¦	¦	¦	°--2	CLI_RIGIDE	1	16.666667
41	¦	¦	¦	°--2	CON_DIAGES	1	100.000000
42	¦	¦	¦	°---1		1	100.000000
43	¦	¦	¦--1		CLI_CONVUL	45	65.217391
44	¦	¦	¦	¦--0	CLI_RIGIDE	5	11.111111

45	¦	¦	¦	¦	°--0	CON_DIAGES	5	100.000000
46	¦	¦	¦	¦	¦---1		3	60.000000
47	¦	¦	¦	¦	¦--5		1	20.000000
48	¦	¦	¦	¦	°--6		1	20.000000
49	¦	¦	¦	¦--1		CLI_RIGIDE	26	57.777778
50	¦	¦	¦	¦	¦--0	CON_DIAGES	5	19.230769
51	¦	¦	¦	¦	¦	¦---1	2	40.000000
52	¦	¦	¦	¦	¦	¦--5	1	20.000000
53	¦	¦	¦	¦	¦	°--6	2	40.000000
54	¦	¦	¦	¦	¦--1	CON_DIAGES	12	46.153846
55	¦	¦	¦	¦	¦	¦---1	1	8.333333
56	¦	¦	¦	¦	¦	¦--2	2	16.666667
57	¦	¦	¦	¦	¦	¦--5	1	8.333333
58	¦	¦	¦	¦	¦	¦--6	4	33.333333
59	¦	¦	¦	¦	¦	¦--8	1	8.333333
60	¦	¦	¦	¦	¦	¦--9	1	8.333333
61	¦	¦	¦	¦	¦	°--10	2	16.666667
62	¦	¦	¦	¦	°--2	CON_DIAGES	9	34.615385
63	¦	¦	¦	¦	¦---1		6	66.666667
64	¦	¦	¦	¦	°--6		3	33.333333
65	¦	¦	¦	°--2		CLI_RIGIDE	14	31.111111
66	¦	¦	¦	¦	¦--0	CON_DIAGES	2	14.285714
67	¦	¦	¦	¦	¦	¦--5	1	50.000000
68	¦	¦	¦	¦	¦	°--6	1	50.000000
69	¦	¦	¦	¦	¦--1	CON_DIAGES	8	57.142857
70	¦	¦	¦	¦	¦	¦---1	2	25.000000
71	¦	¦	¦	¦	¦	¦--3	1	12.500000
72	¦	¦	¦	¦	¦	¦--6	4	50.000000
73	¦	¦	¦	¦	¦	°--7	1	12.500000
74	¦	¦	¦	¦	°--2	CON_DIAGES	4	28.571429
75	¦	¦	¦	¦	¦---1		2	50.000000
76	¦	¦	¦	¦	¦--1		1	25.000000

```
77   ¦   ¦   ¦          °--6                          1  25.000000
78   ¦   ¦   °--2                   CLI_CONVUL      18  26.086957
79   ¦   ¦       ¦--1                CLI_RIGIDE       8  44.444444
80   ¦   ¦       ¦   ¦--0            CON_DIAGES       1  12.500000
81   ¦   ¦       ¦   ¦   °--1                         1 100.000000
82   ¦   ¦       ¦   ¦--1            CON_DIAGES       2  25.000000
83   ¦   ¦       ¦   ¦   ¦--5                         1  50.000000
84   ¦   ¦       ¦   ¦   °--6                         1  50.000000
85   ¦   ¦       ¦   °--2            CON_DIAGES       5  62.500000
86   ¦   ¦       ¦       ¦---1                        3  60.000000
87   ¦   ¦       ¦       °--6                         2  40.000000
88   ¦   ¦       °--2                CLI_RIGIDE      10  55.555556
89   ¦   ¦           ¦--0            CON_DIAGES       1  10.000000
90   ¦   ¦           ¦   °---1                        1 100.000000
91   ¦   ¦           ¦--1            CON_DIAGES       4  40.000000
92   ¦   ¦           ¦   ¦---1                        2  50.000000
93   ¦   ¦           ¦   °--5                         2  50.000000
94   ¦   ¦           °--2            CON_DIAGES       5  50.000000
95   ¦   ¦               ¦---1                        2  40.000000
96   ¦   ¦               ¦--1                         1  20.000000
97   ¦   ¦               ¦--2                         1  20.000000
98   ¦   ¦               °--8                         1  20.000000
99   ¦   °--2                       CLI_VOMITO       7   6.194690
100  ¦       °--... 2 nodes w/ 17 sub                NA         NA
101  °--... 2 nodes w/ 310 sub                       NA         NA
```

Figura 20: Árbol de decisión con 6 variables

Cabe reseñar que para representar gráficamente el árbol, podríamos haber utilizado la función plot, también de la librería data.tree. Pero después de probarlo, desechamos esta opción, ya que debido al gran tamaño del árbol, la representación gráfica es aún peor y es imposible ver nada, ni siquiera si hacemos zoom en la imagen del árbol.

Figura 21: Representación gráfica del árbol con 6 variables

5. VALIDACIÓN DE RESULTADOS OBTENIDOS CON EL ÁRBOL DE DECISIÓN

5.1 FUNCIÓN DE PREDICCIÓN DEL MÁXIMO VALOR

La predicción en un árbol de decisión consistirá en comprobar si el valor real de un diagnóstico de un paciente coincide con el valor predicho por el árbol de decisión.

Por tanto, para realizar la validación de los obtenidos con el árbol de decisión construiremos una función de predicción. La función de predicción consistirá en ir recorriendo los distintos niveles del árbol para finalmente llegar al valor predicho o al diagnóstico predicho por el árbol. Una vez aquí, comprobaremos si coincide el valor predicho por el árbol y el valor real que tendremos en el conjunto de datos.

La función de predicción, que llamaremos PredictMaxVal, tendrá 2 atributos de entrada. Estos serán tree, que es el árbol de decisión que hemos construido anteriormente, y features, que serán las características de un individuo en cuestión. Esta función nos devolverá el valor predicho por el árbol y en el caso de que el árbol nos prediga más de un valor, devolveremos el valor que presente más observaciones.

En la figura 22, vemos el código de esta función de predicción:

```
#Funcion de prediccion, poniendo como parametros, el arbol y las
#caracteristicas de decision
PredictMaxVal <- function(tree, features) {
  #Estudiamos el caso en que el nodo sea raiz
  if (tree$children[[1]]$isLeaf) {
    num<- tree$count
    max<- 1
    #Elegimos el nodo que más observaciones tenga
    for(j in 1:num) {
      if(tree$children[[max]]$obsCount < tree$children[[j]]$obsCount) {
        max <- j
      }
    }
```

```
    #Devolvemos el nombre del nodo que mas observaciones tenga
    return (tree$children[[max]]$name)
  }
  caract<-names(tree$children)[1]
  for(i in 1:tree$count){
    if(features[[tree$feature]]==names(tree$children)[i])
      caract<- names(tree$children)[i]
  }

  child <- tree$children[[caract]]
  return (PredictMaxVal(child, features))
}
```

Figura 22: Código de la función de predicción del máximo valor

Veamos un ejemplo de uso para el primer árbol construido en el apartado anterior. Se introduce por pantalla lo que vemos en la figura 23:

```
features <- c(CLI_CEFALE='1', CLI_FEBRE='1', CLI_VOMITO='1',
CLI_CONVUL='2',
+               CLI_RIGIDE='1', CLI_KERNIG='2', CLI_ABAULA='2',
CLI_COMA='2',
+               CLI_PETEQU='2', CLASSI_FIN='1' )
> PredictMaxVal(tree,features)
```

Figura 23: Entrada de un ejemplo de la función de predicción del máximo valor

y nos devolverá lo que vemos en la figura 24:

```
[1] "6"
```

Figura 24: Salida de un ejemplo de la función de predicción del máximo valor

Esta función de predicción tiene el gran inconveniente de que cuando el árbol nos dé más de un valor para diagnóstico, como hemos puesto la condición de que nos devuelva el valor con mayor número de observaciones, el resto de valores bien predichos realmente no los devolverá como bien predichos.

5.2 FUNCIÓN DE PREDICCIÓN DE TODAS LAS OPCIONES

La principal motivación de este apartado es corregir un inconveniente que tiene la anterior función de predicción. Este inconveniente es debido a que hay un gran número de casos en que árbol predice bien, pero la función de predicción nos dirá lo contrario. Esto ocurrirá cuando para un conjunto de datos de un paciente, el árbol nos devuelva más de un valor. Para cuando el árbol nos devuelva varios valores, la anterior función de predicción nos devolvía el valor con mayor número de observaciones, pero el resto no es contemplado.

Debido a esto, pensamos en la idea de crear una nueva función de predicción de la misma forma que la anterior, pero en vez de devolver el valor con mayor número de observaciones, que nos dé como salida con conjunto con todos los valores predichos.

A esta función de predicción, la llamaremos PredictAllOptions, ya que contemplará todas las opciones que nos dé el árbol de decisión. Esta función tendrá dos parámetros de entrada como son: tree, que es el árbol de decisión ya obtenido anteriormente, y features, que son las características de un paciente en cuestión. Esta función funcionará igual que la anterior con la diferencia de que iremos construyendo un conjunto con todos los valores predichos por el árbol. Este conjunto será la salida de la función.

A continuación, vemos en la figura 25 el código de esta nueva función de predicción:

```
#Funcion de prediccion(cogiendo todas las posibles opciones), poniendo como
#parametros, el arbol y las caracteristicas de decision
PredictAllOptions <- function(tree, features) {
  if (tree$children[[1]]$isLeaf) {
    num<- tree$count
    conj <- c()
    #Creamos un conjunto con las observaciones que haya en el arbol para la
    #tupla dada
    for(j in 1:num) {
      conj <- c(conj, tree$children[[j]]$name)
    }
    #Devolvemos el vector con los nombres de los nodos
```

```
    return (conj)
  }
  #Tomamos como arbol, el hijo, por ejemplo el que sea CLI_CEFALE=1 y hacemos
  #recursividad
  caract<-names(tree$children)[1]
  for(i in 1:tree$count){
    if(features[[tree$feature]]==names(tree$children)[i])
      caract<- names(tree$children)[i]
  }
  child <- tree$children[[caract]]
  return (PredictAllOptions(child, features))
}
```

Figura 25: Código de la función de predicción de todas las opciones

Veamos a continuación un ejemplo de uso de la función para el árbol construido en el apartado anterior. Se introduce por pantalla como se ve en la figura 26:

```
features <- c(CLI_CEFALE='1', CLI_FEBRE='1', CLI_VOMITO='1',
CLI_CONVUL='2',
+               CLI_RIGIDE='1', CLI_KERNIG='2', CLI_ABAULA='2',
CLI_COMA='2',
+               CLI_PETEQU='2', CLASSI_FIN='1' )
> PredictAllOptions(tree,features)
```

Figura 26: Entrada de un ejemplo de la función de predicción de todas las opciones

y nos devolverá lo siguiente como vemos en la figura 27:

```
[1] "1"  "2"  "4"  "5"  "6"  "7"  "8"  "10"
```

Figura 27: Salida de un ejemplo de la función de predicción de todas las opciones

Esta función de predicción a simple vista no presenta ningún inconveniente, pero una vez que pasemos a la validación cruzada, comprobaremos que no es posible obtener una matriz de confusión. Esto se debe a que como obtenemos un vector con

los valores que predice el árbol, el cual es posible que tenga más de un valor. Como es posible que el vector tenga más de un valor predicho, no tiene sentido construir una matriz de confusión, lo cual nos dificultaría la comparación con otros métodos. Únicamente, como veremos en los próximos apartados, podremos sacar el porcentaje de acierto y de fallo.

5.3 INTRODUCCIÓN AL CONCEPTO DE VALIDACIÓN CRUZADA

La validación cruzada o cross-validation se trata de una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Esta técnica consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar como de preciso es un modelo que se llevará a cabo en la práctica. Es una técnica muy utilizada en proyectos relacionados con la inteligencia artificial para validar modelos generados.

A continuación, vemos en la figura 28 un ejemplo de cross-validation con k=4:

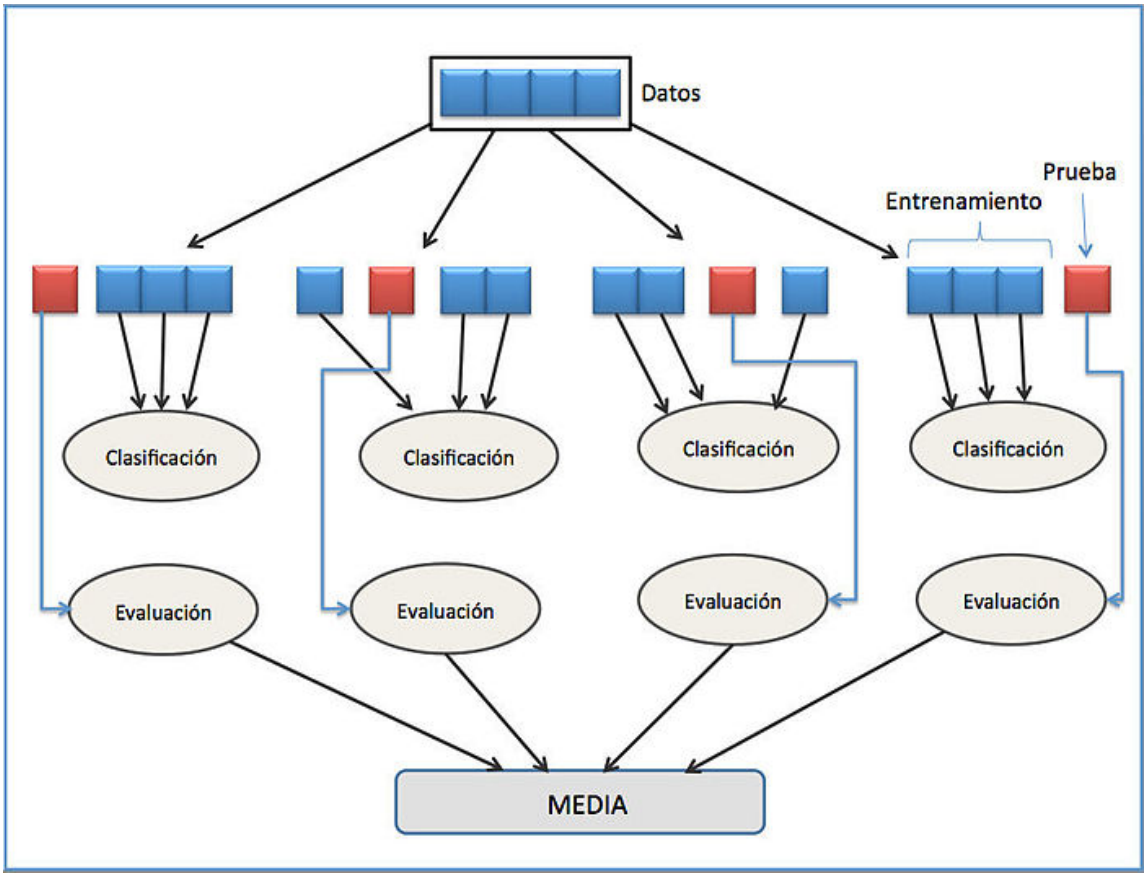


Figura 28: Proceso de cross-validation para k=4

La validación cruzada tiene su origen en el método de retención. El método de retención consiste en dividir en dos conjuntos complementarios los datos de la muestra, realizar el análisis de un subconjunto(llamado datos de entrenamiento), y validar el análisis en el otro subconjunto(llamado datos de prueba). De esta forma, la función de aproximación sólo se ajusta con el conjunto de datos de entrenamiento y a partir de aquí calcula los valores de salida para el conjunto de datos de prueba. Este método realiza únicamente una iteración, con lo cual es muy rápido respecto a la computación. Pero tiene el inconveniente de que no es demasiado preciso debido a la variación de los resultados obtenidos para diferentes datos de entrenamiento. Aquí en la figura 29, vemos una representación gráfica de este método.

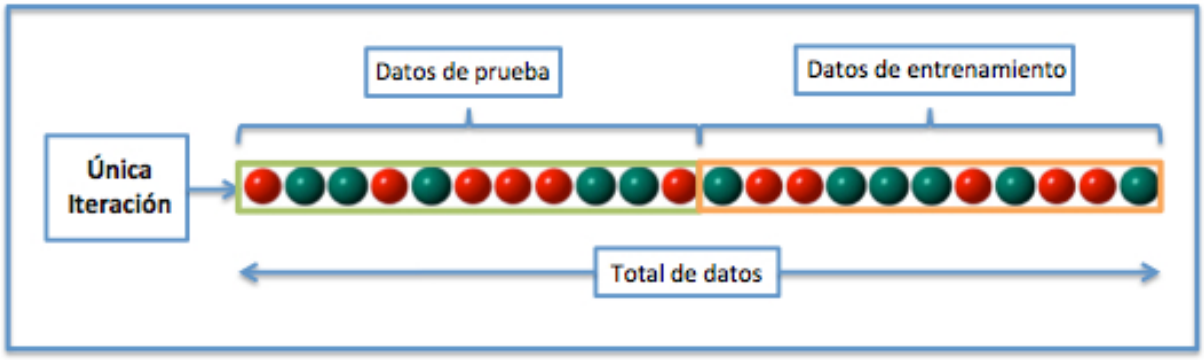


Figura 29: Método de retención

Como consecuencia de los inconvenientes del método de retención surge la validación cruzada o cross-validation.

Existen varios tipos de validaciones cruzadas como la validación cruzada de K iteraciones, la validación cruzada aleatoria o la validación cruzada dejando uno fuera, pero nos vamos a centrar en la validación cruzada de K iteraciones o K-fold cross-validation, que será la que vamos a utilizar.

Esta técnica consiste en dividir los datos en varios conjuntos de datos y luego elegir uno de ellos para "testear" y el resto para entrenar el árbol de decisión. Esto se hace de forma repetida hasta "testear" con cada conjunto de datos, guardando el resultado de cada iteración en una tabla para luego analizar la eficiencia de la predicción. Este método es bastante preciso ya que evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, aunque presenta la desventaja de ser computacionalmente lento. En la práctica, la elección del número de iteraciones depende de la medida del conjunto de datos, aunque la más usada habitualmente es la 10-fold cross-validation (k=10).

En la figura 30, vemos un ejemplo de validación cruzada con k=4.

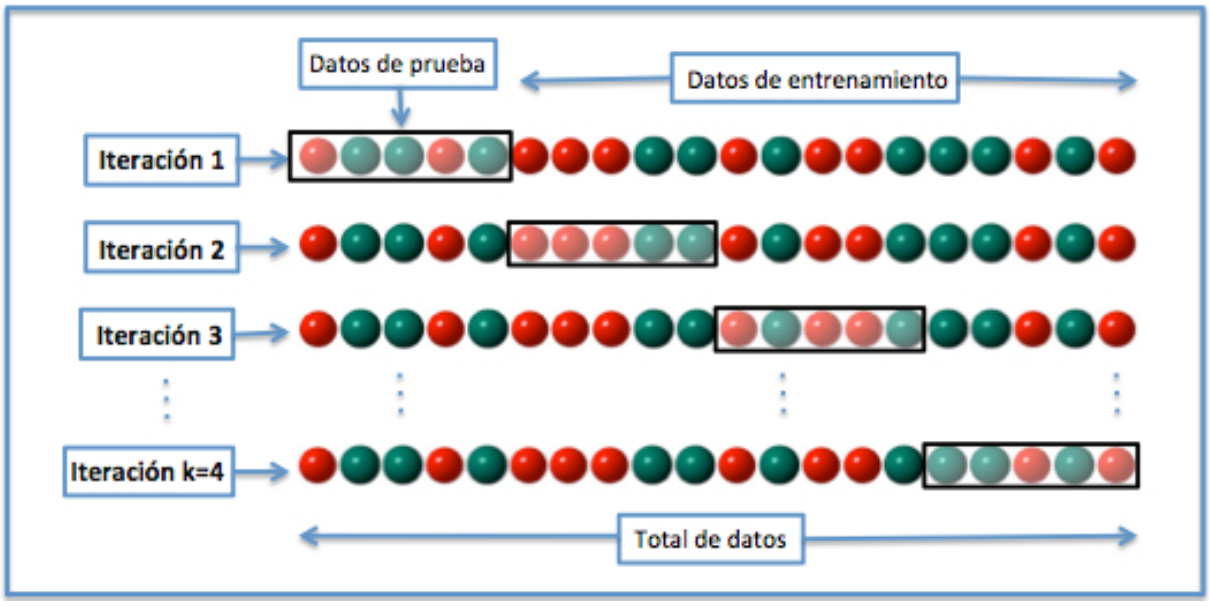


Figura 30: Selección de conjunto de entrenamiento y prueba para cross-validation con k=4

5.4 VALIDACIÓN CRUZADA PARA FUNCIÓN DE PREDICCIÓN DEL MÁXIMO VALOR

En la función que vamos a ver a continuación, lo que realizamos es todo el proceso de validación cruzada, obteniendo todos los árboles para cada iteración y mediante estos árboles de decisión obtenidos se realiza la predicción. Al realizar la predicción, sacaremos el porcentaje de acierto y de fallo del método, y la matriz de confusión que usaremos para comparar con los otros algoritmos ya existentes. En este caso, la función de predicción que utilizaremos será PredictMaxVal.

Cabe destacar que esta función realiza en su interior la construcción del árbol de decisión y la predicción, en cada una de las iteraciones. Es decir, aquí unimos todo lo visto hasta aquí.

A esta función, que llamaremos CrossValMaxVal, le pasaremos 3 atributos de entrada. Estos serán: kfolds, que será el número de iteraciones que realizaremos en la cross-validation; datos, que será el conjunto de datos de ejemplos, que para nuestro caso serán los pacientes; variables, que serán las variables utilizadas para la construcción del árbol en cada nivel. Como salida, vamos a imprimir varios resultados como son el vector de acierto, con los porcentajes de acierto del árbol en cada iteración, el porcentaje total de acierto, el porcentaje de fallo y la matriz de confusión. También devolveremos una lista con 3 elementos que serán la matriz de confusión, un vector con los datos reales de la tabla y otro vector con los valores obtenidos a partir de la función de predicción. Esta función la tenemos representada en la figura 31.

```
CrossValMaxVal <- function(kfolds, datos, variables) {
  #Inicializamos la matriz de confusion
  len <- length(variables)
  lon <- length(levels(datos[, len]))
  mat<- matrix(0,nrow=lon,ncol=lon)
  colnames(mat) <- levels(datos[, len])
  rownames(mat) <- levels(datos[, len])

  tamMuestra <- nrow(datos)/kfolds
  probs <- c()
```

```
  #Estas dos variables nos servirán para crear la curva ROC posteriormente
  vectDatosReales <- c()
  vectDatosPredichos <- c()
  for(i in 1:kfolds) {
    conj <- ((tamMuestra*(i-1))+1):(tamMuestra*(i))
    train <- datos[-conj, ]

    #Creamos el arbol
    tree <- Node$new("enfermedad")
    arbolDecMarcos(variables, train, tree)
    contador <- 0
    for (j in conj) {
      #Obtenemos el valor predicho
      val <- PredictMaxVal(tree,datos[j,])
      #Insertamos en la matriz de valores reales y predichos, los valores
      #reales y predichos

      vectDatosReales <- c(vectDatosReales, as.character(datos[j,
variables[len]]))
      vectDatosPredichos <- c(vectDatosPredichos, val)

      #Vemos si el valor predicho coincide con el dato de la tabla
      if(val==datos[j, variables[len]]) {
        contador <- contador +1
      }
      k <- as.character(datos[j, variables[len]])
      l <- as.character(val)
      mat[k,l] <- mat[k,l]+1
    }
    probs <- c(probs, contador)

  }
```

```
  #Devolvemos un vector de probabilidades
  vect<- probs/tamMuestra
  print(vect)
  print("Porcentaje de acierto: ")
  print(sum(vect)*10)
  print("Porcentaje de fallo: ")
  print(100- (sum(vect)*10))
  #Devolvemos la matriz de confusion
  print("Matriz de confusion: ")
  print(mat)
  list(matConfusion= mat, vectDatosReales=as.numeric(vectDatosReales),
vectDatosPredichos=as.numeric(vectDatosPredichos))
}
```

Figura 31: Función de cross-validation para la función de predicción del máximo valor

A continuación, vamos a ver un ejemplo de uso de esta función. Utilizaremos el conjunto de datos que estamos utilizando durante todo el trabajo. Como variables para la construcción del árbol, emplearemos todas las variables como explicativas, menos CON_DIAGES, que será la variable explicada que pondremos en último lugar. Introducimos por pantalla, lo que vemos en la figura 32.

```
variables <- c("CLI_CEFALE", "CLI_FEBRE", "CLI_VOMITO", "CLI_CONVUL",
+                "CLI_RIGIDE", "CLI_KERNIG", "CLI_ABAULA", "CLI_COMA",
+                "CLI_PETEQU", "CLASSI_FIN", "CON_DIAGES")
> kfolds <- 10
> CrossValMaxVal(kfolds, datos, variables)
```

Figura 32: Entrada de un ejemplo de la función de cross-validation del máximo valor

y en la figura 33, vemos que obtendremos lo siguiente:

```
[1] 0.34 0.49 0.44 0.39 0.27 0.40 0.44 0.39 0.33 0.59
[1] "Porcentaje de acierto: "
[1] 40.8
[1] "Porcentaje de fallo: "
```

```
[1] 59.2

[1] "Matriz de confusion: "

    -1 1  2 3 4  5  6  7 8 9 10

-1 238 6 10 1 4 16 29 13 1 0  2

1    4 8  4 0 0  0  7  3 0 0  1

2   11 4  5 0 0  4 26  5 1 0  2

3    0 3  1 0 0  0  6  0 1 0  0

4    2 0  0 0 0  0  5  0 0 0  0

5   12 0  4 1 0  2 51 39 0 0  4

6   24 4 12 5 2 15 85 61 1 1  3

7    9 1  2 1 1 27 79 70 1 0  0

8    1 0  1 1 0  2  7  8 0 0  0

9    0 0  0 0 0  0  2  2 0 0  0

10   1 4  2 0 1  7 11 10 0 0  0

$matConfusion

    -1 1  2 3 4  5  6  7 8 9 10

-1 238 6 10 1 4 16 29 13 1 0  2

1    4 8  4 0 0  0  7  3 0 0  1

2   11 4  5 0 0  4 26  5 1 0  2

3    0 3  1 0 0  0  6  0 1 0  0

4    2 0  0 0 0  0  5  0 0 0  0

5   12 0  4 1 0  2 51 39 0 0  4

6   24 4 12 5 2 15 85 61 1 1  3

7    9 1  2 1 1 27 79 70 1 0  0

8    1 0  1 1 0  2  7  8 0 0  0

9    0 0  0 0 0  0  2  2 0 0  0

10   1 4  2 0 1  7 11 10 0 0  0


$vectDatosReales

   [1] -1  5 -1  5  7 -1  2 -1  6  6 10  2  6  8  5 -1  6 10  6 -1  5  6
6 -1  7  6  6

  [28]  2  6  6  5  6  6  5 -1  2  8  7  5 10  6  6  7  2  6  5  5  6 10 -
1 -1 -1  7  7
```

```
  [55]  2 10 -1 -1  6  1 -1 -1  4 -1 -1 -1  1 -1  6 -1  6  8 -1  6  7 -1
7  7  8  7  7

  [82] 10 -1  5  5  7  7  7  6 -1  7  6 -1  6  2  1  5  5 10  6  6  7  7
6 -1  7  7  7

 [109]  5 10  7 -1  6 -1 -1 -1  6  6 -1  6  6  6  6  6  2 -1 -1 -1  7  6 -
1  6  3 -1  7

 [136]  6  5 10  2  7  7  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
7  7  7  7  7

 [163]  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
7  7  7  7  7

 [190]  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  2  7 -1  6  6 -1 -
1  6 -1 -1  4

 [217]  7  9  7  7 -1  6  2 -1 -1  7  7 -1  5 10  6  6  6  6  1  6  6 -1
6  6  2 -1  6

 [244] -1  6  6  7  7 -1 -1  7  7  6  2  6  7  6  6 -1  7  5  2  1  6  5 -
1 10 -1  6 -1

 [271]  6 10  2  6  3  6  7  7  6 -1  6 -1 -1  5  6 -1  6 -1  5  1  7  7 -
1  5  6 -1  7

 [298]  7  7  2  2  5 -1  3  5  7  7  5 -1  6  2  7  2 -1  7 -1  7  5 -1 -
1  1  5  8  3

 [325]  5  7  5  7  4  6 -1  5  7  5  5  7 -1  5  5  7  5  7 -1  6  5  7
2  5  7 -1 -1

 [352]  1  6  6 -1 -1 -1  7 -1  6  2  6 -1  6  2  2  7  3  4  6  2  5 -1 -
1  3 -1 -1 -1

 [379] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  5 10  7 10  6 -1  1  4  5 -1
7  5 -1  6  2

 [406]  7  6  6  5  2  5 10 -1 -1 -1  6  7  7  7  5  7  6  1  6  6  6  6
6  6 -1  7  2

 [433]  2  5  2 -1  6 -1  6  6  6  8  6  6 -1  5  5  2  5  5  7 -1  5  7 -
1 10  4  1  6

 [460]  5 -1  6 -1  7  7 10 -1 -1  6  2 -1  6  2 -1  6 10  7  6  6  6  6 -
1  7  9  7  7

 [487] 10 -1  6  6 -1  8  2  8  6 -1 -1  1  7  7  6  7 -1 -1 -1  6 -1  6
6  2  6  6  6

 [514]  6  6 -1  6  6  7  5  6  6 10 10 -1 -1  1  5  5  2 -1  5  2  2 -1
2  6 -1  6  6

 [541] -1  6  6 -1  5  5 -1  5 -1  5  8  5  6  5  5 -1 -1  2 -1 -1  5 -1
5 -1  6  7  5
```

```
 [568]  6  6  7  7  5  5  7  7  5  7 -1  5  5  7  7  5 -1  6  1  8 -1  5
6 -1  5  7 10

 [595]  5  6 -1  6  8 -1 -1 -1  6  6 -1  6 -1  6 -1 -1  2 -1 -1  6  6  6 -
1  9  7 10 -1

 [622] -1  6  7 -1  7  7  7  7  7 -1  5 -1 -1  6 -1 -1 -1  6  1  6  6 -1 -
1  5 -1 -1 -1

 [649] -1 -1  6  6 -1  6  6 -1 -1  8  3 10  2  6 -1 -1  5 10 -1  5  5  7
5 -1  5 -1  5

 [676] 10  5  5  5 -1  5 -1 -1 -1  6 -1  6 -1 -1  5 -1 -1 -1  6  2 -1  6
6  8  8 -1  5

 [703] -1  8 -1  6 -1 -1  1  5  2  5  5  7  5  2 -1 -1  5 -1  6  7  7  6
6 -1  3  6 -1

 [730]  6  2  6  6 -1  6  6  6  7  7 -1 -1  7  6  1  6 -1  5 -1  5  6  6
6  6  6 -1 -1

 [757]  6  6 10  6 -1  2  6  6 -1  6  6  7  6  6  2  5 -1  4  2  5  9  5 -
1  7  6  6 -1

 [784] 10  2  7 -1  6  6  6 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  5 -1 -1 10 -1
5  5  8 -1 -1

 [811]  1 -1  6  5 -1  2  6 -1 -1 -1  7  5  7  7  7  7 -1  7 -1  6  6  7
1 10  7 -1 10

 [838]  1 -1  1  5  1 -1  1  5  7 -1 -1 10 10  1  6  3  6  6  5 -1  5 10 -
1 -1 -1 -1 -1

 [865]  6  6  6  6 -1  2  2 -1 -1  6  6  8  6 10 -1  6  2  6 10 -1  1  2 -
1  5 -1 -1  5

 [892]  5 -1 -1  6 -1 -1  6 -1  6  2  7  8  7 -1  6  6  7 -1 -1  6  7  5
8  6  6 -1  5

 [919] -1 -1 -1  2  7  8  1  7 -1 -1  3  7 -1  6 -1 -1  5 -1  7 -1  2 -1 -
1 -1  6 -1  7

 [946] -1  5  3  7  7  7  7 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
1 -1 -1 -1 -1

 [973] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  7 -1 -1 -1 -1 -1 -1 -1
2 -1 -1 -1  1

[1000] -1


$vectDatosPredichos

   [1] -1  7  6  7  7 -1 -1 -1  5  5  1  1  7  7  6  2  7  6  7 -1  6  7
7 -1  6  7  6
```

```
  [28]  5  6  2 -1  7  7  7  6  2  6  7  7  7 -1  7  6  6  6  7  6  7  6 -
1 -1 -1  1  7

  [55]  6  7 -1 -1  7  1  1 -1  6 -1 -1 -1  6 -1 -1 -1  7  6 -1  1  7 -1
7  7  3  7  5

  [82]  7  1  7  7  6  7  2  3 -1  7  7 -1  7 -1 -1 -1  7  7  9  1  6  7
7 -1  7  5  6

 [109]  6  1  6 -1  5  2 -1 -1  5 -1  6  7  7  5  6  6 -1 -1 -1  2  6  7 -
1  6  6 -1  7

 [136]  6  7  4  5  6  5  7  7  7  7  5  7  7  7  5  5  6  5  7  7  7  7
6  7  6  7  5

 [163]  7  7  5  7  7  6  7  6  6  7  5  6  7  8  7  5  7  6  6  7  5  5
7  7  7  7  6

 [190]  7  7  6  7  5  7  7  7  7  6  7  7  6  5 -1  6  6  7  5  6  5 -1 -
1  6 -1 -1  6

 [217]  6  6  7  6  6  6  6  7 -1  7  6 -1  6  2  7  7  6  6  2  6  7 -1
6  6  6  6  6

 [244] -1  6  6 -1  5 -1 -1  4  6  5  6  6 -1  6  6 -1  7  7  6  7  8  6 -
1  5  8 -1  5

 [271]  6  6  6  6  6  7  6  6  6  5  6 -1  6  7  6 -1  6 -1  6  1  6  7 -
1  6  5  5  6

 [298]  7  6 -1  6  5 -1  1  6  5  6 -1 -1  6  7  6  7 -1  6 -1  6  6 -1 -
1  1  6  6  2

 [325]  6  6  7  5  6  1  1  6  2  7  7  6  5  2  7  6  6  6 -1  6  6  5
8  6  6 -1 -1

 [352]  6  3  6 -1  5 -1 -1 -1  2 -1  6 -1  6  6  1  7  8  6  7  7  6 -1 -
1  1 -1 -1 -1

 [379] -1 -1 -1 -1 -1 -1 -1 -1 -1  6 -1 -1 10  5  5  1 10  5  1 -1  7 -1
5 -1  6  5  6

 [406]  6  6  7  6  7  6  5 -1  6 -1  5  7  6  7  6  6  7 -1 -1  6  7 -1
7  6 -1  6  6

 [433]  6  7  6 -1  7  5  5  4  1  7  2  7 -1  7 10  5  6  6  6  7  6  6
6  7 -1  6  6

 [460] -1 -1  7  1  5  6  6  7 -1  2  6 -1  6  6 -1  6  7  7  6  7  6  6
6  6  6  6  5

 [487]  5 -1  5  6  5  7 -1  5 -1 -1 -1  1  6  7  7  7  6 -1 -1  6  5  6
2 10  7  6  6

 [514]  6  4 -1  6  7  6  6  6  6  6  6  2  7  7  7  7  2 -1  6 10  6 -1 -
1  7 -1  6  7
```

```
 [541]  5  6  7 -1 -1  3 -1  6 -1  5  7 -1  7  6  6 -1 -1  1 -1 -1  6 -1
7 -1  6  7  6

 [568]  7  3  6  6  6  6  6  6  2  6 -1  6  7  6  6  7 -1  6  1  7 -1  6
6 -1  6  5  6

 [595]  7  6 -1  6  6  4  7 -1 -1  6 -1  7 -1  6 -1 -1  6  6 -1  7  7 -1
2  7  6  7  6

 [622] -1  2  6 -1  7  7  5  7  7 -1  6 -1 -1 10 -1 -1 -1  2  6 -1  6  6
7  7 -1 -1 -1

 [649] -1 -1  6  7  6 -1 -1 -1 -1  2  1  2 -1  7 -1  6  6  7 -1  6  2  6 -
1 -1  7 -1  7

 [676]  5  7 10 -1 -1  6 -1  4 -1 -1 -1  7  4 -1  7 -1 -1 -1  6  2  6  6
7  6 -1 -1  6

 [703] -1  6  7  6 10  7  2  7  6  7 -1  7  7  2  3 -1  6 -1  3  6 -1  6
6 -1  6  5 -1

 [730]  7  2 -1 -1 -1  7  6  6  6  6 -1 -1  7  5 10  6  5  7 -1  6  2 -1
6  7  6 -1 -1

 [757]  7 -1  5  7 -1  7  7  6 -1  7 -1  3  3 -1  1  7  6  6  6  7  7  6
7 -1  6  7 -1

 [784]  7  6  6 -1 -1  7  7 -1 -1 -1 -1 -1  7 -1 -1 -1 -1  7  4 -1  5 -1
6  6  5 -1 -1

 [811]  6 -1 -1  6 -1 -1  6 -1 -1  6  6  7  6 -1  6  6 10  7  1  6  6  5
2  1  6  6  6

 [838]  7 -1  1 10  2 -1 -1  6  6  7  6  7  6  6  6  6 -1  6  6 -1  2  6
6 -1 -1 -1  5

 [865] -1  6  7 -1  5  6 -1 -1 -1  6  2  7 10 -1 -1  2  5  2  6 -1 -1  6 -
1  6  6  2 -1

 [892]  6 -1 -1  6 -1  2  2 -1  5  6  6  7  6 -1  7  6  6  5 -1  6  5  6
6  6  7  5  7

 [919] -1 -1  6 -1  7  7  6  6  7 -1  6  7 -1  6 -1 -1 -1  6  6 -1  6 -1 -
1  2  7 -1 -1

 [946] -1  7  6 -1  6  6  7 -1 -1 -1 -1  6 -1 -1 -1 -1  2 -1 -1 -1 -1 -1 -
1 -1 -1 -1  2

 [973] -1 -1 -1 -1 -1 -1  6 -1 -1 -1 -1 -1  6 -1  6  1 -1 -1 -1 -1  7 -1
6 -1 -1 -1  1

[1000] -1
```

Figura 33: Salida de un ejemplo de la función de cross-validation del máximo valor

5.5 VALIDACIÓN CRUZADA PARA FUNCIÓN DE PREDICCIÓN DE TODAS LAS OPCIONES

La siguiente función realizamos el proceso de validación cruzada para la segunda función de predicción construida. Obtendremos un árbol de decisión para cada iteración y con estos árboles realizaremos la predicción. Una vez realizada la predicción, sacaremos el porcentaje de acierto y de fallo del método, y la matriz de confusión que emplearemos para comparar con los otros algoritmos ya existentes. En este caso, la función de predicción que utilizaremos será PredictAllOptions.

A esta función que llamaremos CrossValAllOptions, le pasaremos 3 atributos de entrada. Estos serán: kfolds, que será el número de iteraciones que realizaremos en la cross-validation; datos, que será el conjunto de datos de ejemplos, que para nuestro caso serán los pacientes; variables, que serán las variables utilizadas para la construcción del árbol en cada nivel. Como salida, vamos a obtener varios resultados como el vector de acierto, con los porcentajes de acierto del árbol en cada iteración, el porcentaje total de acierto y el porcentaje de fallo. El código de esta función lo tenemos representado en la figura 34.

```
CrossValAllOptions <- function(kfolds, datos, variables) {
  len <- length(variables)
  tamMuestra <- nrow(datos)/kfolds
  probs <- c()
  for(i in 1:kfolds) {
    conj <- ((tamMuestra*(i-1))+1):(tamMuestra*(i))
    #Tomamos un conjunto de entrenamiento
    train <- datos[-conj, ]

    #Creamos el arbol
    tree <- Node$new("enfermedad")
    arbolDecMarcos(variables, train, tree)
    contador <- 0
    for (j in conj) {
      #Obtenemos un vector de los datos predichos
      val <- PredictAllOptions(tree,datos[j,])
```

```
      #Vemos si el valor del conjunto de datos está en el vector de
los datos predichos
      if(datos[j, variables[len]] %in% val) {
        contador <- contador +1
      }
    }
    probs <- c(probs, contador)
  }
  #Devolvemos un vector de probabilidades
  vect<- probs/tamMuestra
  print(vect)
  print("Porcentaje de acierto: ")
  print(sum(vect)*10)
  print("Porcentaje de fallo: ")
  print(100- (sum(vect)*10))
}
```

Figura 34: Función de cross-validation para la función de predicción de todas las opciones

Ahora, vamos a ver un ejemplo de uso de esta función. En este caso, introduciremos por pantalla lo que tenemos en la figura 35.

```
variables <- c("CLI_CEFALE", "CLI_FEBRE", "CLI_VOMITO", "CLI_CONVUL",
+                "CLI_RIGIDE", "CLI_KERNIG", "CLI_ABAULA", "CLI_COMA",
+                "CLI_PETEQU", "CLASSI_FIN", "CON_DIAGES")
> kfolds <- 10
> CrossValAllOptions(kfolds, datos, variables)
```

Figura 35: Entrada de un ejemplo de la función de cross-validation de todas las opciones

y obtendremos como salida lo que vemos en la figura 36.

```
[1] 0.70 0.77 0.67 0.68 0.54 0.72 0.61 0.63 0.50 0.75
[1] "Porcentaje de acierto: "
```

```
[1] 65.7

[1] "Porcentaje de fallo: "

[1] 34.3
```

Figura 36: Salida de un ejemplo de la función de cross-validation de todas las opciones

Con esta función, como vemos no es posible obtener la matriz de confusión, con lo que no será posible poder realizar comparaciones con los resultados de otros algoritmos sobre árboles de decisión.

6. COMPARACIÓN DE RESULTADOS

6.1 INTRODUCCIÓN

El objetivo de este apartado será realizar una comparación de los resultados obtenidos con nuestro árbol de decisión relacionados con la predicción, con los resultados que se obtienen con otros métodos que ya existen.

Para obtener los resultados que compararemos, realizaremos una función en R que nos devolverá ciertos resultados. Mientras que con los otros métodos ya existentes, utilizaremos la herramienta Weka, debido a que la forma en que nos devuelve los resultados es más comprensible para el usuario y para poder comparar mejor resultados.

6.2 VARIABLES NECESARIAS PARA LA COMPARACIÓN DE RESULTADOS

Para la comparación de resultados hemos seleccionado varias variables, para las cuales realizaremos una función que la calcule. Definiremos estas variables orientándolas a nuestro problema.

Antes de definir estas variables, vamos a considerar un problema de predicción binario, ya que tomaremos un síntoma de un individuo y veremos si ese síntoma se da realmente o no. Estos resultados lo clasificaremos como positivos(P) o negativos(N). Ahora vamos a considerar 4 casos resultantes de las combinaciones entre los valores reales y predichos, que como hemos comentado, se han clasificado como P o N:

- Si el valor real es P y el valor predicho es P, tendremos un verdadero positivo.
- Si el valor real es P y el valor predicho es N, tendremos un falso negativo.
- Si el valor real es N y el valor predicho es P, tendremos un falso positivo.
- Si el valor real es N y el valor predicho es N, tendremos un verdadero negativo.

Estos cuatro valores lo podemos expresar en una matriz de confusión 2x2 como vemos en la figura 37:

		Clase real	
		Clase referencia	**Clase no referencia**
Clase estimada	Clase referencia	TP	FP
	Clase no referencia	FN	TN

Figura 37: Matriz de confusión 2x2

Estos 4 valores serán utilizados en el cálculo de las variables que veremos a continuación.

La sensibilidad es la probabilidad de clasificar correctamente a un individuo con un síntoma, es decir, la probabilidad de que para un sujeto con un síntoma se obtenga en la prueba un resultado positivo. La sensibilidad se trata, por tanto, de la capacidad del test para detectar la enfermedad. También es conocida como fracción de verdaderos positivos. Utilizaremos la fórmula siguiente:

$$Sensibilidad = \frac{VP}{VP + FN}$$

La especificidad es la probabilidad de clasificar correctamente a un individuo que no tenga un determinado síntoma, es decir, la probabilidad de que para un sujeto sin un síntoma se obtenga un resultado negativo. También es posible definirla como la capacidad de detectar a los pacientes sanos. Es posible llamarla también fracción de verdaderos negativos. Para calcular la especificidad, haremos uso de la siguiente fórmula:

$$Especificidad = \frac{VN}{VN + FP}$$

La razón de falsos positivos vendrá dada por:

$$Razón\ de\ Falsos\ Positivos = \frac{FP}{VN + FP}$$

La exactitud (accuracy) nos indicará el porcentaje de individuos que está bien clasificado.

$$Exactitud = \frac{VP + VN}{Total\ de\ individuos}$$

La precisión o valor predictivo positivo nos indicará la probabilidad de tener el síntoma si el resultado de la prueba es positivo.

$$Precisión = \frac{VP}{VP + FP}$$

En R, hemos creado una función que nos calcula estos valores, que será la que vemos en la figura 38:

```
DerivadosMatConfusion <- function(mat) {
  sens <- c()
  razonFP <- c()
  accuracy <- c()
  especificidad <- c()
  precision <- c()
  for(i in 1:nrow(mat)) {
    vp <- VerdaderosPositivos(mat,i)
    fn <- FalsosNegativos(mat,i)
    vn <- VerdaderosNegativos(mat,i)
    fp <- FalsosPositivos(mat,i)
    sensi <- vp/(vp+fn)
    preci <- vp/(vp+fp)
    sens <- c(sens, sensi)
    razonFP <- c(razonFP, fp/(fp+vn))
    accuracy <- c(accuracy, (vp+vn)/(vp+vn+fp+fn))
    especificidad <- c(especificidad, 1- (fp/(fp+vn)))
    precision <- c(precision, preci)
    ##rMeasure <- c(rMeasure, (2*sensi*preci)/(sensi+preci))
  }
  print("Sensibilidad")
  print(sens)
  print("Razon de falsos positivos")
  print(razonFP)
  print("Accuracy")
  print(accuracy)
  print("Especificidad")
  print(especificidad)
  print("Precision")
```

```
  print(precision)
  ##print("R-Measure")
  ##print(rMeasure)
}
```

Figura 38: Función que calcula valores para comparar con otros métodos

A esta función le pasaremos una matriz de confusión, y nos devolverá los valores de las variables comentados anteriormente. Para nuestro caso, la matriz de confusión será la obtenida mediante la validación cruzada.

Finalmente, vamos a hablar sobre la curva ROC. Una curva ROC (Receiver Operating Characteristic, o Característica Operativa del Receptor) es una representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario según se varía el umbral de discriminación. Ha tenido un gran desarrollo en los últimos año en el proceso diagnóstico. Es muy útil para test con datos cuantitativos.

Tradicionalmente, cuando se tenía un test cuantitativo, se elegía el mejor punto de corte, que combinaba la mejor sensibilidad y especificidad del test (mayor rendimiento). Con esto, teníamos una pérdida de información, cuando transformábamos una variable continua en una dicotómica.

Ahora con las curvas ROC, podemos graficar los diferentes puntos de corte de las características operativas del test, es decir, la sensibilidad y la especificidad. El gráfico será una relación entre los verdaderos positivos (VP) y los falsos positivos (FP). En el eje Y, tendremos la sensibilidad, mientras que en el X, tendremos el valor de 1-especificidad. Esta curva nos va a ser muy útil porque nos da una idea del rendimiento del test.

Veamos en la figura 39, un ejemplo de un gráfico de la curva ROC:

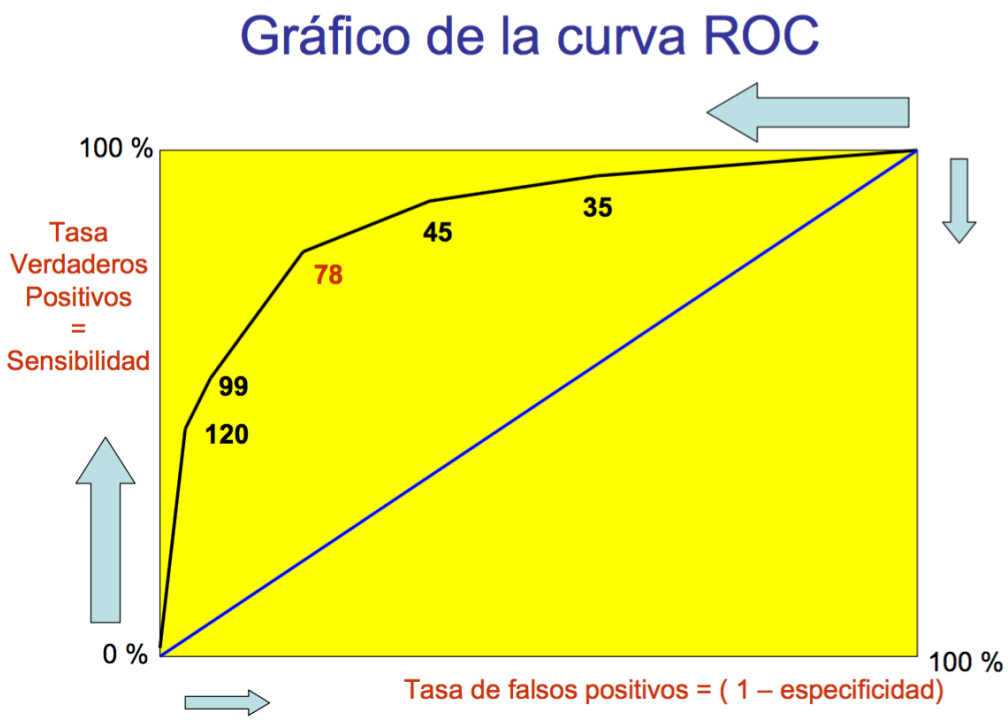


Figura 39: Gráfico de la curva ROC

Un dato que será muy útil, que será el que nos servirá para la comparación, va a ser el área bajo la curva. A mayor valor, mejor es el test. Veamos algunos valores:

- Máximo 1 -> Test perfecto
- Mayor de 0.9 -> Excelente test
- 0.8 a 0.9 -> Buen test
- 0.7 a 0.8 -> Regular test
- 0.5 a 0.7 -> Mal test
- 0.5 -> No relación
- Menor de 0.5 -> Relación inversa

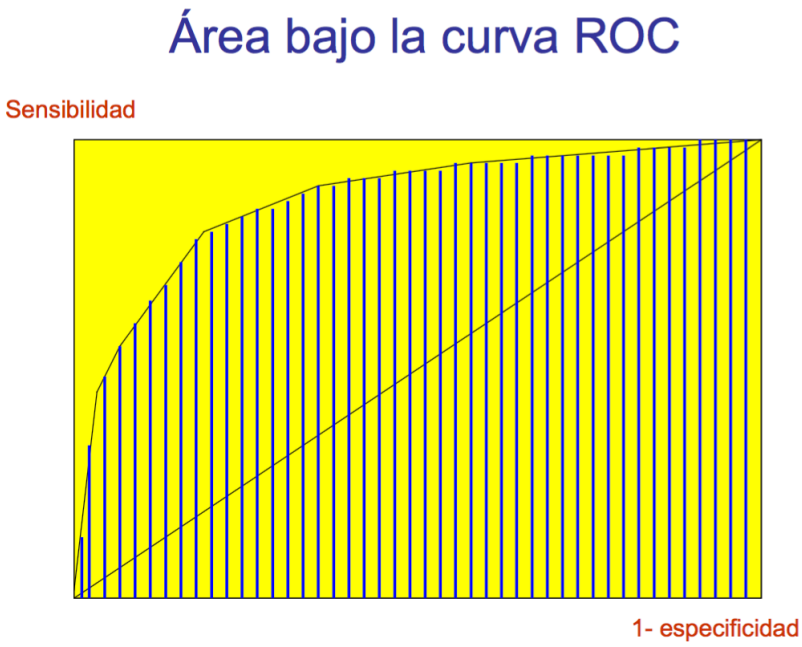


Figura 40: Área bajo la curva ROC

Veamos un gráfico, en la figura 41, con la comparación de varias curvas ROC y donde se encontraría una mejor y otra peor respectivamente.

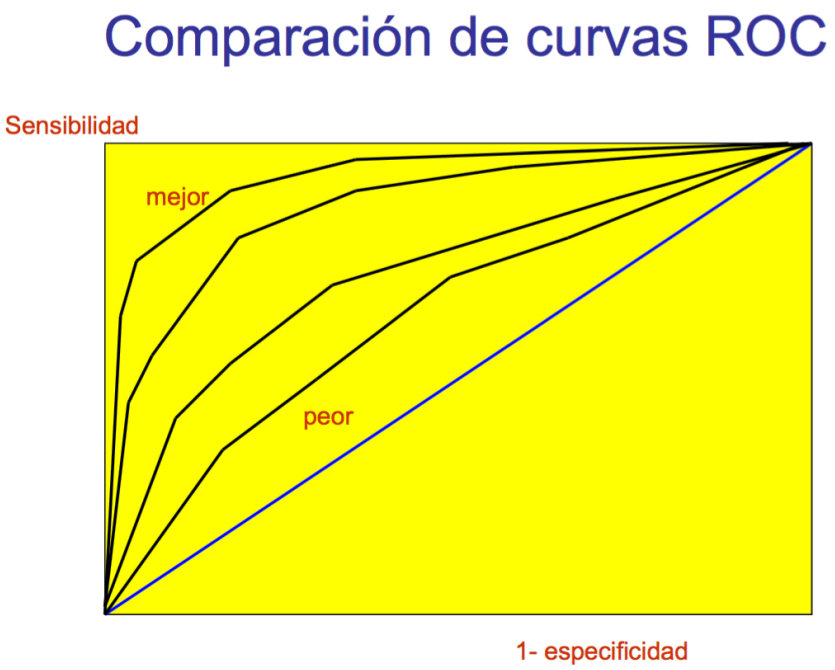


Figura 41: Gráfico de comparación de curvas ROC

Para el cálculo de la curva ROC y su área, hemos realizado la siguiente función, que calcula dicho valor. Para ello, hemos utilizado en la función la librería implementada en R, ROCR, que la cargaremos al inicio del código.

A esta función, le hemos pasado como parámetros de entrada, el conjunto de datos, y los vectores de valores reales de la tabla de datos y de valores predichos obtenidos con la función de predicción. Como salida, nos devolverá un vector con el área bajo la curva ROC para cada uno de los valores de la clase. En la figura 42, vemos el código de la función:

```
curvaROC <- function(vectReal, vectPred, datos) {
 len <- length(datos)
 niv <- levels(datos[, len])
 niv <- as.numeric(niv)
 long <- length(vectReal)
 valorRoc <- c()
 for(i in 1:length(niv)) {
   vectRealAlter <- c()
   vectPredAlter <- c()
   for(j in 1:length(vectReal)) {

     if(vectReal[j]==niv[i]) {
       vectRealAlter <- c(vectRealAlter, 1)
     }
     else {
       vectRealAlter <- c(vectRealAlter, 0)
     }
   }
   for(k in 1:length(vectPred)) {
     if(vectPred[k]==niv[i]) {
       vectPredAlter <- c(vectPredAlter, 1)
     }
     else {
       vectPredAlter <- c(vectPredAlter, 0)
     }
   }
```

```
        predict.roc <- prediction(vectPredAlter, vectRealAlter)
        perf.roc <- performance(predict.roc, "tpr", "fpr")
        plot(perf.roc)
        auc <- as.numeric(performance(predict.roc, "auc")@y.values)
        valorRoc <- c(valorRoc, auc)

      }
      return (valorRoc)
    }
```

Figura 42: Función que calcula el área de la curva ROC

6.3 COMPARACIÓN DE MÉTODOS EXISTENTES

Para realizar la comparación de resultados de nuestro método con otros métodos existentes, vamos a utilizar los datos y el conjunto de variables que estamos empleando durante toda la memoria. Para obtener los resultados con los diferentes métodos existentes utilizaremos la herramienta Weka y obtendremos los resultados para los algoritmos que Weka se llaman Id3, J48, NBTree, Naive Bayes, BayesNet.

En primer lugar, si estudiamos el porcentaje de cada método observamos que los algoritmos implementados en Weka rondan entre 50 y 60% de acierto (Árbol Id3 51%, Árbol J48 57.9%, NBTree 56.4%, Clasificador NaiveBayes 55.8%, Clasificador BayesNet 56.3%). Con nuestro método con la primera función de predicción tiene un 40.8% de acierto, que se aleja un poco del porcentaje de acierto de los métodos existentes, pero esto se debe a que existen muchas instancias que se han clasificado correctamente por el árbol pero la función de predicción como tenemos que escoger el valor con el máximo de observaciones, pues pierde un poco de eficiencia. Mientras que con el segundo método de predicción observamos que se obtiene un 65.7% de acierto, que mejora en varios puntos la eficiencia de los métodos anteriores. El problema de este método es que no podemos obtener la matriz de confusión y por consiguiente la curva que nos dará una mejor visión acerca de la eficiencia del método.

A partir de ahora, los resultados se compararán únicamente con la primera función de predicción.

En relación con el valor de la curva ROC, con nuestro método obtenemos que el área bajo la curva ROC es de 0.6358279, lo que nos dice que es un test regular, pero esto es debido a los datos que estamos utilizando y al problema comentado

anteriormente. Este valor se obtiene haciendo la media entre los valores del área bajo la curva ROC para el valor de cada clase, ponderándolas con el número de observaciones de cada tipo. En el método Id3, se obtiene que el área bajo la curva es 0.751, que nos indica que el test es regular. Mientras que con el resto de métodos, obtendremos valores entre 0.8 y 0.9 (Árbol J48 0.811, NBTree 0.838, Clasificador NaiveBayes 0.837, Clasificador BayesNet 0.839). Esto nos indicará que el test con estos métodos es bueno.

El resto de valores como la especificidad, sensibilidad, exactitud y algunos más calculados con la función de DerivadosMatConfusion, que nos reafirman en lo comentado anteriormente. En la figura 43, vemos que los resultados con nuestra función de predicción son los siguientes:

```
      [1] "Sensibilidad"

       [1] 0.78807947 0.26666667 0.12195122 0.00000000 0.00000000 0.02739726
0.27597403

       [8] 0.33175355 0.00000000 0.00000000 0.00000000

      [1] "Razon de falsos positivos"

       [1]  0.117478510  0.019587629  0.055265902  0.011099899  0.007056452
0.119741100 0.184971098

       [8] 0.153358682 0.020100503 0.004004004 0.036437247

      [1] "Accuracy"

       [1] 0.854 0.959 0.911 0.980 0.985 0.818 0.649 0.738 0.975 0.995 0.952

      [1] "Especificidad"

       [1]  0.8825215  0.9804124  0.9447341  0.9889001  0.9929435  0.8802589
0.8150289 0.8466413

       [9] 0.9798995 0.9959960 0.9635628

      [1] "Precision"

       [1] 0.74375000 0.29629630 0.08620690 0.00000000 0.00000000 0.01769912
0.39906103

       [8] 0.36649215 0.00000000 0.00000000 0.00000000
```

Figura 43: Resultados obtenidos a partir de la función DerivadosMatConfusion

7. CONCLUSIONES Y LÍNEAS FUTURAS

7.1 CONCLUSIONES

Como resultado del desarrollo de este TFG se pueden extraer las siguientes conclusiones:

1. **Obtener una visión general sobre los árboles de decisión existentes hasta la actualidad y estudio sobre las librerías existentes en R o en Weka que implementen estos métodos.** En relación a este apartado, se ha realizado un estudio sobre algunas de las técnicas sobre árboles de decisión como ID3, C4.5 y una breve introducción sobre los métodos bayesianos. También, hemos explicado las librerías que tenemos ya implementadas en R, como rpart, party, etc.
2. **Implementar un árbol de decisión con ciertas características**. Hemos creado un árbol de decisión a partir de un conjunto de árboles y un conjunto de variables que nos indican la decisión a tomar en cada momento. Cabe destacar que el código de esta función es válido para cualquier conjunto de datos y no únicamente para el conjunto de datos con el que hemos trabajado durante el trabajo.
3. **Realizar una función de predicción para el árbol de decisión creado.** Se han realizado dos funciones para la predicción en lugar de una como habíamos previsto. El motivo de realizar dos funciones se debe a separar el caso en que tengamos más de un valor predicho. En este caso, elegiremos el valor con más observaciones o un vector con todas las observaciones.
4. **Implementar el método de validación cruzada para el árbol y la función de predicción construidas.** Para la validación cruzada, hemos creado dos funciones diferenciando las dos funciones de predicción creadas.
5. **Realizar una comparación entre los métodos existentes y el método creado.** En la comparación realizada, obtenemos que nuestro con la primera función de predicción es algo peor que los demás en relación con la eficiencia, sobre todo podemos verlo al obtener el área bajo la curva ROC. Con la segunda función de predicción, se obtiene un mejor porcentaje de acierto que en el resto de métodos, pero no es posible obtener el gráfico de la curva ROC con esta función de predicción.

7.2 LÍNEAS FUTURAS

En relación a las líneas futuras, sería interesante mejorar el porcentaje de acierto para la función de predicción del máximo valor. En relación para la función de predicción de todas las opciones, sería interesante poder realizar más comparaciones con otros métodos, que no únicamente con el porcentaje de acierto y fallo. En

resumen, la idea sería construir una función de predicción que unificara la idea de ambas funciones de predicción.

También estaría interesante incorporar alguna función más a la API, sobre todo orientada a la fase de preprocesamiento de los datos.

Bibliografía

1. Documentación y descarga de RProject. Disponible en: http://www.r-project.org
2. **ZUUR**, A.; **IENO**, E.N.; **MEESTERS**, E. (2009). A Beginner's Guide to R. Springer.
3. Documentación y descarga de Weka. Disponible en: http://www.cs.waikato.ac.nz/~ml/weka/
4. **ALER,** R. (2010). Tutorial sobre Weka. Curso Herramientas de la inteligencia artificial. [Consultado en Septiembre 2016]. Disponible en: http://ocw.uc3m.es/ingenieria-informatica/herramientas-de-la-inteligencia-artificial/contenidos/transparencias/TutorialWeka.pdf/view
5. **SANTANA**, J.S; **MATEOS**, E. (2014). El arte de programar en R: un lenguaje para la estadística. Instituto Mexicano de Tecnología del Agua.
6. **MITCHELL**, T. (1997). Machine Learning. McGraw-Hill.
7. **MURPHY**, K.P.; Machine learning : a probabilistic perspective. Cambridge, 1970
8. **RIPLEY,** B. (2015). Documentación del paquete "rpart". [Consultado en Septiembre 2016]. Disponible en: https://cran.r-project.org/web/packages/rpart/index.html
9. **RIPLEY,** B. (2016). Documentación del paquete "tree". [Consultado en Septiembre 2016]. Disponible en: https://cran.r-project.org/web/packages/tree/index.html