Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Estudio de Bootstrapping en Algoritmos de Clasificación

Autor: Gonzalo Franco Ceballos
Tutor: Teodoro Álamo Cantarero

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Estudio de Bootstrapping en Algoritmos de Clasificación

Autor:
Gonzalo Franco Ceballos

Tutor:
Teodoro Álamo Cantarero
Catedrático de Universidad

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Grado: Estudio de Bootstrapping en Algoritmos de Clasificación

Autor: Gonzalo Franco Ceballos

Tutor: Teodoro Álamo Cantarero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mis padres

AGRADECIMIENTOS

Agradezco a mi profesor y tutor Teodoro Álamo por toda su ayuda e increíble disposición a lo largo del desarrollo de todo el proyecto, y más aún por brindarme la posibilidad de colaborar con él estos últimos tres años. Ha sido una experiencia muy enriquecedora y motivadora que me ha ayudado a entender mejor numerosos campos de interés del área de Ingeniería de Sistemas y Automática.

También agradezco a mi familia por su incondicional apoyo durante toda la carrera, especialmente a mis padres, que han estado junto a mí transmitiéndome ánimos durante todo el camino.

RESUMEN

Este documento recoge un estudio de la aplicación de técnicas basadas en *bootstrapping* a problemas de clasificación.

La primera técnica, el *bagging*, pretende mejorar los resultados y la estabilidad del clasificador mediante un conjunto de expertos entrenados con elementos muestreados mediante *bootstrapping*. La segunda técnica tiene como objetivo realizar una predicción del ratio de acierto de un clasificador entrenado mediante *bagging* con una confianza determinada.

Para la realización del estudio se ha probado cada una de las técnicas en tres problemas de clasificación distintos, dos de ellos multiclase, con el objetivo de comprobar si las técnicas estudiadas tienen impactos diferentes en cada uno de los problemas.

El objetivo final del estudio es el desarrollo de estos algoritmos de mejora de resultados y la comprobación de que realmente se consigue una mejora sustancial de los mismos.

ABSTRACT

The following report analyzes a study of the application of several techniques based on *bootstrapping* to a classification problem.

The first of these techniques, the *bagging technique*, seeks to improve the results and the stability of the classifier by means of a group of experts trained with elements sampled using *bootstrapping* techniques. The second technique's goal is to make a prediction of the success rate of the classifier trained using *bagging* techniques with a designed confidence.

To conduct the study, each of these techniques have been tested on three different classification problems, two of them being multiclass, in order to observe if their impact varies from one problem to another.

The main goal of the study is the development of the algorithms and checking that there is actually a notable improvement in the results obtained.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Durante mis años de colaboración en el departamento de Ingeniería de Sistemas y Automática he tenido la oportunidad de profundizar en el tema del procesamiento de datos y de información. Gracias a esto, he investigado y trabajado en varios campos que tratan grandes cantidades de datos con distintos objetivos. Uno de ellos es el *Aprendizaje Automático*.

El proceso mediante el cual un sistema informático puede aprender a distinguir patrones, predecir sucesos o inferir relaciones es realmente interesante. Un aspecto que me llamó inmediatamente la atención es lo sujeto que están unos buenos resultados a la buena intuición del programador. A menudo no es sencillo distinguir cuál puede ser la mejor aproximación a un problema, y está en la mano de la persona que programa el algoritmo el elegir correctamente tanto las técnicas de inferencia, clasificación, etc. como las de tratamiento de datos con el fin de aumentar el rendimiento y la precisión del programa.

Es en este segundo aspecto en el que se centra este estudio.

1.1 Objetivos

A continuación se describen los objetivos fijados para la realización del estudio. Se muestran en orden cronológico, tal como se han ido alcanzando a lo largo del proceso.

1- Entendimiento de un algoritmo de clasificación. No sólo el que se utiliza en este estudio, durante mi tiempo de colaboración en el departamento de Ingeniería de Sistemas y Automática, he investigado sobre varios algoritmos de clasificación, algunos de invención propia y otros ampliamente utilizados, como son los *algoritmos de regresión*, las *máquinas de vectores de soporte*, los *k-vecinos más cercanos* y los *random forests*.

2- Selección del método de tratamiento de datos. El objetivo del proyecto es la utilización de algún método que ayude a mejorar un algoritmo de clasificación. La técnica de *bootstrapping* aplicada al algoritmo correcto ilustra muy bien cómo se pueden mejorar los resultados con un tratamiento de los datos.

3- Selección de los datos. Se intentará utilizar *sets* de datos vistosos desde el punto de vista del problema de clasificación. Ya que este estudio no se centra en el problema de clasificación sino en cómo mejorarlo, he creído conveniente elegir datos con un alto índice de acierto por parte del clasificador.

4- Elección de las herramientas de programación. Se buscará tanto el lenguaje de programación más adecuado como el entorno de programación, con el fin encontrar el conjunto de herramientas que permita llevar a cabo el estudio de la forma más cómoda y eficiente posible.

5- Programación de los algoritmos.

6- Investigar el impacto que tiene la aplicación de la técnica de tratamiento de datos en los resultados, aplicando clasificador. Una vez elegidos el clasificador y el método de tratamiento de datos, analizan los resultados obtenidos, comparándolos con los que se

alcanzan sin la técnica de tratamiento de datos.

1.2 Organización de la memoria

En el Capítulo 2 se hace una introducción de los conceptos aprendidos en el punto 1 de los objetivos del proyecto. Se explica lo que es un algoritmo de clasificación y cómo funciona, y se presentan dos ejemplos sencillos en los que se ilustra la utilidad de un clasificador. A continuación se explica el proceso de desarrollo de un clasificador cualquiera.

Se profundiza en el clasificador que se va a utilizar en el estudio: el *árbol de decisión*, y se continúa explicando la técnica de mejora de resultados escogida para el estudio, el *bootstrapping*, así como su aplicación a un problema de clasificación. Por último se describe el segundo método que es objeto de estudio: la estimación del ratio de acierto mediante *bootstrapping*.

En el Capítulo 3 se presentan los tres problemas de clasificación escogidos y se describen sus principales características.

El Capítulo 4 recoge toda la información sobre el trabajo realizado en el estudio. Se describen las herramientas de trabajo utilizadas y se justifica su elección. Se explica cada uno de los algoritmos y se describen los elementos más destacables del código. Se enumeran y explican las funciones más importantes que se han utilizado, tanto las incluidas en el paquete de trabajo escogido como las desarrolladas especialmente para el estudio. También se detallan los bloques más importantes, que están programados en *scripts*.

En el capítulo 5 se muestran los resultados del trabajo para cada uno de los métodos de clasificación descritos a lo largo de este documento. Aquí también se incluye el análisis de los resultados para cada uno de los casos, así como explicaciones y aclaraciones de los distintos métodos al ser aplicados a cada caso concreto.

El Capítulo 6 incluye un análisis de conjunto de los resultados mostrados en el Capítulo 5, y una comparativa de los distintos métodos con el objetivo de cuantificar la mejora conseguida con las técnicas estudiadas.

Por último, en el Capítulo 7 se muestra el código desarrollado.

2 CONCEPTOS TEÓRICOS

Si bien el objeto de este proyecto no es el de profundizar en las distintas técnicas de clasificación, sí que he creído conveniente una breve explicación de los conceptos básicos. En este primer capítulo se define el problema de clasificación, a continuación se explica el concepto de algoritmo de clasificación y se justifica su necesidad, y finalmente se describe en detalle el funcionamiento de un clasificador concreto: el árbol de decisión, que será el utilizado a lo largo de este estudio. Posteriormente se presenta la técnica de mejora de resultados elegida, el *bootstrapping*, y se detalla su aplicación en el problema de clasificación a través de la técnica de *bagging*.

2.1 El Problema de clasificación

El problema de clasificación es aquel que tiene como objetivo predecir o estimar a qué categoría pertenece una determinada observación. Para ello, previamente se ha llevado a cabo un número de observaciones sobre un conjunto de muestras cuya clase se conoce a priori, con el objetivo de aprender las diferencias entre ellas. Una vez finalizadas estas observaciones, y en base a ellas, se intenta identificar a qué clase pertenece la nueva muestra.

Para llevar a cabo esta tarea, se desarrollan algoritmos de clasificación, también llamados clasificadores. Estos encuentran y aproximan de manera automática la relación entre las características del conjunto de muestras y su salida, o clase.

2.1.1 Ejemplo 2.1: problema de clasificación.

Consideremos ahora el problema de clasificación que se ilustra en la *figura 2.1.1*, donde se tiene un determinado número de observaciones. Cada una de ellas pertenece a la clase 1 o a la clase 2. Posteriormente se recibe una nueva muestra y se desea saber a qué clase pertenece. Como se ha explicado anteriormente, primero se deben observar las muestras conocidas y realizar algún tipo de distinción entre ellas.

En este ejemplo he optado por separar el espacio en dos subregiones, una para cada clase, mediante una ecuación cuadrática en el sentido de los mínimos cuadrados (ver *figura 2.1.2)*. Se observa que la curva calculada no separa perfectamente las dos clases, por lo que resultaría lógico preguntarse si existe alguna otra que separe el espacio mejor. En efecto, se podría, por ejemplo, trazar una línea recta que divida perfectamente a las dos clases. No existe una única forma de clasificar, del mismo modo que no existe un único clasificador para un problema de clasificación. En la mayoría de las veces no es sencillo predecir qué tipo de algoritmo de clasificación va a funcionar mejor para un problema dado.

Llegados a este punto, sólo resta observar en qué región se encuentra la nueva muestra y, así, hacer la clasificación. En el ejemplo, la nueva muestra resulta pertenecer a la clase 2. Esto no es más que una estimación o predicción, pudiendo ser la clase 1 la verdadera clase de la muestra.

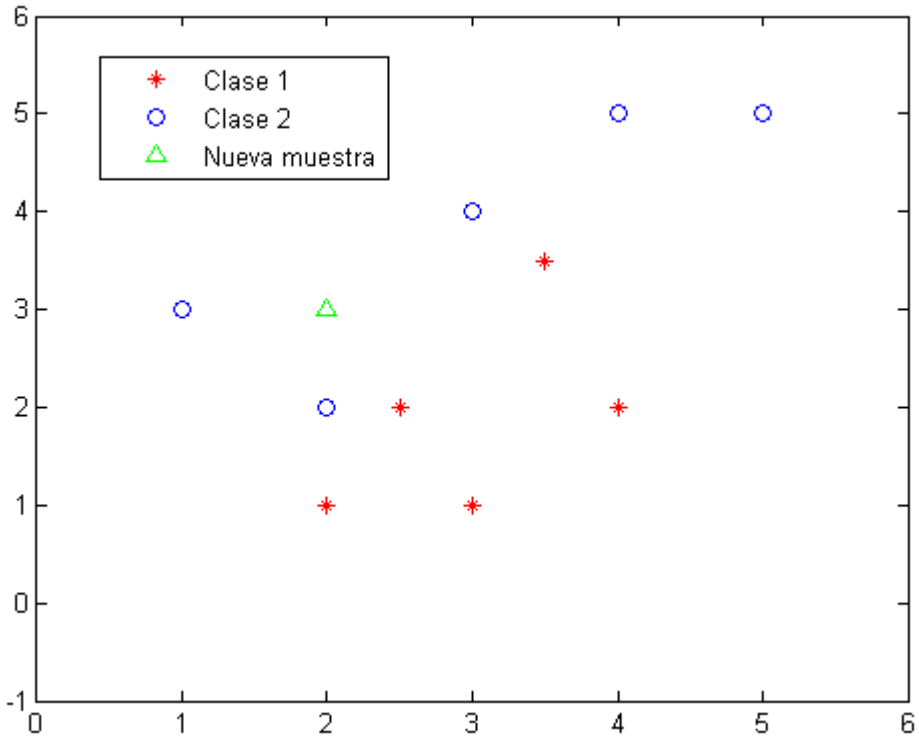

Figura 2.1.1 - Ejemplo 1: problema de clasificación.

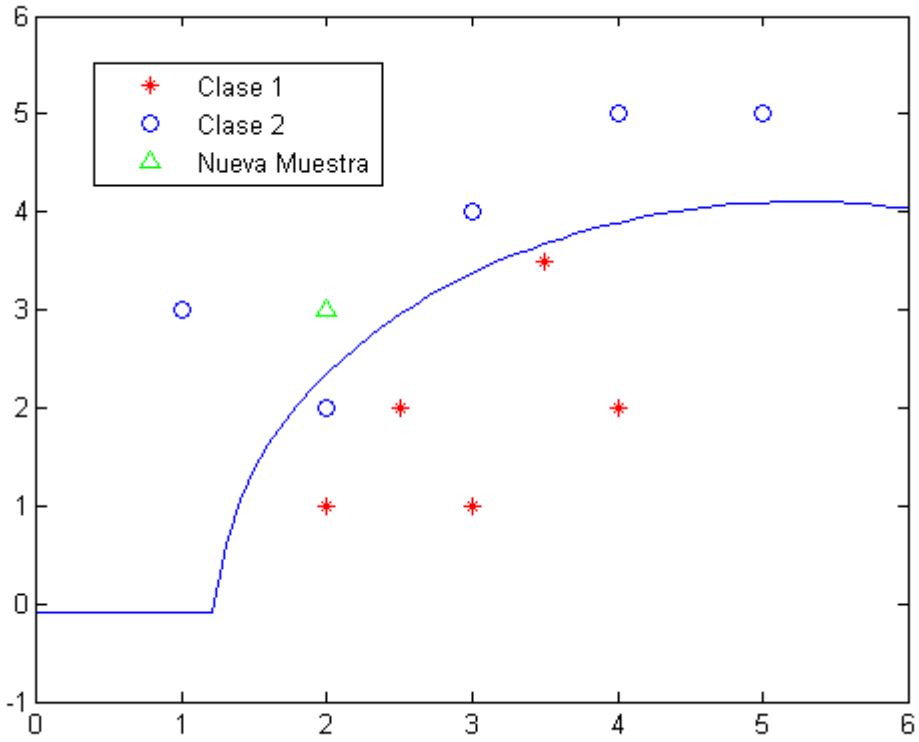

Figura 2.1.2 - Aproximación por una curva cuadrática.

2.1.2 Ejemplo 2.2: problema de clasificación.

Un problema de clasificación puede ser mucho más complicado que el descrito en el ejemplo 1. En la *figura 1.1.3* se muestra un problema en el que las dos clases no están claramente diferenciadas: existe una región del espacio donde se encuentran solapadas. Este hecho va a conllevar un inevitable porcentaje de muestras clasificadas de manera errónea al aplicar el clasificador a un nuevo set de muestras.

En este caso he optado por separar las clases mediante elipses (ver *figura 2.1.4*). Al estar las muestras pertenecientes a la clase 2 más concentradas que las de la clase 1, se clasificará cualquier futura muestra como clase 2 si cae dentro de la elipse que encierra todas las muestras de la clase 2, y clase 1 en el caso contrario.

Los problemas descritos en los ejemplos sólo dependen de dos variables con objeto de ilustrar los conceptos de una manera gráfica. En la realidad raramente ocurre esto, teniendo en muchos casos centenares o miles de variables de las que depende el resultado del problema. Para estos casos a menudo se hacen necesarias técnicas más avanzadas de clasificación que quedan fuera del alcance de este proyecto.

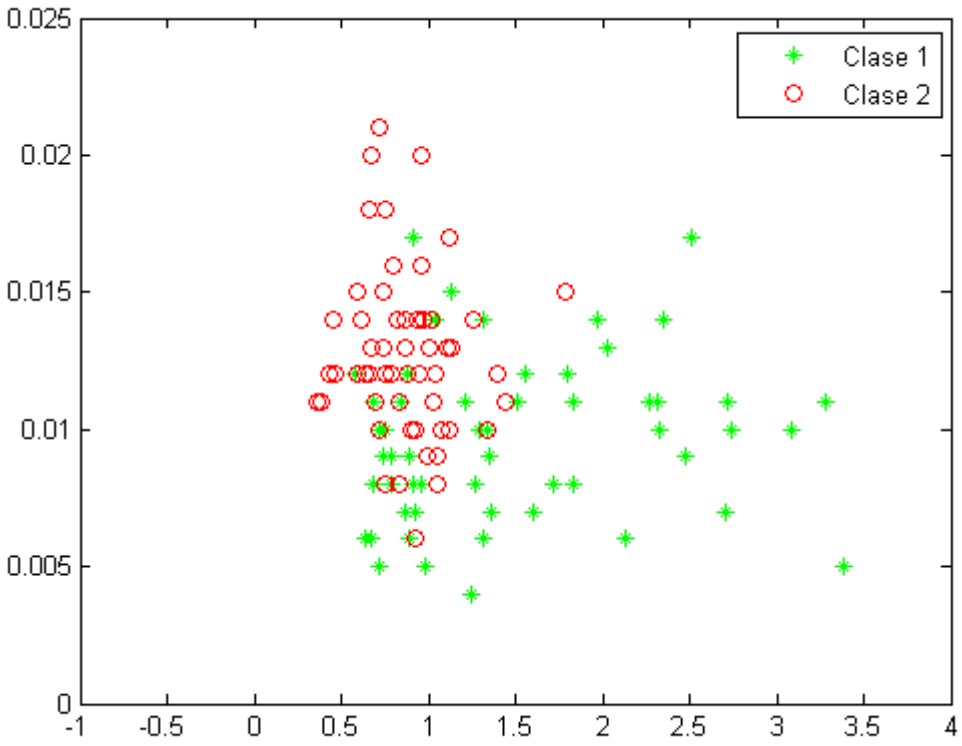

Figura 2.1.3 - Ejemplo 2: problema de clasificación.

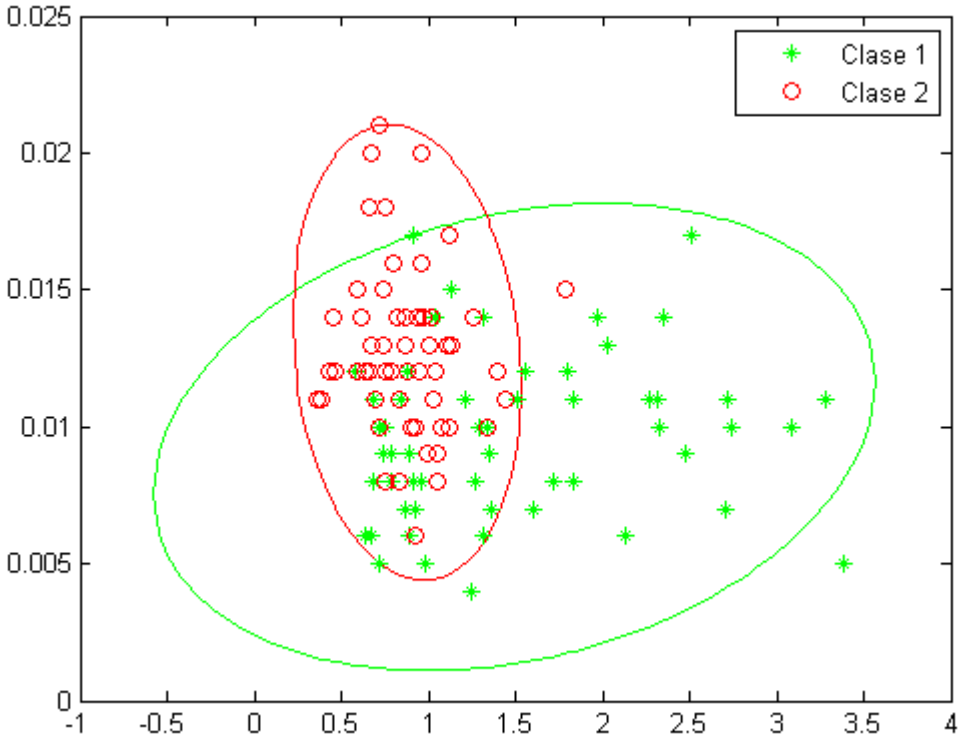

Figura 2.1.4 - Clasificación usando elipses.

2.2 Algoritmos de clasificación

A la vista de los ejemplos anteriores, resulta útil el desarrollo de algún tipo de algoritmo que, de forma automática, sea capaz de aprender a partir de un determinado número de observaciones, para luego poder predecir o estimar la clase de una nueva muestra. Esto se hace absolutamente necesario en el caso de tener problemas complejos con un elevado número de muestras y de variables de las que depende el resultado.

A continuación se describe el proceso de desarrollo de un clasificador cualquiera.

2.2.1 Descripción

Se parte de dos conjuntos de muestras cuyas clases son conocidas. Uno de esos conjuntos – llamado *set de entrenamiento* - se utilizará para entrenar al clasificador, y el otro – *set de validación* - se utilizará para hacer predicciones y comprobar el grado de acierto del clasificador.

Formalmente, se tiene un set de entrenamiento,

$$\{x_i, y_i\}, \quad i = 1, \dots, N,$$

Donde $x_i \in \boldsymbol{R}^{n_x}$ denota el vector de variables de la muestra i, $y_i \in \{clase1, clase2, \dots\}$ es la salida discreta, o clase de cada muestra i.

Se tiene un clasificador cualquiera, C.

Y se tiene un set de validación,

$$\{x_{vi}, y_{vi}\}, \quad i = 1, \dots, N_v \ ,$$

Donde $x_{vi} \in \boldsymbol{R}^{n_x}$ denota el vector de variables de la muestra i, $y_{vi} \in \{clase1, clase2, \dots\}$ es la salida discreta, o clase de cada muestra i.

A las variables también se les llama *marcadores*, *atributos* o *características*.

2.2.2 Fases de desarrollo de un clasificador

El desarrollo de un clasificador se lleva a cabo en tres fases:

1- Fase de entrenamiento.

2- Fase de validación.

3- Fase de ajuste.

En la **fase de entrenamiento** se llevan a cabo todas las operaciones necesarias para ajustar el set de entrenamiento, $\{x_i, y_i\}$, a un modelo. Este modelo trata de representar la relación entre x_i e y_i, separando el espacio de $\boldsymbol{R}^{n_x}$ dimensiones en subregiones, cada una de ellas asociada a una de las clases posibles, de manera que dada una nueva muestra, la clase estimada de ésta será aquella correspondiente a la región donde se encuentre.

Posteriormente, en la **fase de validación**, se utiliza el set de validación, $\{x_{vi}, y_{vi}\}$, para comprobar cómo de bien clasifica el algoritmo. Esto se hace haciendo una predicción para cada muestra $x_{vi} \in \boldsymbol{R}^{n_x}$, y comparando el resultado con su salida real $y_{vi} \in \{clase1, clase2, \dots\}$.

Si tras la validación se observa que los resultados no son adecuados se llega a la **fase de ajuste**, donde se procede a modificar el clasificador ya sea cambiando la técnica de clasificación, modificando el set de entrenamiento, o ajustando los parámetros de los que pueda depender el clasificador.

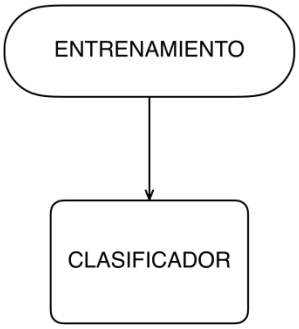

Figura 2.2.1 – Fase de entrenamiento.

Figura 2.2.2 – Fase de validación.

Como se observa, el clasificador sólo aprende a partir de un subconjunto de muestras, y luego se comprueba su efectividad aplicando lo aprendido a un segundo subconjunto. Es importante que el aprendizaje y la validación se hagan con conjuntos distintos para poder garantizar la imparcialidad del clasificador. Si no fuera así, se daría el caso de clasificar muestras cuya clase ya

se conoce y se ha utilizado para entrenar al algoritmo, obteniéndose unos resultados mejores que los que correspondería y falseando los resultados.

Visto de otro modo: un clasificador lleva a cabo una transformación $\boldsymbol{R}^n \rightarrow \boldsymbol{N}$, siendo *n* el número de parámetros de los que dependen las muestras, y $\boldsymbol{N} = \{c_1, c_2, \ldots, c_c\}$ las posibles clases, de manera que cada nueva muestra tiene una transformación sobre $\boldsymbol{N}$, siendo su valor el resultado del proceso de clasificación.

2.3 Árboles de decisión

En este apartado se describe el funcionamiento de un *Árbol de decisión*, que es el clasificador que he elegido en este estudio. Este clasificador tiene una serie de ventajas que se discutirán más adelante.

2.3.1 Funcionamiento

Para describir el funcionamiento de un *Árbol de decisión*, imaginemos el siguiente ejemplo. Se desea predecir qué modelo de coche comprará un cliente en base a su perfil. Para ello se cuenta con un registro de clientes anteriores que compraron uno de los dos modelos que se muestran en la *tabla 2.3.1*. La técnica a utilizar es muy sencilla: ir haciendo preguntas sobre los atributos de la nueva muestra (cliente), y así ir acotando la solución del problema hasta llegar a una solución final.

Observación	Hijos	Salario anual	Sexo	Edad	Modelo
1	2	20.500	H	32	Deluxe
2	0	18.000	H	23	Berlina
3	4	27.000	M	35	Berlina
4	1	19.500	M	28	Berlina
5	1	40.000	H	58	Deluxe
6	0	12.000	H	37	Berlina
7	2	25.000	H	25	Berlina
8	1	14.600	M	25	Berlina
9	1	21.000	H	29	Deluxe
10	3	17.000	H	40	Berlina

Tabla 2.3.1 - Clasificación de clientes.

La primera pregunta que se puede hacer es si el cliente es hombre o mujer, pues se observa que sólo los hombres compran el modelo *deluxe*. Así, si la respuesta es *mujer*, se puede dar por finalizado el problema prediciendo que su compra será la del modelo *berlina*. En caso de ser hombre, se procede a la siguiente pregunta: ¿es su sueldo anual mayor o menor que 20.000? pues se observa que todos los clientes que eligieron la clase *deluxe* tenían un sueldo superior a 20.000. Por último se podría hacer la siguiente pregunta: ¿es su edad superior a 27? Si es así, la clase asociada a ese cliente será *deluxe*, en caso contrario, *berlina*.

Las preguntas que se han elegido y el orden en el que se han formulado no son únicos, siendo

realmente difícil encontrar la combinación óptima. En este ejemplo parece bastante claro qué preguntas hacer y en qué orden para alcanzar una solución con el menor número de preguntas posible, pero en la mayoría de los casos no resulta tan evidente, dejándose que sea el propio algoritmo el que decida qué preguntas hacer y cómo organizarlas en base a unas reglas que se explican más a delante.

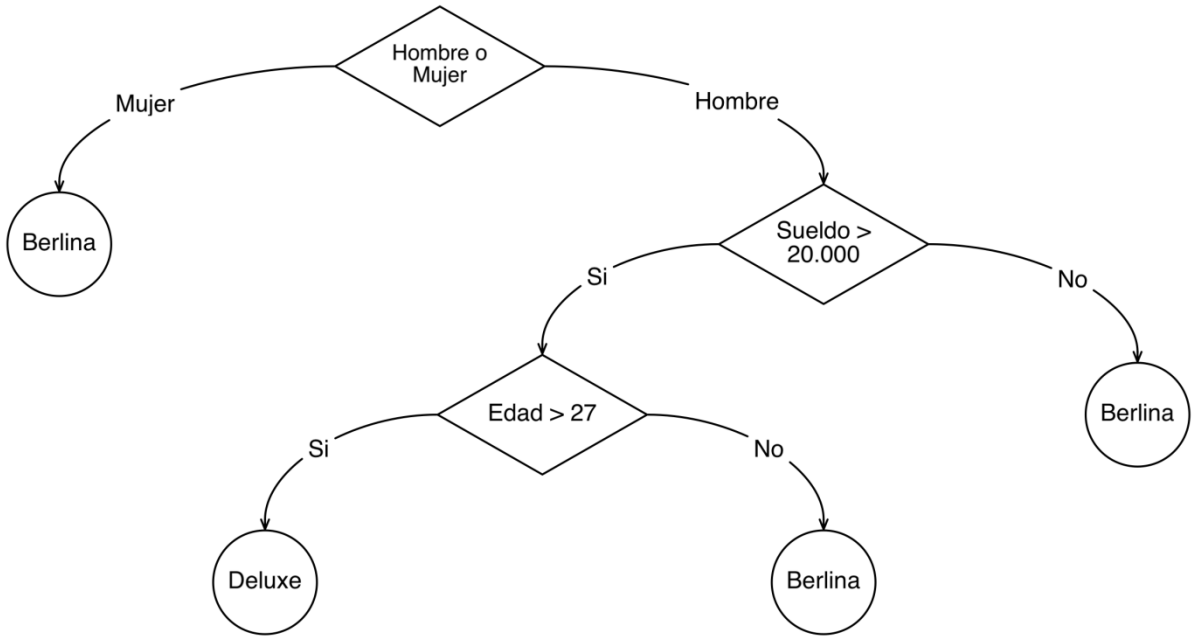

Figura 2.3.1 - Árbol de decisión

Un árbol de decisión está compuesto por nodos, representando cada uno de ellos una condición. Cada nodo contiene todas las muestras provenientes de la condición anterior.

Los nodos están organizados en una jerarquía de arriba abajo, y se pueden clasificar en:

1- Nodo raíz: es el comienzo del árbol. No tiene ningún arco de entrada.

2- Nodo interno: se encuentran dentro del árbol. Tienen un arco de entrada y dos o más de salida.

3- Nodo hoja: se encuentran al final del camino. Tienen un arco de entrada y ninguno de salida. Tienen asociada una etiqueta con el resultado obtenido en caso de ser alcanzados.

Los nodos internos y raíz dividen grupos de muestras con características diferentes para ir obteniendo subgrupos más *puros*. Este concepto de pureza se explica más adelante. El objetivo de un árbol de decisión es conseguir subdividir las muestras en grupos que sólo contengan una de las clases. Esto, a pesar de ser posible en muchos casos, no se llega a alcanzar, imponiéndose condiciones de parada adicionales que evitan el fenómeno de sobre-ajuste u *overfitting*.

2.3.2 Algoritmo de Hunt

Como se ha explicado anteriormente, existen numerosas formas de generar un árbol de decisión. En el ejemplo anterior quedaba de manifiesto que hay formas de seleccionar y organizar las preguntas mejores que otras. Lamentablemente, a medida que crece el problema lo hace también el árbol, haciéndose imposible desde un punto de vista computacional dar con la organización óptima.

Pues bien, el algoritmo de Hunt ofrece un método recursivo para desarrollar un árbol de decisión alcanzando una solución óptima local[1], manejando el concepto de *pureza* mencionado anteriormente.

El algoritmo tiene la siguiente forma:

Para cada M_i, grupo de muestras en el nodo *i*, e $y = \{clase1, clase2, \dots\}$, distintas clases posibles:

1- Si todas las muestras M_i pertenecen a la misma clase; ese nodo se convierte en un nodo *final*, o nodo *hoja*.

2- Si las muestras M_i pertenecen a más de una clase; seleccionar otro atributo para subdividir M_i en grupos más pequeños y más puros, dando lugar a tantos nodos *hijo* como clases distintas había en M_i.

3- Aplicar el algoritmo a cada nodo.

Hay, además, dos condiciones adicionales a tener en cuenta,

1- Al subdividir un grupo en varios, es posible que alguno de los grupos resultantes esté vacío. Esto es, es posible que se genere un nodo *hijo* que no contenga muestras. Esto puede ocurrir cuando el número de hijos es mayor de dos, y ninguna de las muestras tiene atributos en el rango de valores requerido para caer dentro del nodo en cuestión. En este caso ese nodo *hijo* se convierte en nodo *hoja*, asignándosele la etiqueta de la clase que es mayoría en el nodo *padre*.

2- En ocasiones es posible que todas las muestras en un nodo tengan los mismos atributos pero pertenezcan a distintas clases. En este caso no es posible seguir particionando, por lo que se convierte el nodo en cuestión en nodo *hoja* y se etiqueta con la clase que es la mayoría en dicho nodo.

La dificultad del algoritmo reside en la elección del atributo que dividirá el nodo que se está analizando. Al tener una estructura jerárquica, una mala elección va a ser arrastrada hasta el final del problema, por lo que se hace más que necesario encontrar una regla apropiada de decisión.

El objetivo, como se ha mencionado anteriormente, es subdividir cada grupo en grupos más puros. Se entiende que un grupo es totalmente puro cuando todas las muestras dentro de él pertenecen a la misma clase. De este modo, el método recursivo decidirá qué atributo utilizar en un nodo para particionar un grupo en base a cómo de puros serán los grupos resultantes.

Volvamos al ejemplo, encontrándonos en la situación inicial, antes de que el algoritmo genere ningún nodo. En este momento se tiene únicamente el nodo raíz, que contiene a la totalidad de las muestras por ser el primero de los nodos. Se observa que las muestras pertenecen a más de una clase, por lo que será necesario formular una condición en uno de los atributos para subdividir el grupo de muestras. De acuerdo al algoritmo de Hunt, debemos elegir el atributo que particione el grupo de muestras en subgrupos lo más puros posibles. Efectivamente, el atributo *sexo* da lugar a subgrupos más puros. Por un lado, todas las muestras pertenecientes al subgrupo *mujeres* pertenece a la clase *berlina*, por lo que se llega directamente a un nodo *hoja* que de como solución la clase *berlina*. Por otro lado, el subgrupo hombres cuenta con tres muestras pertenecientes a la clase *deluxe* y cuatro a la clase *berlina*, por lo que necesitará ser particionado de nuevo para obtener grupos más puros.

[1] Como se ha comentado, no es posible garantizar un óptimo global.

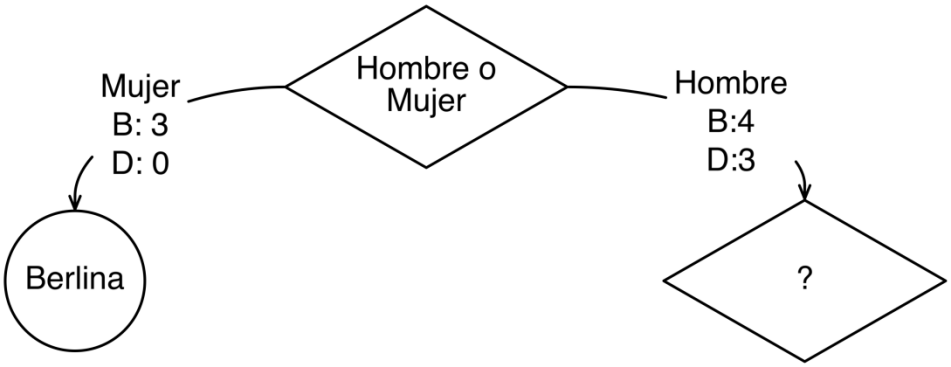

Figura 2.3.2 - Particionado de un nodo.

2.3.3 Trabajando con valores continuos

El ejemplo discutido contiene datos categóricos (sexo), discretos (hijos, edad) y continuos (sueldo). Éstos merecen una mención especial pues, como se verá más adelante, son el tipo de datos que se manejan en el estudio.

Hay dos formas de expresar una condición cuando se trata de números continuos.

La primera es la utilizada en el ejemplo: seleccionar un atributo y dividir el espacio en dos regiones:

$$M_i | A_{ik} < \alpha$$
$$M_j | A_{jk} \geq \alpha$$

Donde M_i , M_j son los subgrupos de muestras resultantes tras aplicar la condición; A_{ik} , A_{jk} son el k-ésimo atributo de de M_i , M_j ; y α es el valor que separa ambos subgrupos.

La segunda opción es subdividir el grupo de muestras en más de dos grupos de la forma:

$$\alpha_i \leq A \leq \alpha_{i+1}, \quad i = 1, \dots, n$$

En este caso se tiene una cantidad *n* de rangos (α_i, α_{i+1}) que subdividirán el grupo de muestras en *n* grupos.

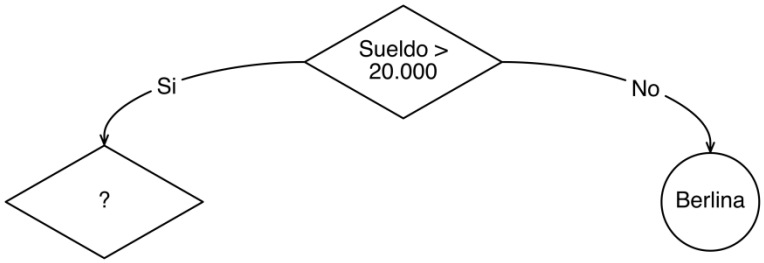

Figura 2.3.3 - Separación binaria según un valor de un atributo continuo.

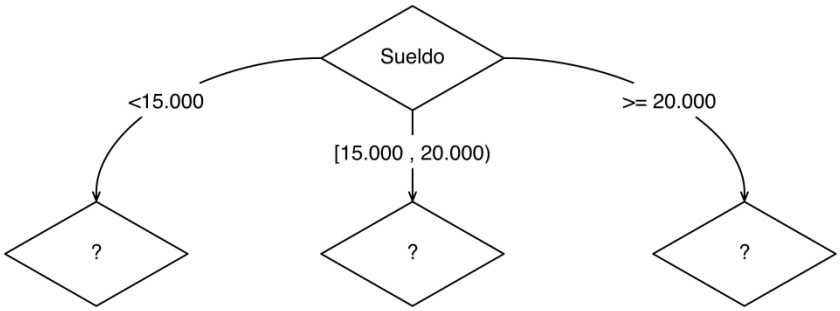

Figura 2.3.4 - Separación en varias clases de acuerdo a un número finito de rangos.

2.3.4 Elección de la partición

La elección de la partición se hace en base al grado de impureza del grupo de muestras antes de dicha partición, y el grado de impureza de los subgrupos generados tras la partición. Para el caso de dos clases el mayor grado de impureza se alcanza cuando se tiene la misma proporción de ambas clases, y se obtiene el menor grado de impureza cuando sólo se tiene una clase.

Sea $p(i|t)$ la fracción de muestras que pertenece a la clase *i* en el nodo t. Se pueden definir las siguientes medidas de impureza:

$$Entropía(t) = -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t) \qquad (1)$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1}[p(i|t)]^2 \qquad (2)$$

$$Error\ de\ clasificación = 1 - \max_i[p(i|t)] \qquad (3)$$

Donde:

(1) es proporcional al grado de desorden del grupo de muestras, entendiendo por perfectamente ordenado un grupo que sólo contiene muestras de una clase. El sumatorio dentro de (2) es el coeficiente de Gini, normalmente utilizado para medir el grado de igualdad entre las fuentes de ingresos de un país y de otro. Este coeficiente alcanza su mínimo en 0 cuando existe una máxima igualdad, esto es, cuando hay el mismo número de muestras de cada una de las clases, y alcanza su máximo en 1 cuando todas las muestras del grupo pertenecen a la misma clase. (3) utiliza el mismo concepto que (2) haciendo uso de una relación lineal.

Para el caso de dos clases, todas estas medidas alcanzan su valor máximo cuando hay la misma proporción de ambas clases, y alcanzan un valor de cero cuando sólo se tiene una clase, tal como se muestra en la *figura 2.3.5*:

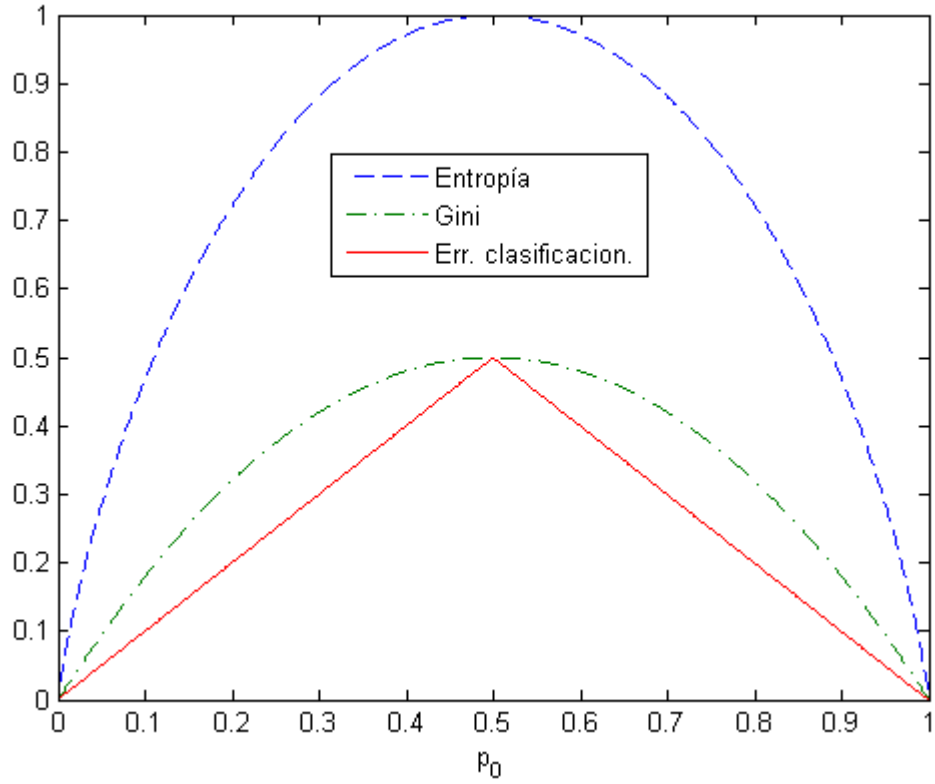

Figura 2.3.5 - Medidas de la impureza.

La bondad de una partición dependerá de la relación entre el grado de impureza del nodo *padre* y el grado de impureza de los nodos *hijos*. Una buena forma de medir esta bondad es mediante el criterio de la ganancia.

Sea, $I(\cdot)$ el grado de impureza de un determinado nodo, *N* el número total de muestras en el nodo padre, *k* el número de atributos, $N(v_j)$ el número de muestras que han caído dentro del nodo v_j, se define la ganancia como:

$$\Delta = I(nodo\ padre) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

El grado de bondad de una partición será tanto mejor como mayor sea el valor de la ganancia asociada a ella. Este concepto se suele usar como criterio de elección de partición. Obsérvese que para cualquier posible partición, tanto la impureza del nodo padre como el número de muestras dentro de él no varían, por lo tanto únicamente hay que buscar una partición tal que se minimice:

$$\sum_{j=1}^{k} N(v_j) \cdot I(v_j) \ , \qquad (4)$$

que es la suma de las impurezas de los nodos hijos ponderadas por su tamaño.

2.3.5 Particionado según valores continuos

Todos los problemas tratados en este estudio tienen atributos continuos, por lo que es conveniente hacer un inciso sobre cómo dividir un nodo en base a este tipo de atributos.

Supóngase el ejemplo anterior, mostrado en la *tabla 2.3.1*.

Siguiendo el método de particionado explicado, se desea subdividir un nodo *padre* en otros dos nodos *hijos* en base al atributo *sueldo*. Para ello, se debe encontrar el valor *v* que mejor divida al nodo *padre* en el sentido del criterio de la ganancia.

Una primera aproximación sería hacer un barrido de todos los posibles valores de *v*, calculando para cada uno de ellos la ganancia, o lo que es lo mismo, la suma ponderada de las impurezas de los nodos *hijos* – ecuación (4). Este procedimiento tiene un coste computacional tan elevado que obliga a buscar un método más fino y óptimo que requiera un menor número de operaciones.

Un método más eficiente es reordenar las muestras del nodo *padre* en función del atributo continuo que se desea evaluar, en este caso el sueldo. Esto se muestra en la *tabla 1.3.2*.

Observación	6	7	8	10	2	4	1	9	3	5
Modelo	Berlina	Berlina	Berlina	Berlina	Berlina	Berlina	Deluxe	Deluxe	Berlina	Deluxe
Sueldo	12.000	13.000	14.600	17.000	18.000	19.500	20.500	21.000	27.000	40.000

Tabla 2.3.2 - Particionado según atributos continuos.

Se tomarán los valores *v* iguales a las medias aritméticas de cada par de muestras consecutivos, más dos valores correspondientes a los extremos. Esto es, *v* = 11.000, 12.500, 13.800,…

Una observación más cuidadosa de los datos muestra que las seis primeras muestras pertenecen a la misma clase, por lo que cualquier partición entre una pareja de ellas daría lugar a un nodo *hijo* puro, y a otro nodo *hijo* no puro. En base a esta observación, se pueden omitir todas las particiones intermedias y pasar directamente a la que garantiza un nodo *hijo* puro mayor, esto es, la división entre la muestra 4 y la 1. Este refinamiento del algoritmo permite reducir aún más el número de operaciones y el coste computacional.

2.3.6 Criterios de parada

Idealmente, el algoritmo termina cuando se consigue dividir todas las muestras en subgrupos totalmente puros. Sin embargo, en la práctica puede convenir fijar un criterio de parada más restrictivo. Los dos principales motivos son el sobreajuste del clasificador (*overfitting*) y el coste computacional.

Los criterios de parada más comunes para dejar de extender una rama dentro del árbol son los siguientes:

1- Todas las muestras del nodo pertenecen a la misma clase. Este es el caso más simple, explicado anteriormente.

2- Se ha alcanzado la profundidad máxima del árbol. Con objeto de limitar el coste computacional, se puede fijar un límite de profundidad para el árbol. Esto es, el número máximo de niveles que puede tener.

3- El número de clases a las que pertenecen las muestras del nodo es menor que un número mínimo de clases prefijado. En el caso de tener un elevado número de clases puede darse el caso de que no se necesite ser tan restrictivo a la hora de cerrar un nodo.

4- Si el mejor criterio para subdividir el nodo es menor que un umbral prefijado, se fija un umbral de pureza a partir del cual se considera que la partición es totalmente pura.

2.4 Bootstrapping

La técnica de *bootstrapping* permite inferir la distribución de una variable estadística a partir de un conjunto de muestras de una población cuya distribución no se conoce. En estadística, el término *bootstrapping* se refiere a cualquier método que hace uso de muestreo con reposición. Un proceso de muestro con reposición es aquel en el que cada muestra tomada de la población de manera aleatoria es devuelta a dicha población, por lo que tendrá las mismas posibilidades de volver a ser elegida que el resto de muestras.

La principal ventaja que presenta esta técnica con respeto a tomar la variable estadística deseada directamente a partir de las muestras es que disminuye en gran medida la desviación típica, permitiendo unos intervalos de confianza mejores.

2.4.1 Método de bootstrapping

Se tiene un conjunto de *N* muestras,

$$M = \{m_1, m_2, \dots, m_N\} ,$$

tomadas de una población cuya distribución se desconoce, y se desea calcular el valor de una variable estadística asociada a dicha población. La forma de proceder es la siguiente:

Se toman *N* muestras de *M* con reposición:

$$X^{(1)} = \left\{x_1^{(1)}, x_2^{(1)}, \dots, x_2^{(1)}\right\} ,$$

y se calcula la variable estadística deseada de ese conjunto de muestras, $S_1(X^{(1)})$, que es una estimación de la variable estadística de la población muestreada.

Se repite este proceso *B* veces:

$$X^{(2)} = \left\{x_1^{(2)}, x_2^{(2)}, \dots, x_2^{(2)}\right\} \rightarrow S_2(X^{(2)})$$
$$X^{(3)} = \left\{x_1^{(3)}, x_2^{(3)}, \dots, x_2^{(3)}\right\} \rightarrow S_3(X^{(3)})$$
$$\vdots$$
$$X^{(B)} = \left\{x_1^{(B)}, x_2^{(B)}, \dots, x_2^{(B)}\right\} \rightarrow S_B(X^{(B)})$$

Se calcula ahora la distribución de la variable aleatoria estimada $\hat{S} = \{S_1, S_2, \dots, S_B\}$ siendo su media la estimación final de la variable aleatoria.

Dicho en otras palabras: se utilizan las muestras tomadas de una población para construir una distribución de esa población, y posteriormente se utiliza esta distribución para calcular la distribución de la variable estadística que se quiere calcular.

Al muestrear el conjunto de datos *M*, esto es lo mismo que muestrear directamente la población, al menos en la medida en que *M* aproxima a dicha población. Así, al muestrear B veces es en cierto modo como si se muestreara la población ese número de veces.

Hacen falta dos condiciones para que todo lo anterior sea cierto:

1- Todas las muestras de *bootstrapping*, $X^{(1)}, X^{(2)}, \dots, X^{(B)}$, deben tomarse del mismo conjunto de muestras, *M*.

2- Todas las muestras tomadas de la población, $\{m_1, m_2, \dots, m_N\}$ deben ser independientes

entre sí.

La técnica de *bootstrapping* se suele utilizar para:

1- Dibujar histogramas.
2- Calcular desviaciones típicas y errores estándar.
3- Crear intervalos de confianza

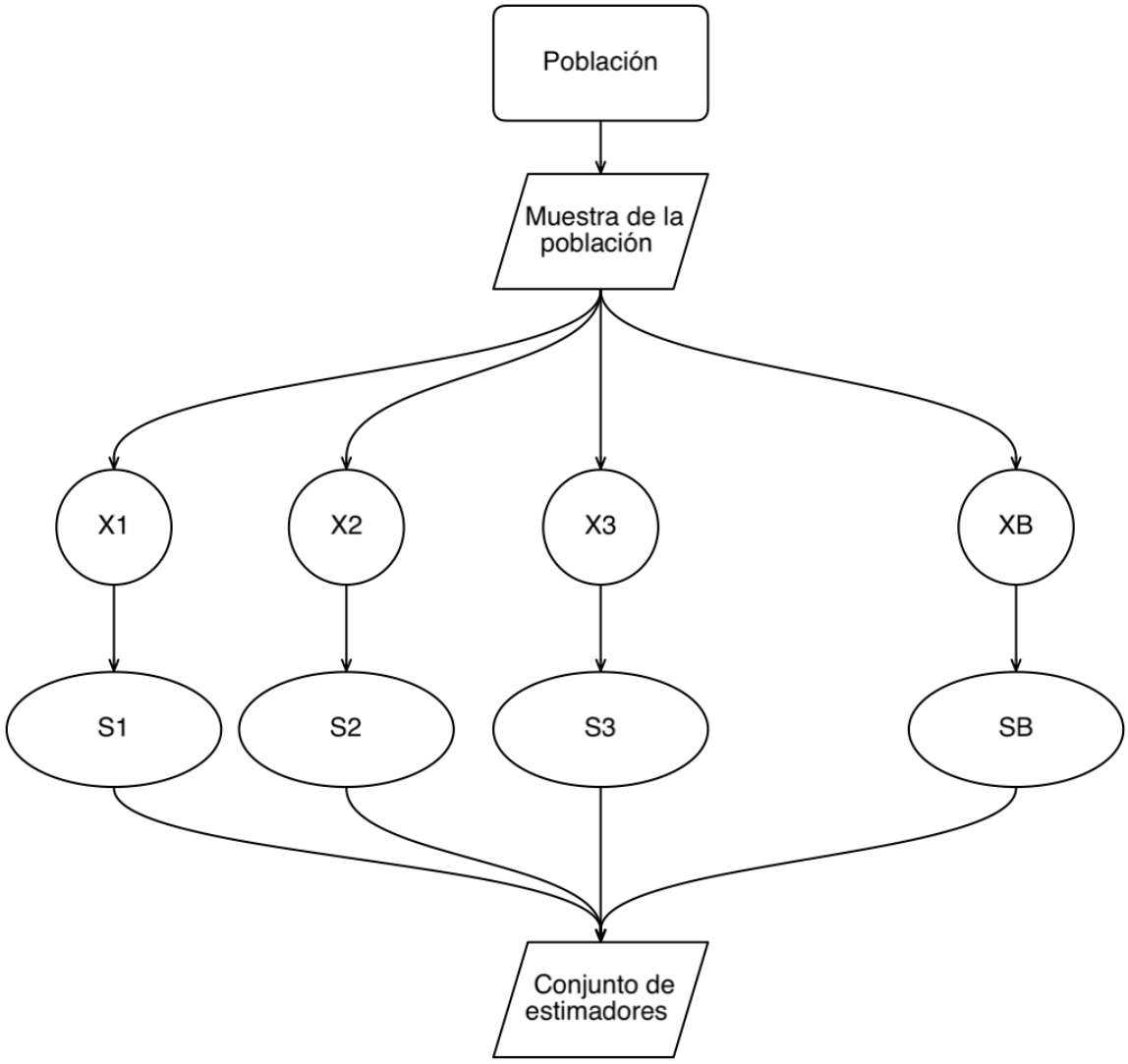

Figura 2.4.1 - Proceso de *bootstrapping*.

2.4.2 Ejemplo 4.1: bootstrapping

Se tiene una población desconocida, a partir de la cual se extrae un conjunto de muestras como el que se muestra en la *figura 2.4.2*. Es importante aclarar de nuevo que esta no es la distribución de la población, sino la de un conjunto de muestras tomadas aleatoriamente de ella.

Si se tratarse de la población no sería necesaria ninguna técnica de estimación, pues se podrían obtener directamente los valores exactos a partir de los datos.

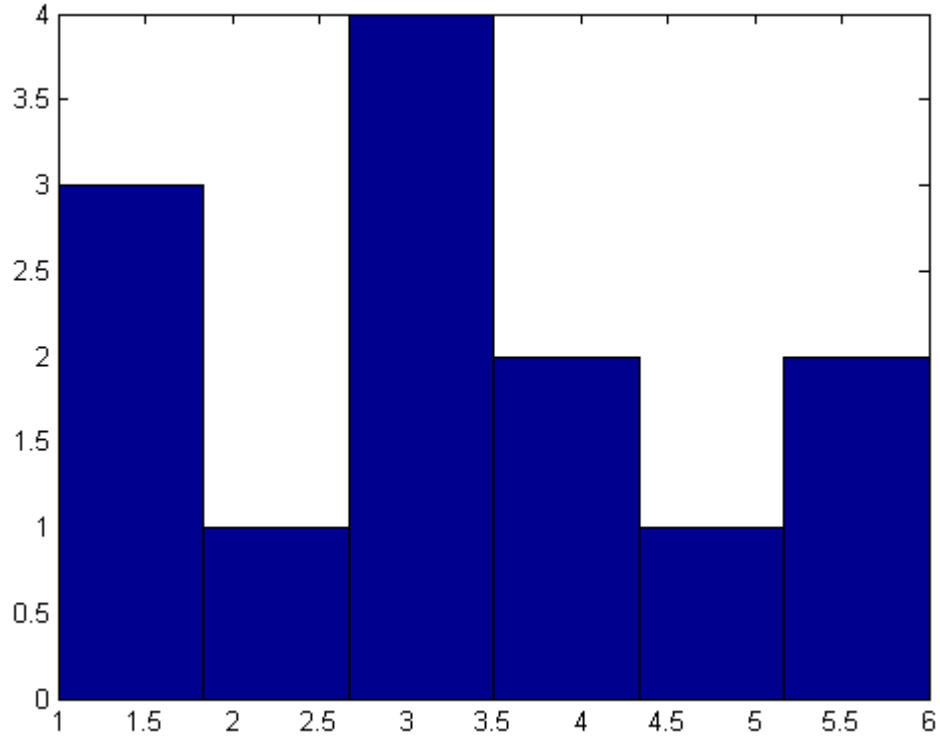

Figura 2.4.2 - Distribución del conjunto de muestras *M*.

Se puede ver que los valores muestreados son:

$$M = \{1, 1, 1, 2, 3, 3, 3, 3, 4, 4, 5, 6, 6, \}$$

Supóngase que se quiere calcular la media de la población. A partir de los datos, esta tendrá un valor de la media $\mu = 3{,}2308$ y la desviación típica $\sigma = 1{,}7394$.

A continuación se muestrea este set de muestras 1.000 veces con reposición, obteniendo por tanto 1.000 sub-*sets* de muestras:

$$X^{(1)} = \{4, 3, 5, 3, 6, 4, 2, 3, 1, 3, 6, 4, 3\} \rightarrow \mu_1 = 3.6154$$
$$X^{(2)} = \{6, 3, 1, 6, 3, 6, 5, 4, 3, 4, 4, 4, 5\} \rightarrow \mu_2 = 4.1538$$
$$\vdots$$
$$X^{(1000)} = \{2, 6, 5, 6, 3, 3, 5, 4, 4, 3, 1, 3, 6\} \rightarrow \mu_{1000} = 3.9231$$

Obteniéndose un conjunto de estimadores,

$$\tilde{\mu} = \{\mu_1, \mu_1, \dots, \mu_{1000}\} = \{3.62, 4.15, \dots, 3.92\}$$

Que, como se puede apreciar en la *figura 2.4.3*, presenta una distribución gaussiana. Esto ocurrirá siempre que el número de muestras de *bootstrapping*, *B*, sea lo suficientemente grande.

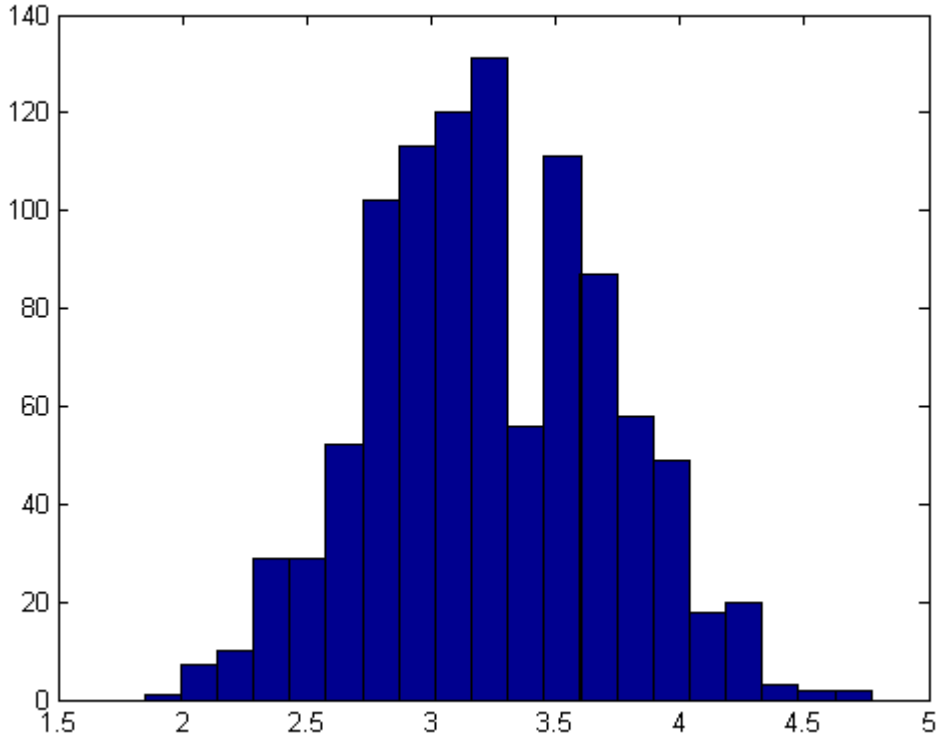

Figura 2.4.3 - Distribución del conjunto de estimadores, $\tilde{\mu}$.

La media del conjunto de estimadores resulta ser $\mu = 3.2412$, aproximadamente igual a la media del conjunto de muestras tomadas de la población, *M*. La desviación estándar de la media es $\sigma = 0.4747$, mucho menor que la desviación estándar de los valores muestreados directamente de la población.

2.5 Bagging

La técnica de *bagging* es una aplicación del concepto de *bootstrapping* en problemas de regresión y de clasificación estadística. El *bagging* mejora la estabilidad y la precisión del clasificador, disminuyendo también la varianza de los resultados y disminuyendo el riesgo de sobre-ajuste. Suele tener buenos resultados en métodos basados en árboles de decisión, motivo por el que se eligió dicho clasificador en primer lugar.

El método de *bagging* muestrea con reemplazamiento *n* veces un conjunto de muestras, con el objetivo de entrenar a *n* clasificadores. Una vez terminado el proceso de muestreo y aprendizaje, se promedian todos los clasificadores - de dudosa precisión - para conseguir un clasificador mejor, media de todos ellos. Al evaluarse una nueva muestra, cada clasificador emitirá un voto, y será la clase con mayor número de votos la que finalmente se elija.

Sea un set de entrenamiento *T* que contiene *n* muestras, se muestrea con reposición *m* veces dicho set, tomando en cada muestreo *n* muestras. Esto da lugar a $B_1, B_2, \ldots, B_m$ *sets* de muestras

por *bootstrapping*. Si *n* es lo suficientemente grande, se puede demostrar que el valor esperado de cantidad de muestras distintas en cada muestreo es de aproximadamente 63,2%, siendo el resto muestras repetidas debido a que el muestreo se hace con reposición.

Debido a esto, es de esperar que el clasificador sea peor que el que se conseguiría si se entrenara directamente con todas las muestras disponibles, ya que se están usando menos datos en la fase de aprendizaje.

Es al unir los *m* clasificadores cuando se consigue un resultado mejor. Al haber muestreado siempre del mismo set de datos los resultados deben ser parecidos, sin embargo, al muestrear un cierto número de veces se reduce la varianza y aumenta la estabilidad. Esto se debe a que se ha conseguido un clasificador más robusto debido a que se ha aprendido en base a unos datos con una varianza menor, como se ha visto anteriormente al explicar el método de *bootstrapping*.

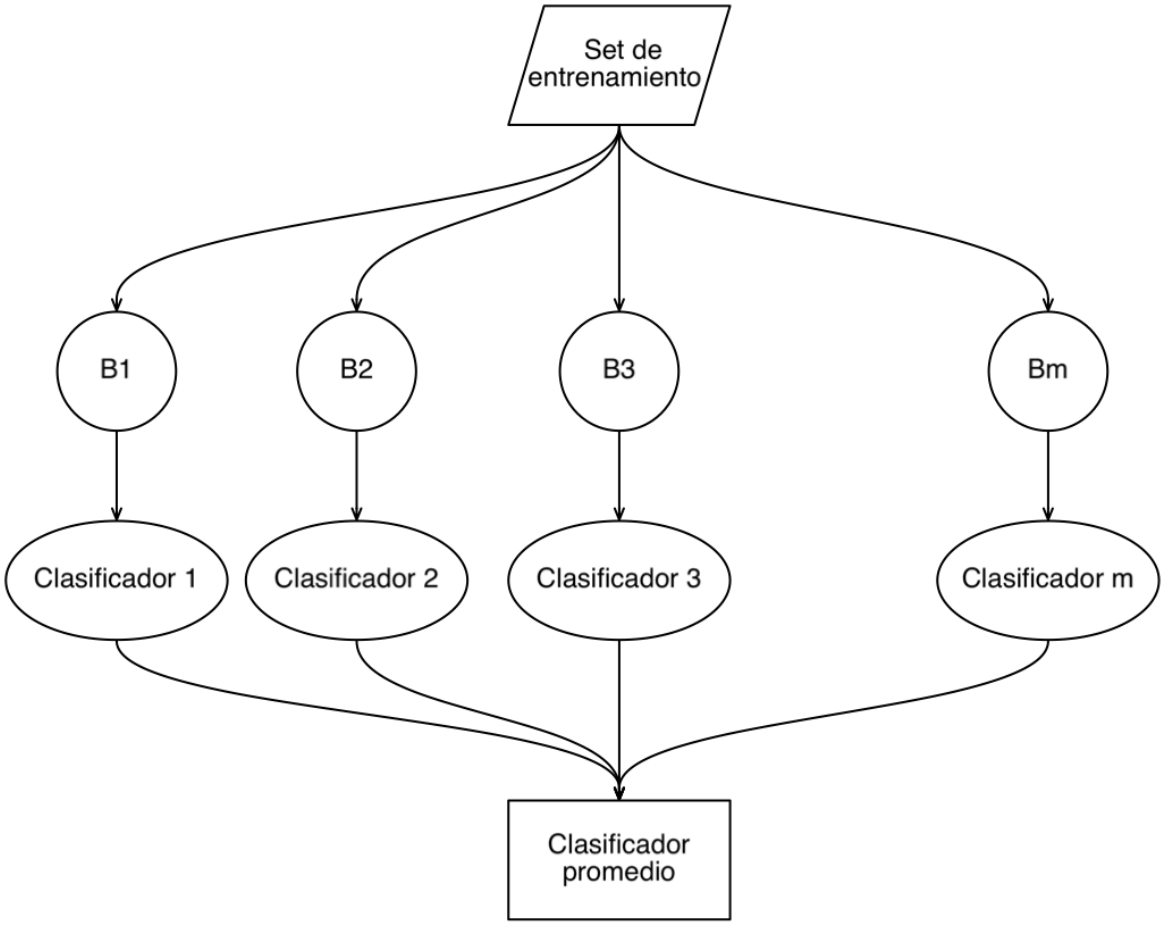

Figura 2.5.1 - Proceso de *bagging*.

2.6 Bagging para la estimación del ratio de acierto

A la vez que se aplica la técnica de *bagging* a un problema de clasificación se puede hacer una estimación el ratio de acierto del clasificador, en nuestro caso el *árbol de decisión*. En cada iteración se muestrea aleatoriamente con reposición, por lo que una determinada cantidad de muestras caerá dentro del set de datos con los que se entrenará al clasificador, quedando las restantes sin utilizar. En este procedimiento se aprovechan estas muestras no utilizadas para

validar el clasificador y, así, obtener una estimación del ratio de acierto que se conseguirá al utilizar el clasificador en el set de validación, tal como se muestra en la *figura 2.6.1*.

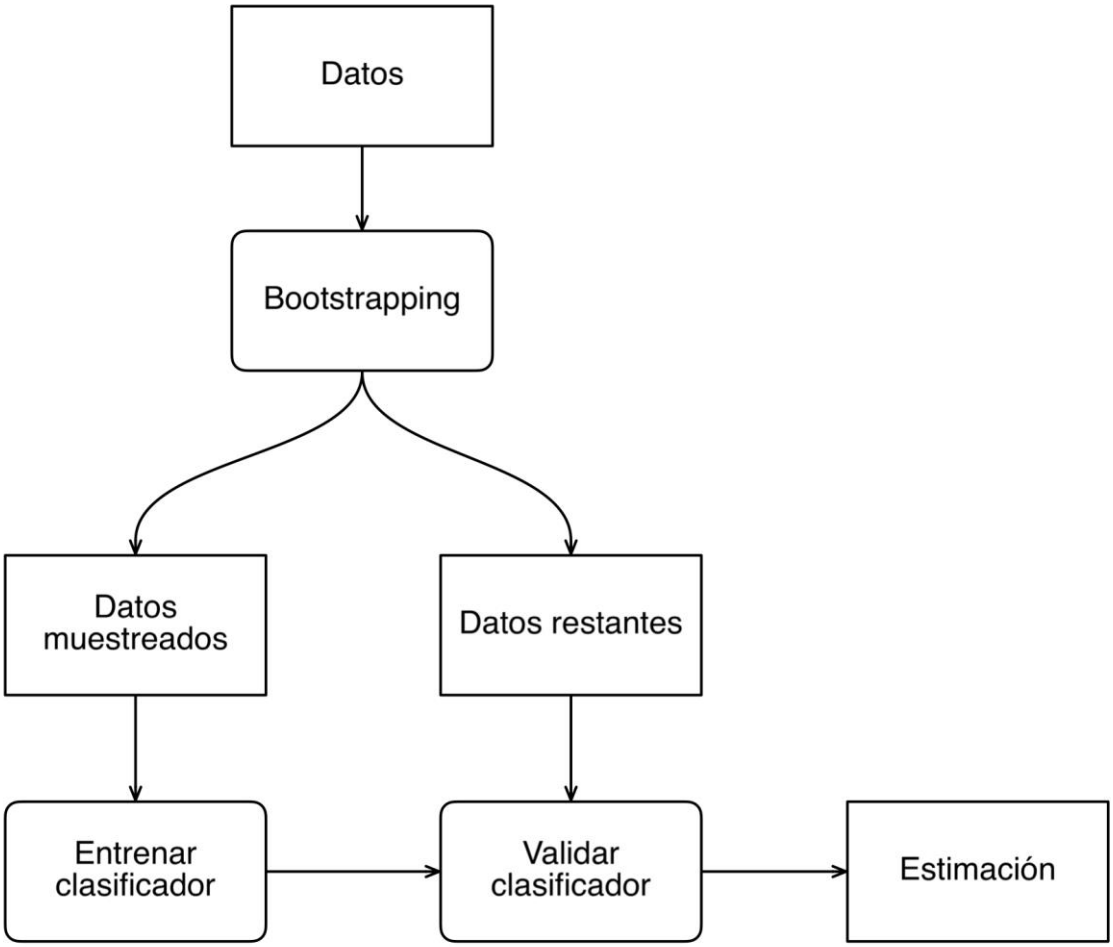

Figura 2.6.1 - Bagging para la estimación del ratio de acierto.

Durante el estudio del clasificador se compara la predicción del ratio de acierto con el resultado final para observar la diferencia entre ambos y buscar intervalos de confianza. Es posible que la predicción sea conservadora en la mayoría de los casos, esto es, consistentemente peor que los resultados finales, consiguiéndose así acotar inferiormente el ratio de acierto. Si no es así, una segunda opción es buscar un intervalo de confianza lo suficientemente grande de la estimación. Todo esto se explica con detenimiento en los capítulos 4 y 5.

3 PRESENTACIÓN DEL PROBLEMA

En este capítulo se presentan los problemas de clasificación elegidos, describiendo cada set de datos, mostrando sus principales características y justificando su elección.

3.1 Selección de los datos

Se han seleccionado tres *sets* de datos. *Flor de iris*, *Cáncer de mama*, y *Cáncer de colón*. Como se ha mencionado en el capítulo de introducción, el problema de clasificación en si no es el objeto de estudio. Por esto se ha intentado elegir problemas que permiten una buena clasificación con objeto de presentar unos resultados más vistosos, no teniendo este factor influencia a priori en el mejor o peor impacto de ninguna técnica de mejora de resultados. Este es el caso de los dos primeros problemas. El problema *Cáncer de colon* se clasifica bastante mal para el caso multiclase, éste se utilizará para comprobar si realmente hay una variación notable en los resultados obtenidos para problemas fácilmente clasificables y problemas que presentan dificultades.

Para la selección de los datos se han realizado pruebas con distintos clasificadores para comprobar si son problemas que de forma consistente *se dejan clasificar* correctamente. Es importante notar que no se ha aplicado ninguna técnica de tratamiento de datos cuyos resultados habrían podido condicionar la elección. Llegados a este punto aún se desconoce si las técnicas que se van a utilizar para mejorar los resultados – que sí se garantiza que serán buenos – van a tener un efecto positivo, negativo o nulo sobre ellos.

Por último, se han escogido problemas con marcadores de valores numéricos continuos.

A continuación se describen los tres *sets* de datos.

3.2 Flor de iris

El set de datos de la flor de iris es uno de los problemas clásicos de clasificación en la literatura. Fue presentado por Ronald Fisher en 1936, y es uno de los ejemplos más utilizados en estudios de clasificación.

Es un problema de clasificación *multiclase* en el que se tienen tres tipos de flor de iris: *Iris setosa*, *Iris virginica* e *Iris versicolor*, y se cuenta con varios marcadores, todos referidos a aspectos morfológicos de la flor: longitud del sépalo, grosor del sépalo, longitud del pétalo y grosor del pétalo. El set de datos cuenta con 50 muestras de cada clase.

Clase	Cantidad
Setosa	50
Virginica	50
Versicolor	50
Total	150

Tabla 3.2.1 - Flor de iris: clases.

A continuación se muestra un extracto de los mismos:

	V1	V2	V3	V4	V5
1	1	5,1	3,5	1,4	0,2
2	1	4,9	3	1,4	0,2
3	1	4,7	3,2	1,3	0,2
4	1	4,6	3,1	1,5	0,2
5	1	5	3,6	1,4	0,2
6	1	5,4	3,9	1,7	0,4
7	1	4,6	3,4	1,4	0,3
8	1	5	3,4	1,5	0,2
9	1	4,4	2,9	1,4	0,2
10	1	4,9	3,1	1,5	0,1
11	1	5,4	3,7	1,5	0,2

Tabla 3.2.2 - Flor de iris: set de datos.

Donde la primera columna se refiere a la clase: 1 *setosa*, 2 *virginica* y 3 *versicolor*, y las columnas restantes a los cuatro marcadores.

3.3 Cáncer de mama

Este segundo set de datos, desarrollado por William Wolberg, Nick Street y Olvi Mangasarian, de la Universidad de Wisconsin (EE.UU.), es también ampliamente utilizado en diversos estudios de *aprendizaje automático*.

Se trata de un problema de dos clases en los que se distingue entre pacientes con dos tipos de cáncer de mama: benigno y maligno. El set de datos cuenta con 569 muestras cuya clase depende de 30 marcadores referidos a distintas características del núcleo de las células cancerígenas. Las clases están repartidas de la siguiente forma:

Clase	Cantidad
Benigno	357
Maligno	212
Total	569

Tabla 3.3.1 - Cáncer de mama: clases.

3.4 Cáncer de colon

El último set de datos corresponde a otro problema de clasificación multiclase. En él se distingue entre pacientes con tres condiciones distintas: cáncer de colon, pólipo y sano.

Se tienen 202 muestras cuya clase depende de 52 marcadores. Las clases están repartidas de la siguiente forma:

Clase	Cantidad
Colon	86
Pólipo	31
Sano	85
Total	202

Tabla 3.4.1 - Cáncer de colon: clases.

4 DESCRIPCIÓN DEL TRABAJO

En este capítulo se explican los procedimientos llevados a cabo en la realización del estudio. En primer lugar se describen las herramientas de trabajo utilizadas, posteriormente se presentan los algoritmos desarrollados para cada una de las técnicas, así como las funciones y scripts utilizados.

4.1 Herramientas de trabajo

4.1.1 MATLAB

Dada la naturaleza del estudio, un punto crucial ha sido la elección de una herramienta adecuada de programación. Se ha decidido utilizar el programa MATLAB, que es una eficaz herramienta para el desarrollo de algoritmos de corte puramente matemático y que es de amplia utilización en ingeniería.

MATLAB incluye un lenguaje de programación propio basado en *scripts*, que permite una gran flexibilidad y rapidez a la hora de hacer pruebas, modificaciones y detectar errores. Además, MATLAB cuenta con una enorme colección de funciones, todo lo cual convierte su lenguaje de programación en una herramienta realmente potente.

Al ser una herramienta orientada al trabajo con matrices, se hace especialmente sencillo el operar con estos elementos tanto por la simple e intuitiva sintaxis como por la gran cantidad de opciones presentes.

Por último, todas las herramientas necesarias para el estudio están incluidas por defecto en el programa. Por todo esto he pensado que MATLAB es la herramienta idónea para el desarrollo del trabajo.

En momentos puntuales se ha hecho uso del programa RStudio para la lectura de bases de datos y su adaptación a MATLAB.

4.1.2 Equipo utilizado

Todo el trabajo, tanto el desarrollo de los algoritmos como la simulación, se ha desarrollado en el siguiente equipo:

- Procesador: Intel Core i7-4702MQ 2.2 GHz.
- Memoria: 6GB RAM.
- Sistema operativo: Windows 8.1 (64bits)
- Versión de MATLAB: R2013a (64bits)

4.2 Algoritmos

En este apartado se muestran los algoritmos desarrollados para el estudio.

4.2.1 Consideraciones importantes

Para el análisis de resultados y de rendimiento de los clasificadores, así como las técnicas de mejora, se ha hecho uso de *validación cruzada*. Se realizan *N* experimentos, y se divide aleatoriamente para cada uno de ellos el conjunto de muestras en dos sub-*sets*: entrenamiento y validación. El set de entrenamiento se usa durante la etapa de entrenamiento del clasificador, y el de validación se usa al final para comprobar su eficacia.

Al dividir el conjunto de muestras en entrenamiento y validación, se mantiene la misma proporción de cada clase en ambos *sets*. Además, se fuerza que el 70% de las muestras caigan en el *set* de entrenamiento con el fin de tener un número lo suficientemente grande para el aprendizaje del algoritmo, manteniendo un tamaño aún razonable para la fase de validación.

Durante la ejecución del algoritmo se hace un seguimiento del acierto sobre cada muestra en validación, de modo que no sólo se obtiene el ratio de acierto del clasificador, sino que también se identifica qué muestras presentan problemas de clasificación. Esta parte del algoritmo no se muestra en los diagramas de flujo por simplicidad.

4.2.2 Algoritmo de árbol de decisión

Se presenta el algoritmo de implementación del árbol de decisión utilizado en el estudio, véase la *figura 4.2.1*.

Se fija un número *N* de experimentos a realizar. Se tiene como resultado N clasificadores, cada uno ofreciendo unos resultados sobre las muestras que hay en el set de validación.

4.2.3 Algoritmo de bagging

A continuación se presenta el algoritmo de *bagging* utilizado en este estudio, véase la figura *4.2.2*.

Tal cual se ha explicado el método, primero se entrena a los árboles y luego, una vez todos están construidos, se clasifican las muestras de validación. En este caso se ha optado por unir ambas etapas en una. Esto es, para cada árbol que se construye se calcula seguidamente la predicción y se almacena, e inmediatamente después se destruye el árbol. Una vez hechas todas las predicciones, se vota y se genera la predicción final para ese set de entrenamiento. De este modo se evita tener que almacenar en memoria todos los árboles hasta el final del bucle. Esto se puede hacer porque sólo interesan los resultados y no los árboles generados, ya que éstos no se van a volver a usar para predicciones futuras.

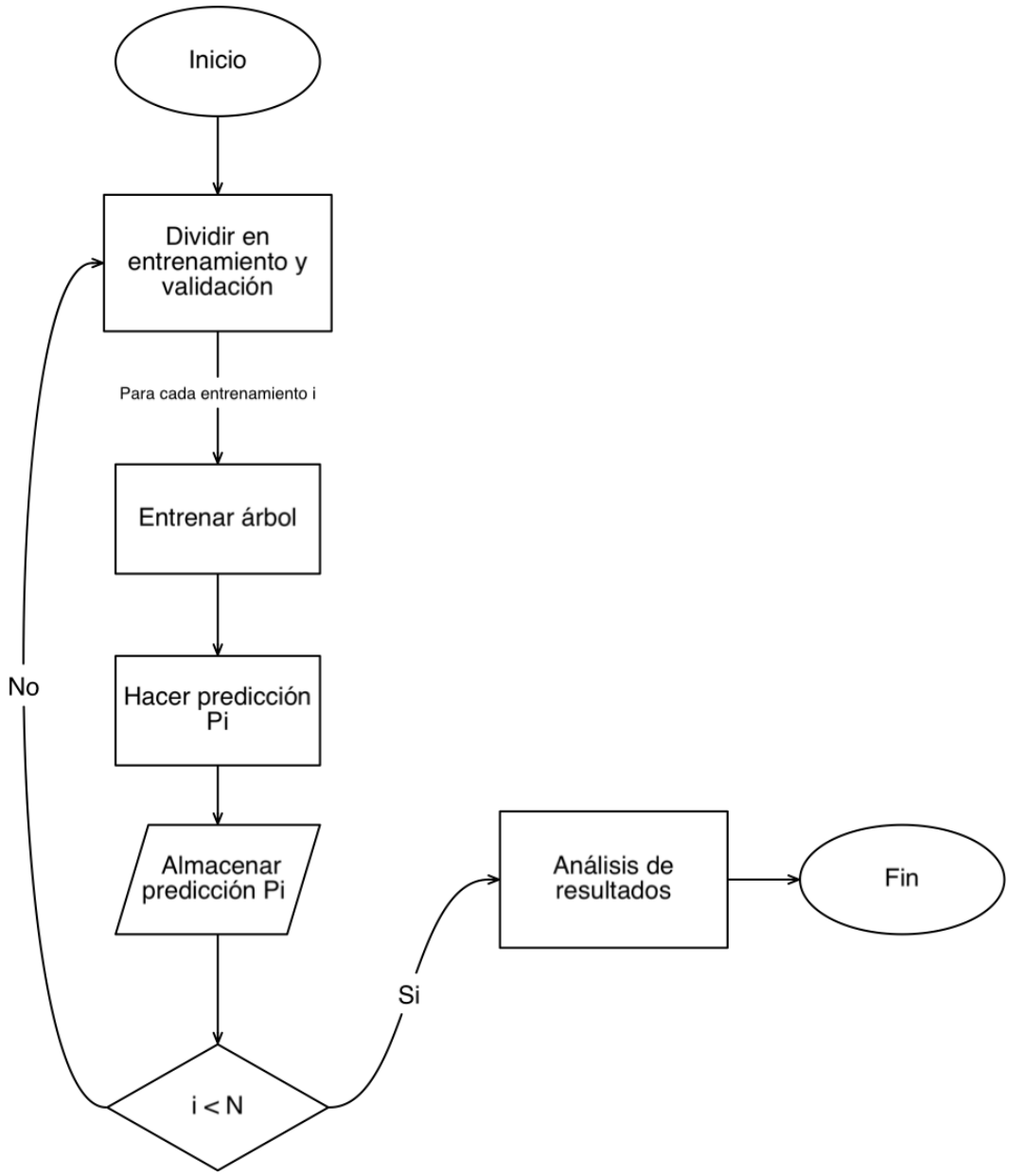

Figura 4.2.1 - Algoritmo de árbol de decisión.

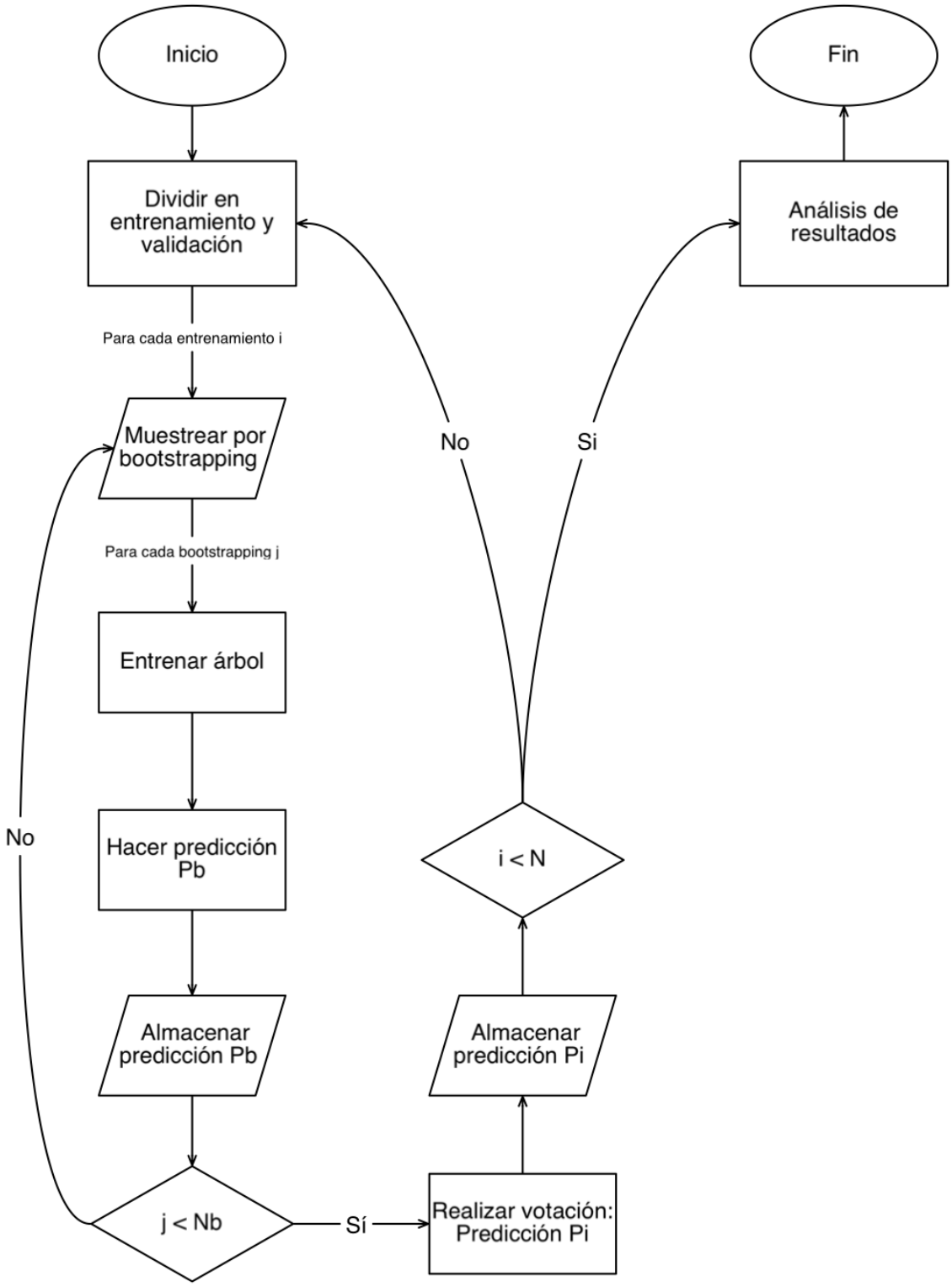

Figura 4.2.2 - Algoritmo de *bagging*.

4.2.4 Algoritmo de estimación del ratio de acierto mediante bagging

Este algoritmo difiere del anterior únicamente en que paralelamente a la predicción se calcula también la estimación del ratio de acierto. Tanto la predicción como el ratio de acierto se almacenan juntos, para poder ser comparados par a par en la fase de análisis de resultados. Véase la *figura 4.2.3*.

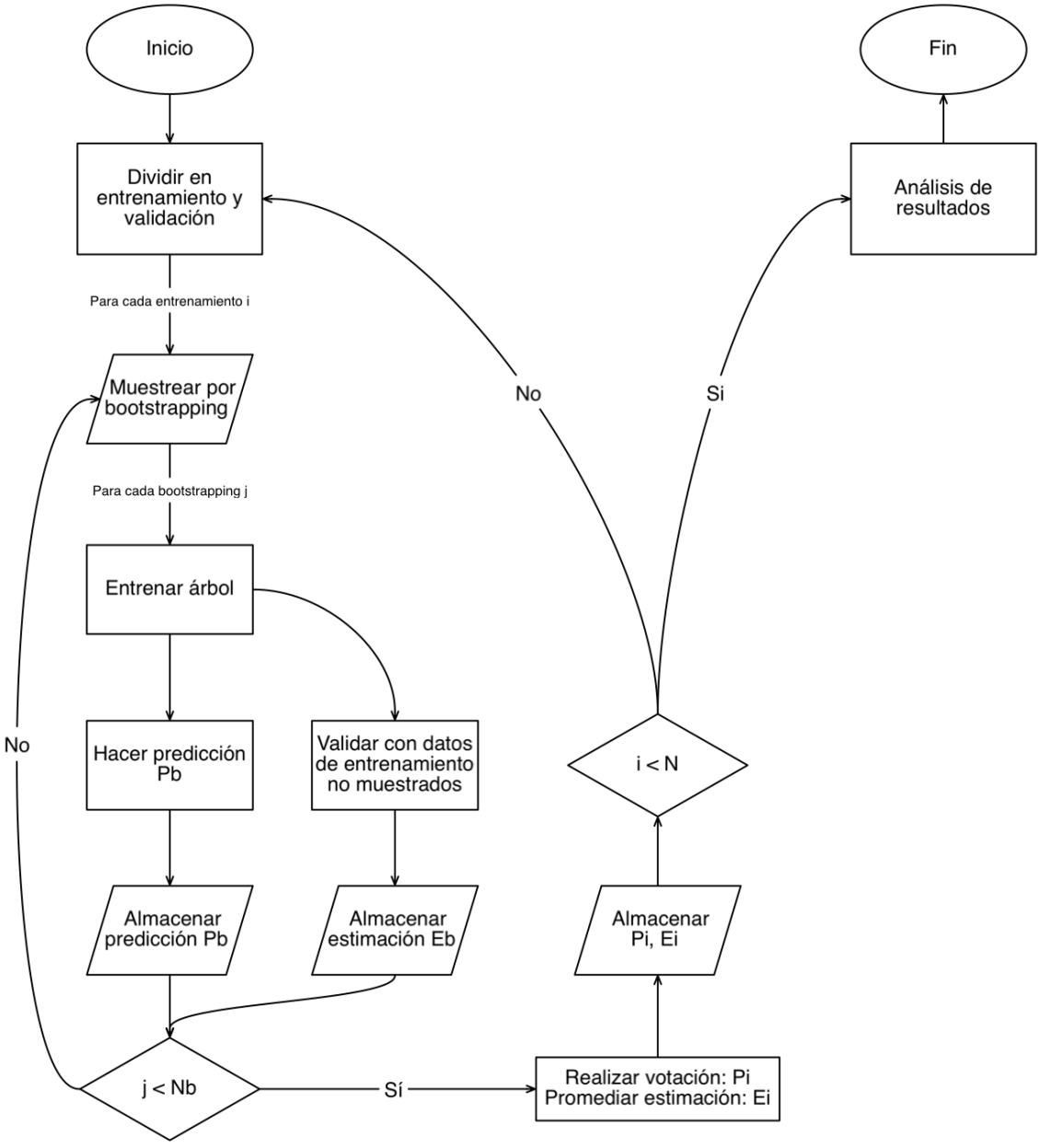

Figura 4.2.3 - Algoritmo para la estimación del ratio de acierto.

4.3 Funciones de MATLAB utilizadas

La mayor parte del código utilizado se ha desarrollado exclusivamente para este estudio, si bien se han utilizado las siguientes funciones de MATLAB con el fin de simplificar el trabajo en las áreas que no son objeto de un estudio detallado.

4.3.1 Función *classregtree*

Para la implementación del árbol de decisión se ha utilizado la función de MATLAB $\boldsymbol{classregtree}(\boldsymbol{X},\boldsymbol{Y})$, que genera un objeto de la clase árbol de decisión a partir de una matriz *X* y un vector columna *Y*. *X* es una matriz *nxm* cuyas filas representan las características de cada muestra. *Y es* de longitud *n* igual al número de muestras, y contiene los valores de las posibles clases a las que puede pertenecer una muestra

Por ejemplo, para el caso de entrenar a un *árbol de decisión* con todas las muestras del set de *Flor de Iris* se obtiene el árbol de decisión mostrado en la *figura 4.3.1*[2]. En la imagen se muestran las variables que ha elegido el algoritmo para particionar cada nodo padre, así como la condición para que una muestra caiga en un *nodo* hijo o en otro. A la clase *Setosa* se ha asignado el valor 1, a la clase *Versicolor* el valor 2, y a la clase *Virginica* el valor 3.

El algoritmo utilizado por la función *classregtree* permite problemas en los que las clases son números ordinales. Esto es, los valores numéricos asignados a cada clase importan, pues se considera que hay clases que están más cerca de otras. Es por esto por lo que dos nodos hoja devuelven como solución un número no entero, al considerar el algoritmo que el resultado será *Versicolor* o *Virginica*, pero nunca *Setosa*. Para esta asignación particular de clases nunca se podría obtener un resultado entre Setosa y Virginica.

Por este motivo, he decidido que cualquier resultado no entero sea considerado un error de predicción.

4.3.2 Función *eval*

Posteriormente, para predecir la clase a la que pertenece un conjunto de muestras se ha utilizado la función de MATLAB $\boldsymbol{eval}(\boldsymbol{T},\boldsymbol{X})$, que recibe como parámetros un objeto *T* de la clase *árbol de decisión*, y una matriz X de tamaño nxm cuyas filas contienen la información de cada una de las muestras, tal y como se ha explicado para la función *classregtree*. La función devuelve una vector columna Y_{eval} con las predicciones para cada una de las muestras en *X*.

[2] Para mostrar el árbol se ha hecho uso de la función *view* (T), donde T es el objeto de clase árbol generado con la función *classregtree*.

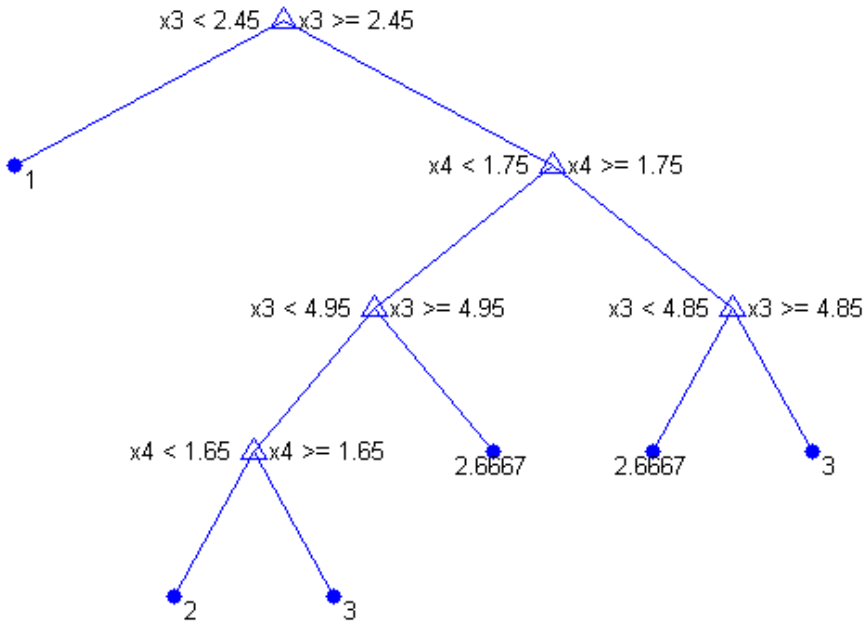

Figura 4.3.1 - Árbol de decisión generado con la función *classregtree*.

4.4 Scripts

A continuación se detallan los *scripts* desarrollados durante el estudio, y que componen los algoritmos descritos más arriba.

Los *scripts* tienen como ventaja que utilizan variables globales, lo que acelera enormemente el desarrollo del código ya que se evita utilizar funciones que reciban parámetros por referencia, como ocurre en otros lenguajes de programación tradicionales como C y C++. Trabajar con variables globales requiere un cuidado especial, pues resulta bastante más difícil seguir la pista de un error al tener varias partes de código acceso a las mismas variables.

4.4.1 Core.m

Es el *script* principal. Desde él se llama al resto de scripts y funciones, y es donde se ejecuta todo el algoritmo.

Cuenta con una primera parte de configuración, donde se seleccionan el tipo de problema que se quiere resolver y el tipo de técnica de mejora de resultados y se cargan los datos. Aquí también se preparan los datos para poder hacer un seguimiento de cada muestra. Esto se hace añadiendo un identificador para cada muestra, de manera que el algoritmo pueda seguir la pista de cada una cuando son aleatoriamente divididas y barajadas dentro de un grupo u otro.

La segunda etapa es el bucle de validación cruzada. En cada iteración se dividen los datos en set de entrenamiento y de validación, y se adaptan los mismos para que puedan ser interpretados por el clasificador. A continuación se ejecuta la técnica de clasificación seleccionada y se almacenan los resultados.

La tercera y última parte del algoritmo está destinada a mostrar los resultados, tanto por pantalla como a través de gráficas.

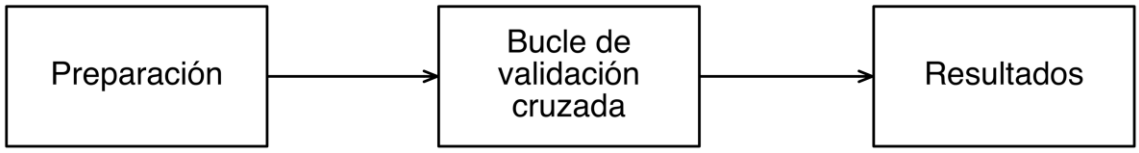

Figura 4.4.1 - Script core.m

4.4.2 SplitData.m

Se ejecuta al inicio de cada experimento. Su función principal es dividir los datos en set de *validación* y de *entrenamiento*.

Permite seleccionar la proporción de muestras que caerá en el set de entrenamiento, que como se ha mencionado anteriormente se ha fijado en 70% para todos los experimentos. Al hacer la división se fuerza que se mantenga la proporción de cada clase en cada uno de los *sets*.

Se separa a su vez cada set en el las matrices *X* e *Y*. La matriz *X* es de dimensión *nxm*, y es la matriz que contiene las *m* características de cada una de las *n* muestras. La matriz *Y* tiene dimensión *nx2*, la primera columna contiene el identificador de cada una de las n muestras, y la segunda columna la clase.

Por último se adapta la clase de cada muestra al clasificador: a la clase 1 se asigna el valor de 1, a la clase 2 se asigna el valor de -1, y a la clase 3 se asigna el valor de 0.

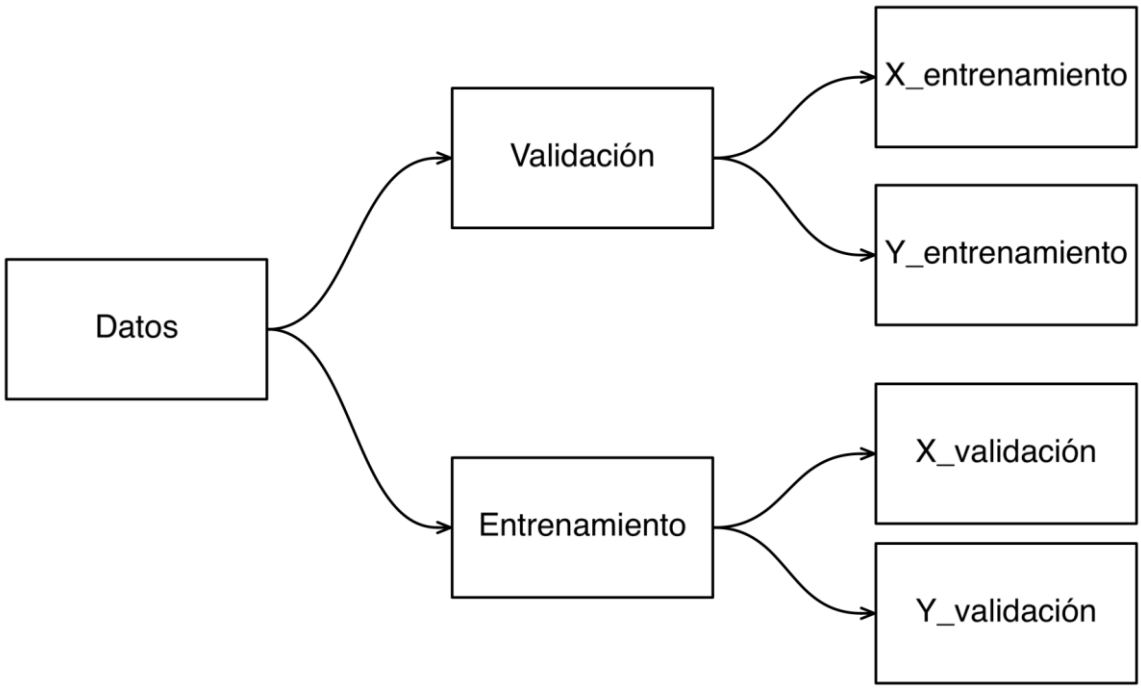

Figura 4.4.2 - Script SplitData.m

4.5 Funciones

Se hace una descripción de las funciones desarrolladas para el estudio.

4.5.1 LoadAllData

Esta función carga los datos del problema de clasificación en MATLAB en una matriz de dimensión *nxm+1*. Cada fila contiene la información de cada una de las *n* muestras, la primera columna contiene la clase de cada una de ellas, y el resto sus *m* características.

Esta función recibe como parámetro una cadena de caracteres con el nombre del problema de clasificación que se quiere cargar, y devuelve la matriz con los datos del problema.

4.5.2 LoadAllData_KeepingTrack

Esta función realiza la misma operación que la función LoadAllData, y además agrupa los datos por clases dentro de la matriz, con el objeto de mostrar una gráfica de dispersión más clara. También se añade una columna a la matriz de datos con el identificador para cada muestra.

Esta función recibe como parámetro una cadena de caracteres con el nombre del problema de clasificación que se quiere cargar, y devuelve la matriz con los datos del problema.

4.5.3 BaggingMethod

Función que implementa el método de *bagging* para un árbol de decisión. Recibe como parámetros el set de entrenamiento, las características de las muestras del set de validación y el número de clasificadores que se quiere construir mediante *bootstrapping*. Devuelve la predicción que es el resultado de los votos de todos los clasificadores.

Para hacer el muestreo de *bootstrapping* se hace una llamada a la función *BootStrapping_sample* descrita más adelante, y para construir el clasificador y hacer la predicción se hace uso de las funciones *classregtree* y *eval* descritas en el apartado 4.3.

4.5.4 BaggingEstimationMethod

Es la función que implementa el método de bagging junto con el de estimación del ratio de acierto. Recibe como parámetros el set de entrenamiento, las características de las muestras del set de validación y el número de clasificadores que se quiere construir mediante *bootstrapping*. Devuelve la predicción que es el resultado de los votos de todos los clasificadores y la estimación del ratio de acierto.

Al igual que hacía la función *BaggingMethod*, se hace una llamada a la función BootStrapping_sample para muestrear el set de entrenamiento. Además, se hace una llamada a la función Bootstrapping_remaining para obtener los datos que no han sido muestreados. Finalmente se llevan a cabo la predicción y la estimación.

4.5.5 BootStrapping_sample

Esta función lleva a cabo el proceso de muestreo mediante bootstrapping, que utiliza la técnica de muestreo con reposición tal y como se ha explicado en el capítulo 2.

La función recibe como parámetros las matrices *Y* y *X* que contienen los identificadores, las clases y las características del set de datos que se quiere muestrear. Devuelve las matrices *Xsample* e *Ysample*, que contienen los datos muestreados.

Para muestrear se hace uso de la función de MATLAB *datasample*.

4.5.6 BootStrapping_remaining

Esta función identifica los elementos que no han sido muestreados en la fase de *bootstrapping*. La función recibe como parámetros el set de entrenamiento completo y los índices de los datos muestreados, y devuelve el sub-set de datos no muestreados.

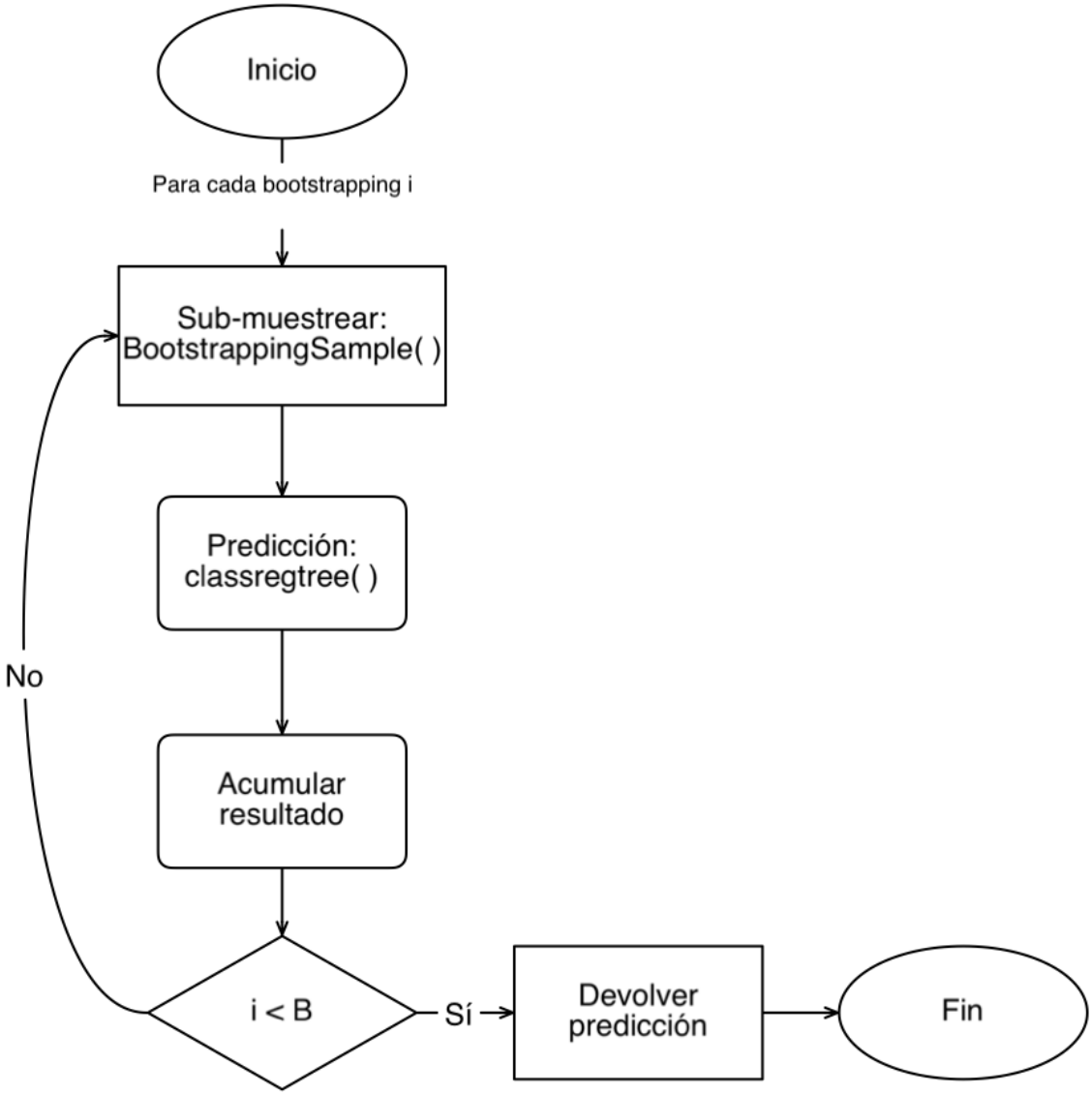

Figura 4.5.1 - Función *BaggingMethod*.

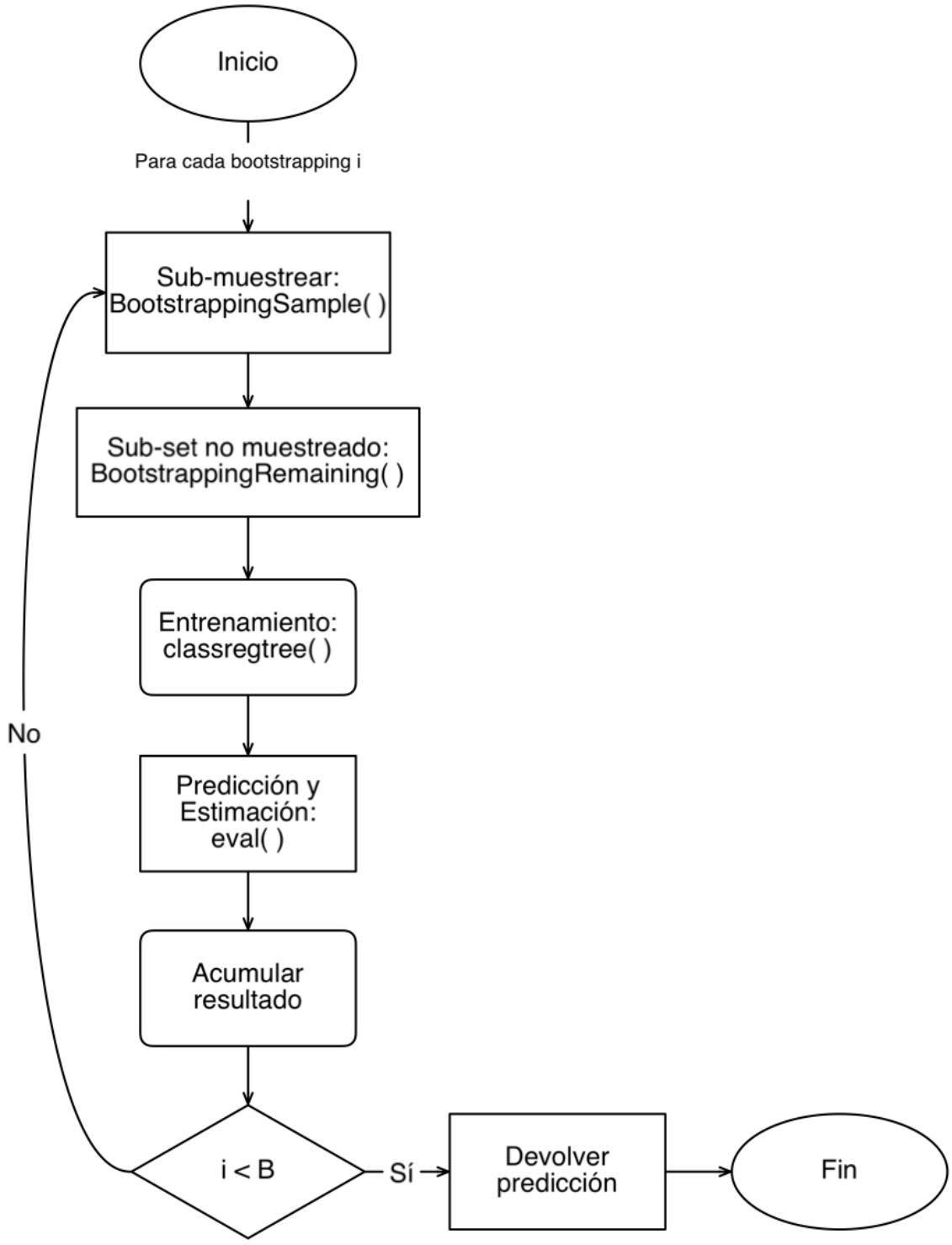

Figura 4.5.2 - Función *BaggingEstimationMethod*.

5 RESULTADOS

En este quinto capítulo se muestran los resultados de cada uno de los métodos utilizados en el estudio aplicados a los tres problemas. Además se hace un análisis de los mismos, dejando para el capítulo 6 el análisis comparativo y las conclusiones del estudio. He pensado que es más fácil seguir las explicaciones a medida que se ven los resultados junto con las gráficas, en lugar de simplemente mostrar los resultados sin comentarios por un lado, y su análisis sin las gráficas por otro.

5.1 Aplicación del árbol de decisión a los problemas de clasificación

A continuación se muestran los resultados obtenidos tras resolver el problema de clasificación utilizando un *árbol de decisión*, y con el algoritmo de validación cruzada arriba descrito. Para todos los casos se han llevado a cabo 10.000 experimentos.

Para cada problema se muestran dos gráficas. La primera es un histograma que muestra la distribución de la función de probabilidad del acierto. La segunda es una gráfica de dispersión donde cada punto representa a una muestra y el porcentaje de veces que se clasificó incorrectamente. Además, para los problemas de clasificación del cáncer de mama y de colon se incluyen histogramas con la información sobre falsos negativos y falsos positivos.

5.1.1 Flor de iris

Se observa que la totalidad de las muestras pertenecientes a la clase *Setosa* se clasifican correctamente, por lo que se infiere que ésta tiene unas características muy distintas a las otras dos.

En la *tabla 5.1.1* se muestra el porcentaje de éxito y su desviación típica, para ser comparados posteriormente con los obtenidos mediante el algoritmo de *bagging*.

% éxito	Desviación típica
92,38	3,50

Tabla 5.1.1 - Resultados de aplicar árbol de decisión al problema *Flor de Iris*.

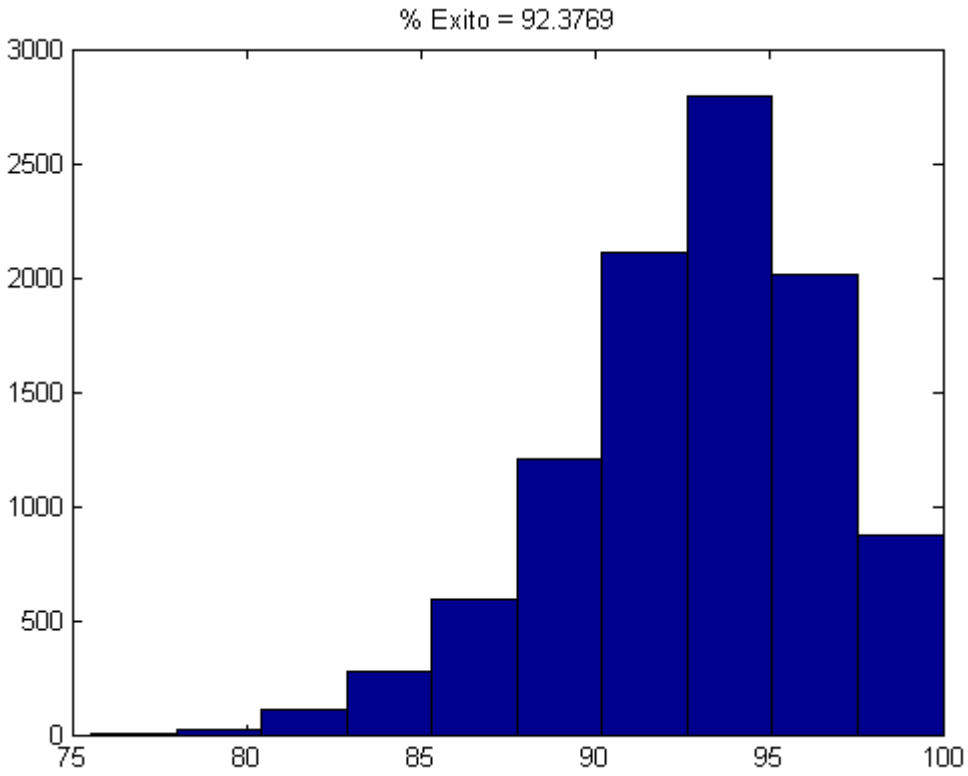

Figura 5.1.1 - Clasificación de Flor de Iris mediante árbol de decisión.

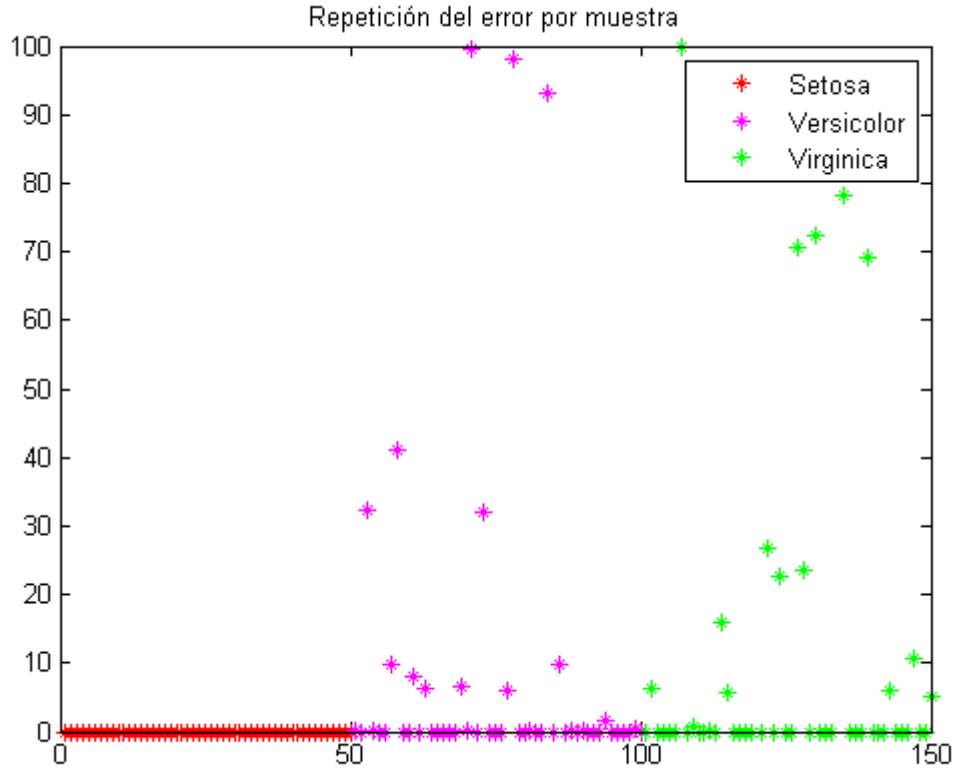

Figura 5.1.2 - Flor de iris. Error por cada muestra usando árbol de decisión.

5.1.2 Cáncer de mama

A pesar de obtenerse unos resultados aparentemente parecidos a los conseguidos para el problema de clasificación *Flor de Iris*, se puede observar que la aleatoriedad de qué muestra será la que se clasifique incorrectamente es mucho mayor (*figura 5.1.6*). En el problema anterior hay unas determinadas muestras que se clasifican incorrectamente de una manera consistente, mientras que en este problema la variabilidad de las muestras clasificadas de manera errónea es mucho mayor.

	%	Desviación típica
Éxito	89,16	2,66
Falso Positivo	6,53	2,76
Falso Negativo	9,92	4,10

Tabla 5.1.2 - Resultados de aplicar árbol de decisión al problema *Cáncer de mama*.

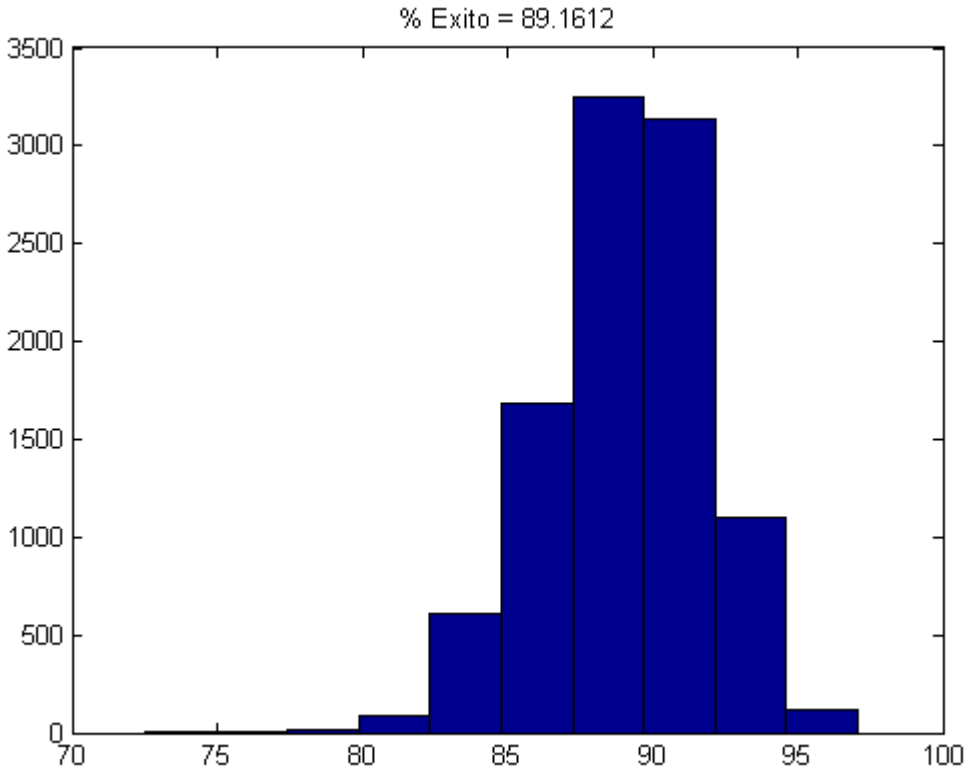

Figura 5.1.3 - Clasificación de *Cáncer de Mama* mediante árbol de decisión.

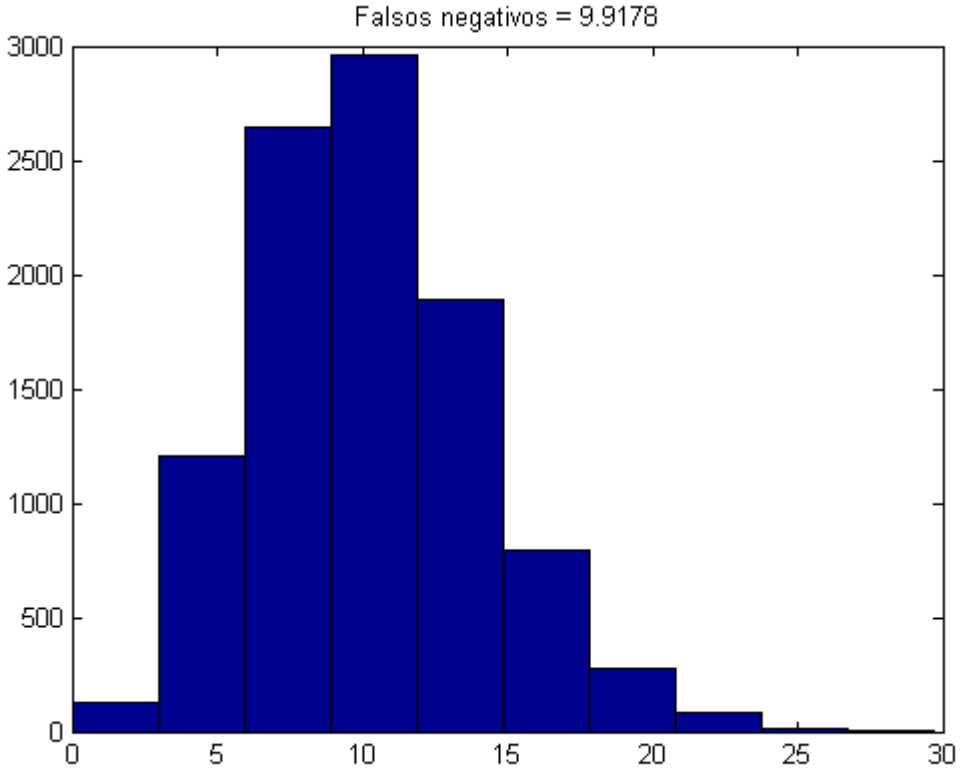

Figura 5.1.4 - Falsos positivos de *Cáncer de Mama* mediante árbol de decisión.

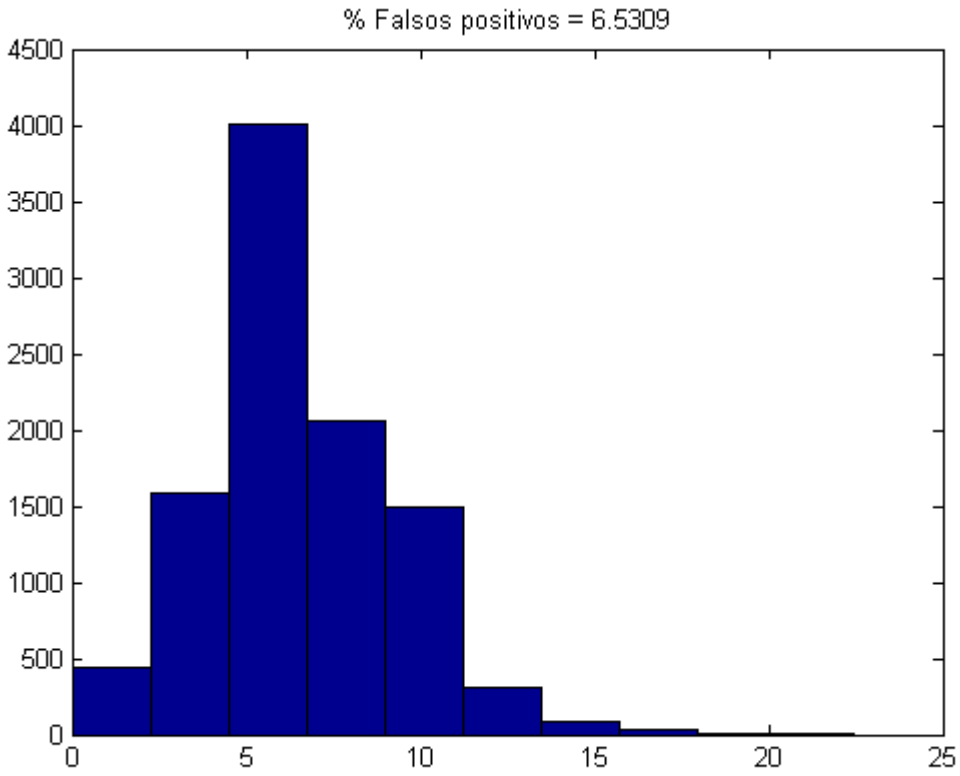

Figura 5.1.5 - Falsos negativos de *Cáncer de Mama* mediante árbol de decisión.

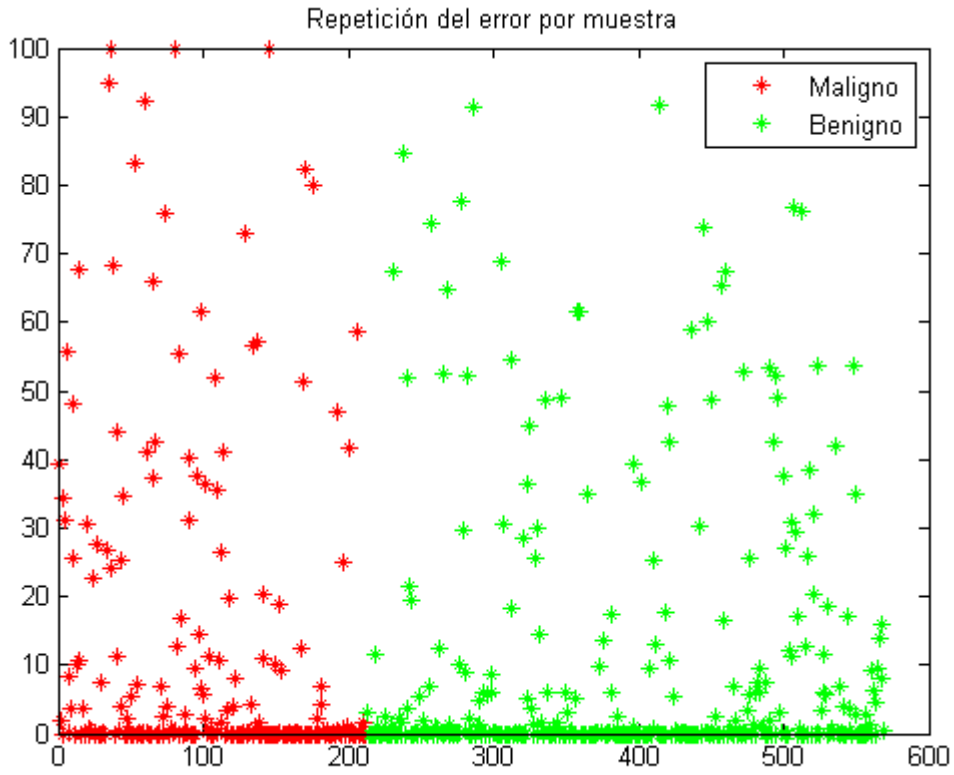

Figura 5.1.6 - Cáncer de mama. Error por cada muestra usando árbol de decisión.

5.1.3 Cáncer de colon

A la vista de los resultados se hace evidente que el algoritmo del árbol de decisión no es suficiente para este problema, se observa que hay muchas muestras que se clasifican mal en un porcentaje relativamente alto de veces. Se espera que al aplicar el método de bootstrapping se consiga reducir dichos errores.

También se observa que el clasificador tiene problemas identificando las muestras de clase *pólipo*. Esto se debe a que hay menos muestras de esta clase y que la mayoría de ellas presentan características parecidas a la clase *cáncer*.

La desviación típica del porcentaje de éxito es también muy elevada, lo que pone de manifiesto la inestabilidad y poca fiabilidad del árbol de decisión para este problema.

	%	Desviación típica
Éxito	51,28	6,68
Falso Positivo	26,41	9,99
Falso Negativo	23,48	8,84

Tabla 5.1.3 - Resultados de aplicar árbol de decisión al problema *Cáncer de mama*.

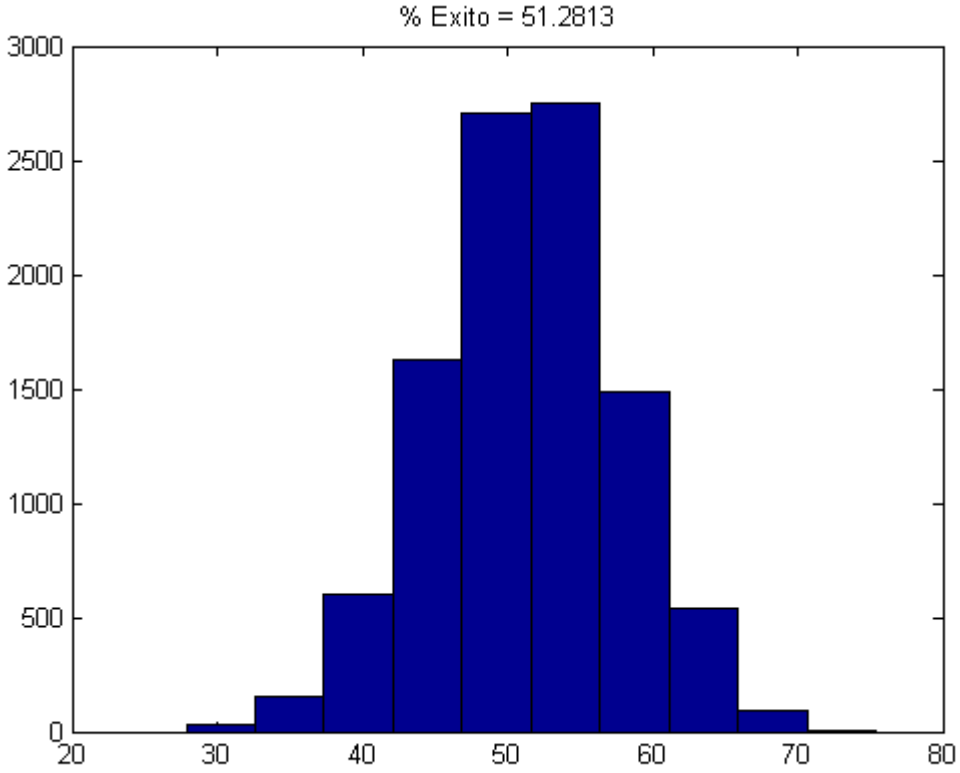

Figura 5.1.7 - Clasificación de *Cáncer de Colon* mediante árbol de decisión.

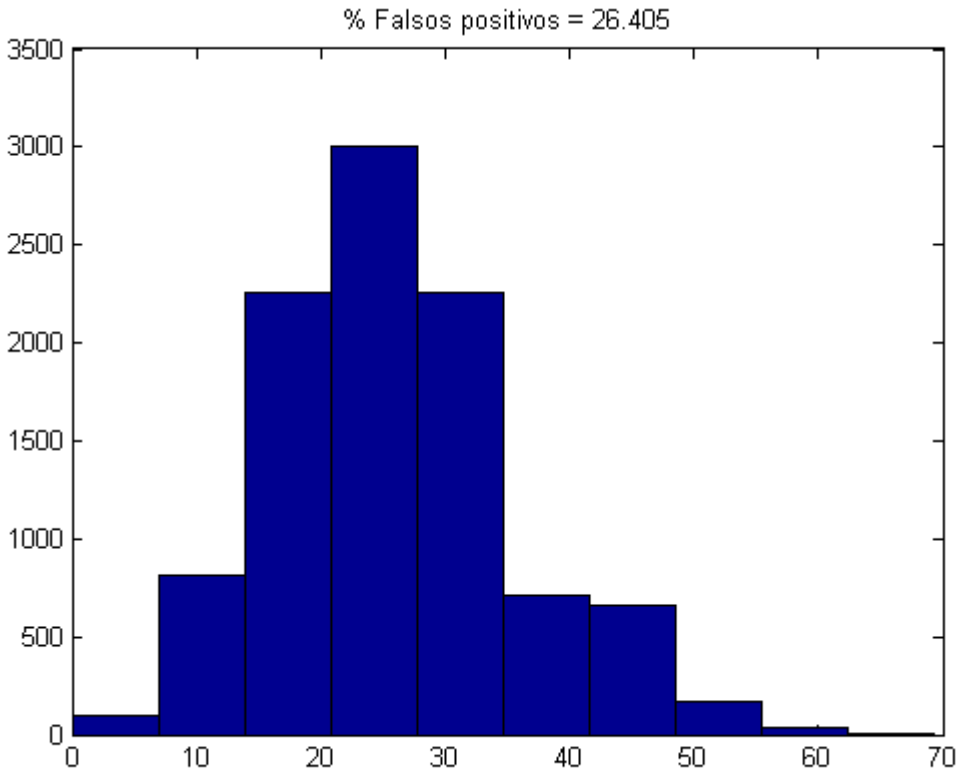

Figura 5.1.8 - Falsos positivos de *Cáncer de Colon* mediante árbol de decisión.

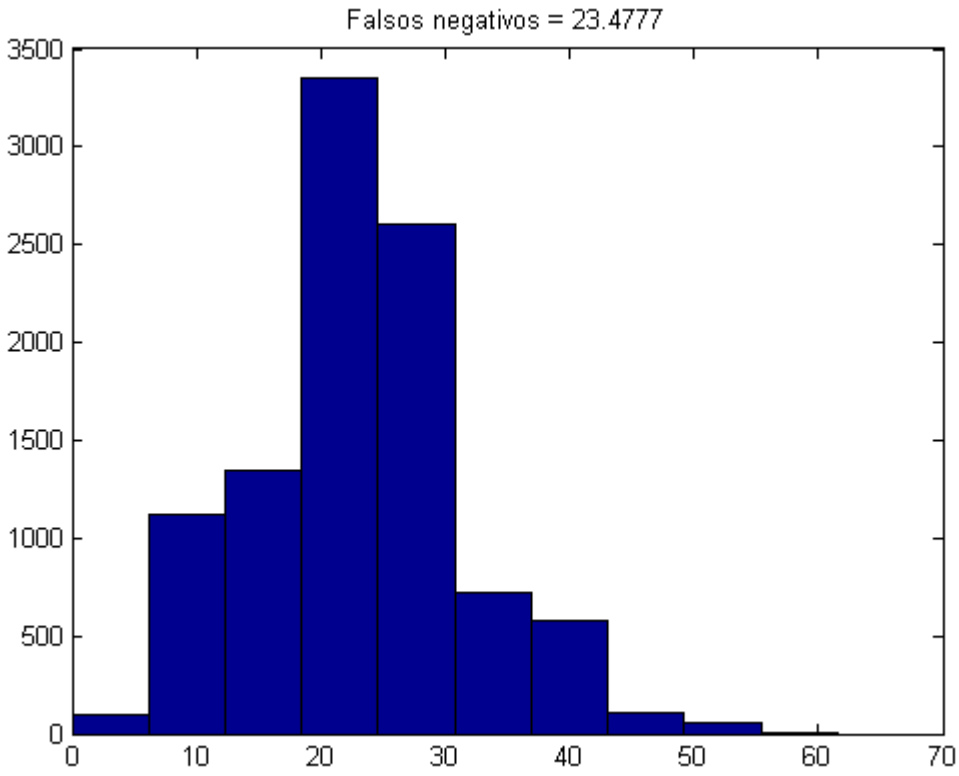

Figura 5.1.9 - Falsos negativos de *Cáncer de Colon* mediante árbol de decisión.

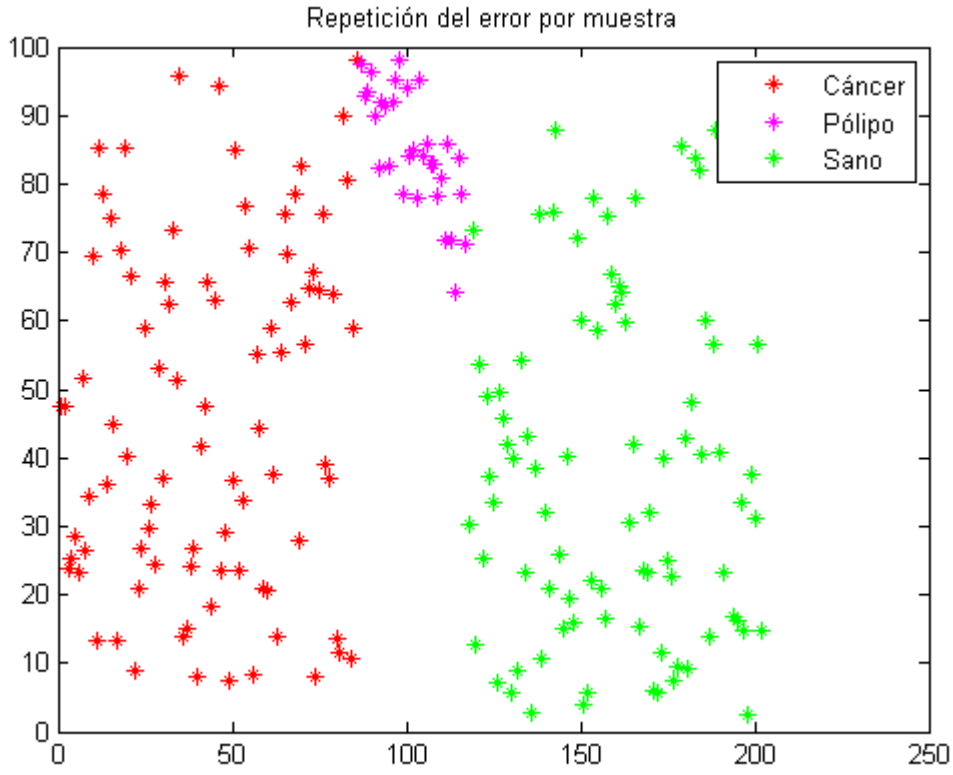

Figura 5.1.10 - Cáncer de colon. Error por cada muestra usando árbol de decisión.

5.2 Aplicación de la técnica de bootstrapping a los problemas de clasificación

Se muestran los resultados obtenidos tras ejecutar el algoritmo de validación cruzada con la técnica de *bagging* aplicada al mismo árbol de decisión que en apartado 5.1. Para los tres casos se han llevado a cabo 10.000 experimentos con 50 muestreos de bootstrapping cada uno. De este modo, cada experimento cuenta con la aportación de 50 clasificadores, siendo la clase con más votos la elegida como solución.

5.2.1 Flor de iris

Se observa una ligera mejora en los resultados, tanto en el porcentaje de acierto como en la desviación típica de dicho porcentaje, lo que dará lugar a unos intervalos de confianza mejores, tal y como predice la técnica de *bootstrapping*. Al ser los resultados realmente buenos aplicando simplemente el árbol de decisión, la diferencia de utilizar el método de bagging o no no es muy relevante.

% éxito	Desviación típica
93,79	3,23

Tabla 5.2.1 - Resultados de aplicar *bagging* al problema *Flor de Iris*.

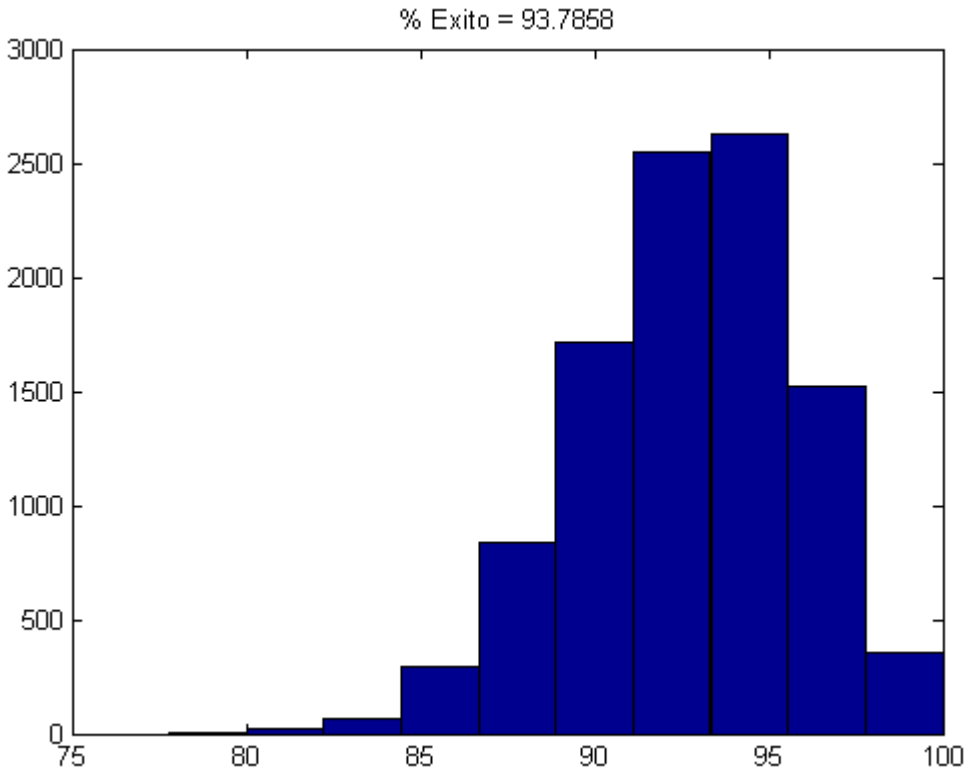

Figura 5.2.1 - Clasificación de *Flor de Iris* aplicando *bagging*.

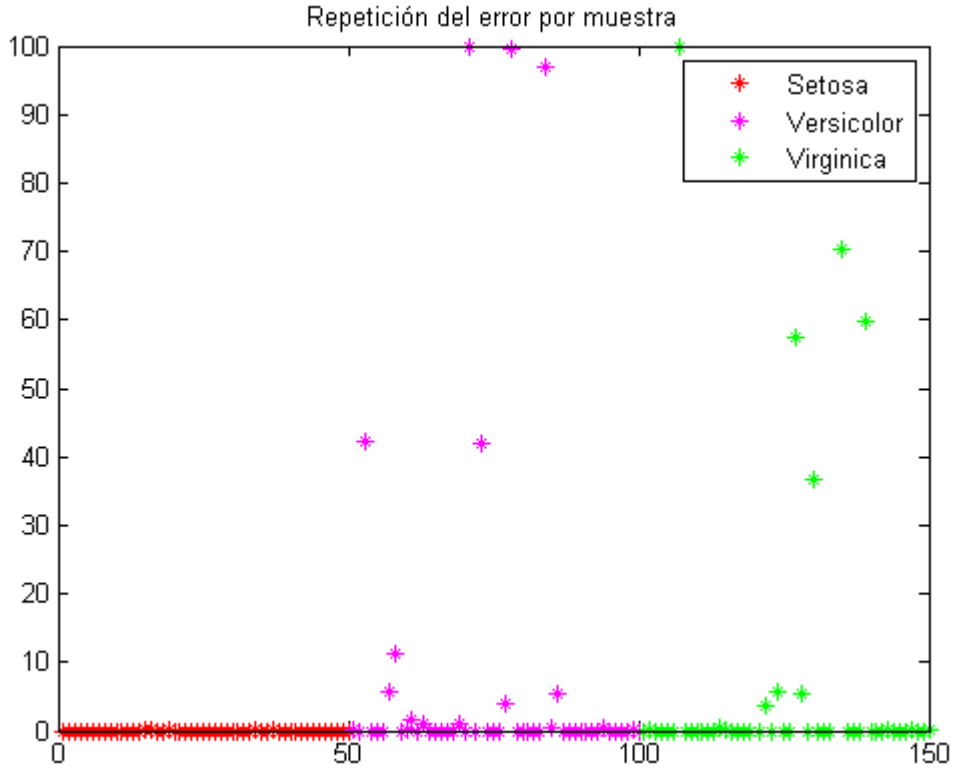

Figura 5.2.2 - Flor de Iris. Error por cada muestra aplicando *bagging*.

5.2.2 Cáncer de mama

A continuación se muestran los resultados para el problema *Cáncer de Mama* aplicando la técnica de *bagging*.

	%	Desviación típica
Éxito	90,70	2,05
Falso Positivo	7,73	2,86
Falso Negativo	4,00	2,62

Tabla 5.2.2 - Resultados de aplicar *bagging* al problema *Cáncer de Mama*.

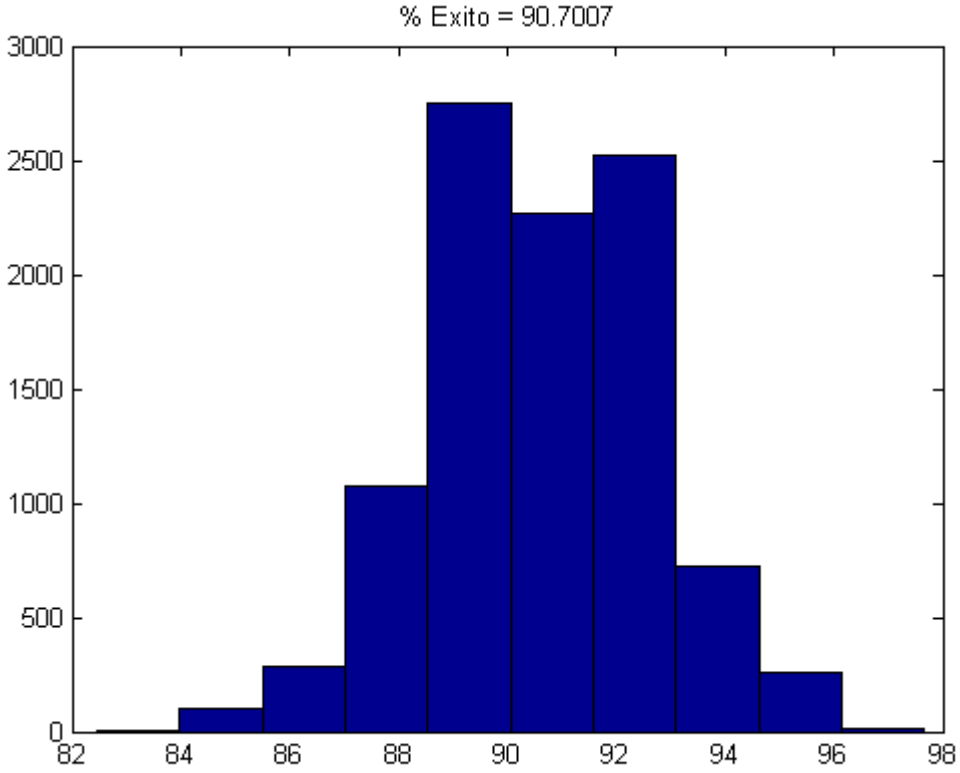

Figura 5.2.3 - Clasificación de *Cáncer de Mama* aplicando *bagging*.

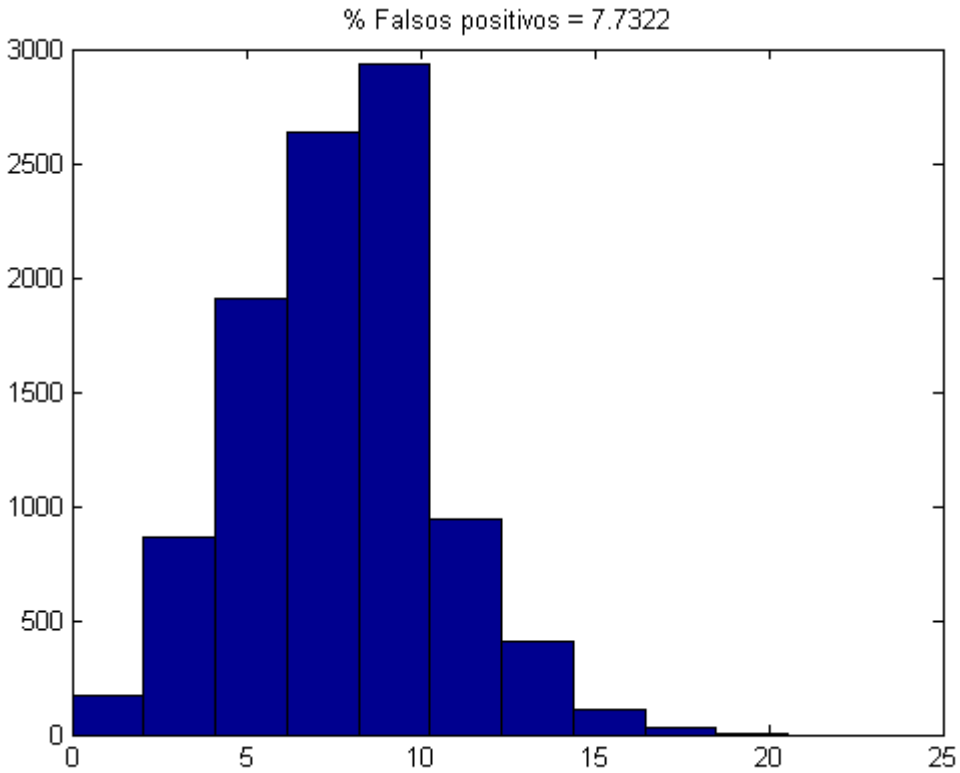

Figura 5.2.4 - Falsos positivos para *Cáncer de Mama* aplicando *bagging*.

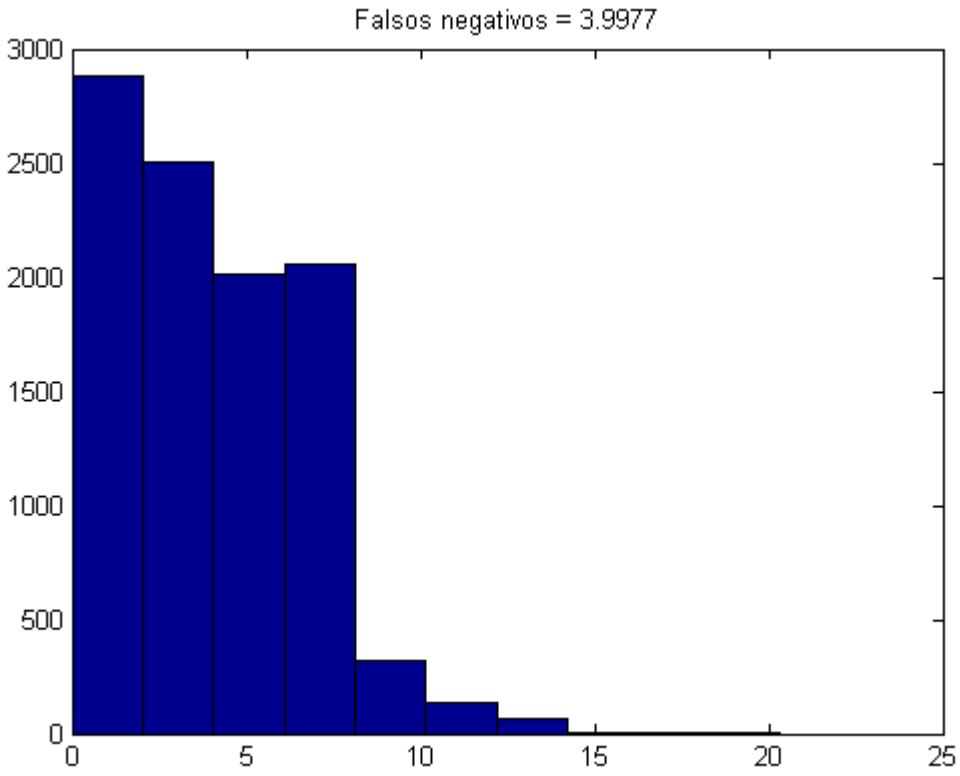

Figura 5.2.5 - Falsos negativos para *Cáncer de Mama* aplicando *bagging*.

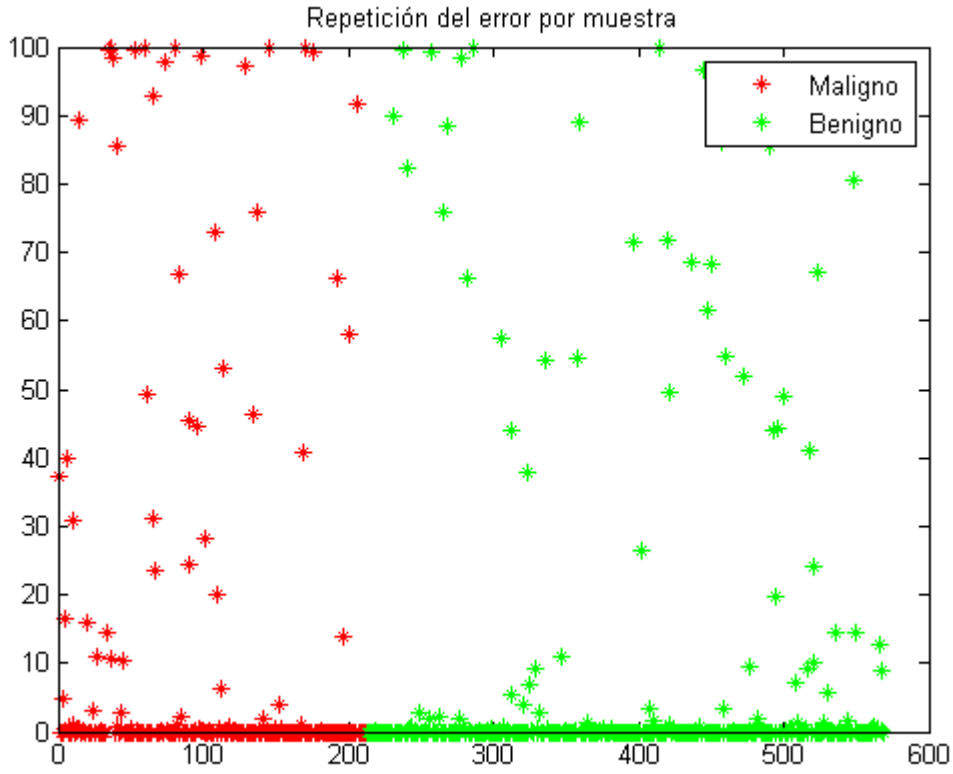

Figura 5.2.6 - Cáncer de Mama. Error por cada muestra aplicando *bagging*.

5.2.3 Cáncer de colon

Se muestran los resultados tras aplicar la técnica de *bagging* al problema *Cáncer de Colon*.

	%	Desviación típica
Éxito	65,22	5,35
Falso Positivo	31,02	9,46
Falso Negativo	6,19	4,81

Tabla 5.2.3 - Resultados de aplicar *bagging* al problema *Cáncer de Colon*.

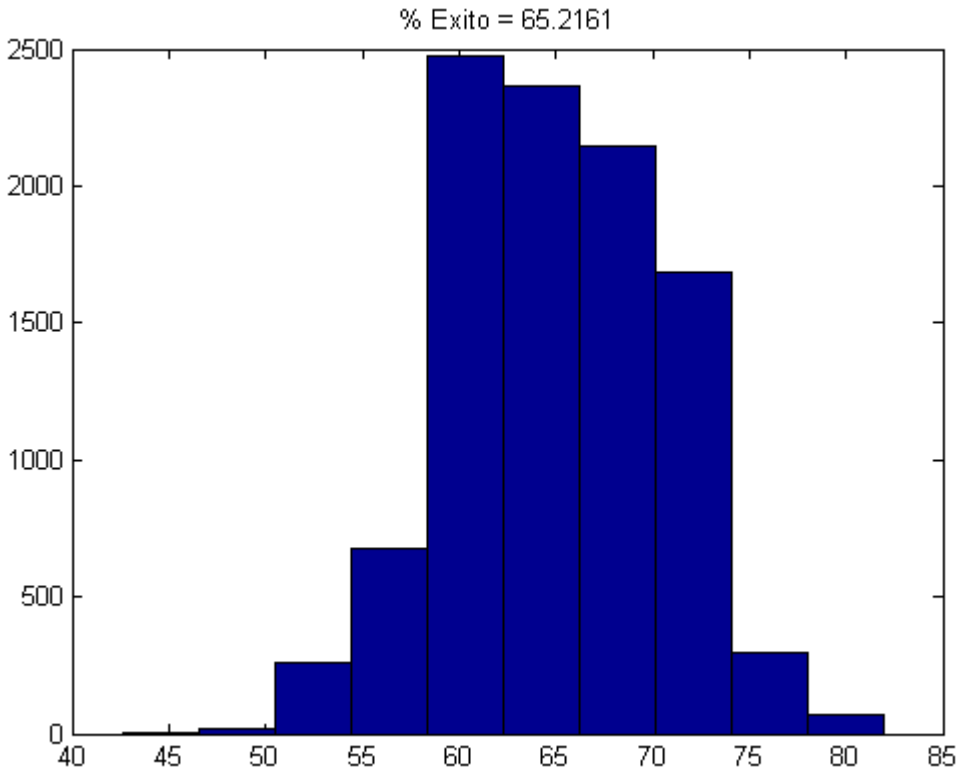

Figura 5.2.7 - Clasificación de *Cáncer de Colon* aplicando *bagging*.

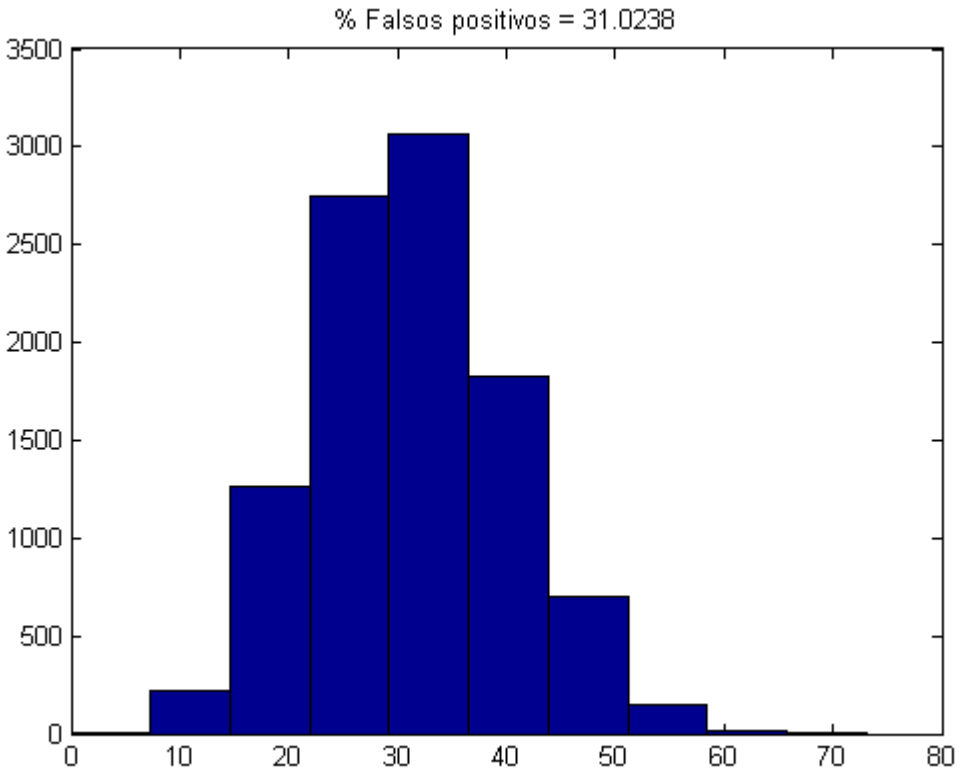

Figura 5.2.8 - Falsos positivos para *Cáncer de Colon* aplicando *bagging*.

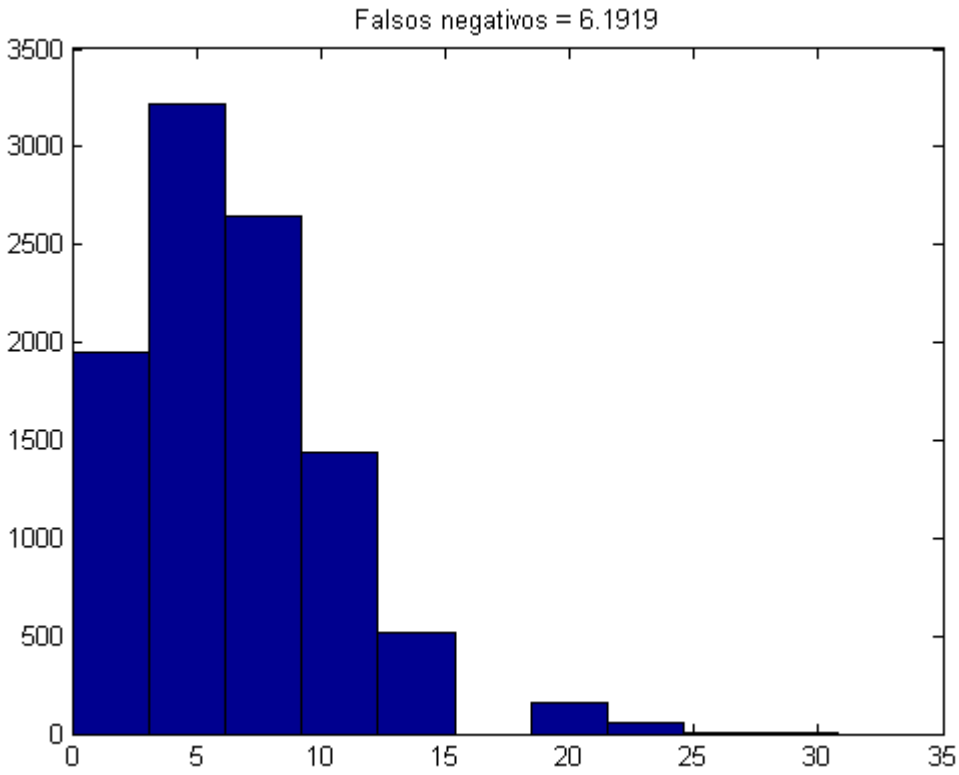

Figura 5.2.9 - Falsos negativos para *Cáncer de Colon* aplicando *bagging*.

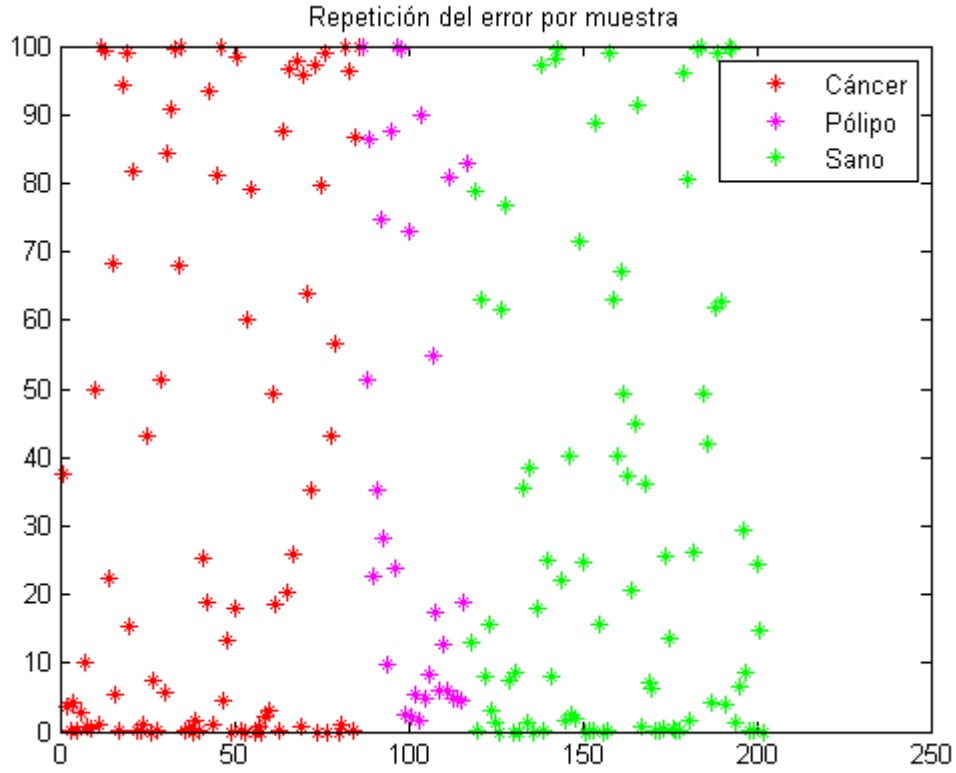

Figura 5.2.10 - Cáncer de Colon. Error por cada muestra aplicando *bagging*.

5.3 Estimación del ratio de acierto mediante bootstrapping

A continuación se muestran los resultados obtenidos tras ejecutar el algoritmo de estimación del acierto por *bootstrapping* aplicado a los problemas de los apartados anteriores usando el árbol de decisión. Para los tres casos se han llevado a cabo 10.000 experimentos con 50 muestreos de *bootstrapping* cada uno. De este modo cada experimento cuenta con la aportación de 50 estimaciones del acierto, siendo la media de todas ellas la estimación final para cada experimento.

En la primera gráfica se muestran los valores de la predicción del acierto y el acierto real para cada experimento junto con una recta a 45 grados para una fácil identificación de qué elemento tiene mayor valor: si el resultado final o la estimación. Los puntos por encima de esta recta corresponden a los experimentos en los que el acierto real mayor que la estimación, y los puntos por debajo de la recta corresponden al caso contrario. Interesa que los puntos estén por encima de esta recta, ya que esto permitiría acotar inferiormente el ratio de acierto mediante la estimación. La segunda recta[3] divide los experimentos en dos sub-conjuntos, el superior conteniendo el 90% de los experimentos y el inferior los restantes.

La segunda gráfica es un histograma que muestra la distribución de la diferencia entre el acierto real y la estimación, en tanto por ciento, para los 10.000 experimentos. Esta gráfica es interesante para ver la desviación típica, la media, y calcular intervalos de confianza. El criterio elegido es que se consiga una seguridad de al menos un 90% para la estimación del ratio de acierto. Esto se explica en más detalle para cada uno de los casos más abajo.

En esta segunda gráfica se muestran únicamente los 1.000 primeros experimentos para que se

[3] Ambas rectas pasan por el origen.

aprecie claramente la densidad de la nube de puntos, ya que para una mayor cantidad da la impresión de que hubiera la misma densidad de puntos entorno a la recta de 45 grados que a distancias más alejadas.

5.3.1 Flor de iris

Se observa que la cantidad de predicciones pesimistas es de un 60%. O sea, en el 60% de las veces el resultado real es mejor que el predicho. Se desearía que este porcentaje fuera lo más alto posible para, así, poder acotar el ratio de acierto con una confianza grande. Este número no es lo suficientemente grande, por lo que en su lugar se buscará un intervalo de confianza del 90%, o lo que es lo mismo, el percentil 10. Esto es igual a cambiar la inclinación de la recta Y=X hasta conseguir la separación deseada tal como se muestra en la *figura 5.3.1*.

Para el cálculo del percentil 10 se utiliza la distribución mostrada en la *figura 4.3.2*, resultando ser -5.34. Esto quiere decir que el 90% de los experimentos estarán por encima de la recta con una pendiente un 5,34% menor que la recta Y=X.

Por ejemplo, si para un determinado experimento se obtiene una estimación del acierto del 95%, se puede garantizar con un 90% de seguridad que el $95 - 5{,}34 = 89{,}66\%$ de las muestras que se pasen al clasificador serán clasificadas correctamente.

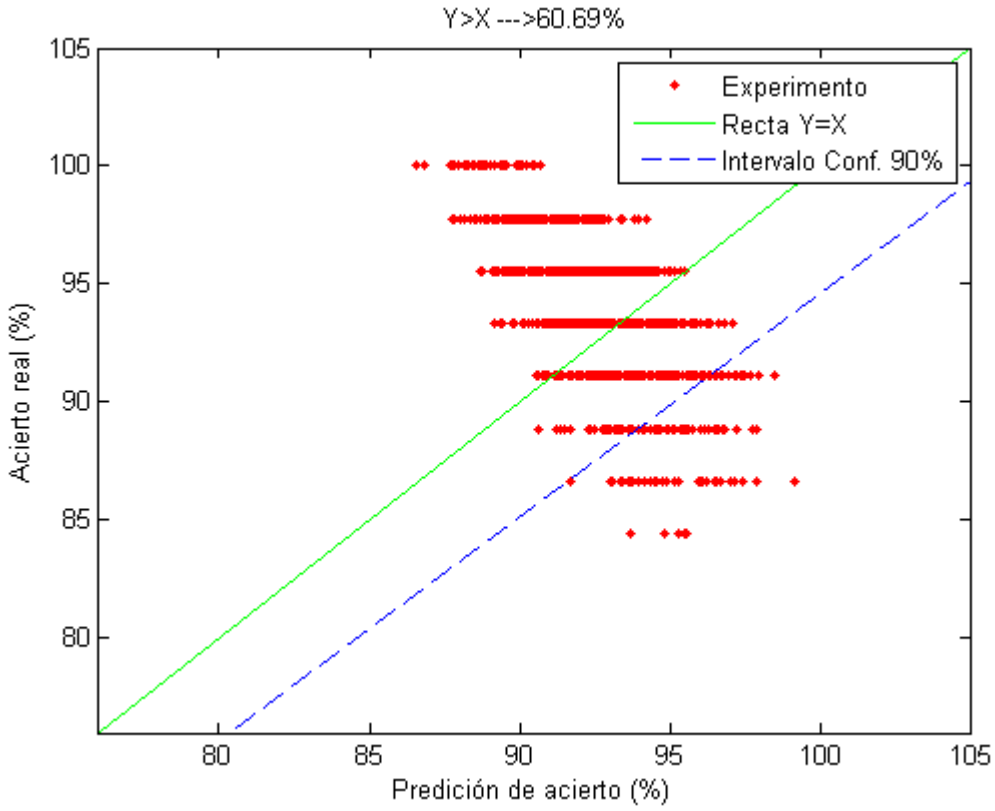

Figura 5.3.1 - Estimación del ratio de acierto real vs estimación para problema *Flor de Iris*.

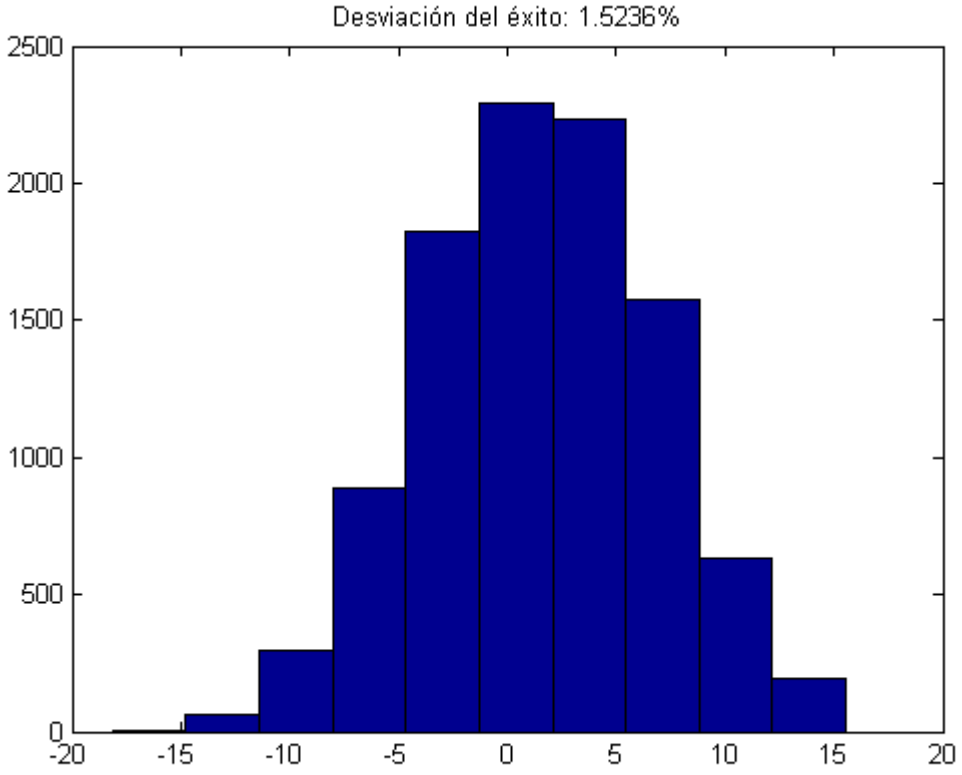

Figura 5.3.2 - Distribución de la desviación de la predicción del acierto para el problema *Flor de Iris*.

5.3.2 Cáncer de mama

En este caso la cantidad de predicciones pesimistas es de un 68% (*figura 5.3.3*). O sea, en el 68% de las veces el resultado real es mejor que el predicho. Al igual que en el caso anterior, se desea que este porcentaje sea lo más alto posible para poder acotar el ratio de acierto con una confianza suficientemente grande. Como este número no es lo suficientemente grande se buscará un intervalo de confianza del 90%. El percentil 10 resulta ser -5.78. Se puede ver la distribución de la desviación de la estimación del acierto en la *figura 5.3.4*.

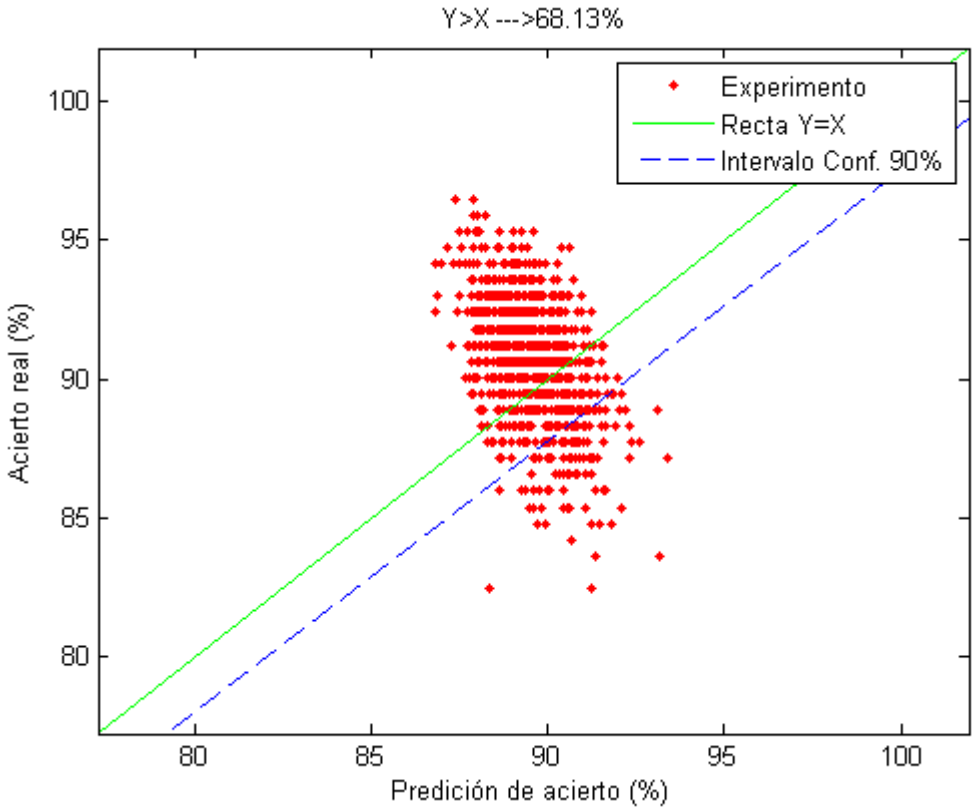

Figura 5.3.3 - Estimación del ratio de acierto real vs estimación para problema *Cáncer de Mama*.

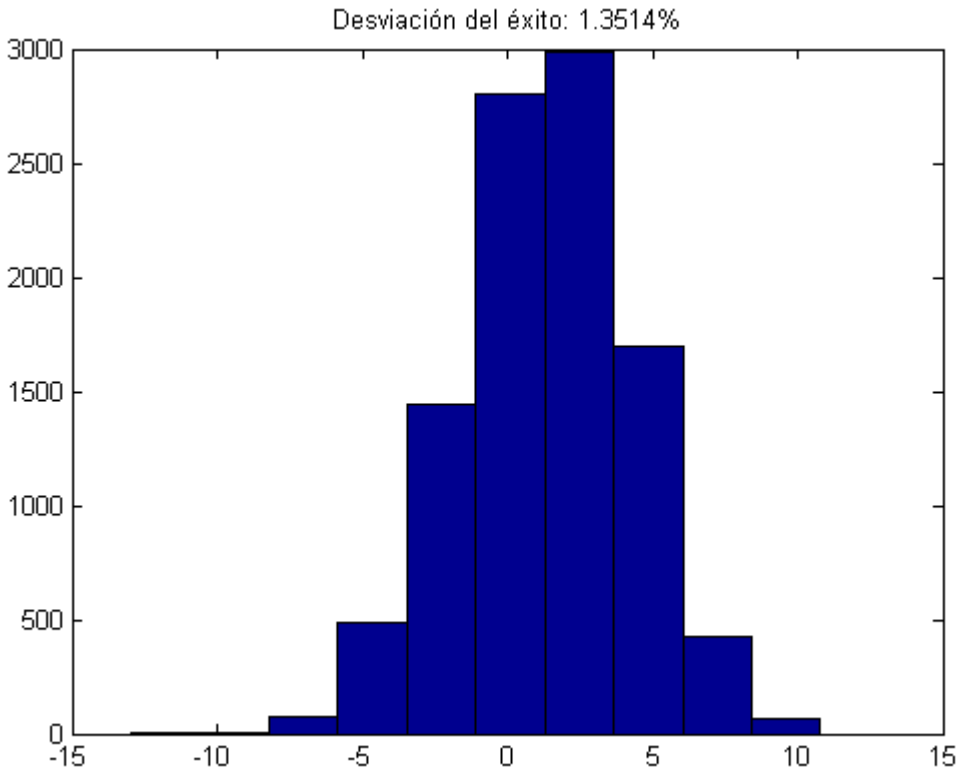

Figura 5.24 – Distribución de la desviación de la predicción del acierto para el problema *Cáncer de Mama*.

5.3.3 Cáncer de colon

En este caso la estimación es más conservadora que el resultado final en más del 90% de los casos. De acuerdo al criterio adoptado no es necesario calcular el intervalo de confianza del 90%, pues la línea que divide los experimentos cae por encima de la recta X=Y, tal como se muestra en la *figura 5.25*.

De acuerdo a los resultados obtenidos, la estimación del acierto será menor que resultado real con una seguridad del 96,88%, por lo que se puede acotar dicha predicción por abajo con una seguridad suficiente. Esto permite garantizar unos intervalos de confianza mayores que los obtenidos en los casos anteriores.

No se debe confundir estos intervalos de confianza con los resultados finales obtenidos. En efecto, estos resultados son peores que los que se consiguen para los problemas *Flor de Iris* y *Cáncer de Mama*. Lo positivo de estos resultados es que no habrá *penalización* en la estimación del ratio de acierto para conseguir el intervalo de confianza deseado que debe ser mayor o igual al 90%.

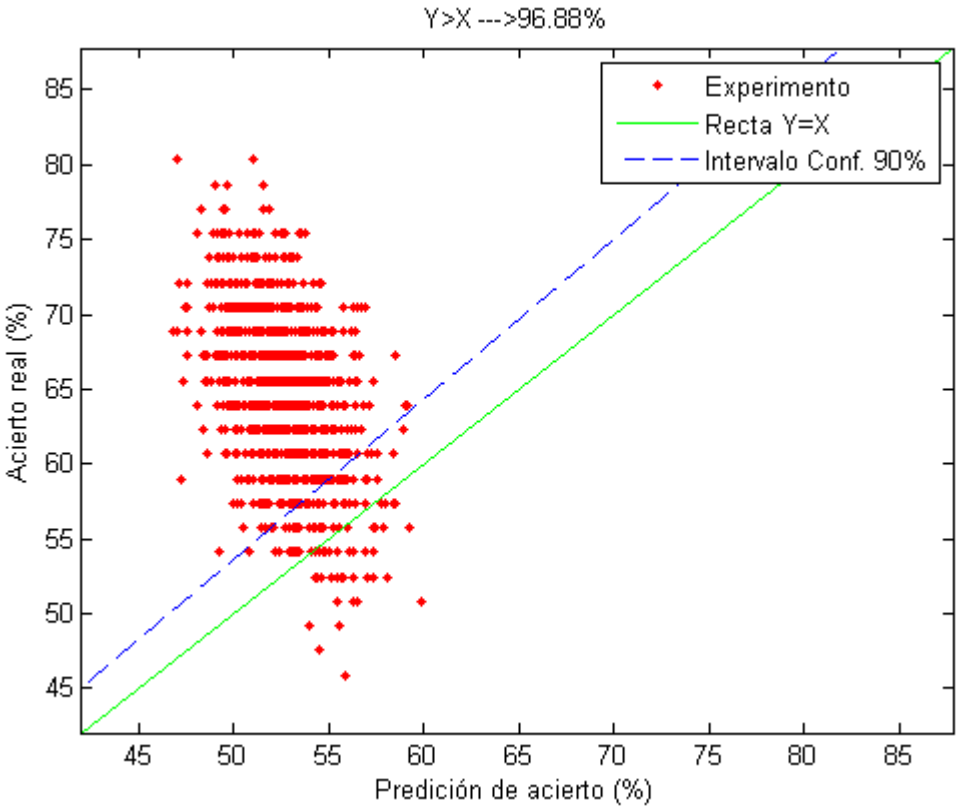

Figura 5.25 – Estimación del ratio de acierto real vs estimación para problema *Cáncer de Colon*.

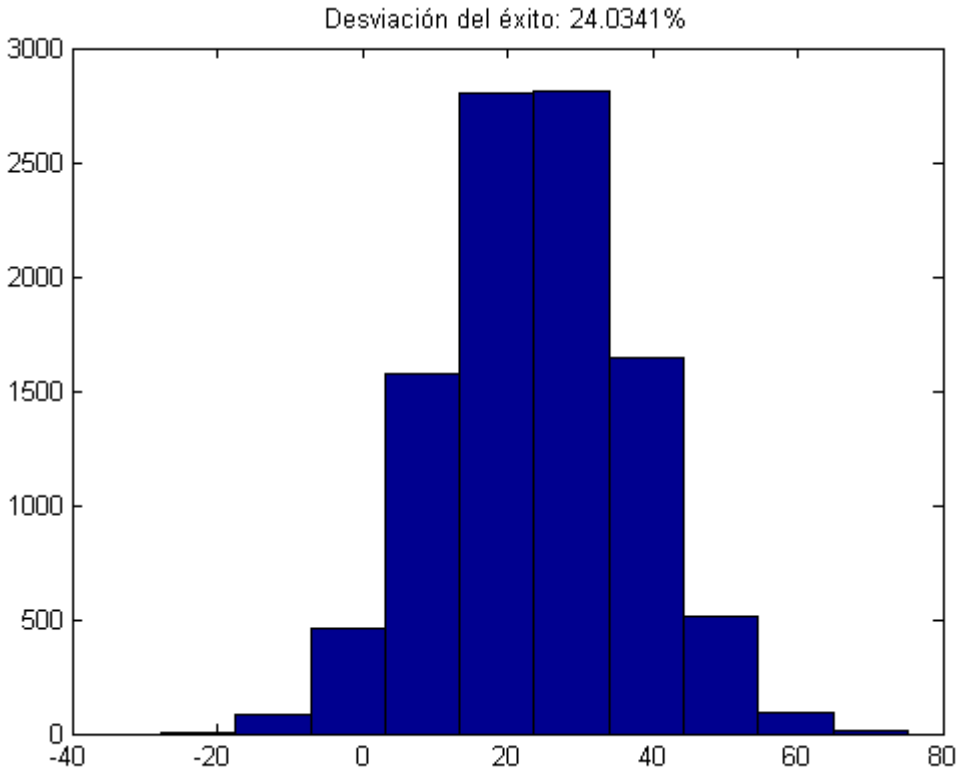

Figura 5.3.4 – Distribución de la desviación de la predicción del acierto para el problema *Cáncer de Colon*.

6 CONCLUSIONES

Este capítulo recoge un análisis de conjunto de los resultados mostrados en el capítulo anterior. Se hace una comparación entre los resultados conseguidos sin aplicar la técnica de *bagging*, y los conseguidos al aplicarla. También se comparan los resultados obtenidos utilizando la técnica de estimación del ratio de acierto en los tres problemas.

6.1 Impacto del uso de bagging

A continuación se hace una comparación de los resultados obtenidos únicamente con el *árbol de decisión* y aplicando la técnica de *bagging*.

El bagging suele mejorar los resultados de cualquier clasificador o, en el peor de los casos, mantenerlos. Como se ha visto en el *capítulo 5*, se cumple esta predicción: los resultados han mejorado en mayor o menor medida cuando se ha utilizado la técnica de *bagging* para ayudar al clasificador.

6.1.1 Flor de iris

Se consigue una ligera mejora tanto del ratio de éxito como de su desviación típica. Debido a que el problema se clasifica muy bien sin necesidad de bagging, el margen de mejora es menor.

	Sin bagging	Con bagging
% éxito	92,38	93,79
Desviación típica	3,50	3,23

Tabla 6.1.1 - Comparación de resultados para el problema *Flor de Iris*.

6.1.2 Cáncer de mama

Al igual que ocurre con en el caso anterior, los resultados mejoran ligeramente. Se observa que a pesar de clasificarse ligeramente peor que el problema *Flor de Iris*, se tiene una desviación típica menor para ambos casos. Esto puede deberse a que hay dos clases para clasificar en lugar de tres.

	Sin bagging		Con bagging	
	%	Desviación típica	%	Desviación típica
Éxito	89,16	2,66	90,70	2,05
Falso Positivo	6,53	2,76	7,73	2,86
Falso Negativo	9,92	4,10	4,00	2,62

Tabla 6.1.2 - Comparación de resultados para el problema *Cáncer de Mama*.

6.1.3 Cáncer de colon

Al obtenerse unos malos resultados al aplicar únicamente el árbol de decisión, era de esperar que éstos se mejoraran sustancialmente al utilizar la técnica de *bagging*. Efectivamente, el ratio de

acierto aumenta un 14% y la desviación típica baja más de un punto.

	Sin bagging		Con bagging	
	%	Desviación típica	%	Desviación típica
Éxito	51,28	6,68	65,22	5,35
Falso Positivo	26,41	9,99	31,02	9,46
Falso Negativo	23,48	8,84	6,19	4,81

Tabla 6.1.3 - Comparación de resultados para el problema *Cáncer de Colon*.

Tal como se esperaba, la desviación típica disminuye al aplicar el método de *bagging*. Al igual que pasa con el ratio de acierto, ésta se consigue reducir en mayor medida en problemas con un margen de mejora mayor.

6.1.4 Conclusión

Se observa que los problemas que tienen un ratio de acierto entorno al 90% a penas mejoran sus resultados al ser clasificados mediante *bagging*. Sin embargo, para el problema *Cáncer de Colon*, que ofrece unos resultados muy pobres, se consigue una mejora sustancial de los mismos.

En vista de estos resultados se deduce que el método de *bagging* es tanto mejor como margen de mejora hay. Sin embargo, al contarse únicamente con tres problemas de clasificación no se tiene una certeza suficiente como para generalizar esta conclusión a cualquier problema de clasificación.

6.2 Análisis de la estimación de ratio de aciertos mediante bootstrapping

Tanto si se consigue acotar directamente el ratio de acierto a partir de los resultados de la estimación, como si se necesita calcular un intervalo de confianza que obligue a reducir el valor de ella, se consigue siempre acotar la estimación del acierto con una seguridad 90%.

Este grado de seguridad podría haberse elegido mayor, dando lugar a un intervalo de confianza mayor y disminuyendo la pendiente de la recta. Esto hará que la penalización sobre la estimación crezca, por lo el valor de la estimación final disminuirá a su vez.

Una nube de puntos concentrada permitirá elegir intervalo de confianza mayores sin necesidad de aumentar la penalización demasiado. Es en problemas con una varianza grande donde habrá una penalización mayor para aumentar el intervalo de confianza. Como se ha explicado, el método de *bootstrapping* disminuye la varianza, por lo que esta penalización debe estar de por sí disminuida.

7 CÓDIGO

A continuación se muestra el código de las funciones y scripts descritos en el *capítulo 4*.

7.1 Script core.m

```
% CORE PROGRAM TO TEST RESULTS

clear all
clc

% TO BE MODIFIED
Num_Iter = 1;               % Number of experiments to be run on every (gam,sig2)
combination
Nb = 50;                        % Number of boorstrap samplings.
problem = 'colon';              % Type of problem: {iris, breast, colon}
method = 'bagging_estimation';  % Options are: {normal, bagging, bagging_estimation}


% CATEGORIES
if(strcmp(problem,'breast'))
    Malign  = 1;
    Benign  = 7;
    Class_A = [Malign];             % Class A will be assigned Y=1
    Class_B = [Benign];             % Class B will be assigned Y=-1
    Class_C = [];                   % Class C will be assigned Y=0;
elseif(strcmp(problem,'iris'))
    Setosa  =   1;
    Versicolor= 2;
    Virginica = 3;
    Class_A = [Versicolor];         % Class A will be assigned Y=1
    Class_B = [Virginica];          % Class B will be assigned Y=-1
    Class_C = [Setosa];             % Class C will be assigned Y=0
elseif(strcmp(problem,'colon'))
    Cancer = 1;
    Polyps = 2;
    Healthy= 7;
    Class_A = [Cancer];      % Class A will be assigned Y=1
    Class_B = [Healthy];            % Class B will be assigned Y=-1
    Class_C = [Polyps];                   % Class C will be assigned Y=0
end


All_Data = LoadAllData(problem);                        % Load original data to messure
its size, before it is shrinked by LoadData function
Error_Tracker = zeros(size(All_Data,1),1);              % Keeps track of how many time
a sample is redicted wrong
Training_Frequency = zeros(size(All_Data,1),1);         % Stores how many times a
sample falls into the validation set
```

```
All_Data = LoadAllData_KeepingTrack(problem);     % Loads dataset

S = zeros(Num_Iter,1);
C = zeros(Num_Iter,3);
for k=1:Num_Iter;

    disp(k);

    % Masks to modify ErrorTracker and Frequency arrays
    mask_error=zeros(length(Error_Tracker),1);
    mask_frequency =zeros(length(Training_Frequency),1);


    % Splitting data set into training and validation keeping track of the original
position of each sample
    SplitData;

    Yt = Y_Training(:,2);
    Yv = Y_Validation(:,2);
    Yv_tracking = Y_Validation(:,1);


    % APPLYING METHOD
    if strcmp(method,'normal')
        T = classregtree(X_Training,Yt);
        Y_predict = eval(T,X_Validation);
    elseif strcmp(method,'bagging')
        Y_predict = BaggingMethod(X_Training,Yt,X_Validation,Nb);
    elseif strcmp(method,'bagging_estimation')
        [Y_predict, s] =
BaggingEstimationMethod(X_Training,Y_Training,X_Validation,Nb);
        S(k) = s;
    end

    I_correct=(Y_predict==Yv);
% Indices of correct predictions
    I_incorrect=(I_correct==0);

    Nval=length(Yv);      % Size of the validation set

    Nv_A=sum(Yv==1);      % Number of Class_A samples
    Nv_B=sum(Yv==-1);     % Number of Class_B samples

    FalsePositive=sum((Yv==-1)&(Y_predict>=0));   % Number of false positives
    FalseNegative=sum((Yv==1)&(Y_predict<0));     % Number of false negatives

    EP_Correct_Val=100*(sum(I_correct)/Nval);   % Estimated Probability of correct
classification in Validation Set.
    EP_FP_Val=100*FalsePositive/Nv_B;           % Estimated Probability of False
Positive in validation set.
    EP_FN_Val=100*FalseNegative/Nv_A ;          % Estimated Probability of False
Negative in validation set.

    % Storing results in A
    C(k,1) = EP_Correct_Val;
    C(k,2) = EP_FP_Val;
```

```
    C(k,3) = EP_FN_Val;

    % Update tracking arrays
    mask_error(Yv_tracking(I_incorrect)) = 1;
    Error_Tracker = Error_Tracker + mask_error;
    mask_frequency(Yv_tracking) = 1;
    Training_Frequency = Training_Frequency + mask_frequency;

    close all
end


% Display tracking results on screen
Error_Tracker = 100 * (Error_Tracker./Training_Frequency);
disp('Error tracking:');
disp('   Position     %error');
disp([(1:length(Error_Tracker))' Error_Tracker]);

nBars = 10;

exito = mean(C(:,1));
fpositivo = mean(C(:,2));
fnegativo = mean(C(:,3));

figure(1)
hist(C(:,1),nBars);
title(['% Exito = ' num2str(exito)]);

figure(2)
hist(C(:,2),nBars);
title(['% Falsos positivos = ' num2str(fpositivo)]);

figure(3)
hist(C(:,3),nBars);
title(['Falsos negativos = ' num2str(fnegativo)]);


disp('RESULTS');
disp('-------------');
disp(['% exito = ' num2str(mean(C(:,1)))]);
disp(['falsos positivos = ' num2str(mean(C(:,2)))]);
disp(['falsos positivos = ' num2str(mean(C(:,3)))]);


% Scatter plot
    figure(4)
if(strcmp(problem,'breast'))
    IMalign = All_Data(:,2)== Malign;
    IBenign = All_Data(:,2) == Benign;
    %Ihealthy = All_Data(:,2) == Healthy;
    plot(All_Data(IMalign,1),Error_Tracker(IMalign),'*r' ,
All_Data(IBenign,1),Error_Tracker(IBenign),'*g');
    title('Repetición del error por muestra');
```

```
    legend('Maligno','Benigno');
end

if(strcmp(problem,'iris'))
    ISetosa = All_Data(:,2)== Setosa;
    IVersicolor = All_Data(:,2) == Versicolor;
    IVirginica = All_Data(:,2) == Virginica;
    plot(All_Data(ISetosa,1),Error_Tracker(ISetosa),'*r' ,
All_Data(IVersicolor,1),Error_Tracker(IVersicolor),'*m' ,
All_Data(IVirginica,1),Error_Tracker(IVirginica),'*g');
    title('Repetición del error por muestra');
    legend('Setosa','Versicolor','Virginica');
end

if(strcmp(problem,'colon'))
    ICancer = All_Data(:,2)== Cancer;
    IPol = All_Data(:,2) == Polyps;
    IHealthy = All_Data(:,2) == Healthy;
    plot(All_Data(ICancer,1),Error_Tracker(ICancer),'*r' ,
All_Data(IPol,1),Error_Tracker(IPol),'*m' ,
All_Data(IHealthy,1),Error_Tracker(IHealthy),'*g');
    title('Repetición del error por muestra');
    legend('Cáncer','Pólipo','Sano');
end


% Bootstrapping check plots
if(strcmp(method,'bagging_estimation'))
    pred_real = C(:,1);
    pred_est = S;
    YY=pred_real-pred_est; I_ok=YY>0; ratio=sum(I_ok)/length(I_ok);
    figure(5)
    a = 0.95*min([min(pred_est) min(pred_real)]);
    b = 1.05*max([max(pred_est) max(pred_real)]);
    plot(pred_est,pred_real,'.r' , [a b],[a b],'g');
    axis([a b a b]);
    xlabel('Predición de éxito (%)')
    ylabel('Éxito real (%)')
    title(['Y>X --->' num2str(100*ratio) '%']);

    figure(6)
    deviation = 100*( (pred_real./pred_est)-1 );
    hist(deviation)
    title(['Desviación del éxito: ' num2str(mean(deviation)) '%']);

    figure(7)
    p90 = prctile(deviation,10);
    x1 = a;  y1 = (1+(p90/100))*x1;
    x2 = b;  y2 = (1+(p90/100))*x2;
    plot(pred_est(1:1000),pred_real(1:1000),'.r' , [a b],[a b],'g' , [x1 x2],[y1
y2],'--b');
    axis([a b a b]);
    xlabel('Predición de acierto (%)')
    ylabel('Acierto real (%)')
    title(['Y>X --->' num2str(100*ratio) '%']);
    legend('Experimento' , 'Recta Y=X' , 'Intervalo Conf. 90%');
```

```
end
```

7.2 Script SplitData.m

```
% Splits dataset keeping the proportion of Class_A/Class_B in both training
% and validation set.
% The xls file must have been loaded before execution of this script

if(strcmp(problem,'breast'))
    Malign = 1;
    Benign = 7;
elseif(strcmp(problem,'iris'))
    Setosa  =   1;
    Versicolor= 2;
    Virginica = 3;
elseif(strcmp(problem,'colon'))
    Cancer = 1;
    Polyps = 2;
    Healthy = 7;
end


N_Split = 0.7;  % ratio Training/Total_Samples

N_Samples=size(All_Data,1);         % Total number of patients

NA = size(Class_A,2);
NB = size(Class_B,2);
NC = size(Class_C,2);

% Computing Index for Class_A
I_A = zeros(N_Samples,1);
for m=1:NA
    I_A = I_A + (All_Data(:,2)==Class_A(m));
end

% Computing Index for Class_B
I_B = zeros(N_Samples,1);
for m=1:NB
    I_B = I_B + (All_Data(:,2)==Class_B(m));
end

% Computing Index for Class_C
I_C = zeros(N_Samples,1);
for m=1:NC
    I_C = I_C + (All_Data(:,2)==Class_C(m));
end

% Turning index arrays into logical arrays
```

```
I_A = I_A==1;
I_B = I_B==1;
I_C = I_C==1;

% Adapting outputs to SVM-friendly values {1,-1,0}
Y = I_A-I_B;

% Amount of samples for each class
N_Class_A = sum(I_A);
N_Class_B = sum(I_B);
N_Class_C = sum(I_C);

% Splitting data into classes A and B
All_Data_A = All_Data(I_A,:);
All_Data_B = All_Data(I_B,:);
All_Data_C = All_Data(I_C,:);
All_Data_ABC = [All_Data_A ; All_Data_B ; All_Data_C];

% Creating random indices
shuffled_a = randperm(N_Class_A);     % Random index to pick Class_A samples for the
training and validation sets
shuffled_b = randperm(N_Class_B);     % Random index to pick Class_B samples for the
training and validation sets
shuffled_c = randperm(N_Class_C);     % Random index to pick Class_C samples for the
training and validation sets

% Indices that will split Class_A samples into training and validation
I_a_t  = shuffled_a(1 : round(N_Split*length(shuffled_a)));
I_a_v = shuffled_a(length(I_a_t)+1 : N_Class_A);

% Indices that will split Class_B samples into training and validation
I_b_t = shuffled_b(1 : round(N_Split*length(shuffled_b)));
I_b_v = shuffled_b(length(I_b_t)+1 : N_Class_B);

% Indices that will split Class_C samples into training and validation
I_c_t = shuffled_c(1 : round(N_Split*length(shuffled_c)));
I_c_v = shuffled_c(length(I_c_t)+1 : N_Class_C);

%Creating training and validation sets
X_Training =   [All_Data_A(I_a_t,3:end) ; All_Data_B(I_b_t,3:end) ;
All_Data_C(I_c_t,3:end)];
X_Validation = [All_Data_A(I_a_v,3:end) ; All_Data_B(I_b_v,3:end) ;
All_Data_C(I_c_v,3:end)];

Y_Training =   [All_Data_A(I_a_t,[1 2]) ; All_Data_B(I_b_t,[1 2]) ;
All_Data_C(I_c_t,[1 2])];
Y_Validation = [All_Data_A(I_a_v,[1 2]) ; All_Data_B(I_b_v,[1 2]) ;
All_Data_C(I_c_v,[1 2])];



Yt_original = Y_Training;
Yv_original = Y_Validation;

I_YA = zeros(length(Y_Training),1);
I_YB = zeros(length(Y_Training),1);
```

```
I_YC = zeros(length(Y_Training),1);
for m=1:NA
    I_YA = I_YA + (Y_Training(:,2)==Class_A(m));
end
for m=1:NB
    I_YB = I_YB + (Y_Training(:,2)==Class_B(m));
end
for m=1:NC
    I_YC = I_YC + (Y_Training(:,2)==Class_C(m));
end
I_YA = I_YA==1;
I_YB = I_YB==1;
I_YC = I_YC==1;
Y_Training(I_YA,2)= 1;
Y_Training(I_YB,2)=-1;
Y_Training(I_YC,2)= 0;


I_YA = zeros(length(Y_Validation),1);
I_YB = zeros(length(Y_Validation),1);
I_YC = zeros(length(Y_Validation),1);
for m=1:NA
    I_YA = I_YA + (Y_Validation(:,2)==Class_A(m));
end
for m=1:NB
    I_YB = I_YB + (Y_Validation(:,2)==Class_B(m));
end
for m=1:NC
    I_YC = I_YC + (Y_Validation(:,2)==Class_C(m));
end
I_YA = I_YA==1;
I_YB = I_YB==1;
I_YC = I_YC==1;
Y_Validation(I_YA,2)= 1;
Y_Validation(I_YB,2)=-1;
Y_Validation(I_YC,2)= 0;
```

7.3 Función LoadAllData.m

```
function ALL_DATA=LoadAllData(problem)

if(strcmp(problem,'breast'))
    ALL_DATA = xlsread('data_breast.xlsx','alldata','B2:AG570');
    %Sort samples by class to have a clear scatter plot
    I1 = ALL_DATA(:,1)==1;
    I7 = ALL_DATA(:,1)==7;
    ALL_DATA = [ALL_DATA(I1,:) ; ALL_DATA(I7,:)];

elseif(strcmp(problem,'iris'))
```

```
    ALL_DATA = xlsread('data_iris.xlsx','alldata','B2:F151');
elseif(strcmp(problem,'colon'))
    ALL_DATA = xlsread('data_colon.xlsx','Hoja1','B2:BB203');
end

end
```

7.4 Función LoadAllData_KeepingTrack

```
function [ALL_DATA]=LoadAllData_KeepingTrack(problem)
% Reads dataset and stores its content in ALL_DATA.
% Also, adds an extra column with the position of every sample so it can be
% tracked throughout the test.

if(strcmp(problem,'breast'))
    ALL_DATA = xlsread('data_breast.xlsx','alldata','B2:AG570');
    % Sort samples by class to have a clear scatter plot
    I1 = ALL_DATA(:,1)==1;
    I7 = ALL_DATA(:,1)==7;
    ALL_DATA = [ALL_DATA(I1,:) ; ALL_DATA(I7,:)];

elseif(strcmp(problem,'iris'))
    ALL_DATA = xlsread('data_iris.xlsx','alldata','B2:F151');

elseif(strcmp(problem,'colon'))
    ALL_DATA = xlsread('data_colon.xlsx','Hoja1','B2:BB203');
end

ALL_DATA = [(1:size(ALL_DATA,1))' ALL_DATA];
```

7.5 Función BaggingMethod

```
function Ypred = BaggingMethod(Xt,Yt,Xv,B)

Ypred = zeros(size(Xv,1),1);
for i=1:B
   [X,Y] = BootStrapping_sample(Xt,Yt); % Sampling with replacement
   T = classregtree(X,Y);               % Trains the tree
   Ypred = Ypred + eval(T,Xv);          % Acumulated prediction
end

% Adapting results
Ypred = Ypred/B;
I1 = Ypred > 1/3;
I2 = Ypred <-1/3;
I12 = I1+I2;
```

```
I3 = ~I12;

Ypred(I1)=1;
Ypred(I2)=-1;
Ypred(I3)=0;

end
```

7.6 Función BaggingEstimationMethod

```
function [ Ypred, S ]= BaggingEstimationMethod( X_Training,Y_Training,X_Validation,B )
% Will provide with an estimation of the accuracy of the prediction based on
% bootstrapping method.

success_r = zeros(B,1);
Ypred = zeros(size(X_Validation,1),1);
for i=1:B
    % Sampling wo/ replacement
    [X,Y] = BootStrapping_sample(X_Training, Y_Training);
    % Remaining samples that didn't fall into bootstrapping sample set
    [Xr,Yr] = BootStrapping_remaining(X_Training, Y_Training, Y);

    T = classregtree(X,Y(:,2)); % Training the tree

    Ypred = Ypred + eval(T,X_Validation);   % Prediction for  bootstrapped samples
    Ypred_r = eval(T,Xr);      % Prediction for samples that were not bootstrapped

    Ir_success = (Ypred_r==Yr(:,2));        % Index of samples predicted successfully
    % Index successfully predicted non-bootstrapped samples
    success_r(i) = 100*(sum(Ir_success)/length(Ypred_r));
end

% Adding up predictions, computing mean, adapting outcome to fixed values {-1 0 1}
S = mean(success_r);
Ypred = Ypred/B;
I1 = Ypred > 1/3;
I2 = Ypred <-1/3;
I12 = I1+I2;
I3 = ~I12;

Ypred(I1) = 1;
Ypred(I2) =-1;
Ypred(I3) = 0;

end
```

7.7 Función BootStrapping_sample

```
function [ Xsample , Ysample ] = BootStrapping_sample( X, Y )
% Ramdomly samples training data wo/ replacement to apply bootstrapping.
N = size(X,1);
i = 1:N;
I=datasample(i,N);

Xsample = X(I,:);
Ysample = Y(I,:);

end
```

7.8 Función BootStrapping_remaining

```
function [Xr,Yr] = BootStrapping_remaining(X_Training,Y_Training,Y)
% Will compute the samples that haven't been selected in the bootstrapping sampling

all = Y_Training(:,1);
selected = Y(:,1);

Xr=[];
Yr=[];

for i=1:length(all)
    if( sum(find(all(i)==selected)) == 0)
        Yr = [Yr ; Y_Training(i,:)];
        Xr = [Xr ; X_Training(i,:)];
    end
end

end
```

BIBLIOGRAFÍA

- Pang-Ning Tan, Michael Steinbach, Vipin Kumar. *Introduction to Data Mining*. 2015.
- Oded Maimon, Lior Rokach. *Data Mining and Knowledge Discovery*. 2010.
- Leo Breiman. *Bagging Predictors*. 1994.
- Alex Smola,S.V.N. Vishwanathan. *Introduction to machine learning*. 2008.
- Ian H. Witten, Eibe Frank, Mark A. Hall. Data Mining: Practical Machine Learning Tools and Techniques, Third Edition. 2011.
- Decision Tree Learning (15 Mayo 2015). Wikipedia, la enciclopedia libre. Consultado 19 Mayo 2015. http://www.riate.org/version/v1/recursos/cursolicenciasnavegable/cmo_citar_la_wikipedia.html
- Bootstrap aggregating (1 Junio 2015). Wikipedia, la enciclopedia libre. Consultado 3 Junio 2015. https://en.wikipedia.org/wiki/Bootstrap_aggregating
- Statistical Classification (13 Junio). Wikipedia, la enciclopedia libre. Consultado 16 Junio 2015. https://en.wikipedia.org/wiki/Statistical_classification
- Set de datos de *Flor de Iris*: UCI Machine Learning Repository. Recuperado 19 Abril 2015. https://archive.ics.uci.edu/ml/datasets/Iris
- Set de datos *Cáncer de Mama*: UCI Machine Learning Repositoyr. Recuperado 21 Abril 2015. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)