

BEYOND TRADITIONAL FRAMEWORKS FOR MODELLING LEARNING DOMAINS

By

Sergi Sancho-Asensio

UNIVERSIDAD RAMON LLULL
GRUADO EN INGENIERIA MULTIMEDIA
DEPARTAMENTO DE INFORMÁTICA

Supervisor:
XAVIER SOLÉ BETETA

MARZO 2015

laSalle

Universitat Ramon Llull

Abstract

Information and communications technology (ICT) is becoming increasingly important in the educational sector. These technologies allow process automation to take advantage of existing computational resources, improving teaching experience. In this context, Machine Learning provides an added value because it is capable of automatically modelling complex problems that used to be solved using traditional methods.

The purpose of this thesis is to research the state of the art in Machine Learning techniques to help teachers model and predict student performance. More specifically, by using Gradient Boosting, a highly competent Ensemble method.

Keywords

Teaching, Artificial Intelligence, Machine Learning, Supervised Learning, Ensemble Method, Gradient Boosting, Cross Validation, Regression, Classification, Educational Data Mining (EDM).

Abstracto

El mundo de las tecnologías de la información resulta cada vez más importantes en el ámbito educativo. Éstas permiten automatizar procesos aprovechando los recursos computacionales existentes mejorando la experiencia de los docentes. Es aquí donde el Aprendizaje Automático aporta un valor añadido puesto que es capaz de modelar problemas complejos de manera automática que tradicionalmente se resolvían de forma artesanal. El propósito del Trabajo Final de Grado es investigar el estado del arte en técnicas de Aprendizaje Automático para ayudar al cuerpo docente a modelar y predecir el rendimiento de los alumnos. Más específicamente, empleando métodos *Ensemble* como es el caso del *Gradient Boosting*.

Palabras clave

Docencia, Inteligencia artificial, Aprendizaje Automático, Aprendizaje Supervisado, método Ensemble, *Gradient Boosting*, *Cross Validation*, regresión, clasificación, *Educational Data Mining (EDM)*.

Abstracte

El món de les tecnologies de la informació resulta cada vegada més important en l'àmbit educatiu. Aquestes permeten automatitzar processos aprofitant els recursos computacionals existents millorant l'experiència dels docents. És aquí on l'Aprenentatge Automàtic aporta un valor afegit degut a que és capaç de modelar problemes complexos de manera automàtica que tradicionalment es resolien de forma artesanal. El propòsit d'aquest Treball Final de Grau, és investigar l'estat de l'art en tècniques d'Aprenentatge Automàtic per ajudar al cos docent a modelar i predir el rendiment dels alumnes. Més específicament, emprant mètodes *Ensemble* com és el cas del *Gradient Boosting*.

Paraules clau

Docència, Intel·ligència artificial, Aprenentatge Automàtic, Aprenentatge Supervisat, Mètode *Ensemble*, *Gradient Boosting*, *Cross Validation*, regressió, classificació, *Educational Data Mining (EDM)*.

Índice general

Abstract	III
Abstracto	V
Abstracte	VII
Índice de figuras	XIII
Índice de tablas	XVII
1. Introducción	1
1.1. Marco	1
1.2. Objetivos	2
1.3. Estructura del documento	3
2. Aprendizaje Automático	5
2.1. Contexto	5
2.2. Notación	6
2.3. Función f	7
2.4. Inferencia	7
2.5. Predicción	8
2.6. Estimación \hat{f}	9
2.7. Paradigmas	11
2.8. Resumen	11
3. Tipos de modelos y clasificadores avanzados	13
3.1. Sistemas clasificadores	13
3.1.1. Aprendizaje Perezoso	14
3.1.2. Redes neuronales	15
3.1.3. Modelos probabilísticos	17
3.1.4. Máquinas de Soporte Vectorial	20
3.1.5. Árboles de decisión	23
3.2. Técnicas avanzadas de clasificación	30

3.2.1. Bagging	31
3.2.2. Boosting	33
3.3. Resumen	37
4. Desarrollo del sistema	39
4.1. Especificación de requisitos <i>software</i>	39
4.1.1. Propósito	40
4.1.2. Enfoque	40
4.1.3. Estructura de la <i>ERS</i>	40
4.2. Descripción general	40
4.2.1. Perspectiva del producto	40
4.2.2. Interfaces	41
4.2.3. Requisitos de rendimiento	41
4.2.4. Funcionalidades del producto	41
4.2.5. Restricciones de diseño, asunciones y dependencias	43
4.2.6. Trabajo adicional	43
4.3. Requisitos específicos	43
4.3.1. Requisitos funcionales	43
4.3.2. Usabilidad	45
4.3.3. Rendimiento	45
4.4. Metodología	45
4.5. Análisis y diseño	47
4.6. Resumen	53
5. Experimentación	55
5.1. Validación cruzada	55
5.2. Validación estadística de los resultados	56
5.3. Experimentación	58
5.3.1. Configuración <i>GBM</i>	58
5.3.2. Comparativa	59
5.4. Resumen	62
6. <i>GBM</i> aplicado a datos docentes	65
6.1. Procesado de los datos	65
6.2. Notas previas a la experimentación	68
6.3. Primer experimento	68
6.3.1. Clase a predecir	68
6.3.2. Análisis de los resultados	70
6.3.3. Modelado del alumnado	72
6.4. Segundo experimento	78
6.4.1. Clase a predecir	78

6.4.2. Análisis de los resultados	78
6.5. Extrapolación al mundo real	79
6.6. Resumen	79
7. Costes temporales	81
7.1. Resumen de los objetivos	81
7.2. Coste en horas	81
8. Resumen y conclusiones	83
8.1. Resumen del Trabajo Final de Grado	83
8.2. Conclusiones	86
9. Líneas de futuro	91
A. Software externo: guía de referencia	93
A.1. Ficheros <i>ARFF</i>	93
A.2. Manual de <i>Weka</i>	94
A.3. Manual de <i>PGUI</i> y <i>ModelToGpv</i>	96
A.3.1. <i>PGUI</i>	96
A.3.2. <i>ModelToGpv</i>	97
A.4. Optimización y depuración del código	98
A.5. <i>Doxygen</i>	105
Bibliografía	107

Índice de figuras

2.1. Ciclo de vida <i>KDD</i> según Fayyad et al. [1996]	6
2.2. <i>Microarray</i> original	8
2.3. <i>Microarray</i> de predicción	9
2.4. Gráfica que representa dos tipos de clasificadores y dos tipos de respuesta [Icke, 2008].	10
2.5. Relación error/tiempo en un caso de sobreajuste	10
3.1. Red neuronal de un sólo perceptrón	15
3.2. <i>Data set</i> de entrada para aplicar el modelo probabilístico <i>Naive Bayes</i>	18
3.3. Modelo probabilístico <i>Naive Bayes</i> con un nuevo objeto para clasificar.	18
3.4. SVM problema linealmente separable	21
3.5. SVM problema no linealmente separable	21
3.6. Representación gráfica del <i>data set</i> de ejemplo en función de los atributos X_1 y X_2	24
3.7. Representación gráfica de la partición del <i>data set</i> de ejemplo según la ganancia en varianza del atributo X_1	25
3.8. Representación gráfica de la partición del <i>data set</i> de ejemplo según la ganancia en varianza del atributo X_2	25
3.9. Primera etapa del árbol generado. Se trata de un árbol con profundidad 1 y con un nodo hoja puro (sin varianza). El valor de la partición es 2.5 y 5/3 es un valor numérico temporal e implica que de los cinco casos posibles quedan tres por clasificar.	26
3.10. Segunda etapa (y final) del árbol generado. El árbol es puro, es decir, el acierto es del 100 % y ha descartado por completo el segundo atributo, significa que éste no aporta nada.	26
3.11. Árbol binario de regresión que modela el <i>data set</i> <i>Iris</i>	28
3.12. Árbol binario de regresión, lectura <i>depth first</i>	28
3.13. Árbol binario de regresión, lectura <i>bread first</i>	28
3.14. Método <i>Ensemble</i> [Polikar, 2008]. El <i>data set</i> tiene dos atributos distintos y el sistema utiliza tres sistemas clasificadores para modelar la información.	30

3.15. Método <i>Bagging</i>	31
3.16. <i>Kinect</i> clasifica las partes del cuerpo mediante el algoritmo <i>Random Forest</i> [Shotton et al., 2013].	32
3.17. Muestra el entrenamiento y predicción de usuarios sintéticos a la izquierda y las predicciones con usuarios reales a la derecha [Shotton et al., 2013].	32
4.1. Modelo en espiral de Boehm [1986].	46
4.2. Diagrama de casos de uso <i>UML</i> del <i>software GBM</i>	48
4.3. Diagrama de casos de uso <i>UML</i> del <i>software PGUI</i>	48
4.4. Diagrama de actividades <i>UML</i> del <i>software GBM</i>	49
4.5. Diagrama de clases <i>UML</i> simplificado de <i>GBM</i>	50
4.6. Clase <i>Config</i>	50
4.7. Clase <i>GradientBoosting</i>	51
4.8. Clase <i>BinaryTree</i>	52
4.9. Clase <i>DataSet</i>	52
4.10. Clase <i>NominalFeature</i>	53
6.1. Diagrama de barras de la clase nominal del <i>data set</i> procesado. El color cyan representa el número de alumnos suspendidos, estos son el 57,33 % de la clase. El color azul marino simboliza los alumnos que han aprobado la asignatura con un suficiente (un total de 12,67 %). El color rojo indica el número de alumnos que han sacado un bien (estos son el 14,67 % de la clase), el color verde apagado representa los alumnos que han sacado un notable de nota final (el 14 %) y por último el color rosa son aquellos alumnos que tienen un excelente de nota final (el 1,33 %).	67
6.2. Strong learner generado para la clase SUSPENDIDO.	73
6.3. Strong learner generado para la clase SUFICIENTE.	74
6.4. Strong learner generado para la clase BIEN.	75
6.5. Strong learner generado para la clase NOTABLE.	76
6.6. Strong learner generado para la clase EXCELENTE.	77
7.1. Desglose del tiempo dedicado en el presente Trabajo Final de Grado. . .	82
A.1. Pestaña Explorer en <i>Weka</i> . Se ha aplicado un <i>10 fold cross-validation</i> con un <i>SMO</i> configurado con un <i>Kernel</i> polimómico de primer grado. El promedio de las medidas <i>F Measure (F1)</i> ha sido 72,80 %.	95
A.2. Plataforma <i>PGUI</i> , muestra la configuración y el output de <i>GBM</i> en utilizado en la sección 6.3.2.	96
A.3. Árbol generado por el código de <i>output.txt</i>	98
A.4. Proceso de compilación en <i>C++</i>	99
A.5. Error al ejecutar el algoritmo.	101
A.6. Uso de Backtrace.	102

A.7. Frame 1.	102
A.8. Comando <i>break 199</i>	103
A.9. <i>GDB</i> indica que la variable <i>rightChild</i> no está declarada.	103
A.10. Fichero <i>analysis.txt</i>	104
A.11. Fragmento de la documentación generada por <i>Doxygen</i>	106

Índice de tablas

3.1.	<i>Data set</i> de ejemplo con cinco instancias. Está formado por los atributos X_1 , X_2 y la clase Y	23
3.2.	Este <i>data set</i> está formado por los atributos X_1 , X_2 y la clase Y . Se han ordenado las cinco instancias según el atributo X_2	25
3.3.	Primera partición del <i>data set</i> , no hay impureza. Por consiguiente, el nodo que se generará será directamente un nodo hoja con una predicción del 100 %.	26
3.4.	Segunda partición del <i>data set</i> , en este caso hay impureza.	26
3.5.	Fragmento del <i>data set Iris</i> formado por cuatro instancias. Este <i>data set</i> está formado por los atributos <i>Sepal length</i> , <i>Sepal width</i> , <i>Petal length</i> , <i>Petal width</i> y la clase (<i>Iris-Setosa</i> , <i>Iris-Versicolor</i> y <i>Iris-Virginica</i>). La columna residuo mapea cada una de las clases.	34
3.6.	Matriz de confusión, fragmento del <i>data set</i> de <i>Iris</i> con 100 instancias. Cada columna representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.	36
4.1.	Parámetros de configuración de <i>GBM</i> . Los valores por defecto de cada parámetro se hallan en la tabla 4.2.	42
4.2.	Parámetros de configuración por defecto de <i>GBM</i>	44
5.1.	Test de <i>Friedman</i> , primer paso. Muestra la puntuación F de una serie de algoritmos (<i>GBM</i> , <i>IBK</i> , <i>SMO</i> , <i>J48</i> y <i>NB</i>) aplicados en tres <i>data sets</i> distintos (<i>Auto Imports Database</i> , <i>BUPA Liver Disorders</i> y <i>Glass Identification Database</i>).	57
5.2.	Test de <i>Friedman</i> , segundo paso. Muestra el promedio de los rankings de las técnicas (<i>GBM</i> , <i>IBK</i> , <i>SMO</i> , <i>J48</i> y <i>NB</i>) y su ranking definitivo.	57
5.3.	Resumen de las características de los <i>data sets</i> utilizados. Muestra el nombre del <i>data set</i> , número de instancias, número de atributos y número de clases.	60
5.4.	Configuración de <i>GBM</i> para que el sistema sea más preciso en general.	60

5.5. Comparativa de algoritmos. Muestra los <i>data sets</i> en los que se hace la comparación y los resultados de la puntuación F en los algoritmos <i>GBM</i> , <i>IBK</i> , <i>SMO</i> , <i>J48</i> y <i>NB</i>	61
5.6. Clasificación de los algoritmos. Muestra tres atributos: algoritmo, <i>ranking</i> de <i>Friedman</i> y <i>ranking</i> global. Los <i>rankings</i> cuanto más bajo sean mejor.	62
5.7. Test de <i>Holm</i> para $\alpha = 0,05$	62
6.1. Atributos del <i>data set</i> original. Se trata de un <i>data set</i> real con las notas del alumnado en una asignatura. Muestra los atributos del <i>data set</i> y su rango de valores.	66
6.2. Se añade grandularidad al sistema mediante <i>dummies</i> , estos mapean las notas <i>NP</i> . Se amplía el <i>data set</i> con los atributos <i>NP EC1</i> , <i>NP EC2</i> , <i>NP Midterm</i> , <i>NP EC3</i> y <i>NP Examen Febrero</i>	66
6.3. Una vez se han añadido los <i>dummies</i> , se les da valor cero si no hay <i>NP</i> y la unidad en caso contrario. Posteriormente se substituyen los valores <i>NP</i> por ceros en las notas.	67
6.4. Criterios de evaluación para definir la clase. Muestra la clase <i>SUSPENDIDO</i> , <i>SUFICIENTE</i> , <i>BIEN</i> , <i>NOTABLE</i> , <i>EXCELENTE</i> y su relación con la nota final.	69
6.5. <i>Data set</i> procesado. Muestra los atributos del <i>data set</i> procesado y su rango de valores. Se ha eliminado el atributo <i>Expediente</i> , se ha modificado el rango de valores de cada atributo, se ha añadido la clase (la nota final de la asignatura) y por último los atributos que mapean los valores <i>NP</i> (<i>dummies</i>).	70
6.6. Configuración <i>GBM</i> para modelar el rendimiento del alumnado.	70
6.7. Comparativa entre <i>GBM</i> , <i>IBK</i> , <i>SMO</i> , <i>J48</i> y <i>NB</i> en el <i>data set</i> docente con y sin <i>dummies</i>	71
6.8. <i>Data set</i> procesado. Muestra los atributos del <i>data set</i> procesado y su rango de valores. Se ha eliminado el atributo <i>Expediente</i> , se ha modificado el rango de valores de cada atributo, se ha añadido la clase (la nota del examen de febrero) y por último los atributos que mapean los valores <i>NP</i> (<i>dummies</i>).	78
6.9. Comparativa de precisiones según la medida F de los distintos algoritmos (<i>GBM</i> , <i>IBK</i> , <i>SMO</i> , <i>J48</i> y <i>NB</i>) en el segundo experimento.	79
7.1. Coste en horas del trabajo. Muestra la fase, el tiempo dedicado en horas y el porcentaje del proyecto en tanto por ciento.	82
8.1. Análisis <i>DAFO</i> de <i>Gradient Boosting</i>	88

Índice de algoritmos

1.	Construcción de un árbol binario de regresión	29
2.	Generación del modelo	29
3.	Fase de entrenamiento de <i>Gradient Boosting</i>	35
4.	Fase de predicción de <i>Gradient Boosting</i>	35

1

Introducción

En el mundo actual las tecnologías de la información resultan cada vez más importantes a la hora de realizar tareas que tradicionalmente las llevaba a cabo un experto de forma tradicional. La manera tradicional de realizar las tareas posee el inconveniente de que el experto tiene que invertir demasiado tiempo y esfuerzo mientras que resulta un trabajo laborioso además de que está altamente influenciado por el criterio subjetivo de éste. La explotación de los recursos computacionales proporciona a las tecnologías de la información una manera de paliar los inconvenientes mencionados. Un entorno donde estas pueden aplicarse y aportar un valor añadido es el campo de la docencia.

1.1. Marco

Gracias a los nuevos avances en el diseño de los procesadores y las altas prestaciones ofrecidas por éstos, en las últimas décadas se ha incrementado el interés en el Aprendizaje Automático (*Machine Learning*) [Mitchell, 1997]. Éste es un campo de la inteligencia artificial cuyo objetivo es desarrollar algoritmos que permitan dotar a las máquinas con la capacidad de “aprender” modelos computacionales tales que, de forma automática, les permitan resolver nuevos problemas o mejorar las soluciones propuestas por técnicas tradicionales. Para llevarlo a cabo, esta disciplina de la computación estudia la creación de programas capaces de generalizar comportamientos a partir de

una información que en ocasiones no está estructurada y que se suministra en forma de ejemplos y/o hechos.

De este modo el Aprendizaje Automático resulta especialmente atractivo en aplicaciones que (1) son demasiado complejas como para que un experto humano las diseñe e implemente manualmente, (2) requieren que el software se mejore o refina a sí mismo continuamente. Por lo tanto, el objetivo final del Aprendizaje Automático es solucionar problemas que son demasiado complejos para solucionar o dar instrucciones acerca de cómo solucionarlos para los seres humanos [Orriols-Puig, 2008].

Existe un gran número de familias de técnicas dentro del campo del Aprendizaje Automático. De éstas, la que ha atraído la atención de los investigadores recientemente debido a su elevada tasa de acierto se halla en los métodos *Ensemble* [Goldbloom, 2010]. Estos agrupan varios sistemas inteligentes llamados clasificadores en uno sólo. Más concretamente, se quiere investigar la aplicación de los sistemas *Gradient Boosting* [Friedman, 2000], el funcionamiento de los cuales se basa en la mejora continua.

En este contexto, el presente Trabajo Final de Grado se centra en estudiar las principales técnicas del Aprendizaje Automático, diseñar e implementar un *Gradient Boosting*, comparar los resultados con otras técnicas y aplicar a datos procedentes del campo de la docencia.

1.2. Objetivos

La meta de este Trabajo Final de Grado consiste en revisar el estado del arte en técnicas de Aprendizaje Automático centrándose en la familia de los métodos *Ensemble* con la finalidad de ayudar al cuerpo docente a modelar y predecir el rendimiento de los alumnos. A partir de este objetivo global, se establecen los siguientes objetivos individuales.

Revisar las principales técnicas del Aprendizaje Automático. El Aprendizaje Automático es el proceso por el cual automáticamente se puede extraer y descubrir nuevos conocimientos útiles implícitos en grandes volúmenes de datos [Gama, 2010, García-Piquer, 2012]. Para ello será necesario introducir su fundamento teórico, revisar diferentes tipos de modelos y posteriormente detallar técnicas avanzadas de clasificación con el designio de hallar la más adecuada para el propósito planteado.

Desarrollar un algoritmo *Gradient Boosting* y compararlo con otras técnicas. De las distintas técnicas estudiadas, destaca *Gradient Boosting*. Para las plataformas *GBM* (implementación del *Gradient Boosting* con árboles de regresión) y *PGUI* (interfaz gráfica para utilizar *GBM* y hacer validaciones cruzadas), será necesario realizar una especificación de requisitos de software (*ERS*), mostrar la metodología y las decisiones de diseño del sistema. Posteriormente comparar

GBM con otros algoritmos proporcionados por *Weka* Witten et al. [2011]. Se (1) demostrará que generalmente un sistema clasificador que construye un solo modelo o *strong learner*, es menos eficiente que uno que construye este a partir de muchos *weak learners* James et al. [2013] y (2) comprobará que *GBM* da buenos resultados. Para ello, se debe hallar una configuración óptima para *GBM*, hacer una *stratified 10 fold cross validation* y por último realizar una validación estadística de los resultados.

Aplicar *Gradient Boosting* a datos docentes con la finalidad de modelar al alumnado. El principal motivo por el cual se lleva a cabo toda esta investigación es automatizar la realización de tareas laboriosas que tradicionalmente se hacían de forma artesanal en el sector docente. Para ello se (1) procesará un *data set* real de las notas del alumnado en una asignatura, (2) buscará una buena configuración para *GBM*, (3) comparará su precisión frente a los principales sistemas clasificadores, y (4) analizará el resultado por el personal docente.

1.3. Estructura del documento

El documento presente se encuentra organizado, a parte del capítulo actual, en ocho capítulos, el anexo y la bibliografía. A continuación se describen sus contenidos.

Capítulo 2. Sitúa al lector en el campo de la Inteligencia Artificial, más concretamente en la disciplina del Aprendizaje Automático. En este capítulo se definen todos los conceptos imprescindibles para facilitar la comprensión del presente Trabajo Final de Grado. Para ello se comienza situando al lector en el contexto del Aprendizaje Automático para seguidamente introducir el formalismo matemático, los objetivos en sí y finalmente se introduce la taxonomía de esta disciplina.

Capítulo 3. Profundiza en el aprendizaje supervisado, introduce los cinco sistemas clasificadores más comunes y las dos principales grandes familias de los métodos avanzados de clasificación *Ensemble*. Estas se denominan *Bagging* y *Boosting*, agrupan distintos sistemas clasificadores para obtener uno mucho más preciso.

Capítulo 4. Especifica los requisitos de software, muestra la metodología y las decisiones de diseño del sistema *GBM* y su interfaz gráfica *PGUI*.

Capítulo 5. Se centra en (1) comprobar que una técnica que combina numerosos *weak learners* para producir un *strong learner* es más precisa que otra técnica que genere un solo *strong learner*, (2) encontrar una configuración óptima para *GBM* y (3) compararlo con otros sistemas clasificadores. Para ello, se realiza una validación cruzada completa y una validación estadística de los resultados.

Capítulo 6. Expone (1) el procesamiento de datos docentes, (2) la aplicación de *GBM* a los datos procesados, (3) la comparación con otros algoritmos, y (4) la supervisión del resultado.

Capítulo 7. Desglosa la estimación del coste del proyecto.

Capítulo 8. Resume el trabajo realizado y expone las conclusiones.

Capítulo 9. Detalla las líneas de futuro.

Además de los capítulos principales, se incluyen cinco anexos.

Anexo 1 Ficheros *ARFF*. Explica la estructura del formato *ARFF*. Éste se ha convertido en un referente debido a su uso masivo.

Anexo 2 Manual de *Weka*. Detalla el funcionamiento de *Weka*. Se trata de una plataforma de libre distribución y difusión que proporciona algoritmos de Minería de Datos. A lo largo de toda la experimentación realizada, se ha utilizado para (1) convertir *data sets* de formato *CSV* a *ARFF* y (2) configurar y aplicar algoritmos de Aprendizaje Automático con la finalidad de compararlos con *GBM*. En esta sección se muestra como se ha hecho este proceso.

Anexo 3 Manual de *PGUI* y *ModelToGpv*. Detalla el funcionamiento de *PGUI* y *ModelToGpv*. *GBM* no tiene interfaz gráfica, así que funciona vía línea de comandos. Por lo tanto se ha construido la interfaz *PGUI*, esta permite al usuario experimentar con *GBM* de forma sencilla y realizar una validación cruzada para evaluar el sistema. *ModelToGpv* es un *script* que traduce el modelo a lenguaje *DOT*.

Anexo 4 Optimización y depuración del código. Muestra las herramientas y procedimientos utilizados para dar efecto a *C++*.

Anexo 5 *Doxygen*. Explica cómo se ha generado la documentación del algoritmo *GBM* mediante *Doxygen*¹.

¹Disponible de forma gratuita en <http://www.stack.nl/~dimitri/doxygen/>.

2

Aprendizaje Automático

Los grandes avances en la gestión de datos digitales han provocado una enorme dificultad en obtener conocimiento útil sobre la información. A medida que se van desarrollando estas tecnologías, resulta más complejo su tratamiento con métodos tradicionales debido a la gran cantidad de datos y diferentes características de estos. Un grupo de técnicas que permite su gestión se halla en el campo del Aprendizaje Automático.

2.1. Contexto

El Aprendizaje Automático, también conocido como Aprendizaje Estadístico (*Statistical Learning*) es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender con la finalidad de solucionar problemas de inferencia, realizar predicciones, decisiones o construir modelos descriptivos a partir de los datos. Para ello, [Fayyad et al. \[1996\]](#) define el término *Knowledge Discovery Database (KDD)* como el proceso de extraer información útil en los datos.

En la figura [2.1](#) se muestran todos los pasos que conforman el *KDD*. Primero se obtienen los datos sin formato o *raw* de los que se quiere extraer el conocimiento, de estos se seleccionan aquellos que cumplen los criterios deseados y posteriormente se preprocesan eliminando posible ruido o variables no deseadas. Estos datos preprocesados se transforman, es decir, se pueden discretizar, normalizar, etcétera, si así se requiere. En este punto, los datos pueden ser almacenados por ejemplo en un fichero *ARFF* para

ser utilizados por un algoritmo de Aprendizaje Automático como *Gradient Boosting* con la finalidad de obtener los patrones y de ahí el conocimiento.

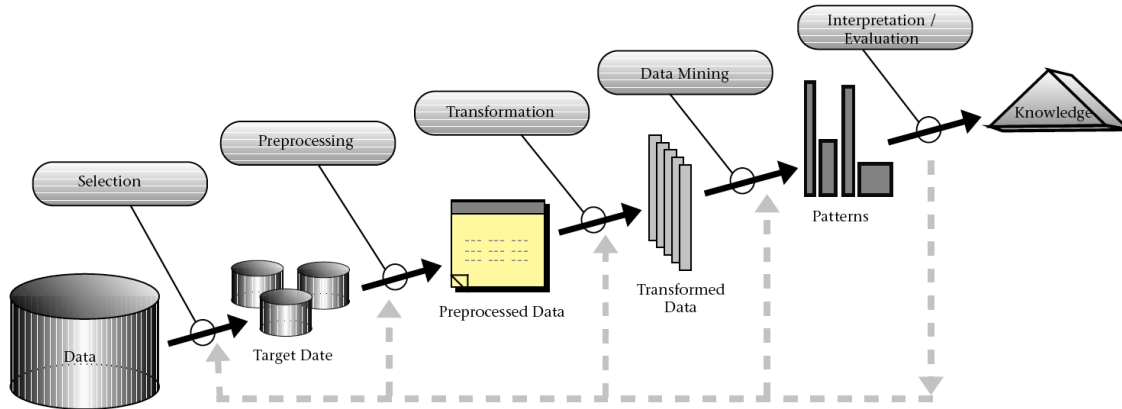


FIGURA 2.1: Ciclo de vida *KDD* según Fayyad et al. [1996].

Matemáticamente el proceso de Aprendizaje Estadístico se traduce en hallar una serie de aproximaciones para estimar una función f que vincula variables de entrada a variables de salida [James et al., 2013].

Una vez expuesto el ciclo de vida *KDD*, a continuación se define la notación formal con la intención de facilitar al lector la comprensión del actual Trabajo Final de Grado.

2.2. Notación

Los problemas a resolver se representan como un conjunto de variables de entrada mediante el símbolo X . Éstas están compuestas por las variables, atributos o predictores X_1, X_2, \dots, X_P es decir, $X = (X_1, X_2, \dots, X_P)$. Las variables de salida, variables dependientes o respuesta se simbolizan mediante Y . Todo dato contiene un error aleatorio que no puede ser tratado. El símbolo ε denota el mencionado error. Éste es independiente de X y su media es cero. El acento circunflejo simboliza una aproximación hacia el propio objeto, por ejemplo \hat{f} es la aproximación de f e \hat{Y} la de Y .

En el capítulo 3, concretamente en los modelos basados en redes neuronales y máquinas de vectores de soporte, se hace uso de hiperplanos. En un espacio p dimensional, un hiperplano es un espacio afín de dimensión $p - 1$, es decir, en un espacio unidimensional (una recta) un hiperplano es un punto que divide una línea en dos. En un espacio

bidimensional es una recta que divide el plano en dos mitades y en un espacio tridimensional es un plano que divide el espacio en dos mitades. Introducida la notación, en la siguiente sección se detalla la relación existente entre X e Y .

2.3. Función f

Como se menciona en el apartado anterior, se asume que existe una relación entre las variables de entrada y salida, esto es entre $X = (X_1, X_2, \dots, X_P)$ e Y . Esta correspondencia se podría definir de la siguiente forma, expresión 2.1.

$$Y = f(X) + \varepsilon \quad (2.1)$$

En este contexto f es una función desconocida y representa la información sistemática que X provee sobre Y . Los dos principales motivos para estimar f son la inferencia y la predicción.

2.4. Inferencia

Estimar f permite estudiar la relación existente entre las variables de entrada y la respuesta o dicho de otro modo, la forma en la que Y es afectada cuando X_1, X_2, \dots, X_P varia, a este hecho se le denomina inferencia. En éste contexto, el objetivo no tiene por qué ser realizar predicciones para Y . Puede interesar conocer qué predictores están relacionados con la respuesta, cuál es la relación entre ésta y cada predictor, si mediante una ecuación lineal se puede describir el vínculo existente entre la respuesta y cada predictor o por contra, si se trata de algo más complejo [James et al., 2013].

Podemos encontrar usos de la inferencia en la industria: supermercados como *7 Eleven* o *Wal-Mart* en Estados Unidos que aplican algoritmos de Inteligencia Artificial sobre la base de datos de su comercio para encontrar una correlación entre las ventas de sus productos en forma de reglas de asociación [Agrawal et al., 1993]. Éstos hallaron una relación entre la venta de pañales y cervezas, además estos productos se vendían principalmente los viernes por la tarde y eran comprados por hombres entre los 25 y 35 años. Se estipula que la razón de este hallazgo es que los paquetes de pañales son voluminosos y las esposas, que en muchos casos hacen la compra de la casa, dejan los pañales para que el marido los compre. Éste compra los pañales especialmente los viernes en compañía de las cervezas para el fin de semana. Como consecuencia, estos supermercados han situado las cervezas al lado de los pañales y el resultado ha sido que las ventas de cerveza se han disparado.

2.5. Predicción

En muchos problemas, hallar las variables de entrada X es una tarea inmediata pero la salida Y no se puede obtener de forma sencilla. En este escenario se debe construir un modelo f capaz de generalizar los datos y el vínculo entre éstos usando los casos ya existentes para poder predecir la salida Y . Un caso real se halla en la predicción de cáncer de próstata [Llorà et al., 2007] donde se propone mediante un algoritmo predictivo, mejorar el acierto que tiene un experto manualmente en el diagnóstico de esta enfermedad. El algoritmo utiliza el tipo de tejido y las coordenadas del *microarray* como variables de entrada y si se trata de cáncer o no a modo de respuesta o variable de salida.

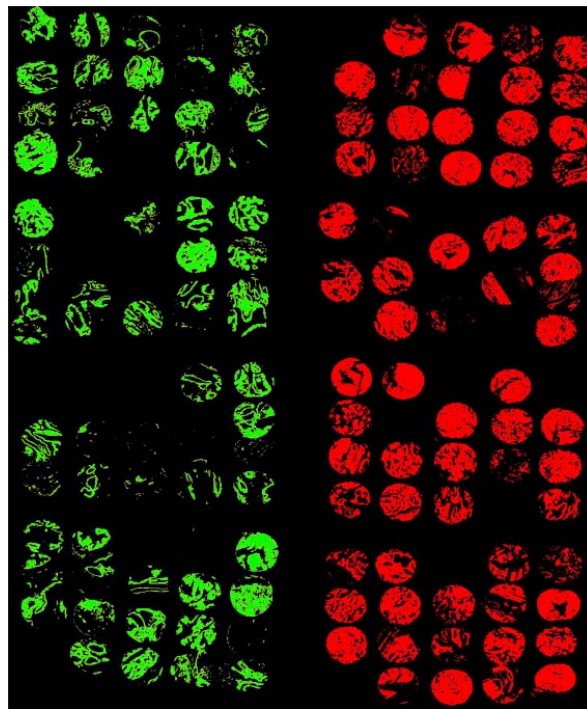
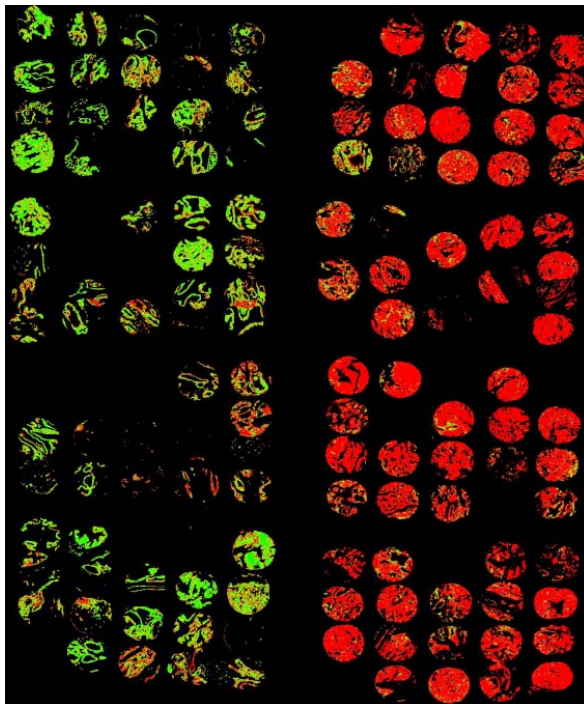


FIGURA 2.2: *Microarray* original [Llorà et al., 2007].

En las figuras 2.2 y 2.3, los píxeles verdes del *microarray* representan regiones de piel donde el tejido no es cancerígeno mientras que los rojos representan cáncer. La figura 2.2 presenta el *microarray* original con el que se ha entrenado el algoritmo y la figura 2.3 presenta la imagen reconstruida basada en las predicciones. El algoritmo es capaz de clasificar los píxeles de forma correcta y de este modo obtener un ratio de error comparable al de los expertos humanos de forma automatizada.

FIGURA 2.3: *Microarray* de predicción [Llorà et al., 2007].

2.6. Estimación \hat{f}

La estimación del modelo \hat{f} es una tarea que no resulta trivial, se desea generalizar de forma automatizada y para ello típicamente se fragmenta una base de datos mediante una validación cruzada (*cross validation*) en dos partes, una de entreno (*train*) y otra de prueba (*test*). La validación cruzada es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre los datos de entrenamiento y prueba. Los datos de entrenamiento constituyen el modelo \hat{f} donde $Y \approx \hat{f}(X)$ para cualquier observación (X, Y) y representan una estimación para f . Finalmente, los datos de prueba se utilizan para realizar predicciones mediante el modelo construido por los de entrenamiento. Si el modelo \hat{f} se adapta excesivamente a los datos de entrenamiento impedirá que éste generalice Leiva Murillo [2007]. Es decir, el algoritmo de aprendizaje quedará ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo f . Éste fenómeno es comúnmente denominado sobreajuste (*overfitting*) y una solución posible es la regularización. La regularización es un factor aplicado en el modelo \hat{f} que simplemente añade un ruido en la aproximación de la respuesta \hat{Y} con la finalidad de no encajar demasiado con los datos de entreno.

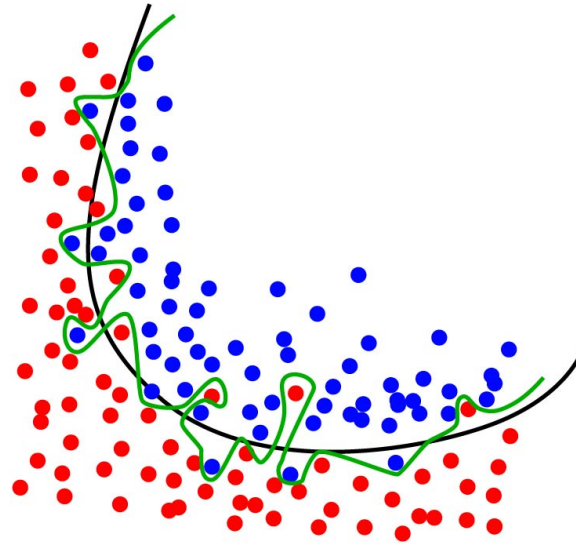


FIGURA 2.4: Gráfica que representa dos tipos de clasificadores y dos tipos de respuesta [Icke, 2008].

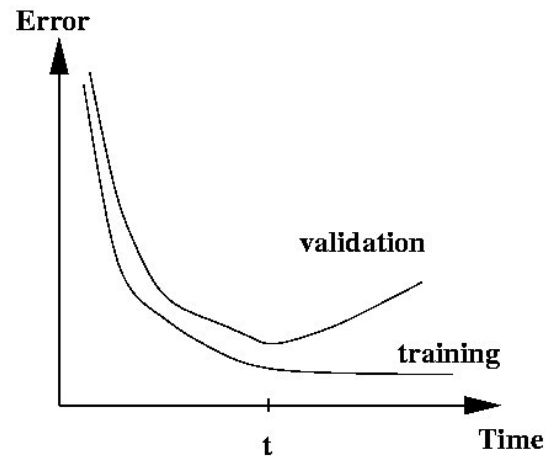


FIGURA 2.5: Relación error/tiempo en un caso de sobreajuste [Orr, 1999].

Las figuras anteriores constituyen casos de *overfitting*. La figura 2.5 representa una relación entre el error y el tiempo transcurrido, inicialmente el error en *test* y *train* del algoritmo es muy elevado ya que no se ha adaptado lo suficiente a los datos y va disminuyendo hasta llegar a un tiempo t donde se ve muy claramente reflejado el problema del sobreajuste ya que el error en la predicción (*validation*) vuelve a aumentar mientras que en el *training* disminuye. La figura 2.4 presenta dos clases distintas, una mostrada en color azul y la otra en rojo. La línea verde como clasificador se adapta mejor a los datos con los que se ha entrenado el algoritmo, pero está demasiado

adaptada a ellos, de forma que ante nuevos datos probablemente arrojará más errores que la clasificación usando el clasificador representado con la línea negra.

2.7. Paradigmas

Los diferentes algoritmos de Aprendizaje Automático se agrupan en tres grandes paradigmas [Bacardit, 2004, Llorà, 2002, Orriols-Puig, 2008]: aprendizaje supervisado, no supervisado y por refuerzo. A continuación se detallan las tres familias de algoritmos.

Se denomina aprendizaje supervisado (*supervised learning*) al proceso de extraer una función o modelo que mapea la relación entre atributos de entrada y salida. Dependiendo de los atributos de salida, el aprendizaje supervisado puede ser de clasificación o de regresión. Se trata de clasificación si el objetivo es hallar un modelo que predice la clase o salida de nuevas instancias, en cambio se trata de regresión cuando interesa hallar una función para predecir un valor continuo de salida (se trata de una generalización de la clasificación). En general, un algoritmo de aprendizaje supervisado tiene que generalizar a través de los datos conocidos y construir una función o un modelo capaz de predecir el valor de salida por cada valor de entrada [Orriols-Puig, 2008].

En el aprendizaje no supervisado (*unsupervised learning*) el algoritmo cuenta con un conjunto de entrenamiento X_1, X_2, \dots, X_P pero no tiene ninguna respuesta asociada. El objetivo es extraer patrones descriptivos a partir de los datos de entrada. El resultado se puede emplear para tomar decisiones y agrupar entradas similares. Destacan las siguientes técnicas en el aprendizaje no supervisado: *Clustering*, Reglas de asociación y Reducción de dimensiones. La técnica *Clustering* se basa en agrupar objetos en regiones donde la similitud mutua es elevada. Las Reglas de asociación se utilizan para descubrir hechos que ocurren comúnmente dentro de un determinado conjunto de datos y por último, Reducción de dimensiones se basa en reducir el espacio de búsqueda de la base de datos quedándose con aquellos atributos que aportan la mayor información en el *data set*.

El aprendizaje por refuerzo (*reinforcement learning*) [Sutton and Barto, 1998] es un híbrido entre el aprendizaje supervisado y no supervisado. Estudia cómo un agente aprende mediante interacción directa con su entorno, sin la intervención de un supervisor que le dice qué tiene que hacer. El aprendizaje es reactivo, aprende a prueba y error, es por esto que hay que definir el algoritmo de forma que se premie o castigue su comportamiento en función de las acciones que vaya realizando el agente.

2.8. Resumen

En este capítulo se han presentado todos los conceptos base para facilitar la comprensión de los capítulos siguientes. Se ha descrito teóricamente y matemáticamente la disciplina del Aprendizaje Automático: consiste en aprender a partir de la experiencia.

Se han presentado los tres principales paradigmas del Aprendizaje Automático, estos son: (1) supervisado, el algoritmo deduce una función a partir de los datos de entrada o entrenamiento, (2) no supervisado, donde no hay conocimiento a priori y (3) por refuerzo, el algoritmo aprende a base de ensayo-error.

En el presente Trabajo Final de Grado se pretende modelar el rendimiento del alumnado, para ello ya se conoce a priori la salida del sistema (la nota final), por lo tanto en el siguiente capítulo se detallan las principales técnicas del aprendizaje supervisado.

3

Tipos de modelos y clasificadores avanzados

En la sección anterior se ha presentado una taxonomía clásica que identifica tres tipos de aprendizaje. La finalidad de este capítulo es exponer los modelos más representativos del Aprendizaje Automático [Wu et al., 2007] para posteriormente proseguir con técnicas avanzadas de clasificación. Para ello se presenta primero los sistemas clasificadores, es decir, Aprendizaje Perezoso, Redes Neuronales, Modelos probabilísticos, Máquinas de Soporte Vectorial y Árboles de decisión y posteriormente los métodos *Ensemble*, los cuales consisten en combinar múltiples modelos entre sí para generar un único método mucho más preciso. Estos métodos constan principalmente de dos grandes familias, *Bagging* y *Boosting*.

3.1. Sistemas clasificadores

El Aprendizaje Automático consta de tres paradigmas, el supervisado, no supervisado y por refuerzo. El propósito de éste capítulo es describir la familia supervisada. Los sistemas supervisados intentan generalizar un modelo a partir de datos previamente etiquetados. Más específicamente se busca un modelo que haga un mapeo entre los atributos de entrada y la respuesta. Existen numerosas técnicas de las cuales se describirán las cinco más comunes en la literatura [Wu et al., 2007].

3.1.1. Aprendizaje Perezoso

En contraste con otras técnicas de aprendizaje que construyen un modelo con un conjunto de datos de entrenamiento, el Aprendizaje Perezoso, *Instance Based Learner*, *Lazy Learner* o *Just-in-time Learning* [Cybenko, 1989], es una técnica que no genera ningún modelo, así que no requiere el aprendizaje de un modelo y se basa en una función de distancia para clasificar. En este tipo de aprendizaje se diferencian dos fases, (1) la fase de entrenamiento, donde el algoritmo almacena todo el conjunto de datos disponibles, y (2) la fase de generalización, donde se extraen de la memoria un conjunto de datos similares al nuevo dato que son empleados para clasificarlo. De este modo todo el cómputo se realiza en el tiempo de clasificación y es por eso que se le denomina Aprendizaje Perezoso. Existe una gran variedad de algoritmos para calcular el grado de similitud entre datos, el más conocido es *KNN* (*k-Nearest Neighbours*) debido a su robustez frente al ruido (para valores altos de k) y su simpleza [Mitchell, 1997]. El número k se utiliza para decidir el valor de salida (por ejemplo, haciendo una votación). Para calcular los vecinos más cercanos de una instancia es muy típico utilizar la distancia Euclidiana [Fornells, 2006], expresión 3.1.

$$d(\vec{X}_i, \vec{X}_j) = \sqrt{\sum_{r=1}^n (X_i[r] - X_j[r])^2} \quad (3.1)$$

En el caso de que los atributos presenten rangos distintos, se normaliza. Una gran ventaja de este algoritmo es que puede tratar datos continuos y discretos. En el caso de usar atributos discretos, se calcula de la siguiente manera. Sea $\langle X, f(X) \rangle$ el conjunto de datos almacenados y sea X_q el nuevo objeto a clasificar [Mitchell, 1997], expresión 3.2:

$$f : \Re^n \rightarrow V \quad (3.2)$$

$$\hat{f}(\vec{X}_q) = \operatorname{argmáx}_{v \in V} \sum_{i=1}^k \delta(v, f(\vec{X}_i))$$

Donde

$$\delta(a, b) = \begin{cases} 1 & \text{si } a = b, \\ 0 & \text{en caso contrario.} \end{cases}$$

Típicamente se añade un peso o grado de contribución w_i a cada muestra, de esta forma se puede refinar el algoritmo dotando de mayor importancia aquellos vecinos más cercanos, expresión 3.3.

$$\hat{f}(\vec{X}_q) = \operatorname{argmáx}_{v \in V} \sum_{i=1}^k w_i \delta(v, f(\vec{X}_i)) \quad (3.3)$$

Donde

$$w_i \equiv \frac{1}{d(\vec{X}_q, \vec{X}_i)^2}, \quad \text{si } d=0 \text{ entonces } w=0 \quad (3.4)$$

En caso de que se trate de atributos continuos, hay que calcular el valor promedio entre los k más cercanos [Mitchell, 1997], expresión 3.5.

$$\hat{f}(\vec{X}_q) = \frac{\sum_{i=1}^k f(\vec{X}_i)}{k} \quad (3.5)$$

Si se tiene en cuenta el peso de cada muestra, la operación matemática queda de la siguiente forma, expresión 3.6.

$$\hat{f}(\vec{X}_q) = \frac{\sum_{i=1}^k w_i f(\vec{X}_i)}{\sum_{i=1}^k w_i} \quad (3.6)$$

Los métodos de Aprendizaje Perezoso presentan las ventajas de ser muy fáciles de implementar y por lo general bastante eficientes en tiempo de ejecución siendo bastante precisos para k elevadas en caso de usar KNN . Este tipo de técnicas no generan un modelo, así que requieren una búsqueda exhaustiva (caso por caso) dentro de la memoria de casos cada vez que aparece nueva información, dado que no se dispone de un modelo de generalización.

3.1.2. Redes neuronales

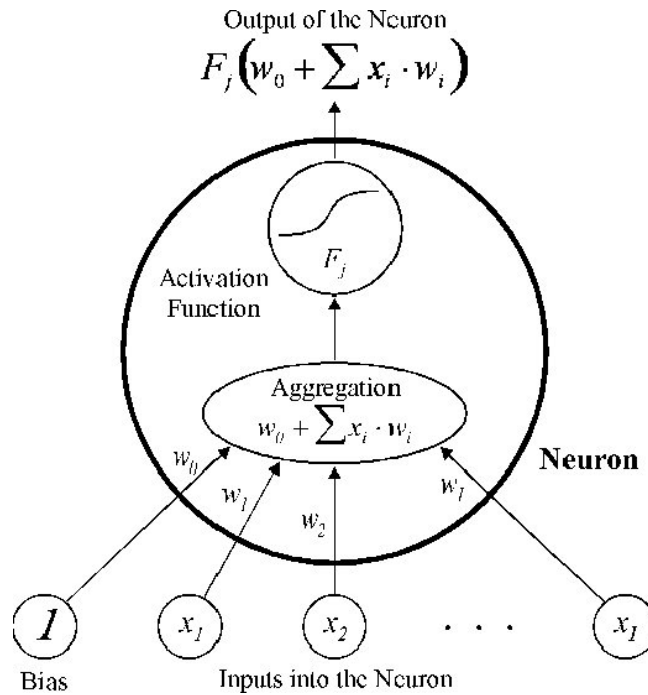


FIGURA 3.1: Red neuronal de un sólo perceptrón [Härdle and Lehmann, 2004].

Las redes neuronales o redes neuronales artificiales (RNA) son un paradigma de aprendizaje y procesamiento automático, inspirado en la forma que funciona el sistema nervioso, para obtener patrones que definen el modelo de aprendizaje, donde las neuronas están interconectadas y colaboran entre sí para producir un estímulo de salida.

En la figura 3.1 se muestra la estructura de un perceptrón o neurona formado por p atributos de entrada $X = (X_1, X_2, \dots, X_p)$ y el Bias. El Bias es un offset que ayuda a posicionar el hiperplano que separa las clases. A todos se les aplica un factor de importancia de la muestra o peso $w_i \{ i \in \mathbb{Z} \mid 0 < i = p \}$, si éste es positivo la conexión se denomina excitatoria y si es negativo inhibitoria. Se suman formulando la función de propagación o excitación $\varphi(x) = w_0 + \sum_{i=1}^p x_i w_i$ y posteriormente se le aplica una función de activación. Típicamente ésta puede ser una función escalón, una función rampa, una sigmoide, una tangente hiperbólica o una Gaussiana y da como resultado el modelo $f(x) = F(w_0 + \sum_{i=1}^p x_i w_i)$.

El algoritmo más común para encontrar los pesos en la fase de entrenamiento es el *Widrow-Hoff delta rule* [Widrow and Lehr, 1990], expresión 3.7.

$$w_i = w_i + \eta(y_i - o_i)x_i \quad (3.7)$$

Donde

$\eta(0 < \eta \leq 1)$ es el ratio de aprendizaje y o_i es la salida de la neurona.

El gran problema que tiene la neurona de un perceptrón es que sólo puede tratar problemas lineales, es decir, problemas linealmente separables. Para mejorar esto se diseñó el concepto de red neuronal, donde varios perceptrones se unen en una misma red generando una red neuronal multicapa. *Back Propagation* es un algoritmo clásico de entrenamiento para la red neuronal multicapa [Widrow and Lehr, 1990]. Este está basado en el perceptrón simple pero escalado a toda una red de neuronas. Se trata de propagar el error desde la capa de salida hacia las entradas, calcular el error cometido e irse ajustando a este de forma progresiva. El algoritmo parte de que la topología (como está construida la red neuronal) esta generada, entonces se propaga la señal de entrada hacia la salida (*feed forward*), se calcula el *output* para cada neurona y este se va propagando hacia las capas superiores. Una vez llega al final, se recalculan los pesos y el error cometido se propaga hacia atrás (hacia la entrada). Este ciclo llega a su fin cuando el error tiende a cero o supera un número determinado de iteraciones.

Hacer uso de redes neuronales presenta una serie de ventajas e inconvenientes [Sancho-Asensio et al., 2014]. Tienen la capacidad de aprender con un alto grado de precisión mediante una etapa de aprendizaje, construyen su propia representación de la información en su interior liberando al analista de este trabajo, son tolerantes a fallos, flexibles, se pueden obtener respuestas a tiempo real y modelan patrones de datos generalizándolos. Son capaces de aproximar una función arbitraria [Cybenko, 1989].

No obstante son cajas negras, es decir, son difíciles de interpretar y su entrenamiento puede tomar mucho tiempo.

Un ejemplo real de su uso lo encontramos en el mundo de los videojuegos, concretamente en *Quake II* [Chapman, 1999]. En este videojuego se aplicó *Neuralbot* a los bots, éstos son programas que simulan a un jugador humano y utilizan una red neuronal artificial para decidir su comportamiento y lo combinan con un algoritmo genético para su aprendizaje.

3.1.3. Modelos probabilísticos

Los modelos probabilísticos, junto con los árboles de decisión y las redes neuronales, han sido los tres métodos más usados en el Aprendizaje Automático durante esta última década en tareas como la clasificación de documentos, filtrado de mensajes de correo electrónico o diagnósticos médicos automatizados [John and Langley, 1995]. Se fundamentan en el razonamiento estadístico y el más famoso es *Naive Bayes* [Wu et al., 2007]. Éste se basa en el teorema de *Bayes* y resulta muy útil porque ofrece (1) valores que pueden intervenir en el problema y un análisis cualitativo de los atributos, es decir, como se relacionan en términos de causalidad y su correlación existente, y (2) la importancia cuantitativa de estos atributos, ya que ofrece una medida probabilística de la importancia de las variables en el problema. Todos los clasificadores basados en *Naive Bayes* asumen que no hay ninguna variable de entrada que dependa de otra. A continuación se muestran los fundamentos matemáticos de la regla de *Bayes* para posteriormente contrastarlos con un ejemplo práctico.

Sea $X = (X_1, X_2, \dots, X_P)$ el conjunto de datos de entrada mutuamente excluyentes y exhaustivos, y tales que la probabilidad de cada uno de ellos es diferente de 0. Sea B un suceso cualquiera del que se conocen las probabilidades condicionadas $P(B|X_i)$. Entonces la probabilidad $P(X_i|B)$ se calcula de la siguiente forma [John and Langley, 1995], expresión 3.8.

$$P(X_i|Y) = \frac{P(Y|X_i)P(X_i)}{P(Y)} \quad (3.8)$$

Donde $P(X_i)$ son las probabilidades a priori, $P(Y|X_i)$ es la probabilidad de B en la hipótesis X_i y $P(X_i|Y)$ son las probabilidades a posteriori.

La probabilidad de suceso Y , basándose en el teorema de la probabilidad total, puede expresarse de la siguiente forma, expresión 3.9.

$$P(Y) = \sum_{k=1}^p P(Y|X_k)P(X_k) \quad (3.9)$$

En lo que sigue se expone un ejemplo práctico de la técnica mencionada anteriormente.

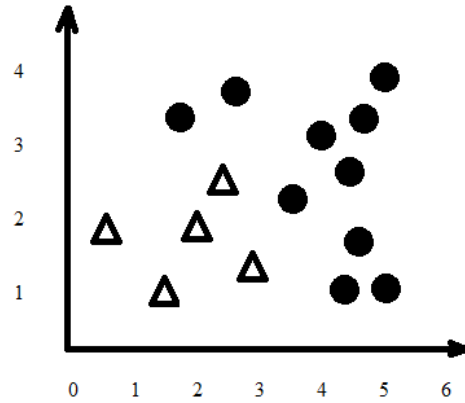


FIGURA 3.2: *Data set* de entrada para aplicar el modelo probabilístico *Naive Bayes*.

En la figura 3.2 se muestra una gráfica que representa un conjunto de datos o *data set* con 15 objetos. De éstos, cinco responden a la clase Δ y los diez restantes a \bullet . Esto significa que la probabilidad a priori de que un objeto nuevo sea Δ o \bullet es la siguiente, expresiones 3.10 y 3.11.

$$P(\Delta) = \frac{\text{Número de } \Delta}{\text{Número total de objetos}} = \frac{5}{15} \quad (3.10)$$

$$P(\bullet) = \frac{\text{Número de } \bullet}{\text{Número total de objetos}} = \frac{10}{15} \quad (3.11)$$

En la figura 3.3 el algoritmo debe clasificar un nuevo objeto (representado por un rombo verde).

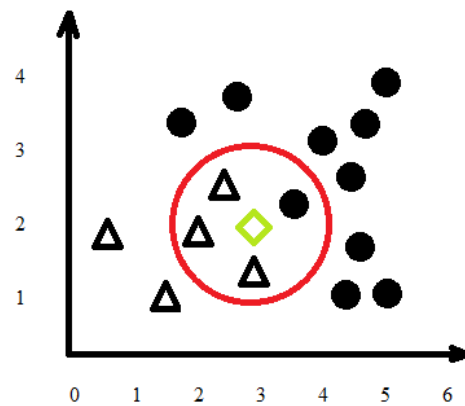


FIGURA 3.3: Modelo probabilístico *Naive Bayes* con un nuevo objeto para clasificar.

Si los objetos están bien agrupados por similitud (*clusterizados*) [James et al., 2013], se puede asumir que el objeto será del mismo tipo que los de su alrededor. En

este caso, para medir su semejanza, se ha trazado un círculo rojo que comprende 4 objetos, expresiones 3.12 y 3.13.

$$\text{Semejanza } (\triangle) = \frac{\text{Número de } \triangle \text{ vecinos}}{\text{Número total de objetos vecinos}} = \frac{3}{4} \quad (3.12)$$

$$\text{Semejanza } (\bullet) = \frac{\text{Número de } \bullet \text{ vecinos}}{\text{Número total de objetos vecinos}} = \frac{1}{4} \quad (3.13)$$

Mientras que la probabilidad general indica que este nuevo objeto probablemente sea un \bullet , dentro del rango trazado con sus vecinos indica lo contrario. En los análisis Bayesianos hay que combinar estas dos fuentes de información, expresiones 3.14 y 3.15.

$$\text{P. post. } (\triangle) = \frac{\text{Número de } \triangle}{\text{Número total de objetos}} \frac{\text{Número de } \triangle \text{ vecinos}}{\text{Número total de objetos vecinos}} = \frac{5}{15} \frac{3}{4} = \frac{1}{4} \quad (3.14)$$

$$\text{P. post. } (\bullet) = \frac{\text{Número de } \bullet}{\text{Número total de objetos}} \frac{\text{Número de } \bullet \text{ vecinos}}{\text{Número total de objetos vecinos}} = \frac{10}{15} \frac{1}{4} = \frac{1}{6} \quad (3.15)$$

Finalmente, se llega a la conclusión mediante el cálculo de la probabilidad posterior que este nuevo objeto se trata de un \triangle ya que su probabilidad es mayor.

Se ha visto un ejemplo sencillo acerca del funcionamiento de este método. Se generaliza, dado una serie de variables de entrada $X = (X_1, X_2, \dots, X_P)$ y un conjunto de posibles salidas $Y = (Y_1, Y_2, \dots, Y_P)$ se puede describir la probabilidad posterior $P(Y_j|X)$ de la siguiente forma, 3.16.

$$P(Y_j|X) = P(Y_j) \prod_{k=1}^p P(X_k|Y_j) \quad (3.16)$$

Donde $P(Y_j)$ es la probabilidad de que sea la propia clase respecto el número total de clases y $\prod_{k=1}^p P(X_k|Y_j)$ asumiendo que las variables son estadísticamente independientes, es una descomposición en forma de productorio de términos que representa el grado de semejanza que tiene cada clase respecto la nuestra en el rango indicado, en el ejemplo anterior, éste es el círculo rojo que comprende los cuatro objetos alrededor del objeto a clasificar. Usando *Bayes* y siguiendo la formulación anterior, se etiqueta un nuevo objeto X con la clase Y_j que tenga mayor probabilidad posterior. El método *Naive Bayes* $P(X_k|Y_j)$ puede ser modelado de distintas formas, entre ellas destaca la gaussiana o normal, ésta se formula de la siguiente manera, expresión 3.17.

$$\frac{1}{\sigma_{kj}\sqrt{2\pi}} e^{\frac{-(X-\mu_{kj})^2}{2\sigma_{kj}^2}} \quad (3.17)$$

$$-\infty < x < \infty, -\infty < \mu_{kj} < \infty, \sigma_{kj} > 0$$

Donde μ_{kj} es la media y σ_{kj} la desviación típica. Los métodos probabilísticos basados en *Naive Bayes* son rápidos de calcular, a pesar de ser simples dan muy buenos resultados en cuanto acierto y son explicativos, dan una certeza en forma de probabilidad. Debido a las asunciones en algunos casos pueden carecer de precisión y asumen una distribución de probabilidad, esto puede ser problemático porque si los datos no siguen esta distribución se generarán falsos positivos o falsos negativos [Wu et al., 2007]. La solución a este último problema es generando el modelo de forma no paramétrica pero con el inconveniente de que puede aumentar mucho su complejidad.

3.1.4. Máquinas de Soporte Vectorial

Las máquinas de Soporte Vectorial o *Support Vector Machines (SVM)* fueron introducidas en 1992 por *Boser, Guyon y Vapnik* y se basan en el principio de minimización del riesgo estructural. Es decir, quieren obtener el menor error de clasificación posible [Mitchell, 1997, Vapnik, 1995]. Son especialmente atractivas para los analistas puesto que superan a las Redes Neuronales en cuanto a su capacidad de generalización y a la ausencia de mínimos locales aunque sufre de otros problemas como la selección de una función *Kernel* [Platt, 1998]. Esta técnica consiste en hallar un hiperplano que tenga el máximo margen con las muestras del *data set* que estén más cerca del mismo. Este margen se entiende como la distancia entre los vectores formados por los puntos más cercanos al hiperplano denominados vectores de soporte, y la finalidad es separar óptimamente los puntos de una clase de la otra de tal forma que los datos etiquetados en una categoría estarán en un lado del hiperplano y los casos que se encuentren en otra categoría estarán al otro lado.

La figura 3.4 presenta un caso idóneo donde las clases están separadas y la solución del problema es inmediata. Se trata de una representación gráfica en función de los atributos de entrada de un *data set* con dos atributos de salida distintos, uno de ellos representado mediante círculos negros y el otro blancos. El Bias o *intercept* está simbolizado mediante el carácter b , el hiperplano está formado por todas aquellas x que satisfacen la ecuación $wx + b = 0$, w es su vector normal y $\frac{b}{\|w\|}$ su *offset* respecto al eje de coordenadas. Lo que se pretende es hallar el mejor hiperplano posible para el problema, el criterio para elegir a éste viene dado por el grado de separación o margen $\frac{2}{\|w\|}$ entre los casos extremos o vectores de soporte $wx + b = 1$ y $wx + b = -1$. Cuanto mayor sea el margen, menor será el error de generalización del clasificador. Nótese que las *SVM* sólo clasifican para atributos de salida definidos en el conjunto $\{-1, 1\}$, o sea sólo distinguen dos tipos distintos de salida, se trata pues de un clasificador binario.

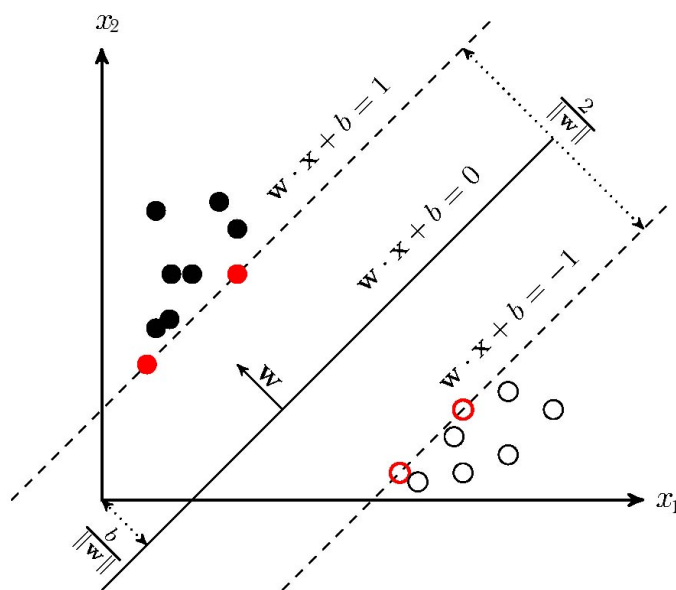


FIGURA 3.4: SVM problema linealmente separable [Peng, 2013].

Puede darse el caso en que los datos no sean completamente separables, y la clasificación se haga en más de dos categorías. En el primer caso, la función *Kernel* permite resolver el sistema no lineal proyectando los datos a un espacio Hilbertiano de una mayor dimensión, como puede verse en la figura 3.5.

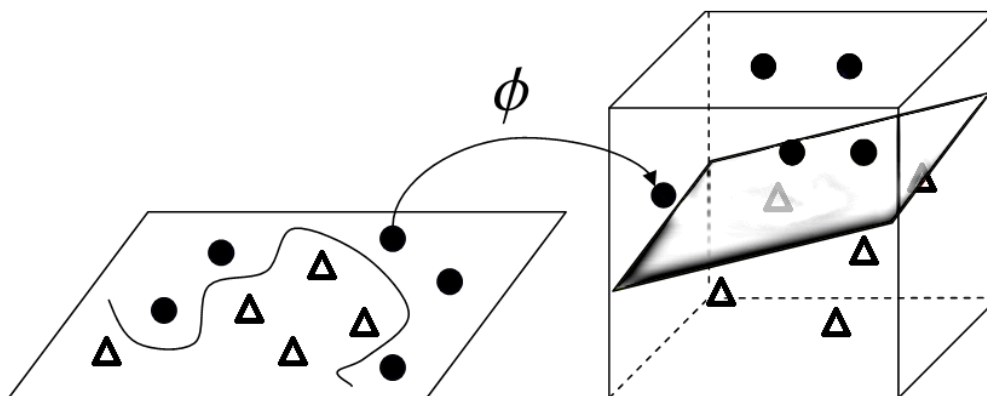


FIGURA 3.5: SVM problema no linealmente separable [Birgi, 2014].

La función *Kernel* Φ permite un mapeo del espacio original de los datos a un espacio mayor donde resulta más factible encontrar un hiperplano capaz de separar ambas clases. Como ejemplos de *Kernel*, los más comunes son el Polinomial (expresión 3.18) y el Radial Gaussiano (expresión 3.19).

$$k(\vec{x}_i + \vec{x}_j) = (\vec{x}_i \vec{x}_j + 1)^d \quad (3.18)$$

Donde d es el grado del polinomio

$$k(\vec{x}_i + \vec{x}_j) = e^{-\gamma \|\vec{x}_i - \vec{x}_j\|^2}, \quad \gamma > 0 \quad (3.19)$$

Donde γ es un parámetro que controla la anchura del *Kernel*.

En el segundo caso, donde el problema a solucionar es multiclase, la estrategia más común es uno contra todos (*one-versus-all*) [Mitchell, 1997]. Se pasa de un problema con n clases a n problemas binarios y lógicamente se necesitaran n Máquinas de soporte vectorial.

El modelo que construye el *SVM* es el que viene representado por la expresión 3.20.

$$f : \frac{1}{2} \|\vec{w}\|^2 \quad (3.20)$$

Sujeto a, para cualquier $i = 1, \dots, n$

$$g : y_i(\vec{w}\vec{x}_i - b) = 1 \quad (3.21)$$

Se trata de un problema de programación cuadrática, hay que minimizar $f(x)$ y $g(x)$ debe ser cero. Esto se soluciona mediante los multiplicadores de *Lagrange* y se optimiza el cálculo mediante el truco Lagrangiano (*Lagrangian trick*), finalmente acaba resultando en la forma dual [Platt, 2000].

Se debe maximizar la expresión 3.22 sujeto a la restricción 3.23:

$$L = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j k(\vec{x}_i + \vec{x}_j) \quad (3.22)$$

$$\begin{cases} w = \sum_{i=1}^m \alpha_i y_i \vec{x}_i \\ \sum_{i=1}^m y_i \alpha_i = 0 \end{cases} \quad (3.23)$$

$$0 \leq \alpha_i \leq C, i = 1, \dots, m$$

Donde α representa los multiplicadores de *Lagrange*, C es un parámetro que simboliza cómo de restrictivo es el algoritmo y k es la función *Kernel*. Se trata de un procedimiento para encontrar los máximos y mínimos de funciones de distintas variables sujetas a restricciones, el algoritmo más conocido para solucionar esta formulación es el *Sequential Minimal Optimization* (*SMO*) [Platt, 1998].

La técnica *SVM* permite un buen comportamiento en problemas de alta dimensionalidad [James et al., 2013], ya que la capacidad de generalización como el proceso de entrenamiento no dependen necesariamente del número de atributos debido a su formulación, pues emplea el principio de minimización del riesgo estructural [Vapnik, 1995]. Es la técnica fuera de la familia Ensemble con más precisión que existe [Wu et al., 2007]. En su contra, se trata de un algoritmo muy complejo de implementar, cuyo exponente más famoso (el *SMO*) tiene un coste computacional elevado, concretamente $O(n^3)$ y la explicación de la solución resulta compleja de interpretar.

3.1.5. Árboles de decisión

Otra forma de modelar la información se halla en una estructura bien conocida en el mundo de la computación denominada árbol de decisión. El árbol de decisión es uno de los métodos más prácticos y utilizados para la inferencia inductiva [Mitchell, 1997]. Los árboles pueden ser de clasificación si el objetivo es predecir etiquetas o de regresión si es predecir una salida continua. Un árbol es una estructura recursiva que está formada por un nodo raíz (*root node*) que dispone de nodos hijos. Cada nodo hijo puede ser a su vez un nodo interno y disponer también de otros nodos hijos o puede ser un nodo hoja (*leaf node*) donde se halla el valor esperado o respuesta. A medida que se van generando nodos internos, se va particionando el espacio de búsqueda hasta que la impureza es mínima. Entiéndase por impureza el criterio que define el modo en que el modelo ha particionado los datos hasta el nodo hoja actual. Así, una partición con una impureza nula implica que la partición es perfecta, no hay clases solapadas. El criterio de impureza puede ser la ganancia en varianza, la entropía, índice *Gini*, etcétera. entre la partición o región R_i y el atributo de salida, de este modo en cada nivel el algoritmo escoge la variable X_1, X_2, \dots, X_P con menos variación. En un árbol de clasificación existe el concepto de poda o *pruning*, el cual se basa en eliminar nodos que pueden causar *overfitting*. Para ello se utiliza una técnica estadística que predice si la rama que se va a podar puede ser problemática o no con cierta probabilidad, este proceso es costoso y un ejemplo de uso se haya en el sistema *C4.5* [Quinlan, 1993].

A continuación se muestra un ejemplo simple de su procedimiento paso a paso.

Atributo X_1	Atributo X_2	Atributo Y
1	1	\triangle
2	3	\triangle
3	1	\bullet
4	2	\bullet
5	1	\triangle

TABLA 3.1: *Data set* de ejemplo con cinco instancias. Está formado por los atributos X_1 , X_2 y la clase Y .

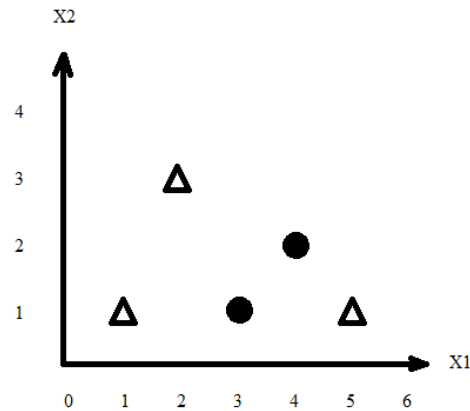


FIGURA 3.6: Representación gráfica del *data set* de ejemplo en función de los atributos X_1 y X_2 .

La tabla 3.1 es una pequeña base de datos, contiene los atributos de entrada X_1 , X_2 y su respuesta asociada Y . La figura 3.6 es la representación gráfica de esta base de datos en función de sus dos predictores. El algoritmo debe seguir los 7 siguientes pasos.

1. Ordenar el primer atributo de menor a mayor. En la tabla anterior ya está ordenado de forma ascendente.
2. Calcular la varianza del atributo actual sin hacer partición (usando todo el *data set*).
3. Calcular la varianza ejemplo a ejemplo del atributo actual guardando la mejor partición hasta el momento (la que contiene menos varianza). En el ejemplo de la figura 3.7, se muestra la partición obtenida usando los datos de la tabla 3.1. Según el atributo X_1 , se generarán tres regiones con varianza mínima, es decir, cada una con un solo tipo de salida (esto es genérico y sirve tanto para clasificación como para regresión). Según la heurística empleada, en este caso se coge el promedio. Por ejemplo la región 1 está comprendida entre los valores 2 y 3 así que el valor intermedio que menos varianza da es 2.5. Nótese que hay una infinidad posible de heurísticas, aunque la más común es esta [Breiman et al., 1984].
4. Ordenar el siguiente atributo en orden ascendente. Como puede verse en la tabla 3.2, la variancia es muy elevada.
5. Seguir el mismo procedimiento que en el punto 2 pero usando la segunda variable, es decir, calcular la varianza de todo el atributo sin hacer partición usando todo el *data set*.

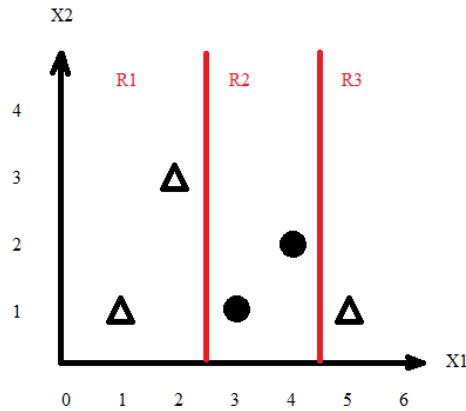


FIGURA 3.7: Representación gráfica de la partición del *data set* de ejemplo según la ganancia en varianza del atributo X_1 .

Atributo X_1	Atributo X_2	Atributo Y
1	1	\triangle
3	1	\bullet
5	1	\triangle
4	2	\bullet
2	3	\triangle

TABLA 3.2: Este *data set* está formado por los atributos X_1 , X_2 y la clase Y . Se han ordenado las cinco instancias según el atributo X_2 .

- Calcular varianza ejemplo a ejemplo al igual que se ha hecho en el punto 3 usando el segundo atributo. En este caso, tal y como se ha explicado anteriormente la varianza es peor y puede comprobarse gráficamente (ver figura 3.8) que comete un error insalvable en la tercera región donde hay dos tipos de respuesta distintos.

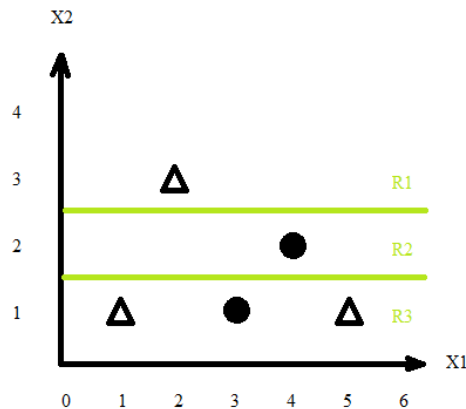


FIGURA 3.8: Representación gráfica de la partición del *data set* de ejemplo según la ganancia en varianza del atributo X_2 .

7. Elegir aquella variable con menor varianza (ver tabla 3.3), en este caso X_1 y generar dos particiones del *data set* inicial.

Atributo X_1	Atributo X_2	Atributo Y
1	1	\triangle
2	3	\triangle

TABLA 3.3: Primera partición del *data set*, no hay impureza. Por consiguiente, el nodo que se generará será directamente un nodo hoja con una predicción del 100 %.

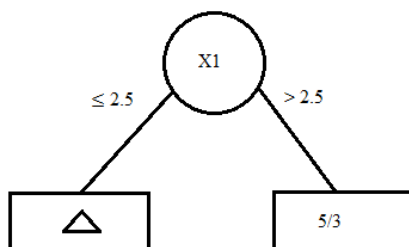


FIGURA 3.9: Primera etapa del árbol generado. Se trata de un árbol con profundidad 1 y con un nodo hoja puro (sin varianza). El valor de la partición es 2.5 y 5/3 es un valor numérico temporal e implica que de los cinco casos posibles quedan tres por clasificar.

Atributo X_1	Atributo X_2	Atributo Y
3	1	\bullet
4	2	\bullet
5	1	\triangle

TABLA 3.4: Segunda partición del *data set*, en este caso hay impureza.

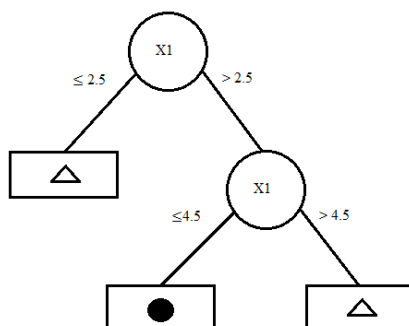


FIGURA 3.10: Segunda etapa (y final) del árbol generado. El árbol es puro, es decir, el acierto es del 100 % y ha descartado por completo el segundo atributo, significa que éste no aporta nada.

En la tabla 3.3 se muestra el resultado de la primera partición de la base de datos y en la figura 3.9 se refleja el árbol generado. Éste árbol tiene profundidad 1 y cuenta con un nodo hoja puro, esto significa que no hay error posible y la varianza es nula. El valor de la partición es 2.5, esto es el valor que ha dado menor variancia según la heurística utilizada. También hay un nodo con el valor numérico temporal $5/3$, esto se refiere a que no es un nodo puro y por lo tanto halla más de un resultado para una misma condición >2.5 , si el usuario desea cortar el árbol en esta etapa, $5/3$ será el valor de la respuesta. Éste valor indica que de cinco posibles muestras quedan tres para clasificar, así que se hace el promedio con la finalidad de simplificar la construcción del árbol. En la tabla 3.4 se muestra el resultado de la segunda partición, en la figura 3.10 se puede ver como el valor temporal de la figura 3.9 ha pasado a ser otro subárbol. Esta es la etapa final en la construcción de esta estructura puesto que el árbol es puro. Se puede observar que el algoritmo ha descartado el segundo atributo, esto significa que éste no aporta nada. Así que la técnica ha sido capaz de reducir en una dimensión los datos de entrada.

En el ejemplo anterior se ha visto paso a paso el funcionamiento de ésta estructura. No ha habido ningún problema debido al reducido tamaño de la base de datos, no obstante el algoritmo puede verse afectado si el número de muestras es gigantesco. Uno de los posibles problemas sería el tamaño descomunal del árbol y el tiempo dedicado a la construcción de éste. Para ello, se utilizan métodos para finalizar el proceso de la construcción del árbol basados en limitar (1) el número de nodos, (2) la profundidad del árbol o (3) la pureza del mismo.

El coste computacional del árbol de decisión binario en promedio es $O(n \log_2 n)$ ya que en cada iteración los datos son particionados en dos trozos distintos [Breiman et al., 1984]. De éste modo, en cada iteración sólo se pueden tomar dos opciones posibles.

En la figura 3.11 se muestra un árbol binario en el cual cada nodo interno dispone de dos nodos hijos y de regresión ya que los nodos hoja o valores esperados son números reales. Modela la base de datos *Iris* [Bache and Lichman, 2013], ésta contiene 50 muestras de tres especies distintas de *Iris* (*Iris setosa*, *Iris virginica* e *Iris versicolor*). En cada una de estas se analizan cuatro atributos (*longitud sépalo*, *anchura sépalo*, *longitud pétalo* y *anchura pétalo*). El árbol tiene profundidad 4 y dispone de 6 nodos hoja. En este caso el algoritmo ha reducido dos dimensiones, los atributos longitud sépalo y anchura sépalo no aportan información útil. El árbol típicamente se puede interpretar en profundidad *depth first* (figura 3.12) o en anchura *bread first* (figura 3.13). Al final de esta sección se muestra el pseudocódigo de alto nivel de un árbol binario de regresión, algoritmos 1 y 2.

Algoritmo 1 Construcción de un árbol binario de regresión

```

1: Requiere dataSet  $\mathcal{D}$  con los datos debidamente cargados.
2: función Generar_árbol_regresión(  $\mathcal{D}$  ) retorna ÁRBOL_BINARIO
3:    $\mathcal{A} = \text{ÁRBOL\_BINARIO.crear\_árbol\_binario}()$ 
4:   Construir_árbol(  $\mathcal{A}, \mathcal{D}$  )
5:   para todo nodo hoja  $i$  hacer
6:      $\mathcal{A}.\gamma_i = \text{calcular\_gamma}(i)$  {Usando la ecuación 3.28}
7:   fin para todo
8:   retorna  $\mathcal{A}$ 
9: fin función

```

Algoritmo 2 Generación del modelo

```

1: Requiere dataSet  $\mathcal{D}$ , árbol  $\mathcal{A}$  y los límites  $\ell$ .
2: procedimiento Construir_árbol(  $\mathcal{D}, \mathcal{A}, \ell$  )
3:   si no hay instancias  $i$  entonces
4:     finalizar
5:   fin si
6:   para todo nodo hoja  $i$  hacer
7:     calcular_ganancia_varianza_global(  $\mathcal{D}, \mathcal{A}, i$  )
8:   fin para todo
9:   si profundidad(  $\mathcal{A}$  )  $\geq \ell_{\text{profundidad}} \vee$  número_nodos(  $\mathcal{A}$  )  $\geq \ell_{\text{número\_nodos}}$  entonces
10:    finalizar
11:   fin si
12:   para todo atributo  $a$  hacer
13:     preparar_corte_arbol(  $\mathcal{D}, a$  )
14:     calcular_ganancia_varianza(  $\mathcal{D}, \mathcal{A}, a$  )
15:      $ma = \text{mejor\_atributo\_según\_varianza}( \mathcal{D}, \mathcal{A}, a )$ 
16:   fin para todo
17:   si ganancia_varianza $_{ma} > 0$  entonces
18:      $\mathcal{D}_{\text{rama\_izquierda}} = \mathcal{D}_{\text{cortar}}( \mathcal{D}, ma )$ 
19:      $\mathcal{D}_{\text{rama\_derecha}} = \mathcal{D}_{\text{cortar}}( \mathcal{D}, ma )$ 
20:     Construir_árbol(  $\mathcal{D}_{\text{rama\_izquierda}}, \mathcal{A}$  )
21:     Construir_árbol(  $\mathcal{D}_{\text{rama\_derecha}}, \mathcal{A}$  )
22:   fin si
23: fin procedimiento

```

3.2. Técnicas avanzadas de clasificación

Llegados a este punto se han expuesto los principales clasificadores del Aprendizaje Automático, los cuales son capaces de construir un modelo generalizando la información más importante de un *data set* para posteriormente predecir una salida. Continuamente se están realizando investigaciones sobre nuevas técnicas que permitan obtener clasificadores más certeros, una de las más utilizadas recientemente se denomina *Ensemble* [James et al., 2013]. Los métodos *Ensemble* se caracterizan por combinar las predicciones de múltiples clasificadores obteniendo así una mayor precisión a uno individual. Las dos familias más populares son *Bagging* y *Boosting* [Friedman, 2000, James et al., 2013, Okun and Valentini, 2009]. Éstas han demostrado que son muy efectivas especialmente con árboles de decisión [Friedman, 2000].

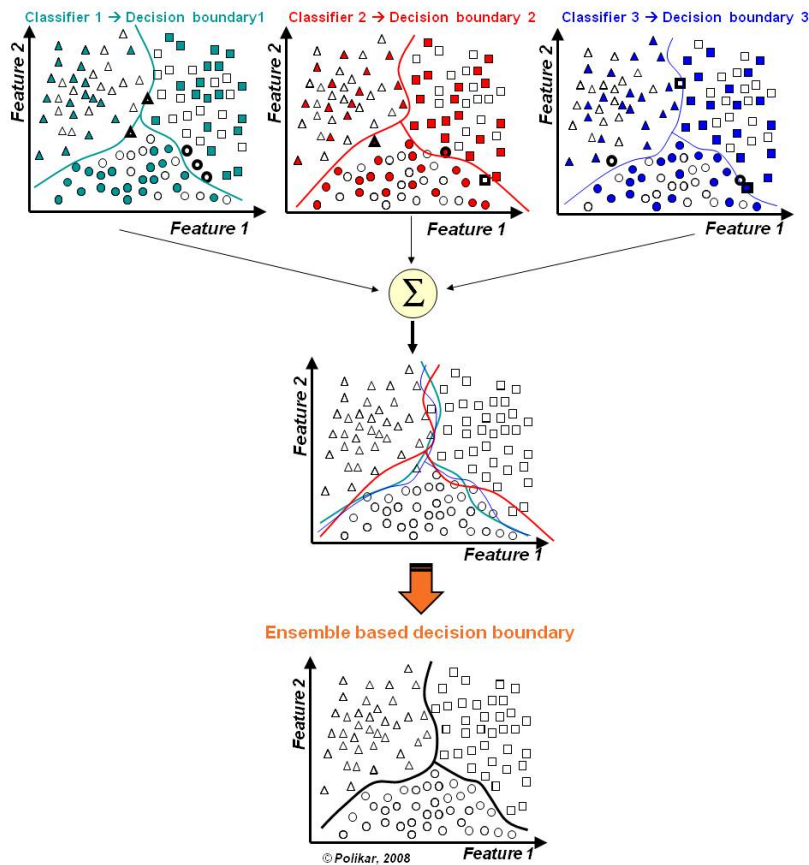


FIGURA 3.14: Método *Ensemble* [Polikar, 2008]. El *data set* tiene dos atributos distintos y el sistema utiliza tres sistemas clasificadores para modelar la información.

La figura 3.14 ilustra el funcionamiento de un problema de clasificación resuelto mediante un método *Ensemble*. Aplica tres sistemas clasificadores, cada uno de estos

modela la información generando el espacio de clasificación mostrado en las gráficas en función de los atributos de entrada $Feature_1$ y $Feature_2$. Finalmente se combinan las predicciones de cada miembro para efectuar una decisión colectiva obteniendo así el modelo representado por la gráfica inferior.

3.2.1. Bagging

Las técnicas Empaquetado, *Bagging* o *Bootstrap Aggregating* son muy utilizadas actualmente debido a que están diseñadas específicamente para minimizar el sobreajuste. Esta técnica estadística permite obtener una aproximación muy acertada a la distribución real que siguen los datos.

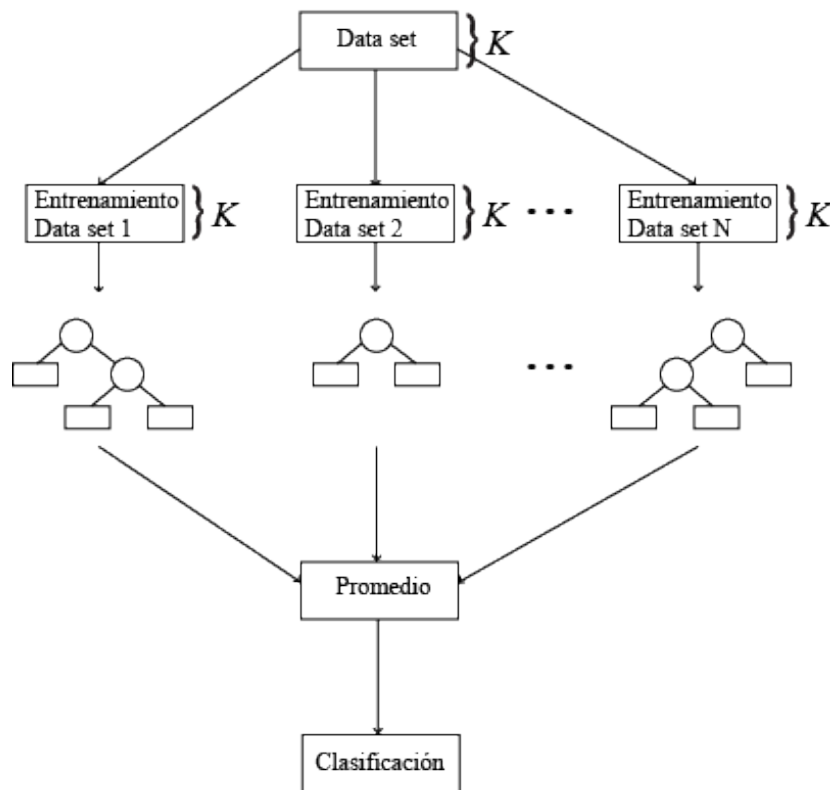


FIGURA 3.15: Método *Bagging*.

En la figura 3.15 se muestra el funcionamiento de *Bagging*. Dado un conjunto de datos de K ejemplos, se generan N *data sets* del original con reemplazo (esto significa que pueden haber elementos repetidos dentro de un mismo *data set*), a cada uno de estos se les aplica un algoritmo de clasificación o regresión, en este caso se trata de árboles de decisión y posteriormente se combinan generando así el clasificador final. Con éste método se consigue evitar el sobreajuste.

Así, se entrenan los distintos clasificadores con sus correspondientes *data sets* y se promedian los resultados, por ejemplo a través de votaciones, cada clasificador es independiente y por lo tanto altamente paralelizable. El algoritmo más famoso que utiliza *Bagging* es el *Random Forest* [Breiman, 2001]. Se trata de un conjunto de árboles de decisión profundos contruidos a partir de muestras generadas por *Bagging* que finalmente se combinan para dar una solución con una alta tasa de acierto [Delgado et al., 2014]. Un caso real se halla en el dispositivo *Kinect* desarrollado por *Microsoft* inicialmente para la videoconsola *Xbox 360*. Éste permite a los usuarios controlar e interactuar sin necesidad de tener contacto físico con un controlador de videojuegos tradicional. Hace uso de un algoritmo *Random Forest* para predecir de forma tridimensional, partiendo de una imagen de profundidad, las partes del cuerpo humano sin utilizar información temporal [Shotton et al., 2013].

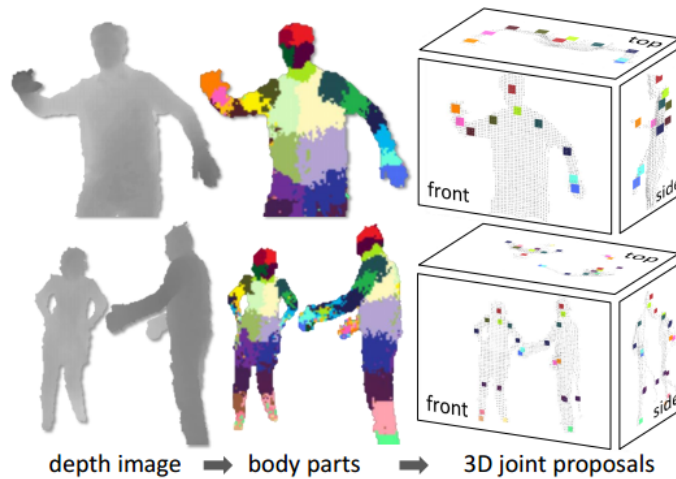


FIGURA 3.16: *Kinect* clasifica las partes del cuerpo mediante el algoritmo *Random Forest* [Shotton et al., 2013].



FIGURA 3.17: Muestra el entrenamiento y predicción de usuarios sintéticos a la izquierda y las predicciones con usuarios reales a la derecha [Shotton et al., 2013].

3.2.2. Boosting

A diferencia del *Bagging*, donde cada clasificador es independiente y finalmente se elige una solución por consenso, en el *Boosting* lo que se busca es generar una mejora continua, esto es, la salida de un clasificador se verá mejorada por el clasificador generado en la iteración siguiente, pues la salida del primero resultará la entrada del segundo. Cada clasificador que se obtiene se denomina débil (*weak Learner*) ya que su error es un poco inferior al 50 %, es decir, es ligeramente mejor que una elección al azar, en el caso de utilizar un árbol, la profundidad del mismo suele ser pequeña. Al combinarlos, resulta en un clasificador fuerte (*strong learner*) con un error muy bajo. La primera técnica en utilizar Boosting es *AdaBoost*. La finalidad de *AdaBoost* es construir un clasificador fuerte $H(x)$ a partir de una combinación lineal de clasificadores simples $h_i(x)$ [Mukherjee et al., 2013], expresiones 3.24 y 3.25.

$$f(x) = \sum_{t=1}^T \alpha_t h_t(x) \quad (3.24)$$

$$H(x) = \text{sign}(f(x)) \quad (3.25)$$

La técnica *Adaboost* fué el primer clasificador en utilizar *Boosting*, sólo hay que afinar el número de iteraciones (el parámetro T), no requiere de un conocimiento previo del clasificador débil y puede combinar distintos modelos entre ellos árboles de decisión y redes neuronales. Por contra es muy susceptible al ruido y fácilmente se alcanza una situación de sobreajuste.

El método *Gradient Boosting* es una evolución de *Adaboost* y pretende obtener mayor precisión [Friedman, 2000]. Esto lo hace mediante el cálculo de los residuos, es decir, la diferencia entre el valor real y y el valor aproximado \hat{y} , esta diferencia se denomina *loss-function* $L(y, F(x))$. La función de pérdida más común es la siguiente, expresión 3.26.

$$L(y, F(x)) = \frac{1}{2} \sum_{k=1}^{\text{ins}} (y_k - \hat{y}_k)^2 \quad (3.26)$$

Donde *ins* es el número de instancias.

A continuación se minimizan los residuos utilizando un gradiente como se muestra en la siguiente fórmula, expresión 3.27.

$$r_{im} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \quad (3.27)$$

Donde instancia $i = 1, \dots, n$ y árbol $m = 1, \dots, M$.

La idea fundamental del algoritmo es que estos residuos tiendan a cero (no haya error).

Sepal length	Sepal width	Petal length	Petal width	Clase	Residuo
5.1	3.5	1.4	0.2	(1) Iris-Setosa	[1,0,0]
7	3.2	4.7	1.4	(2) Iris-Versicolor	[0,1,0]
6.3	3.3	6	2.5	(3) Iris-Virginica	[0,0,1]
4.9	3	1.4	0.2	(1) Iris-Setosa	[1,0,0]

TABLA 3.5: Fragmento del *data set* *Iris* formado por cuatro instancias. Este *data set* está formado por los atributos *Sepal length*, *Sepal width*, *Petal length*, *Petal width* y la clase (*Iris-Setosa*, *Iris-Versicolor* y *Iris-Virginica*). La columna residuo mapea cada una de las clases.

En la tabla 3.5 se muestra un *data set* compuesto por cuatro instancias del *data set* *Iris*. Cada instancia contiene cuatro atributos (*Sepal length*, *Sepal width*, *Petal length* y *Petal width*) y su etiqueta o clase correspondiente (*Iris-Setosa*, *Iris-Versicolor* o *Iris-Virginica*). El residuo mapea cada una de estas clases del siguiente modo, es un *array* real de tantas casillas como número de clases (en este caso 3) inicialmente con todas las casillas a cero excepto la que corresponde al número de la clase, el valor de esta será la unidad (*One versus all*). El residuo se verá modificado a la vez que se genera el árbol de regresión mediante el cálculo de la predicción *gamma* (expresión 3.28) tal y como se ha visto anteriormente en este mismo capítulo en el pseudocódigo del apartado de los árboles de decisión [James et al., 2013].

$$\gamma = \eta \frac{c-1}{c} E \quad (3.28)$$

Donde η es el ratio de aprendizaje, c es el número de clases y E el valor esperado.

De este modo el *Gradient Boosting* puede utilizar árboles de regresión para problemas de clasificación. En lo que sigue se muestra el pseudocódigo a alto nivel de la fase de aprendizaje (algoritmo 3) y predicción (algoritmo 4) de un *Gradient Boosting* que utiliza árboles de regresión. Se utilizan árboles de decisión ya que dan una mayor precisión con esta técnica pese a su coste relativamente elevado (en caso de que se configuren muchos árboles por clase) en la fase de entrenamiento ya que su coste a la hora de predecir es muy bajo. Es por ello que almacenan en disco el modelo y entonces la predicción es muy rápida.

Algoritmo 3 Fase de entrenamiento de *Gradient Boosting*

```

1: Requiere matrizÁrboles  $\mathcal{MA}$ , dataSet  $\mathcal{D}$  y fichero  $\mathcal{F}$ .
2: procedimiento Entrenamiento(  $\mathcal{MA}$ ,  $\mathcal{D}$ ,  $\mathcal{F}$  )
3:    $\mathcal{F}$ .abre()
4:   para todo número de árboles  $i$  hacer
5:     para todo número de clases  $j$  hacer
6:        $\mathcal{MA}_{ij} = \mathcal{A}.$ Generar_árbol_regresión(  $j$ ,  $\mathcal{D}$  )
7:        $\mathcal{MA}_{ij} = \mathcal{A}.$ Guardar(  $\mathcal{F}$  )
8:       para todo número de instancias  $k$  hacer
9:         calcular_residuals(  $k$ ,  $j$ ,  $\mathcal{MA}_{ij}.\gamma$  ) {Usando la expresión 3.27.}
10:      fin para todo
11:    fin para todo
12:  fin para todo
13:   $\mathcal{F}.$ cierra()
14: fin procedimiento

```

Algoritmo 4 Fase de predicción de *Gradient Boosting*

```

1: Requiere matrizÁrboles  $\mathcal{MA}$ , dataSet  $\mathcal{D}$  y fichero  $\mathcal{F}$ .
2: procedimiento predicción( $\mathcal{MA}$ ,  $\mathcal{D}$ ,  $\mathcal{F}$ )
3:    $\mathcal{F}.$ leer()
4:   clasePredicha, prediccionesFallidas, claseReal = 0
5:   para todo número de instancias  $i$  hacer
6:     ejemplo =  $\mathcal{D}.$ coger_atributos(  $i$  )
7:     claseReal =  $\mathcal{D}.$ coger_clase(  $i$  )
8:     para todo número de árboles  $a$  hacer
9:       para todo número de clases  $k$  hacer
10:         $prediccionClase_k = \mathcal{MA}_{ak}.\gamma(\text{ejemplo})$ 
11:      fin para todo
12:    fin para todo
    {Este proceso se denomina Soft Max. Permite pasar de un rango arbitrario a una
    probabilidad, es decir, un número entre 0 y 1.}
13:     $probabilidadClase = \text{round} \left( \text{argmáx}_j \frac{e^{\text{probabilidadClase}_j}}{\sum_{k=0}^{\text{númeroClases}} \text{probabilidadClase}_k} \right)$ 
14:     $matrizConfusion_{\text{clasePredicha}, \text{claseReal}} = \text{matrizConfusion}_{\text{clasePredicha}, \text{claseReal}} +$ 
    1
15:    si clasePredicha  $\neq$  claseReal entonces
16:      prediccionesFallidas = prediccionesFallidas + 1
17:    fin si
18: fin procedimiento

```

En el algoritmo 4 se introduce la matriz de confusión. Esta matriz es una herramienta que indica que clases se están confundiendo. Comúnmente las columnas indican como se ha clasificado y las filas la clase a la que pertenece.

	Iris-Setosa (Clasificado)	Iris-Versicolor (Clasificado)
Iris-Setosa (Pertenece) (P)	49 (VP)	1 (FP)
Iris-Versicolor (Pertenece) (N)	6 (FN)	44 (VN)

TABLA 3.6: Matriz de confusión, fragmento del *data set* de *Iris* con 100 instancias. Cada columna representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

La tabla 3.6 muestra una matriz de confusión de un *sub data set* de *Iris* con dos clases. En ella puede observarse como una instancia perteneciente a *Iris-Setosa* se ha clasificado como *Iris-Versicolor* y seis instancias pertenecientes a la clase *Iris-Versicolor* se han clasificado como *Iris-Setosa*. De este modo, una matriz de confusión sin error en un *data set* con las clases balanceadas (misma proporción de instancias para ambas clases) es aquella donde todas las celdas son cero excepto la diagonal donde cada valor es el número de instancias dividido por el número de clases, en este caso el valor ideal es 50. Existe un *Framework* denominado *Característica Operativa del Receptor* o *Receiver Operating Characteristic (ROC)* para un sistema clasificador binario que mide como de preciso es el sistema. Éste define verdaderos positivos (*VP*) como éxitos, verdaderos negativos (*VN*) como rechazos correctos, falsos positivos (*FP*) como falsas alarmas o error tipo *I* y falsos negativos (*FN*) como error tipo *II*. Esto permite obtener una medida estadística del acierto del clasificador empleando las siguientes expresiones.

1. Ratio de verdaderos positivos, *recall* o *sensitivity* (*VPR*), expresión 3.29.

$$VPR = \frac{VP}{VP + FN} \quad (3.29)$$

2. Ratio de falsos positivos o *fallout* (*FPR*), expresión 3.30.

$$FPR = \frac{FP}{FP + VN} \quad (3.30)$$

3. Especificidad (*SPC*), expresión 3.31.

$$SPC = 1 - FPR \quad (3.31)$$

4. Valor predictivo positivo o *precision* (*PPV*), expresión 3.32.

$$PPV = \frac{VP}{VP + FP} \quad (3.32)$$

5. Valor predictivo negativo (NPV), expresión 3.33.

$$NPV = \frac{VN}{VN + FN} \quad (3.33)$$

6. Ratio de falsos descubrimientos (FDR), expresión 3.34.

$$FDR = \frac{FP}{FP + VP} \quad (3.34)$$

7. 0/1 Accuracy (ACC), expresión 3.35.

$$ACC = \frac{VP + VN}{VP + FP + FN + VN} \quad (3.35)$$

8. Puntuación F o F Measure ($F1$), expresión 3.36.

$$F1 = \frac{2VP}{2VP + FP + FN} \quad (3.36)$$

9. Coeficiente de correlación de Matthew (MCC), expresión 3.37.

$$MCC = \frac{VPVN - FPFN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}} \quad (3.37)$$

Gradient Boosting es una técnica que se fundamenta en uno de los paradigmas con más acierto que existe actualmente dentro del estado del arte del Aprendizaje Automático, usa el concepto de *Ensemble* para mejorar iterativamente los resultados obtenidos en las anteriores iteraciones. Es una de las técnicas con más acierto que hay ahora mismo en la bibliografía y se fundamenta en unas técnicas estadísticas muy sólidas. Por contra, estas técnicas tienden a resultar en *overfitting*. Normalmente se utilizan con árboles de decisión, el coste computacional de todos estos es elevado, pero se mejora paralelizando y ejecutándolo de forma distribuida. Las técnicas *Gradient Boosting* han cobrado mucha importancia recientemente debido a su gran precisión y rapidez en predecir. Este algoritmo se comporta mejor utilizando árboles de decisión como base [Friedman, 2000], además de este modo se genera un modelo relativamente interpretable. Es por ello que interesa por encima de las demás técnicas estudiadas, aplicarla al terreno del campo docente.

3.3. Resumen

En el capítulo actual se han expuesto los cinco principales sistemas clasificadores y los métodos *Ensemble*. Estos últimos agrupan múltiples sistemas clasificadores generando de este modo uno más preciso, se comportan mejor con árboles de decisión

como base y las dos principales familias son *Bagging* y *Boosting*. *Bagging* se caracteriza por generar *data sets* a partir del original con el mismo número de instancias elegidas al azar (con reemplazo). Entonces se aplica un clasificador para cada uno de estos y finalmente se someten a un proceso de votación. *Random Forest* es el algoritmo más conocido. A diferencia de *Bagging*, *Boosting* construye *weak learners* iterativamente hasta generar el *strong learner* o modelo final. Esta técnica se basa en la mejora continua, es decir, cada *weak learner* mejora al anterior y para predecir se tienen todos estos en cuenta. *Gradient Boosting* consigue ser el algoritmo más preciso de esta familia. La técnica que más se adapta a los objetivos planteados, esto es modelar el rendimiento del alumnado, es *Gradient Boosting* utilizando árboles de decisión como base ya que se trata de una técnica muy precisa y resulta relativamente interpretable.

En el siguiente capítulo se especifican los requisitos de software, metodología y el diseño del sistema.

4

Desarrollo del sistema

En los capítulos anteriores se ha descrito el concepto de Aprendizaje Automático y se han presentado distintas técnicas; desde sistemas clasificadores hasta métodos *Ensemble*. Estos últimos generalmente se utilizan con árboles de decisión [Friedman, 2000] y se caracterizan por combinar distintos clasificadores con la finalidad de ser más precisos. *Gradient Boosting* pertenece a la familia *Boosting* de los métodos *Ensemble* y es (1) una de las técnicas más precisas del momento [Goldbloom, 2010], (2) robusto frente al ruido y (3) rápido a la hora de predecir. Estas características hacen que *Gradient Boosting* sea un método muy a tener en cuenta dentro del campo *Educational Data Mining* (EDM), más concretamente para extraer conocimiento de datos docentes con la finalidad de predecir el rendimiento del alumnado. En este capítulo se detalla la especificación de requisitos del *software*, se explica la metodología de desarrollo elegida y se expone el análisis de diseño del sistema.

4.1. Especificación de requisitos *software*

La estructura de esta sección sigue el estándar IEEE-STD-830-1998 [1998]. La especificación de requisitos *software* (ERS) es una descripción del comportamiento del sistema a construir, define de manera clara y precisa todas las funcionalidades y restricciones. A continuación se describe el propósito, enfoque y la estructura de los siguientes apartados.

4.1.1. Propósito

El propósito de esta sección es indicar todas las funcionalidades de *software* de las aplicaciones *GBM* y *PGUI*.

4.1.2. Enfoque

El sistema *GBM* se crea con la finalidad de modelar el rendimiento del alumnado. Para estudiar su comportamiento y validar el sistema con el objetivo de comparar su precisión frente a otros algoritmos, se construye la plataforma *PGUI*.

4.1.3. Estructura de la *ERS*

Esta *ERS* está compuesta principalmente por dos secciones, estas son la descripción general y los requisitos específicos del *software*.

Descripción general. Detalla la perspectiva del producto, las interfaces (sistema, usuario, *hardware* y *software*), los requisitos de rendimiento, las funcionalidades del producto, las restricciones de diseño, asunciones, dependencias y el trabajo adicional.

Requisitos específicos. Expone los requisitos funcionales, la usabilidad y el rendimiento del sistema.

4.2. Descripción general

A continuación se expone la perspectiva del producto, las interfaces, los requisitos de rendimiento, las funcionalidades del producto, las restricciones de diseño, asunciones, dependencias y finalmente el trabajo adicional.

4.2.1. Perspectiva del producto

PGUI y *GBM* son aplicaciones que permiten realizar experimentos con *Gradient Boosting*. *GBM* proporciona al usuario la posibilidad de experimentar directamente con *Gradient Boosting*; Generar un modelo a partir de datos de entrenamiento y realizar predicciones mediante éste. *PGUI* es una interfaz gráfica, se construye con el designio de facilitar al usuario el uso del programa *GBM* y permitir realizar una validación cruzada. Mediante la interfaz gráfica el usuario configura los parámetros de entrada del algoritmo y gestiona las llamadas necesarias a *GBM*.

4.2.2. Interfaces

Interfaz del sistema. El único requisito para *GBM* es un compilador de *C++* actual, siguiendo el estándar de *C++11*. *PGUI* también es independiente de la plataforma, así que el único requisito es un intérprete *Python 3.4.3* que soporte la librería gráfica *tkinter*.

Interfaz de usuario. *GBM* no tiene interfaz gráfica, funciona vía línea de comandos. El algoritmo recibe la configuración deseada mediante parámetros de entrada y una vez se ejecuta, si se configura en modo entrenamiento se guarda a disco un fichero binario con el modelo de entrenamiento del *Gradient Boosting*. *PGUI* es una interfaz gráfica para que el usuario pueda experimentar con *GBM* de forma sencilla. Su funcionamiento es similar a *Weka* [Witten et al., 2011].

Interfaz *Hardware* y *software*. Para utilizar *GBM* o *PGUI* no se necesita una interfaz *hardware*. La interfaz *software* que se necesita es un sistema operativo con un compilador *C++* estándar (por ejemplo *Visual Studio 2013* o *g++*) y un intérprete *Python*.

4.2.3. Requisitos de rendimiento

El ordenador debe contar con 2GB de memoria *RAM* y el procesador debe ser un *Pentium 4* o su equivalente en *AMD*.

4.2.4. Funcionalidades del producto

GBM es una herramienta orientada a la experimentación de *Gradient Boosting* con árboles de decisión. El algoritmo recibe mediante línea de comandos, la configuración necesaria. Las variables que componen esta configuración son las siguientes (tabla 4.1): (1) seleccionar modalidad *train* o *test*, (2) elegir la ruta del fichero *ARFF* a cargar, (3) el mínimo número de instancias por nodo hoja, (4) el máximo número nodos hoja del árbol, (5) profundidad máxima del árbol, (6) la varianza mínima, (7) el número de árboles, (8) el ratio de aprendizaje, (9) si se desea mostrar por pantalla los árboles generados y (10) si se desea realizar un entrenamiento y a continuación una predicción con el mismo fichero.

El programa debe contar con las siguientes funcionalidades.

1. Modelar los datos mediante un fichero *ARFF*.
2. Aplicar algoritmo *Gradient Boosting* con árboles de regresión teniendo en cuenta la configuración del usuario.
3. Generar las particiones de cada uno de estos árboles según la ganancia en varianza.

Parámetro
<i>Train</i> o <i>test</i>
Ruta del fichero <i>ARFF</i> a cargar
Mínimo número de instancias para generar nodos hoja
Máximo número de nodos hoja
Profundidad máxima
Varianza mínima
Número de árboles
Ratio de aprendizaje
Mostrar modelo por pantalla
Múltiples experimentos (<i>train</i> y <i>test</i>)

TABLA 4.1: Parámetros de configuración de *GBM*. Los valores por defecto de cada parámetro se hallan en la tabla 4.2.

4. Guardar los árboles de regresión.
5. Guardar en un fichero binario el modelo generado en la etapa de entrenamiento.
6. Realizar predicciones.
7. Calcular la matriz de confusión y da medidas del error (*ACC* y *F1*).
8. Mostrar el modelo generado.

El siguiente código ilustra los parámetros de entrada de *GBM* con el formato correcto (cada parámetro va seguido de dos guiones).

```
--train=../GradientBoosting/DataSet/iris.arff
--minimumnumberofinstancesperleaf=4
--maximumnumberofleaves=0
--maximumdepth=3
--minimumvariance=0.00001
--numberoftrees=3
--learningrate=0.1
--verbose=T
--multipleexperiments=T
```

Se carga en modo *train* el *data set Iris*, se limita el algoritmo a cuatro instancias por nodo hoja y a profundidad 3 (si el algoritmo recibe un 0 no hará caso de la restricción, caso ilustrado por el máximo número de nodos hoja en el ejemplo anterior). La varianza mínima es 0.00001, el ratio de aprendizaje 0.1, los arboles generados se mostrarán por pantalla y el algoritmo realiza más de un experimento, es decir, automáticamente hace

una predicción una vez ha construido el modelo. Si el usuario no introduce alguno de los parámetros, estos se usan en sus valores por defecto (exceptuando el *data set*).

PGUI es una interfaz gráfica para que el usuario no familiarizado con el uso de la línea de comandos pueda utilizar *GBM* y realizar una validación cruzada. A continuación se listan las funcionalidades de la interfaz gráfica.

1. Configurar los parámetros de entrada para el algoritmo GBM.
2. Visualizar el modelo generado.
3. Realizar una *stratified 10 fold cross validation*.
4. Visualizar la matriz de confusión y las medidas del error.
5. Calcular la desviación típica en las medidas del error en la validación cruzada.

4.2.5. Restricciones de diseño, asunciones y dependencias

GBM y *PGUI* no tienen restricciones de diseño. *GBM* se programa en *C++*. Esto significa que se requiere instalar un compilador *C++* estándar. La plataforma *PGUI* se construye en *Python*, así que se necesita un intérprete *Python*.

4.2.6. Trabajo adicional

El proceso de creación de árboles de forma iterativa en *GBM* es una tarea muy costosa, en un futuro se puede paralelizar la creación de los *weak learners* para minimizar el tiempo de ejecución del algoritmo.

4.3. Requisitos específicos

En lo que sigue se muestran los requisitos específicos del diseño del *software*.

4.3.1. Requisitos funcionales

En esta sección se presenta una breve descripción de cada uno de los requisitos funcionales que definen la expectativa del usuario en el sistema.

El sistema *GBM* debe permitir al usuario:

1. Configurar todos los parámetros de entrada sin la necesidad de recompilar todo el código fuente. Si algún parámetro no se proporciona, éste tomará su valor por defecto (ver tabla 4.2).

Parámetro	Valor por defecto
<i>Train</i> o <i>test</i>	Test
Ruta del fichero <i>ARFF</i> a cargar	“
Mínimo número de instancias para generar nodos hoja	4
Máximo número de nodos hoja	0
Profundidad máxima	1
Varianza mínima	0.000001
Número de árboles	10
Ratio de aprendizaje	0.1
Mostrar modelo por pantalla	Falso
Múltiples experimentos (<i>train</i> y <i>test</i>)	Falso

TABLA 4.2: Parámetros de configuración por defecto de *GBM*.

2. Experimentar con *Gradient Boosting* sobre cualquier *data set* del tipo *ARFF*. En caso de que haya algún campo vacío o valor no deseado, el algoritmo lo gestionará automáticamente.
3. Generar un modelo a partir de los datos de entrenamiento y almacenarlo a disco en un fichero binario nombrado *model.bin*.
4. Mostrar por pantalla el modelo generado por los árboles de regresión.
5. Realizar predicciones basándose en el modelo anteriormente generado.
6. Medir el error de cada predicción mediante la matriz de confusión, precisión (*ACC*) y puntuación *F* (*F1*).

El sistema *PGUI* debe permitir al usuario:

1. Configurar los parámetros de entrada del programa *GBM* y utilizar el algoritmo de forma cómoda.
2. Evaluar la precisión del algoritmo *GBM* aplicando *stratified 10 fold cross validation*. Se realizan 10 experimentos (pares *train-test*) tanto para la fase de entreno como para la predicción [Orriols-Puig \[2008\]](#).
3. Mostrar por pantalla la desviación típica de las medidas de error en la validación cruzada.
4. Visualizar por pantalla la matriz de confusión y el error mediante *ACC* y *F1*.
5. Visualizar por pantalla el modelo generado.

4.3.2. Usabilidad

Como *GBM* es un algoritmo que funciona mediante línea de comandos, es decir, el usuario tiene la libertad de elegir la configuración necesaria tecleando los parámetros de entrada, no se han establecido requisitos de usabilidad. *PGUI* debe ser lo más parecido a la plataforma *Weka* [Witten et al., 2011] ya que es muy conocida por la comunidad de la Minería de Datos.

4.3.3. Rendimiento

El algoritmo *Gradient Boosting* a implementar hace uso de árboles de decisión, esto implica que iterativamente el programa genera árboles de poca profundidad (*weak learners*) hasta construir el modelo final (*strong learner*), este proceso puede comportar un alto coste computacional. Es por ello que *C++* resulta un lenguaje orientado a objetos idóneo debido a su alto rendimiento. Para asegurar un *software* eficiente, se instala *Debian* en una máquina virtual con el siguiente *software* libre: *GDB*, *Valgrind* y *Gprof*.

GDB es el debugger de *GNU*, permite depurar los programas realizados en *C++*.

Valgrind es un simulador de sistema operativo el cual permite capturar errores de memoria como fugas de memoria y errores de la escritura o lectura en punteros.

Gprof es un *profiler*, permite identificar cuellos de botella en el programa.

PGUI se encarga de interactuar con el usuario y ejecutar *GBM* según sus preferencias.

4.4. Metodología

En la sección anterior se ha descrito la Especificación de Requisitos *Software* (*ERS*), a continuación se muestra la metodología de desarrollo que se ha elegido.

Tomando en consideración las características del sistema a desarrollar así como la naturaleza del presente trabajo, se ha decidido seguir un desarrollo en espiral para la construcción del sistema. Este tipo de modelo permite tener en cuenta el riesgo que aparece al momento de desarrollar *software*, se (1) evalúan diferentes alternativas de procesos, (2) selecciona el riesgo más asumible y (3) hace un ciclo de la espiral. Así sucesivamente hasta que el *software* diseñado no necesite mejorarse [Boehm, 1986] (ver figura 4.1).

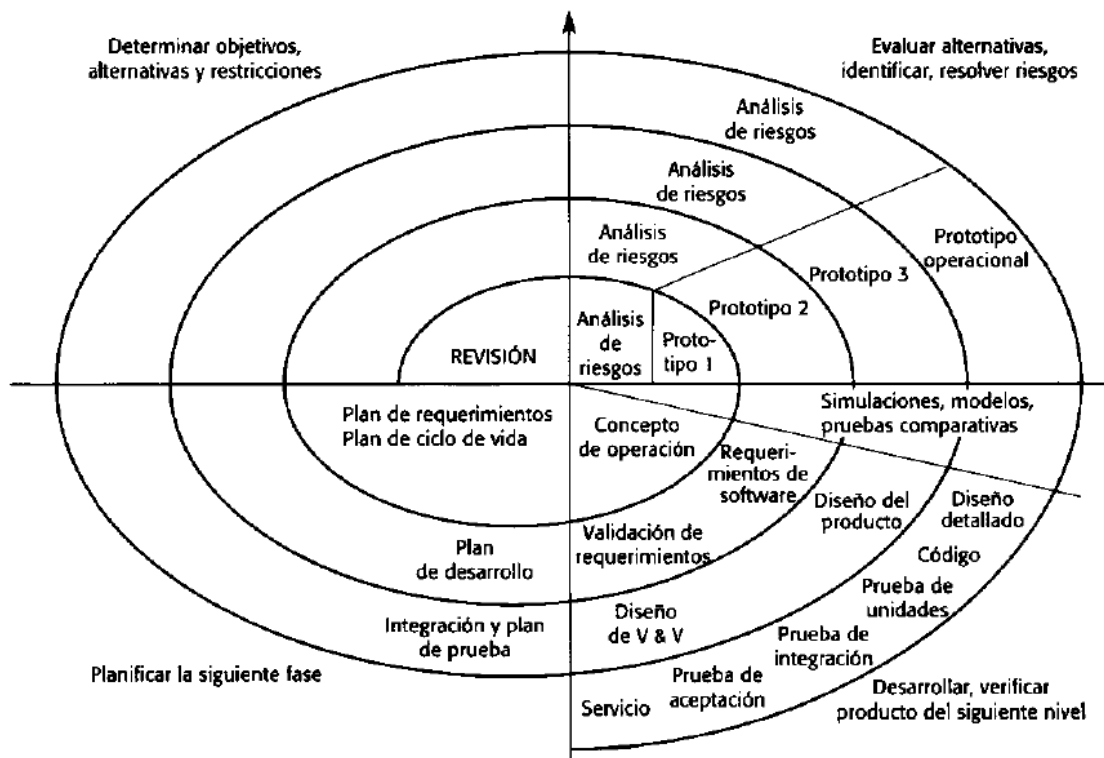


FIGURA 4.1: Modelo en espiral de Boehm [1986].

En una primera iteración de la espiral se diseña el sistema *GBM* desglosándolo en siete partes, estas proporcionan todos los requisitos necesarios para que este funcione. Las siete partes son (1) cargar el *data set ARFF*, (2) actualizar configuración del algoritmo según los parámetros de entrada del usuario, (3) inicializar arboles de regresión, generar modelo y guardarlo a disco, (4) cargar modelo y realizar predicciones, (5) implementación de *Gradient Boosting*, (6) calcular resultados de la predicción, (7) creación de una interfaz gráfica que permita utilizar *GBM* y realizar *cross validation*. De todas estas alternativas, el riesgo más asumible para comenzar a construir el algoritmo es la carga del *data set*, un algoritmo encargado de cargar campo a campo todo el fichero *ARFF* a variables del programa (memoria *RAM*) garantizando que los datos son leídos correctamente, por ejemplo si se lee un entero en un campo, asegurar que todos los datos de ese campo sean del mismo tipo.

En una segunda iteración se implementa un árbol de regresión para modelar los datos de entrada guardándolos a disco y predecir una salida. Para ello, previamente se implementa un árbol binario para estudiar su comportamiento y se utiliza para realizar búsquedas, se trata de una estructura de datos recursiva que

imita la forma de un árbol, es decir, un conjunto de nodos conectados. En esta iteración se ve muy clara la utilidad del ciclo de vida en espiral puesto que se tiene que añadir a la clase encargada de cargar los datos *ARFF* nuevas funcionalidades como por ejemplo ordenar datos según un atributo para poder implementar correctamente el árbol.

En una tercera iteración de la espiral se construye el *Gradient Boosting*, de nuevo se debe añadir funcionalidades a la clase encargada de cargar el *data set* como el cálculo de los residuos y a la clase que gestiona los árboles de regresión como por ejemplo el cálculo de la *gamma* a medida que el árbol se genera.

En una cuarta iteración se implementa (1) el cálculo los resultados de la predicción a partir de la matriz de confusión y (2) la posibilidad de que el usuario pueda modificar la configuración del algoritmo mediante parámetros introducidos vía línea de comandos.

En la quinta y última iteración se desarrollado una interfaz gráfica *PGUI* que permite facilitar al usuario el uso de *GBM* y comprobar la precisión del algoritmo mediante una validación cruzada completa.

4.5. Análisis y diseño

Esta sección muestra los aspectos más remarcables del sistema *GBM* y la plataforma *PGUI* pertenecientes a las fases de análisis y diseño.

En el sistema *GBM* (figuras 4.2 y 4.4), el usuario introduce vía comandos la ruta del *data set* y la configuración del algoritmo, en caso de que la ruta sea errónea el algoritmo finaliza y si hay algún parámetro que no se ha introducido, éste toma su valor por defecto. Según la configuración de entrada, el algoritmo realiza la fase de entrenamiento o predicción. En la fase de entrenamiento el algoritmo hace tres tareas distintas, primero carga el archivo *ARFF* a memoria *RAM*, seguidamente modela iterativamente el *data set* mediante *weak learners* de tal forma que la salida de cada uno es la entrada del siguiente (mejora continua) hasta que alguna condición de salida se cumpla. Finalmente se guarda el modelo final o *strong learner* a disco. En la fase de predicción se carga el modelo generado anteriormente, se calcula la predicción recorriendo el *strong learner* y el error mediante la matriz de confusión. Mediante esta matriz se dan las medidas precisión (*ACC*) y Puntuación *F* (*F1*) del error cometido.

La plataforma *PGUI* (figura 4.3) muestra una interfaz gráfica, en ésta el usuario puede configurar los parámetros de entrada de *GBM*, lanzar experimentos y mostrar por pantalla la salida del algoritmo y medidas del error.

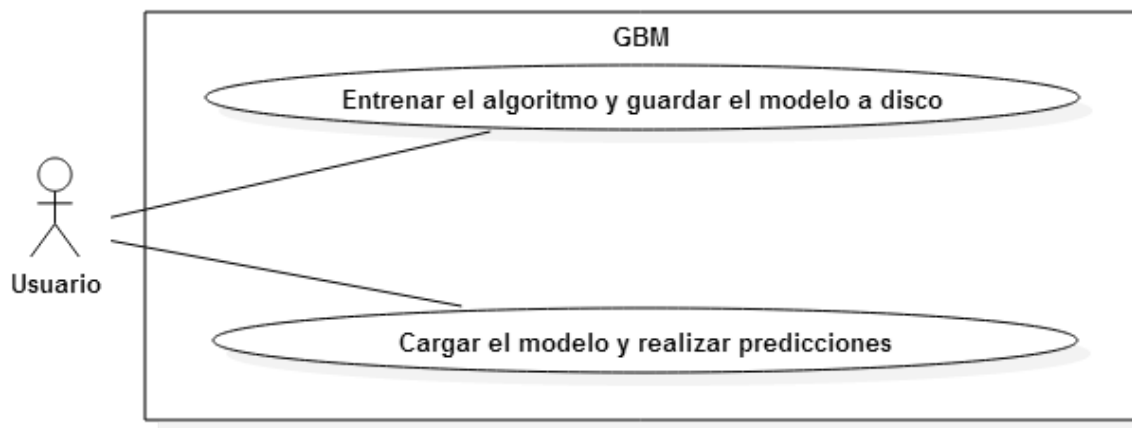


FIGURA 4.2: Diagrama de casos de uso *UML* del *software GBM*.

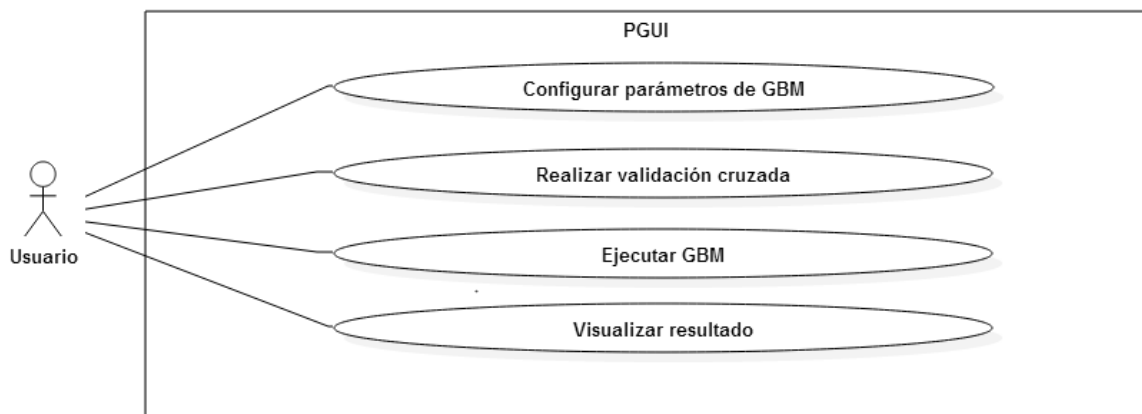
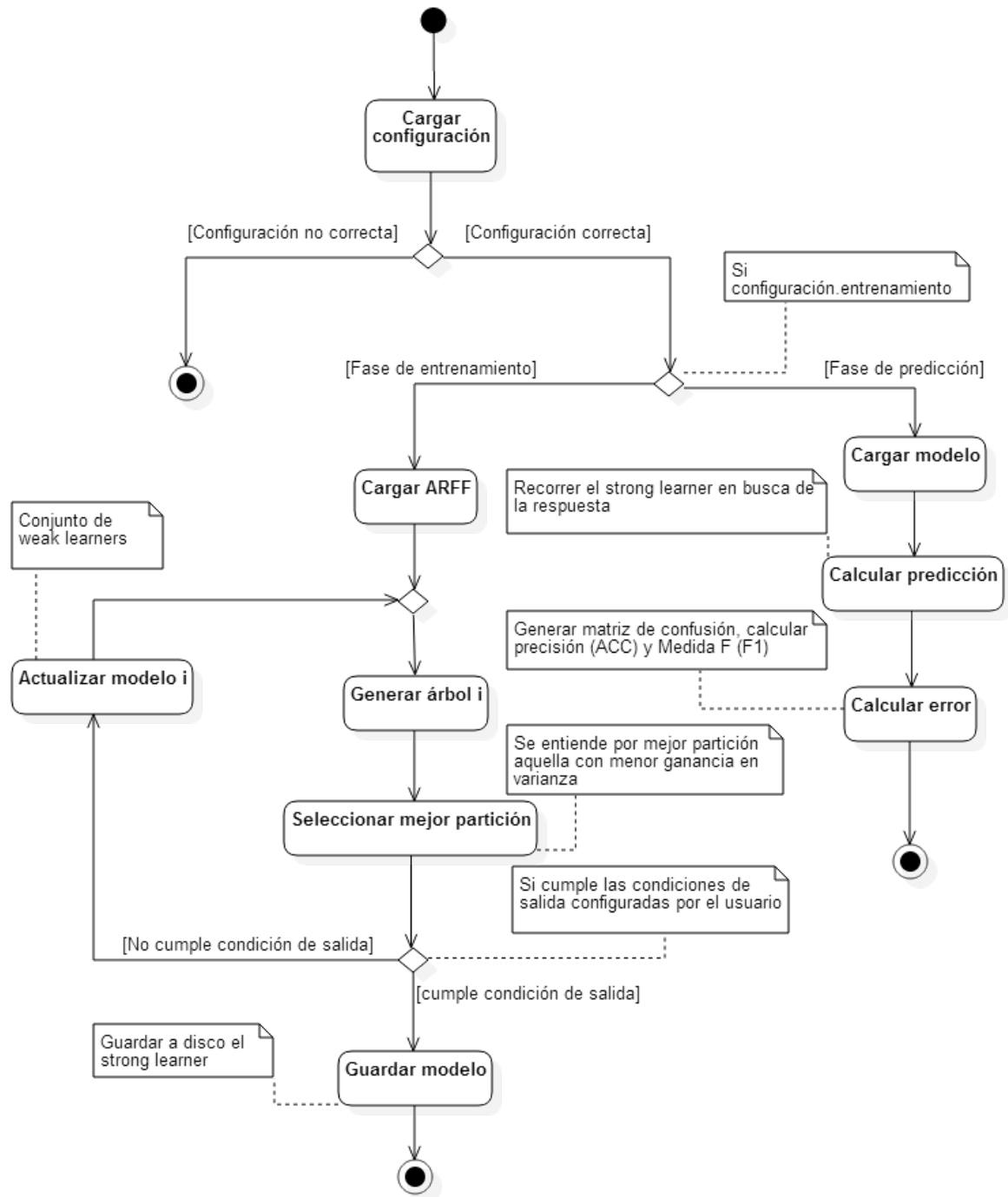
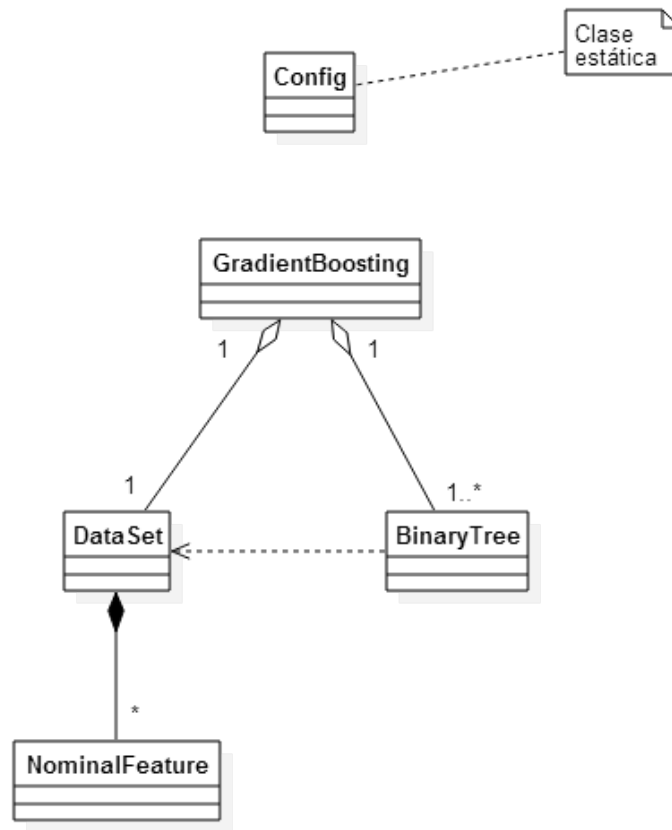
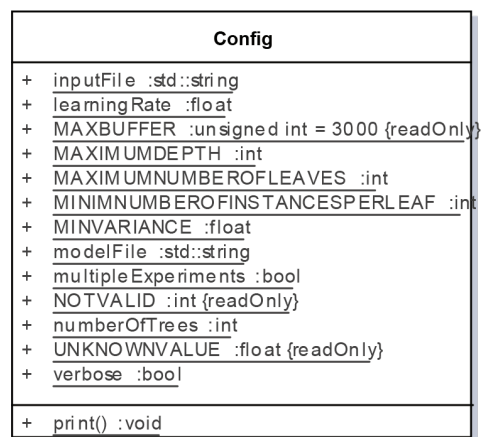


FIGURA 4.3: Diagrama de casos de uso *UML* del *software PGUI*.

En la figura 4.5 se muestra el diagrama de clases simplificado de *GBM* y posteriormente se detallan cada una de las clases que componen el diagrama.

FIGURA 4.4: Diagrama de actividades *UML* del software *GBM*.

FIGURA 4.5: Diagrama de clases *UML* simplificado de *GBM*.FIGURA 4.6: Clase *Config*.

Como se puede observar en la figura 4.5, la clase *GradientBoosting* es la principal y encargada de controlar todo el proceso tanto de carga de datos a través de la clase *DataSet* como de aplicar modelos de aprendizaje a través de la clase *BinaryTree*.

La clase *Config* (figura 4.6) es estática y se encarga de almacenar la configuración leída del fichero, de esta forma la configuración es accesible para todas las clases que componen *GBM*. Nótese que los parámetros de configuración los determina el usuario a través de comandos.

GradientBoosting	
-	boostingTrees :BinaryTree***
-	genericDataset :DataSet*
-	computePrediction(example :float*, labelPredictions :float*, labelProbabilities :float*) :int
-	computeSoftMax(labelPredictions :float*, labelProbabilities :float*) :void
-	FMeasure(confusionMatrix :int**) :float
-	FMeasureAt(classIndex :int, confusionMatrix :int**) :float
+	GradientBoosting()
+	~GradientBoosting()
-	precision(confusionMatrix :int**) :float
-	precisionAt(classIndex :int, confusionMatrix :int**) :float
-	readModelFromFile() :void
-	recall(confusionMatrix :int**) :float
+	test() :void
+	train() :void
-	truePositives(classIndex :int, confusionMatrix :int**) :float

FIGURA 4.7: Clase *GradientBoosting*.

La clase *GradientBoosting* (figura 4.7) se encarga de obtener los datos del fichero *ARFF* a través de la clase *DataSet* e inicializa los distintos árboles mediante la clase *BinaryTree*. Según la configuración se puede tratar de (1) fase de entrenamiento, donde el algoritmo genera el *strong learner* y lo guarda a un fichero binario, y (2) fase de predicción, donde carga el fichero binario anteriormente generado, calcula la predicción y da medidas del error, estas medidas son *precision*, *recall*, *F Measure* y *Accuracy*.

BinaryTree (figura 4.8) genera el modelo predictivo, este representa un árbol binario de regresión. A través del método *generateTree* se genera el mencionado árbol de manera recursiva. El método más complejo es *variableSplit*, este decide cómo particionar la variable en cada nodo del árbol. Esta clase también se encarga de guardar a disco los árboles en formato binario.

La clase *DataSet* (figura 4.9) gestiona los datos de entrada. Carga un fichero en formato *ARFF* a memoria principal (*RAM*), la clase de este *data set* se debe llamar *class*. Esta clase se asegura de que los datos leídos correspondan con los valores esperados, utiliza la clase *NominalFeature* para procesar los datos y darles un formato entendible para el algoritmo.

BinaryTree
<pre> - classLabel :int - expectedValue :float - feature :int - gamma :float - leaves :std::vector<BinaryTree*> - leftChild :BinaryTree* - numberOfRegions :int - proportionOfInstances :float ([2]) - rightChild :BinaryTree* - splitPoint :float - sumOfSQErrors :float - sumOfWeights :float + BinaryTree() + ~BinaryTree() + buildRBinaryTree(depth :int, expectedValue :float, sumOfWeights :float, minVariance :float, data :DataSet*, splitsInTheTree :int) :int + computeVariance(value :float, sumSQValue :float, weight :float) :float + deallocateMemory(data :BinaryTree*) :void + deallocateMemory(data :DataSet*) :void + deallocateMemory(data :float*) :void + deallocateMemory(data :float**, lengthOfFeatures :int) :void + deallocateTree() :void + generateTree(classLabel :unsigned int, data :DataSet*) :void + getGamma(example :float*) :float + getNumberOfRegions() :int + makeAPrediction(example :float*) :float + print(data :DataSet*) :void + print(depth :int, data :DataSet*) :void + printToFile(data :DataSet*) :void + printToFile(depth :int, data :DataSet*, pGraphvizData :std::string&) :void + readFromBinaryFile() :void + readFromBinaryFile(mFile :std::ifstream&) :void + saveToBinaryFile() :void + saveToBinaryFile(ofFile :std::ofstream&) :void + setLeaves(fraction :float) :void + variableSplit(feature :int, data :DataSet*, gainValues :float*, expectedValues :float**, SSEs :float**, totalSubSetWeights :float**, proportions :float**) :float </pre>

FIGURA 4.8: Clase *BinaryTree*.

DataSet
<pre> - classPosition :unsigned int - data :float** - features :std::vector<NominalFeature> - featureType :std::vector<std::string> - numberOfClasses :unsigned int - numberOfFeatures :unsigned int - numberOfInstances :unsigned int - residuals :float** - stack :std::stack<int> + computeResidual(exampleIndex :unsigned int, classIndex :unsigned int, value :float) :void + DataSet(fileName :std::string&) + DataSet(dataSet :DataSet*) + DataSet(feature :unsigned int, symbol :std::string&, threshold :float, dataset :DataSet*) + ~DataSet() + extractValue(line :std::string, i :unsigned int&, output :char*) :void + getClassPosition() :unsigned int + getExampleAt(index :unsigned int, example :float*) :float + getFeatureName(index :unsigned int) :std::string {query} + getNumberOfClasses() :unsigned int + getNumberOfFeatures() :unsigned int + getNumberOfInstances() :unsigned int + getNumberOfInstancesWithThresholdAt(feature :unsigned int, symbol :std::string&, threshold :float) :int + getOutputVariableAt(index :unsigned int) :float + getResidualAt(index :unsigned int, label :unsigned int) :float + getValueAt(index :unsigned int, feature :unsigned int) :float + getWeight() :float + isAComment(line :std::string&) :bool + loadData(fileName :std::string&) :void + print() :void + processNominalFeature(line :std::string&) :void + processNumericFeature(line :std::string&) :void + readData(fileName :std::string&) :void + readHeader(fileName :std::string&) :void + readLine(myFile :std::ifstream&, line :std::string&) :bool + setResidualAt(index :unsigned int, label :unsigned int, value :float) :void + sort(feature :unsigned int) :void + trim(line :std::string&) :std::string </pre>

FIGURA 4.9: Clase *DataSet*.

NominalFeature	
-	fromIntToStringMap :std::map<int, std::string>
-	fromStringToIntMap :std::map<std::string, int>
-	name :std::string
-	numberOfvalues :unsigned int
<hr/>	
+	add(featureValue :std::string&) :void
+	getIntValue(value :std::string) :int
+	getName() :std::string
+	getNumberOfValues() :unsigned int
+	getStringValue(index :int) :std::string
+	NominalFeature()
+	NominalFeature(featureName :std::string&)
+	~NominalFeature()
+	setName(name :std::string) :void

FIGURA 4.10: Clase *NominalFeature*.

NominalFeature (figura 4.10) mapea los atributos nominales a numéricos, es decir, se asegura de que los atributos nominales estén transformados en un formato entendible para el sistema.

4.6. Resumen

En este capítulo se han expuesto los requisitos *software* siguiendo el estándar [IEEE-STD-830-1998 \[1998\]](#), la metodología y el diseño del sistema *GBM* y su interfaz gráfica *PGUI*.

GBM es una implementación de la técnica *Gradient Boosting* con árboles de binarios de regresión como base, recibe la configuración vía línea de comandos, genera un modelo a partir de los datos de entrenamiento y realiza predicciones mediante éste. *PGUI* es una interfaz gráfica que facilita al usuario el uso de *GBM*, permite realizar una *stratified 10 fold cross validation* y obtener medidas del error.

En el siguiente capítulo se aplica *GBM* en *data sets* complejos y se compara su rendimiento frente otros métodos, para ello se hace uso de *PGUI*.

5

Experimentación

En el capítulo anterior se han especificado los requisitos software y se ha presentado el diseño del sistema. El objetivo del presente capítulo es aplicar el algoritmo *GBM* en *data sets* complejos y comparar su rendimiento frente otros métodos. Para realizar todos estos experimentos de forma correcta es necesario un buen particionado de cada *data set*, a este proceso se le denomina validación cruzada o *cross validation*.

5.1. Validación cruzada

Data Complexity Library (*DCoL*) es una herramienta creada por Albert Orriols y Núria Macià que proporciona una implementación de medidas de la complejidad de cada *data set* [Ho and Basu, 2002] y que además permite realizar ciertas manipulaciones estadísticas, como un particionado *cross validation*. La validación cruzada o *cross validation*, como se ha detallado en el capítulo 2, es una técnica que permite evaluar la validez de un modelo. *DCoL* garantiza que el particionado es estadísticamente sólido y divide cada base de datos en 10 pares *train-test*, es decir, parte un *data set* de entrada en 20 sub datasets donde la mitad se usan para entrenar y la otra mitad para predecir. A este proceso se le denomina *stratified 10 fold cross validation*. Este proceso es estratificado, esto significa que cada *fold* mantiene la misma proporción de instancias que el fichero original, así se disminuye el sesgo o *bias* lo que implica que los resultados son más fiables. Un ejemplo de sesgo tiene lugar en un fichero de entrenamiento con dos clases donde una de éstas tiene una frecuencia de aparición diez veces más alta que la

otra (aparecen diez veces más ejemplos de clase uno que de la clase dos) y sin embargo en el *test* esto no ocurre, sino que se dispone de la misma proporción de instancias para las dos clases. En este entorno se favorecerá a la primera clase pues se está teniendo un sesgo hacia esta, ya que las clases no están siendo balanceadas entre los ficheros de *train* y *test*. Esto se evita estratificando.

Esta herramienta funciona bajo sistemas operativos *Unix*.

```
./dcol -i mnk.arff -o ./CV/mnk -cv
```

El comando anterior realiza una validación cruzada mediante *DCoL*. Parte el *data set* *mnk.arff* y guarda estas particiones dentro de una carpeta llamada *CV*. El parámetro *-i* indica el fichero de entrada, *-o* el fichero de salida y *-cv stratified 10 fold cross validation*.

Para calcular como de bien funciona el sistema, se aplica *GBM* a cada uno de los pares *train-test* y finalmente se promedian los resultados. Este proceso se puede hacer de forma sencilla mediante la interfaz gráfica *PGUI*.

5.2. Validación estadística de los resultados

Una vez se obtienen los resultados de *stratified 10 fold cross validation* para cada algoritmo, interesa (1) generar un *ranking* y (2) medir si hay diferencias significativas entre estos. Para ello se utiliza el contraste de hipótesis: se trata de un test estadístico para contrastar la validez de esta. Se fundamenta en declarar una hipótesis nula consistente en que todos los algoritmos se comportan de la misma forma, entonces se tendrá que calcular una estadística que demuestre con cierta probabilidad si esto se cumple o no. Por consiguiente, es necesario configurar un nivel de significancia α , este valor indica si el resultado obtenido es debido a la aleatoriedad o sin embargo se trata de un resultado robusto. Valores típicos de nivel de significancia son $\alpha = 0,05$ y $\alpha = 0,1$ [García and Herrera, 2008].

El test estadístico más utilizado es el test de *Friedman* [Demšar, 2006]. Además, el test de Friedman permite calcular el *ranking* de cada algoritmo para cada uno de los *data sets* [Friedman, 1937]. Para cada fila, se asigna casilla a casilla un número entero según el valor que haya en esta. Esto lo hace de la siguiente forma, a la casilla con el número más elevado de la fila se le asigna un uno, al segundo más elevado un dos y así sucesivamente, en caso de empate se hace el promedio. Un ejemplo del resultado del procedimiento se muestra en la tabla 5.1.

<i>Data set</i>	GBM	IBK	SMO	J48	NB
Auto Imports Database	84.48 (1)	66.95 (4)	68.95 (3)	84.05 (2)	55.45 (5)
BUPA Liver Disorders	69.66 (1)	62.50 (3)	42.85 (5)	65.85 (2)	54.25 (4)
Glass Identification Database	70.16 (1)	69.40 (2)	52.45 (4)	66.45 (3)	45.55 (5)

TABLA 5.1: Test de *Friedman*, primer paso. Muestra la puntuación F de una serie de algoritmos (*GBM*, *IBK*, *SMO*, *J48* y *NB*) aplicados en tres *data sets* distintos (*Auto Imports Database*, *BUPA Liver Disorders* y *Glass Identification Database*).

Una vez ya se ha confeccionado esta tabla, se procede a calcular el *ranking*. Para cada algoritmo, se (1) calcula el promedio de los resultados de toda la columna y (2) asigna un *ranking* teniendo en cuenta que los promedio más pequeños ocuparan la primeras posiciones tal y como se muestra en la tabla 5.2.

	GBM	IBK	SMO	J48	NB
Promedio	$\frac{1+1+1}{3} = 1$	$\frac{4+3+2}{3} = 3$	$\frac{3+5+4}{3} = 4$	$\frac{2+3+3}{3} = 2.$	$\frac{5+3+5}{3} = 4.$
Ranking	1	3	4	2	5

TABLA 5.2: Test de *Friedman*, segundo paso. Muestra el promedio de los rankings de las técnicas (*GBM*, *IBK*, *SMO*, *J48* y *NB*) y su ranking definitivo.

Se tiene que comprobar que el valor crítico (el resultado de la estadística de *Friedman*) esté por encima de un valor crítico. Si esto ocurre, significa que hay diferencias significativas y se hará un *post-hoc*, más concretamente el procedimiento de *Holm's*. El valor crítico o *p-value* se calcula usando una tabla de distribuciones ya existente [Sheskin, 2004]. Para dar paso al procedimiento de *Holm's*, la estadística de *Friedman* x_F^2 debe ser mayor que el valor crítico y este debe ser más pequeño que el nivel de significancia α , expresión 5.1.

$$x_F^2 = \frac{12N}{k(k+1)} \left(\sum_{i=1}^k R_i^2 - \frac{1}{4}k(k+1)^2 \right) \quad (5.1)$$

$$p < \alpha$$

Donde N es el número de *data sets*, R_i es el *ranking* de cada *data set*, k es el número de algoritmos en la comparación y p es el valor crítico.

El procedimiento de *Holm's* mide cómo son las diferencias entre parejas de algoritmos. Para ello calcula dos valores p , la p de la diferencia entre los dos algoritmos y la p de *Holm*. Si la p de la diferencia es más pequeña que la p de *Holm*, entonces hay una diferencia estadísticamente significativa. Mediante el *ranking* de *Friedman* se puede ver cuál es el mejor algoritmo.

5.3. Experimentación

En la sección previa se ha descrito el proceso para obtener una medida del acierto de un sistema. En esta sección, se (1) haya una buena configuración para *GBM* y (2) realizan comparativas del sistema *GBM* con algoritmos proporcionados por *Weka* [Witten et al., 2011]. *Weka* o *Waikato Environment for Knowledge Analysis* es una plataforma de software libre que recopila algoritmos para Aprendizaje Automático y herramientas de preprocesado de datos.

5.3.1. Configuración *GBM*

Para poder experimentar y comparar *GBM* con otros algoritmos, primero se deben seleccionar unos parámetros de configuración adecuados. Cada *data set* tiene características distintas, esto provoca que sea necesario una configuración diferente para sacar el máximo partido a *GBM*. Como se quieren métodos robustos que se comporten eficazmente en promedio, se busca la configuración que mejor resultado da en general para todos los *data sets*, esto es: la configuración se mantiene constante. Hay seis parámetros configurables, los cuales se detallan a continuación.

Mínimo número de instancias para generar nodos hoja. Si el mínimo número de instancias para generar nodos hoja es muy bajo, provoca que el árbol crezca y que cada nodo hoja del árbol regrese sobre pocos valores. Esto no es necesariamente negativo pero puede inducir al *overfitting*. En cambio, si este valor es muy elevado, el árbol generado tenderá a ser muy pequeño, esto puede generar *underfitting*. En general es mejor un valor pequeño que uno grande, por ejemplo 4.

Máximo número de nodos hoja. Este parámetro determina el número de nodos hoja por árbol. Si este valor es muy pequeño los árboles serán diminutos y el sistema tiende a *underfitting*. En cambio, si este valor es muy grande, los árboles serán inmensos provocando posible *overfitting*. La configuración ideal dependerá de cada problema, pero en general un número intermedio, por ejemplo 32. En general, si este parámetro está activo, no se usa la profundidad máxima.

Profundidad máxima. La profundidad del árbol viene determinada por este parámetro. Si este parámetro es muy bajo, los árboles serán muy pequeños, esto puede generar *underfitting*. Por el contrario, si este parámetro es muy elevado se generan *strong learners* en vez de *weak learners*, es decir, cada árbol será gigante y se cumple las premisas de los *weak learners*, esto conduce con toda seguridad a *overfitting*. El valor típico en la literatura está entorno al 6 [Friedman, 2000]. No es necesario utilizar la restricción del máximo número de nodos hoja si se utiliza este parámetro.

Varianza mínima. Este valor representa un umbral por el cual continuar particionado o no el árbol. Un valor muy elevado hará que no se genere el árbol mientras que un valor que tienda a cero hará que siempre se genere el árbol. No es un parámetro crítico, un valor típico es 0.000001.

Número de árboles. Este parámetro configura el número de árboles que habrá por etiqueta o clase. Si el valor es muy pequeño, se generará muy pocos árboles con lo que el sistema tendría a *overfitting*, sin embargo si este valor es elevado se generarán muchos árboles por clase y esto puede inducir a *overfitting*. Este parámetro está relacionado con el ratio de aprendizaje, si este parámetro es muy elevado y el ratio de aprendizaje es muy bajo, entonces se puede minimizar el efecto de *overfitting*. Una configuración óptima intenta balancear estos dos parámetros.

Ratio de aprendizaje. Es el valor que determina cómo de importante son las nuevas aportaciones de los nuevos árboles respecto de la solución ya encontrada. Es un parámetro muy importante y depende ampliamente del problema a tratar, si éste tiende a cero le dará poca importancia a los nuevos resultados con lo que puede conllevar a *underfitting*, por el contrario, si este valor es muy elevado dará mucha importancia a los nuevos resultados con lo que puede llevar a *overfitting*. Este parámetro determina el *steepest descent*, es decir, como de grande son los pasos por el gradiente de la función objetivo. Se recomienda un valor pequeño, por ejemplo 0.25.

En general no hay una mejor configuración para cualquier tipo de problema, así que lo que se recomienda en la bibliografía es hacer uso de *cross validation* con distintos parámetros según el problema a tratar. De esta forma se encuentra la mejor configuración específica. Finalmente se realiza un test estadístico que determina cual es la mejor de todas las configuraciones [García and Herrera, 2008].

De cara a hacer comparativas con otras técnicas, la obtención de los parámetros óptimos no se realiza de esta forma ya que interesa ver el comportamiento general dada una misma configuración. Para ello se realiza una variante de esta estrategia que consiste en coger un sub conjunto de todos los problemas que se compararan y ejecutar una validación cruzada simple (por lo tanto mucho más rápida que la completa o *stratified 10 fold cross validation*) que estima el acierto general del sistema.

5.3.2. Comparativa

Para llevar a cabo la experimentación se han elegido varios problemas de los repositorios *KEEL* [Alcalá-Fdez et al., 2011] y UCI [Bache and Lichman, 2013]. Estos se muestran en la tabla 5.3.

<i>Data set</i>	Número de instancias	Número de atributos	Número de clases
Auto Imports Database	205	25	7
BUPA Liver Disorders	345	6	2
Glass Identification Database	214	9	7
Heart Disease Data Set	303	13	5
Statlog (Heart) Data Set	270	13	2
The Monk's Problems	432	6	2
Pima Indians Diabetes Data Set	768	8	2
Lymphography Domain	148	18	4
Sonar, Mines vs. Rocks	208	60	2
Vehicle silhouettes	846	18	4
Deterding Vowel Recognition Data	990	13	11
Tao	1888	2	2
Microcalcifications	216	21	2

TABLA 5.3: Resumen de las características de los *data sets* utilizados. Muestra el nombre del *data set*, número de instancias, número de atributos y número de clases.

Se ha experimentado con *GBM* teniendo en cuenta lo detallado anteriormente. La tabla 5.4 muestra los parámetros que consiguen que el sistema sea más preciso en general. Esta configuración se mantiene constante para todos los *data sets*.

Parámetro	Valor
Mínimo número de instancias para generar nodos hoja	4
Máximo número de nodos hoja	0
Profundidad máxima	6
Varianza mínima	0.000001
Número de árboles	1000
Ratio de aprendizaje	0.25

TABLA 5.4: Configuración de *GBM* para que el sistema sea más preciso en general.

El objetivo de este capítulo es comparar *GBM* con el estado del arte del Aprendizaje Automático. Por ello se ha seleccionado las técnicas más relevantes de la literatura [Wu et al., 2007]. Estas técnicas han sido configuradas siguiendo los pasos de [Sancho-Asensio et al., 2014].

IBK. Este sistema clasificador forma parte de la familia *Lazy Learner's*, más concretamente *KNN* [Aha et al., 1991]. Es decir, utiliza el concepto de vecinos más cercanos para obtener la respuesta a la clase a predecir. El parámetro *K* indica el número de vecinos que elegidos para tomar una decisión de clasificación, se ha elegido $K = 3$.

SMO. Se trata de *Secuencial Minimal Optimization*, pertenece a la familia de *Support Vector Machine* [Platt, 1998]. Se ha configurado con un *Kernel* polinómico de grado uno, expresión 5.2.

$$K(x_1, x_2) = (x_1^T x_2 + 250007)^1 \quad (5.2)$$

J48. Es la versión de *Weka* del *C4.5* revisión 8 [Quinlan, 1993, Witten et al., 2011]. Se trata de un árbol de clasificación. En este sistema se configura el número mínimo de instancias por nodos hoja a 2 y el factor de confianza a 0.25 para podar el árbol.

NB. Naive Bayes [John and Langley, 1995]. Es un clasificador estadístico que se basa en el teorema de *Bayes*. No requiere ningún parámetro de configuración.

En la tabla 5.5 se muestran los resultados obtenidos de aplicar *stratified 10 fold cross validation* en cada uno de los *data sets*. Para que la comparación sea precisa, se ha utilizado (1) la puntuación *F* (F1) como medida del acierto y (2) la misma configuración en todos los problemas.

<i>Data set</i>	GBM	IBK	SMO	J48	NB
Auto Imports Database	84.48	66.95	68.95	84.05	55.45
BUPA Liver Disorders	69.66	62.50	42.85	65.85	54.25
Glass Identification Database	70.16	69.40	52.45	66.45	45.55
Heart Disease Data Set	77.51	81.85	82.90	74.05	83.15
Statlog (Heart) Data Set	79.15	78.35	83.45	78.80	83.85
Microcalcifications	66.37	65.60	65.40	61.85	64.85
The Monk's Problems	79.42	80.30	53.90	88.40	53.85
Pima Indians Diabetes Data Set	75.98	73.55	75.80	73.25	75.80
Lymphography Domain	84.51	81.10	87.05	77.95	83.65
Sonar, Mines vs. Rocks	84.26	83.05	76.70	71.90	67.40
Tao	88.83	96.40	83.95	95.45	80.80
Vehicle Silhouettes	70.37	70.35	72.90	71.50	42.85
Deterding Vowel Recognition Data	80.81	96.75	71.00	79.75	64.00

TABLA 5.5: Comparativa de algoritmos. Muestra los *data sets* en los que se hace la comparación y los resultados de la puntuación *F* en los algoritmos *GBM*, *IBK*, *SMO*, *J48* y *NB*.

Para determinar cuál es el algoritmo más preciso, se ha hecho un test estadístico mediante la herramienta *SCI2S multiple test*. Este programa permite comparar la precisión de múltiples algoritmos del Aprendizaje Automático [García and Herrera, 2008]. Realiza el test de *Friedman* y el procedimiento de *Holm's*. El programa *SCI2S multiple test* realiza el test tanto por $\alpha = 0,05$ como por $\alpha = 0,1$.

```
java Friedman GBM.csv >GBM.tex
```

En el comando anterior muestra el funcionamiento del *multiple test* en *Windows*. Recibe un archivo *csv* con una tabla como la tabla 5.5 y lo transforma en un archivo

en formato *tex*. En este último están los resultados del test estadístico, resumidos en la tabla 5.6.

Algoritmo	Ranking de <i>Friedman</i>	Ranking global
GBM	2.0000000000000004	1
IBK	2.8461538461538463	2
SMO	2.9615384615384626	3
J48	3.3076923076923075	4
NB	3.8846153846153846	5

TABLA 5.6: Clasificación de los algoritmos. Muestra tres atributos: algoritmo, *ranking* de *Friedman* y *ranking* global. Los *rankings* cuanto más bajo sean mejor.

El test de *Friedman* (tabla 5.6) muestra que *GBM* es el algoritmo con mejor *ranking* de los probados para los *data sets* utilizados. En segunda posición está *Instance Based Classifier (IB3)*, en tercera *Secuencial Minimal Optimization (SMO)*, en cuarta *J48* y por último *Naive Bayes (NB)* en quinta posición.

El programa *multipletest* retorna el valor p , en este escenario resulta $p = 0,0423$, menor que $\alpha = 0,05$. Esto significa que hay diferencias estadísticamente significativas, por lo tanto se puede proceder a detectarlas mediante el test de *Holm's*.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
10	NB vs. GBM	3.038850997435606	0.0023748228039272377	0.005
9	C4.5 vs. GBM	2.1085904880165427	0.03497994463544384	0.005555555555555556
8	IB3 vs. NB	1.6744689169543139	0.09403847826910407	0.00625
7	SMO vs. GBM	1.5504341823651067	0.12103733948230683	0.0071428571428571435
6	NB vs. SMO	1.4884168150704997	0.13664100480748356	0.008333333333333333
5	IB3 vs. GBM	1.3643820804812925	0.1724473447164437	0.01
4	C4.5 vs. NB	0.9302605094190637	0.3522362191881913	0.0125
3	C4.5 vs. IB3	0.7442084075352502	0.45675040242316967	0.016666666666666666
2	C4.5 vs. SMO	0.558156305651436	0.5767376547948123	0.025
1	IB3 vs. SMO	0.18605210188381416	0.8524039031542188	0.05

TABLA 5.7: Test de *Holm* para $\alpha = 0,05$.

En la tabla 5.7 se muestra el test de *Holm*. En este se comparan los algoritmos en pareja y se calculan los valores p de (1) la diferencia entre los algoritmos y (2) *Holm*. Puede observarse que la diferencia de p y p de *Holm* de *Naive Bayes* y *GBM* ($i = 10$) es estadísticamente significativa ($p < p_{Holm}$). No se puede afirmar lo mismo para los demás. Se confirma que un sistema clasificador que produce un sólo *strong learner* es menos preciso que un sistema que genera múltiples *weak learners* [Friedman, 2000].

5.4. Resumen

En este capítulo se ha medido la precisión de *GBM*. Para ello, se ha buscado una configuración óptima y se ha comparado con cuatro sistemas clasificadores que previamente se han mencionado en el capítulo 3: se trata de (1) un algoritmo de la familia

del Aprendizaje Perezoso denominado *Instance Based Classifier (IB3)*, (2) un algoritmo que pertenece a las Máquinas de Soporte Vectorial llamado *Secuencial Minimal Optimization (SMO)*, (3) un árbol de decisión (*J48*) y (4) un modelo probabilístico *Naive Bayes (NB)*. Este experimento se ha realizado con 13 *data sets* mediante una configuración constante y se ha calculado el promedio de la puntuación F (F1) sobre el conjunto de todos los pares *train-test* del *stratified 10 fold cross validation*. Se ha calculado el *ranking* a través del test de *Friedman* y posteriormente se ha medido si existe una diferencia significativa entre los algoritmos con el test de *Holm*. Finalmente se ha concluido con que *GBM* (1) es el algoritmo más preciso para los *data sets* del experimento y (2) tiene una diferencia estadísticamente significativa respecto *NB*.

En el siguiente capítulo se alcanza la meta del proyecto, es decir, aplicar *GBM* a datos docentes reales para modelar el rendimiento del alumnado.

6

GBM aplicado a datos docentes

En el capítulo 3 se ha remarcado que *Gradient Boosting* es una de las técnicas con más acierto actualmente existentes en la bibliografía [Goldbloom, 2010], este algoritmo se comporta mejor con árboles de decisión como base [Friedman, 2000] y se fundamenta en técnicas estadísticas muy sólidas. Por ello se ha implementado *GBM* (capítulo 4), este sistema aplica un *Gradient Boosting* con árboles de regresión. Posteriormente, en el capítulo 5 se ha realizado una comparativa con otros cuatro algoritmos utilizando 13 *data sets* y se ha comprobado que este algoritmo es el más preciso en promedio para los *data sets* usados. Los objetivos del presente capítulo son (1) aplicar *GBM* a datos docentes con la finalidad de modelar al alumnado y (2) supervisar el resultado por el personal docente. Para aplicar *GBM*, los datos deben estar ya procesados con el formato idóneo, así pues el primer paso es transformar el *data set* de entrada siguiendo el marco de trabajo propuesto por la disciplina *Knowledge Discovery in Databases (KDD)* con la finalidad de facilitar a *GBM* la extracción de conocimiento.

6.1. Procesado de los datos

En el capítulo anterior se ha validado el buen comportamiento de *GBM* frente a las técnicas tradicionales. En promedio mejora significativamente los resultados debido a las características de los métodos *Ensemble*. Estos problemas del *UCI repository*, pese a ser problemas del mundo real, son utilizados como *benchmarks* y por ende el modelado

de los datos no resulta excesivamente problemático. En este capítulo se utiliza todo el conocimiento que se ha adquirido a lo largo del Trabajo Final de Grado para modelar el rendimiento académico del alumnado. Esto resulta en un problema muy complejo debido a que es muy difícil modelar el comportamiento de los alumnos. Para llevar a cabo este modelado, se realizan dos tipos de experimento utilizando un *data set* real de una asignatura de la universidad procesado de dos formas distintas, una para cada experimento.

El personal docente ha proporcionado una base de datos anonimizada con las notas de una asignatura (ver tabla 6.1). El *data set* está en formato *.csv* y compuesto por seis atributos (Expediente, Ejercicio EC1, Ejercicio EC2, Midterm, Ejercicio EC3 y Examen Febrero) y 150 instancias. Hay un total de 150 alumnos matriculados, el atributo Expediente es un número entero único que hace referencia a cada alumno, de esta forma se consigue el anonimato, este atributo es totalmente prescindible así que se ha eliminado. Los ejercicios EC1, EC2 y EC3 hacen referencia a la nota de cada uno de los tres ejercicios de evaluación continua. Por último, Midterm y Examen Febrero son las notas de los exámenes de mitad y final de semestre respectivamente. Cada una de estas notas se representa mediante un *NP* (no presentado) o un número real del cero al diez.

Columna	Valores [Rango]
EXPEDIENTE	Numeral
EJERCICIO EC1	Numeral [0,10] y NP
EJERCICIO EC2	Numeral [0,10] y NP
MIDTERM	Numeral [0,10] y NP
EJERCICIO EC3	Numeral [0,10] y NP
EXAMEN FEBRERO	Numeral [0,10] y NP

TABLA 6.1: Atributos del *data set* original. Se trata de un *data set* real con las notas del alumnado en una asignatura. Muestra los atributos del *data set* y su rango de valores.

El sistema *GBM* no se puede aplicar directamente sobre este *data set*, ya que este no tiene definido el atributo de salida, y no está en formato *ARFF*.

No tiene definido el atributo de salida. La clase es el atributo de salida o respuesta del sistema, en este *data set* no existe una clase nominal. Por ello se debe generar un campo llamado *class*, secciones 6.3.1 y 6.4.1.

EC1	EC2	Midterm	EC3	Examen	Febrero	NP	EC1	NP	EC2	NP	Midterm	NP	EC3	NP	Examen	Febrero
7.9	NP	6.5	2	5.5	0	1	0	0	0	0	0	0	0	0	0	0
NP	NP	4.6	NP	NP	1	1	0	1	1	0	1	1	1	1	1	1

TABLA 6.2: Se añade granularidad al sistema mediante *dummies*, estos mapean las notas *NP*. Se amplía el *data set* con los atributos *NP EC1*, *NP EC2*, *NP Midterm*, *NP EC3* y *NP Examen Febrero*.

Procesado de notas *NP*. Las notas *NP* corresponden a los alumnos que no se han presentado al examen o no han presentado algún ejercicio de evaluación continua. Como este valor no es un número y forma parte de las notas (donde todas están representadas mediante un atributo numeral del cero al diez), se ha hecho uso de *dummies*: Se han generado cinco columnas más en el *data set* (una para cada una de las notas), estas pueden valer cero si no hay ningún *NP* y uno en caso contrario (ver tabla 6.2). Cada *NP* representa un cero en la nota (ver tabla 6.3).

EC1	EC2	Midterm	EC3	Examen	Febrero	NP	EC1	NP	EC2	NP	Midterm	NP	EC3	NP	Examen	Febrero
7.9	0	6.5	2	5.5	0	1	0	0	0	0	0	0	0	0	0	0
0	0	4.6	0	0	1	1	0	1	0	1	1	0	1	1	1	1

TABLA 6.3: Una vez se han añadido los *dummies*, se les da valor cero si no hay *NP* y la unidad en caso contrario. Posteriormente se substituyen los valores *NP* por ceros en las notas.

De este modo no se pierde información, es decir, se consigue tener en cuenta los valores *NP* y cada uno de estos representa un cero en la nota. Este proceso se denomina ingeniería de los atributos o *Feature Engineering*.

Las clases están desbalanceadas. En la figura 6.1 se puede observar que el número de alumnos suspendidos y con nota excelente es muy distinto a los demás, esto significa que las clases están desbalanceadas. Esto afecta negativamente a la precisión de *GBM*, ya que este considera que la distribución de clases es uniforme.

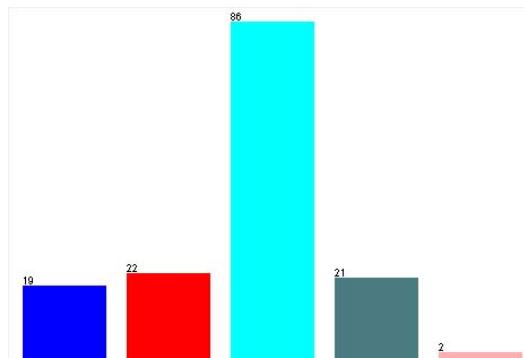


FIGURA 6.1: Diagrama de barras de la clase nominal del *data set* procesado. El color cyan representa el número de alumnos suspendidos, estos son el 57,33 % de la clase. El color azul marino simboliza los alumnos que han aprobado la asignatura con un suficiente (un total de 12,67 %). El color rojo indica el número de alumnos que han sacado un bien (estos son el 14,67 % de la clase), el color verde apagado representa los alumnos que han sacado un notable de nota final (el 14 %) y por último el color rosa son aquellos alumnos que tienen un excelente de nota final (el 1,33 %).

6.2. Notas previas a la experimentación

En la sección anterior se ha explicado el procesamiento del *data set* siguiendo el ciclo de vida *KDD*. Como se ha visto en el apartado 6.1, para poder realizar los dos experimentos es necesario construir previamente el atributo clase.

El primer experimento consiste en predecir la nota final de curso de los alumnos en una asignatura, este problema no resulta tan complicado para los clasificadores ya que (1) la clase es una combinación lineal de los atributos de entrada siguiendo los criterios de evaluación de la asignatura y (2) se proporciona el atributo de entrada con mayor relevancia al algoritmo de aprendizaje, es decir, la nota del examen final. Este primer experimento sirve para calibrar los distintos algoritmos que se utilizan en la comparativa y verificar la eficacia de los *dummies*.

El segundo experimento trata de predecir la nota del examen final de un alumno teniendo en cuenta sus notas parciales a lo largo de la asignatura. Esto resulta tremendamente útil ya que la nota final es el atributo más importante para aprobar la asignatura y el personal docente puede predecir si un alumno necesitará ayuda extra antes de realizar el examen final. Este problema resulta muy complicado ya el comportamiento de los alumnos no siempre es fácil de predecir.

6.3. Primer experimento

En este primer experimento se procesa el *data set* de entrada siguiendo el ciclo de vida *KDD*, posteriormente se busca una configuración adecuada para *GBM* y se compara con los algoritmos clásicos de la literatura. Finalmente se muestra el modelo generado por *GBM*.

6.3.1. Clase a predecir

Para acabar de procesar los datos es imprescindible tener un atributo de salida que corresponda a la clase que se quiere predecir. En este experimento corresponde a la nota final de la asignatura teniendo en cuenta todos los otros atributos y puede tomar cinco valores posibles SUSPENDIDO, APROBADO, BIEN, NOTABLE, EXCELENTE tal y como se muestra en la tabla 6.4.

Para poder etiquetar correctamente cada alumno con su nota final correspondiente, se ha debatido con el personal docente sobre los criterios de evaluación. A continuación se muestran estos cuatro criterios.

1. El Examen de febrero se pondera como un 60 % de la nota final y para hacer media con las demás notas se necesita una nota superior o igual a 3.5.
2. El Midterm se pondera como un 20 % de la nota final.

<i>class</i>	Nota final
SUSPENDIDO	$Nota < 5$
SUFICIENTE	$5 \geq Nota < 5,5$
BIEN	$5,5 \geq Nota < 6,5$
NOTABLE	$6,5 \geq Nota < 8,5$
EXCELENTE	$Nota \geq 8,5$

TABLA 6.4: Criterios de evaluación para definir la clase. Muestra la clase *SUSPENDIDO*, *SUFICIENTE*, *BIEN*, *NOTABLE*, *EXCELENTE* y su relación con la nota final.

3. El conjunto de todos los ejercicios de evaluación continua se ponderan como un 20 % de la nota final, como todos cuentan lo mismo, cada ejercicio de evaluación continua por separado se pondera como un 6,67 de la nota final.
4. No hay datos de exámenes recuperación.

Se ha utilizado *Microsoft Excel* para procesar el *data set*, para ello se han generado seis columnas temporales con el siguiente código en cada una de ellas.

```
=IF(Fi<>'NP',IF( Fi>=3.5, 0.6 * Fi, 0), 0) // Evaluación Examen Febrero
=IF(Di<>'NP',0.2 * Di, 0) // Evaluación Midterm
=IF(Bi<>'NP', (0.2/3) * Bi, 0) // Ejercicio Evaluación Continua 1
=IF(Ci<>'NP', (0.2/3) * Ci, 0) // Ejercicio Evaluación Continua 2
=IF(Ei<>'NP', (0.2/3) * Ei, 0) // Ejercicio Evaluación Continua 3
=Ni+Oi+Pi+Qi+Ri // Nota final
```

Donde i es el número de fila, F es la columna o atributo Examen Febrero y D corresponde al Midterm. Las columnas B , C y E corresponden a los ejercicios de Evaluación Continua 1, 2 y 3 respectivamente. De este modo se han evaluado todas las notas teniendo en cuenta los criterios de evaluación proporcionados por el personal docente. Seguidamente se ha etiquetado el resultado final en una nueva columna llamada *class*.

```
=IF(Mi<5, 'SUSPENDIDO', IF(AND(Mi>=5,Mi<5.5),
'APROBADO', IF(AND(Mi>=5.5,Mi<6.5), 'BIEN', IF(AND(Mi>=6.5,Mi<8.5),
'NOTABLE', 'EXCELENTE'))))
```

Donde M es la columna del resultado del cálculo de la nota final. Finalmente se han borrado las seis columnas temporales anteriores (ver tabla 6.5).

Columna	Valores [Rango]
EJERCICIO EC1	Numeral [0,10]
EJERCICIO EC2	Numeral [0,10]
MIDTERM	Numeral [0,10]
EJERCICIO EC3	Numeral [0,10]
EXAMEN FEBRERO	Numeral [0,10]
NP EC1	Numeral [0,1]
NP EC2	Numeral [0,1]
MIDTERM	Numeral [0,1]
NP EC3	Numeral [0,1]
NP EXAMEN FEBRERO	Numeral [0,1]
CLASS	Nominal [SUSPENDIDO, SUFICIENTE, BIEN, NOTABLE, EXCELENTE]

TABLA 6.5: *Data set* procesado. Muestra los atributos del *data set* procesado y su rango de valores. Se ha eliminado el atributo *Expediente*, se ha modificado el rango de valores de cada atributo, se ha añadido la clase (la nota final de la asignatura) y por último los atributos que mapean los valores *NP* (*dummies*).

6.3.2. Análisis de los resultados

En la sección anterior se ha mostrado el proceso de transformación del *data set* para que las técnicas de Aprendizaje Automático tengan más información y consecuentemente puedan tener más precisión. En este apartado se (1) realiza una *stratified 10 fold cross validation*, (2) busca una configuración óptima para usar *GBM* con los datos docentes, (3) compara su precisión frente los algoritmos del capítulo 5. Para tener una estimación de la dificultad del problema se realiza una validación cruzada completa. Para ello, se ha hecho uso de la herramienta *DCoL* del siguiente modo:

```
./dcol -i notas.arff -o ./CV/notas -cv
```

Donde *notas.arff* es el *data set* donde se hallan los datos docentes procesados como se ha visto en el apartado anterior.

Una vez se ha hecho el proceso de validación cruzada, se ha tomado un par de muestras *train-test* con las que se han ido probando distintas configuraciones de *GBM*. La tabla 6.6 muestra los parámetros más óptimos hallados.

Parámetro	Valor
Mínimo número de instancias para generar nodos hoja	4
Máximo número de nodos hoja	0
Profundidad máxima	1
Varianza mínima	0.000001
Número de árboles	10
Ratio de aprendizaje	0.1

TABLA 6.6: Configuración *GBM* para modelar el rendimiento del alumnado.

Seguidamente se ha comparado la precisión del sistema utilizando la puntuación *F*

($F1$) con los algoritmos *Instance Based Classifier* (*IBK*), *Secuencial Minimal Optimization* (*SMO*), *J48* y *Naive Bayes* (*NB*) con la misma configuración para cada uno de ellos detallada en el capítulo 5.

<i>Data set</i>	GBM	IBK	SMO	J48	NB
Notas alumnos sin <i>dummies</i>	70.19	57.00	57.60	41.80	56.50
Notas alumnos con <i>dummies</i>	86.98	75.20	72.80	79.00	74.80

TABLA 6.7: Comparativa entre *GBM*, *IBK*, *SMO*, *J48* y *NB* en el *data set* docente con y sin *dummies*.

En los resultados (tabla 6.7) destaca que *J48* consigue el resultado más similar al obtenido por *GBM* debido a que la representación del modelo de conocimiento en ambos casos es muy parecida. La gran diferencia está en el concepto de combinar múltiples *weak learners* para obtener un único *strong learner* (*GBM*) o utilizar un único *strong learner* (*J48*). Combinando diez *weak learners* diferentes por cada clase (un total de 50) se consigue evitar el *overfitting* a la vez que se mejora mucho la calidad del modelo. De esta forma se puede evitar casos extremos por ejemplo que un alumno que tenga un promedio de diez en todas las notas, no se presente al examen final y por consecuencia suspenda. Este caso es muy complicado de detectar debido a la compleja casuística. El problema de clasificar la nota final de los alumnos es mucho más compleja de lo que a priori puede parecer, por ejemplo una técnica tan robusta como *SMO* está resultando inferior al resto de las técnicas analizadas. La dificultad de modelar los alumnos es la causa del rendimiento, por ello se ha procesado la información añadiendo variables *dummy* para enriquecer el *data set* ya que el hecho de tener valores *NP* no se pueden cuantificar de manera directa y los algoritmos no pueden generalizar. El hecho de añadir esta información hace que las puedan generalizar de una manera más efectiva (se está añadiendo granularidad a la información).

$$\frac{(86,98 - 70,19) + (75,20 - 57,00) + (72,80 - 57,60) + (79,00 - 41,80) + (74,80 - 56,50)}{5} = 21,14\% \quad (6.1)$$

Añadiendo esta granularidad al *data set* para los cinco algoritmos se consigue un promedio de 21,14% más precisión (ver expresión 6.1, el cálculo se realiza mediante los datos de la tabla 6.7). Se puede observar que la parte más importante del estudio realizado se halla en el procesado del *data set* ya que provoca una mejora muy elevada del acierto, mucho mayor que la diferencia de precisión entre cada uno de los algoritmos presentados.

6.3.3. Modelado del alumnado

En la sección anterior se ha hallado una buena configuración de *GBM* para resolver el problema. En este apartado se visualiza el modelo generado por el primer par *train-test* de la validación cruzada (de los diez totales), esto significa que se ilustran 50 de los 500 modelos generados en total, en general todos los modelos son prácticamente iguales. A medida que se van generando árboles para cada una de las clases, los residuos disminuyen. *GBM* utiliza la técnica *One Versus All* combinada con *Soft Max* para poder clasificar más de una clase.

El modelo se ha generado mediante la herramienta *ModelToGpv*. La sección [A.3.2](#) detalla la nomenclatura utilizada.

En la figura [6.2](#) se puede observar que el atributo más importante (el que tiene menor ganancia en varianza) para predecir si la etiqueta es suspendido, es la nota del examen de febrero.

Los atributos más importantes para predecir la etiqueta suficiente (figura [6.3](#)) son la nota del examen de febrero y la nota del primer ejercicio de evaluación continua. Si la nota del examen de febrero es inferior o igual a 3.475 es muy probable que ese alumno no consiga un suficiente de nota final.

El atributo más importante según el modelo construido a partir de los datos de entrenamiento (figura [6.4](#)) para determinar si un alumno obtiene una nota final de bien o no, es la nota del examen de febrero.

En la figura [6.5](#) se puede observar de nuevo que el atributo con menos ganancia en varianza es la nota del examen de febrero. Este determina si un alumno obtiene un notable o no de nota final.

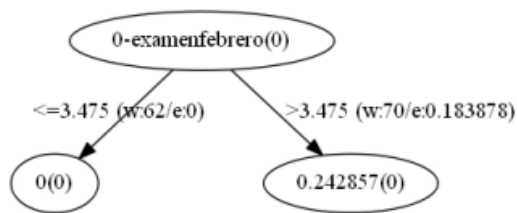
El algoritmo ha detectado que el atributo más importante para clasificar si un alumno obtiene un excelente o no de nota final (figura [6.6](#)), es la nota del segundo ejercicio de evaluación continua. Si la nota es inferior o igual a 9.5 en éste, el alumno seguro que no tiene un excelente de nota final, en caso contrario no se puede afirmar nada. En este caso sólo es necesario el primer árbol puesto que ya acierta completamente el caso.

En este experimento se ha calibrado *GBM* con la finalidad de obtener la mayor precisión posible para el *data set* y se ha comprobado la eficacia de los *dummies*.

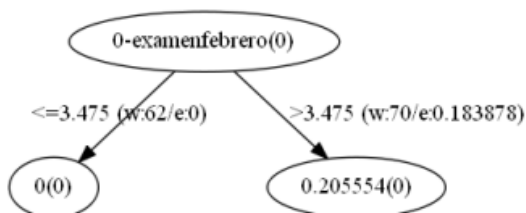
En la siguiente sección se realiza un experimento mucho más complejo, el que permitirá al cuerpo docente predecir de manera efectiva el rendimiento de los alumnos en el examen final de la asignatura.



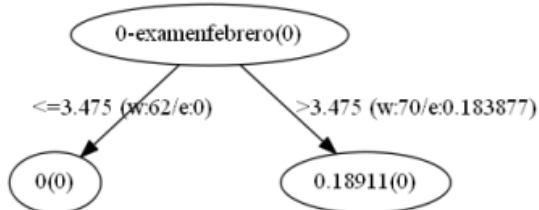
FIGURA 6.2: Strong learner generado para la clase SUSPENDIDO.



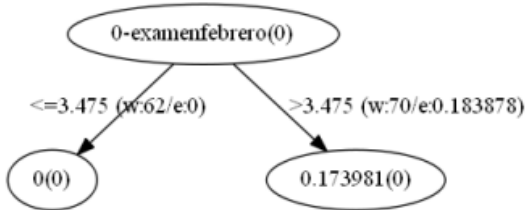
Árbol 1.



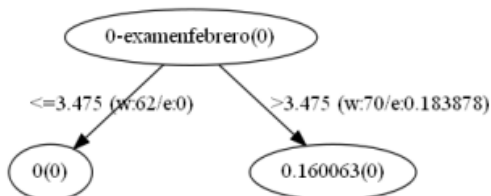
Árbol 2.



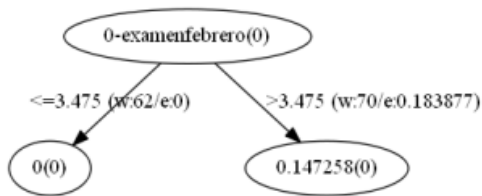
Árbol 3.



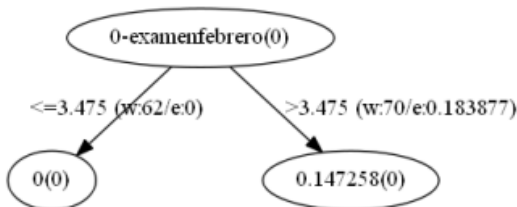
Árbol 4.



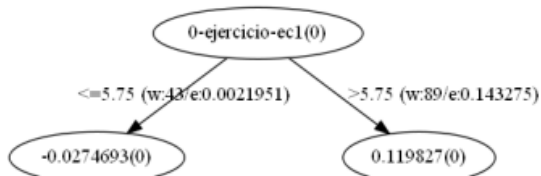
Árbol 5.



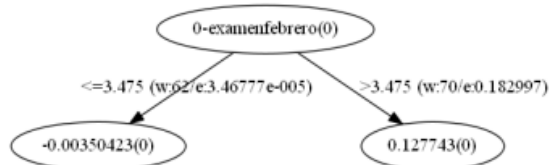
Árbol 6.



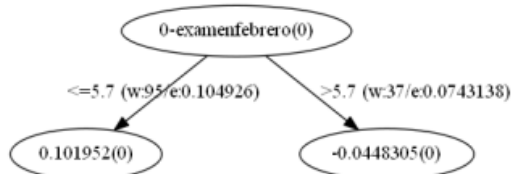
Árbol 7.



Árbol 8.



Árbol 9.



Árbol 10.

FIGURA 6.3: Strong learner generado para la clase SUFICIENTE.



FIGURA 6.4: Strong learner generado para la clase BIEN.

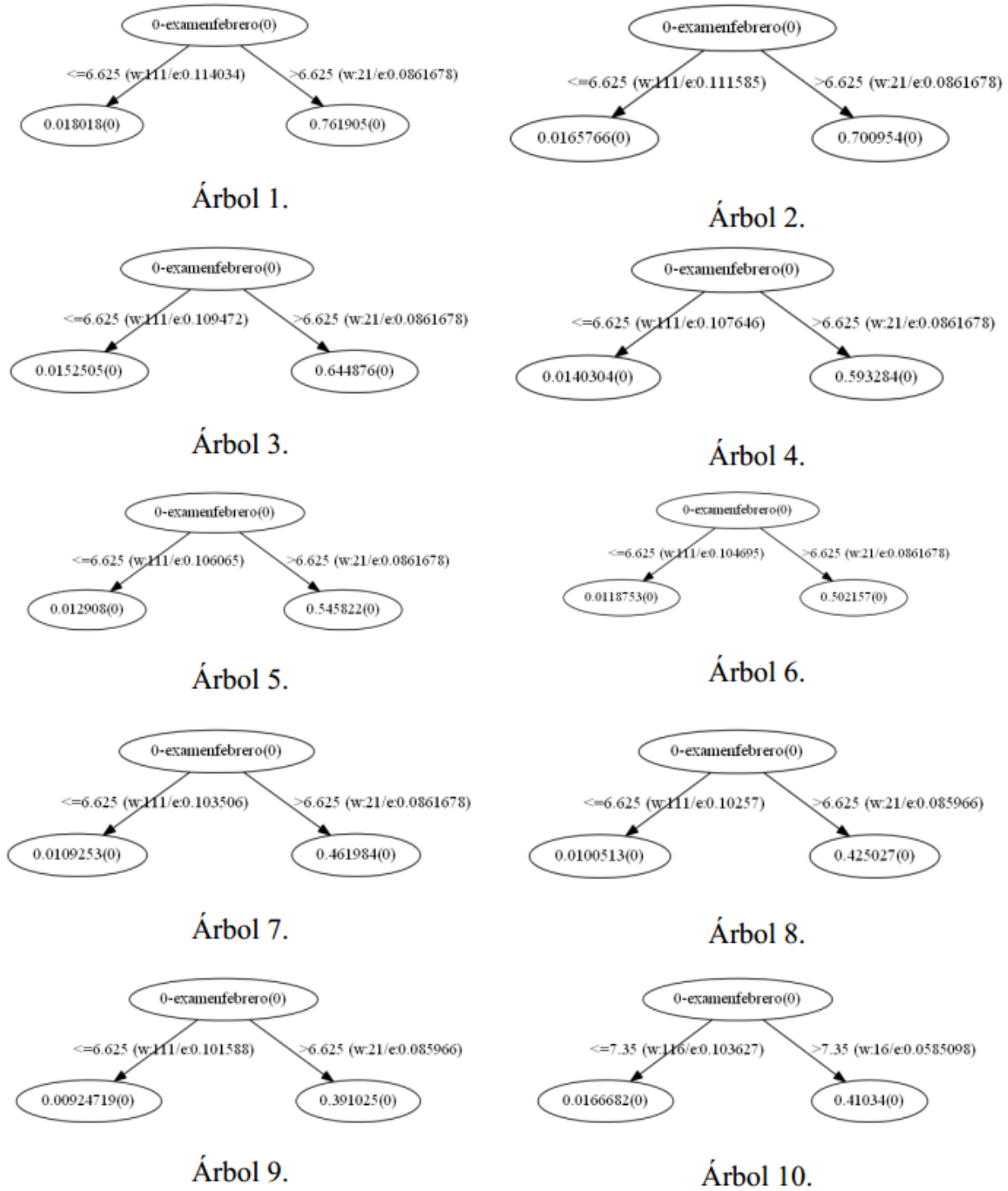
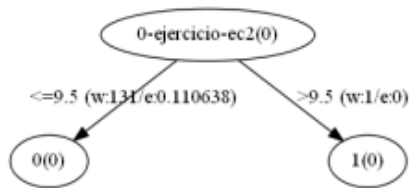
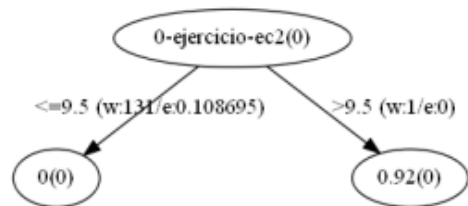


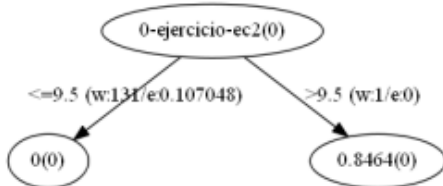
FIGURA 6.5: Strong learner generado para la clase NOTABLE.



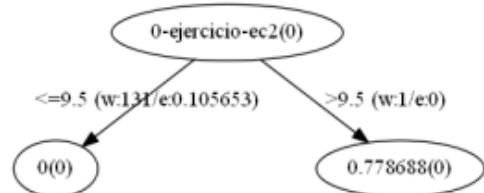
Árbol 1.



Árbol 2.



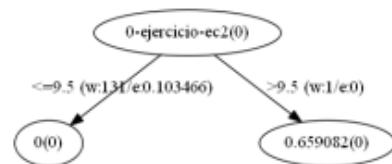
Árbol 3.



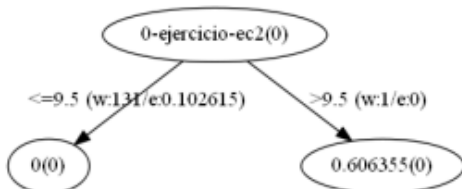
Árbol 4.



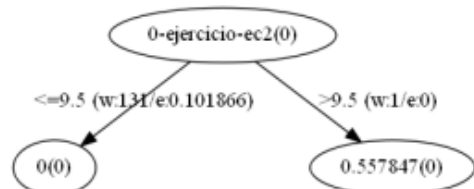
Árbol 5.



Árbol 6.



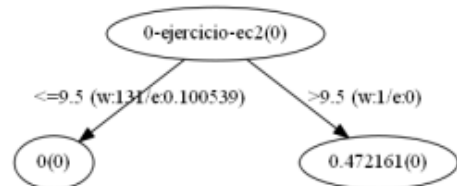
Árbol 7.



Árbol 8.



Árbol 9.



Árbol 10.

FIGURA 6.6: Strong learner generado para la clase EXCELENTE.

6.4. Segundo experimento

En el primer experimento se ha calibrado *GBM* y se ha visto su potencial frente a problemas complejos como por ejemplo el *data set* docente sin *dummies*. De todas formas, es más útil para el personal docente predecir la nota del examen final en vez de la nota final de la asignatura. Este problema resulta muy complicado debido a que para predecir la nota del examen final se disponen de pocos atributos que sean descriptivos, es decir, que aporten una *ganancia de la información* elevada. Así los distintos atributos de entrada aportan poca información sobre la nota final, por lo que resulta necesario exprimir al máximo las características de generalización de los algoritmos de aprendizaje. Por otra parte existen variables *ocultas* como por ejemplo la vida personal del alumno que no se pueden medir en este estudio (problemas familiares, de salud, etcétera).

6.4.1. Clase a predecir

Para poder llevar a cabo el experimento se debe construir el atributo clase. En este caso, la clase a predecir es la nota final del examen. A diferencia del primer experimento

- la nota del examen de febrero se ha convertido a nominal (la clase a predecir) y
- se ha eliminado el atributo nota final de la asignatura.

En la tabla 6.8 se muestra la descripción de los atributos.

Columna	Valores [Rango]
EJERCICIO EC1	Numeral [0,10]
EJERCICIO EC2	Numeral [0,10]
MIDTERM	Numeral [0,10]
EJERCICIO EC3	Numeral [0,10]
NP EC1	Numeral [0,1]
NP EC2	Numeral [0,1]
MIDTERM	Numeral [0,1]
NP EC3	Numeral [0,1]
CLASS	[SUSPENDIDO, SUFICIENTE, BIEN, NOTABLE, EXCELENTE]

TABLA 6.8: *Data set* procesado. Muestra los atributos del *data set* procesado y su rango de valores. Se ha eliminado el atributo *Expediente*, se ha modificado el rango de valores de cada atributo, se ha añadido la clase (la nota del examen de febrero) y por último los atributos que mapean los valores *NP* (*dummies*).

6.4.2. Análisis de los resultados

Una vez se han procesado los datos, se ha procedido a la experimentación con *GBM*, *IBK*, *SMO*, *J48* y *NB*. Para *GBM* se ha utilizado la configuración hallada en

<i>Data set</i>	GBM	IBK	SMO	J48	NB
Examen Febrero con <i>dummies</i>	75.99	43.50	47.40	45.80	52.00

TABLA 6.9: Comparativa de precisiones según la medida F de los distintos algoritmos (GBM , IBK , SMO , $J48$ y NB) en el segundo experimento.

el experimento de calibración (ver tabla 6.6) y para las otras técnicas, la configuración detallada en el capítulo 5. Los resultados del experimento se muestran en la tabla 6.9.

Como se ha mencionado al comienzo del capítulo este segundo experimento es muy complicado, esto se refleja en la tabla 6.9 donde los algoritmos IBK , SMO , $J48$ y NB obtienen precisiones muy bajas (rondando el 50 % en la medida F). Destaca notablemente GBM dando unos resultados muy competitivos, no es de extrañar debido a que las técnicas *Ensemble* consiguen una mayor precisión al combinar distintos modelos para generar la predicción.

Gracias a la precisión de GBM en este problema real, el personal docente puede hacer un seguimiento de cada alumno y predecir con confianza que nota obtendrá en el examen final. En caso de que esta sea baja, el profesor podrá avisar con suficiente antelación al alumno para ofrecerle atención personalizada.

6.5. Extrapolación al mundo real

En los experimentos anteriores se ha hecho una predicción sobre algo que ya se conocía de ante mano, es decir, se ha realizado una validación cruzada sobre datos supervisados (ya se conocía la respuesta), en el mundo estadístico se conoce como *back testing* y sirve para probar la validez de los modelos [Sheskin, 2004]. En el mundo real no se conoce la clase a extrapolar pues no se puede predecir algo que aún no ha ocurrido, un ejemplo se halla en las competiciones de *Kaggle* [Goldbloom, 2010] donde se proporciona a cada participante dos ficheros, el de entrenamiento con las clases debidamente etiquetadas (de aquí es donde se realiza la validación cruzada) y un fichero de *test* generalmente con muchas más instancias el cual los administradores son los únicos que conocen como van etiquetados, de esta forma se evalúa el acierto de las técnicas propuestas por los participantes. El sentido de las competiciones *Kaggle* es superar el *benchmark* proporcionado por los administradores, un ejemplo lo encontramos en la *Higgs Kaggle Competition 2014* donde se proponía superar la precisión de las técnicas utilizadas en el *CERN*.

6.6. Resumen

En este capítulo se ha aplicado GBM para modelar al alumnado. Para ello se han realizado dos experimentos, un primero para calibrar GBM y otro para modelar el comportamiento de los alumnos en el examen final de la asignatura. El *data set*

utilizado está formado por notas reales de los alumnos en una asignatura y siguiendo el ciclo *KDD*, éste se ha procesado dos veces (una por experimento) para simplificar el tratamiento de la información por parte de los sistemas clasificadores: se han eliminado los atributos prescindibles y se le ha definido el atributo de salida según los criterios de evaluación, se le ha añadido granularidad y se ha transformado a *ARFF*.

En el primer experimento, para modelar el *data set* procesado, se ha hallado una configuración óptima para que *GBM* evite el *overfitting* y posteriormente se ha comparado su puntuación F con los algoritmos *IBK*, *SMO*, *J48* y *NB*. La técnica más precisa es *GBM* y en segundo lugar *J48* ya que utiliza una representación similar del conocimiento y se ajusta muy satisfactoriamente al problema. El procesamiento de los datos así como la granularidad añadida, provoca que la puntuación F se incremente en un 22,5 % en promedio, esto es una diferencia mucho mayor que la diferencia de precisión entre los algoritmos en promedio, por lo tanto es crucial realizar un buen procesamiento de los datos antes de aplicar un algoritmo de aprendizaje.

En el segundo experimento se ha partido del mismo *data set* pero la clase a predecir es la nota del examen final. Éste resulta mucho más complicado que el anterior. El único algoritmo fiable en este experimento ha sido *GBM* debido a su alta tasa de precisión. Gracias a estos resultados, el personal docente puede predecir cómo irá el examen final al alumnado y reforzar con actividades sus conocimientos para ayudar a superar la asignatura de forma satisfactoria.

En el siguiente capítulo se desglosa el coste temporal del presente Trabajo Final de Grado.

7

Costes temporales

En este apartado se muestra el coste en horas aproximado del presente Trabajo Final de Grado.

7.1. Resumen de los objetivos

La finalidad del presente Trabajo Final de Grado ha sido aplicar una de las técnicas más precisas del Aprendizaje Automático para modelar el rendimiento académico del alumnado. Para ello, ha sido necesario revisar las técnicas tradicionales del Aprendizaje Automático y los métodos *Ensemble*, diseñar e implementar un *Gradient Boosting*, validar el algoritmo y por último aplicarlo a datos docentes.

7.2. Coste en horas

Para llevar a cabo los objetivos mencionados anteriormente, el presente Trabajo Final de Grado ha conllevado la realización de varias tareas las cuales pueden clasificarse según las etapas (1) investigación, (2) diseño, (3) implementación, (4) documentación y (5) experimentación. Éstas se describen a continuación.

Investigación. Esta ha sido la etapa más larga y crítica para el proyecto, ha consistido en la lectura de documentación técnica sobre el Aprendizaje Automático.

Análisis y Diseño. En esta etapa se ha diseñado todo el *software* para garantizar su correcta implementación.

Implementación. La codificación de los distintos algoritmos se ha llevado a cabo en esta fase. Estos son *GBM*, *PGUI* y *ModelToGpv*.

Documentación. Esta etapa comprende la documentación escrita de todo el trabajo realizado incluyendo la elaboración del documento presente.

Experimentación. Esta fase comprende la realización de tres tipos distintos de experimentos, se ha (1) hallado una configuración óptima para *GBM* y se ha comparado con las técnicas tradicionales en 13 *data sets* del *UCI Repository* [Bache and Lichman, 2013], (2) calibrado *GBM* en un problema con datos reales y (3) aplicado con esta calibración para predecir la nota del examen final de una asignatura.

En la tabla 7.1 y figura 7.1 se desglosa el tiempo dedicado para cada una de las fases anteriormente mencionadas.

Fase	Tiempo dedicado (h)	Porcentaje (%)
Investigación	272	34.21
Diseño	131	16.48
Implementación	117	14.72
Documentación	205	25.79
Experimentación	70	8.80
Total	795	100

TABLA 7.1: Coste en horas del trabajo. Muestra la fase, el tiempo dedicado en horas y el porcentaje del proyecto en tanto por ciento.

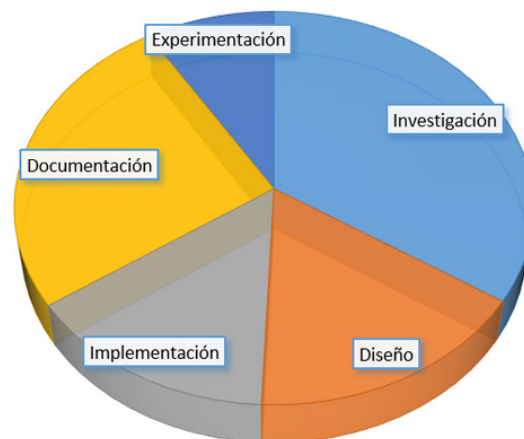


FIGURA 7.1: Desglose del tiempo dedicado en el presente Trabajo Final de Grado.

En el siguiente capítulo se resume todo el trabajo realizado y se presentan las conclusiones.

8

Resumen y conclusiones

Uno de los campos más exitosos de las tecnologías de la información es el Aprendizaje Automático, una disciplina centrada en el estudio de algoritmos capaces de aprender de la experiencia para resolver problemas que de manera tradicional (Ingeniería del Software) no tenían solución o su obtención es sumamente costosa. La disciplina del Aprendizaje Automático es especialmente adecuada para modelar problemas complejos como es el caso de la docencia. El propósito de este Trabajo Final de Grado ha sido investigar el estado del arte en técnicas de Aprendizaje Automático para ayudar al cuerpo docente a modelar y predecir el rendimiento de los alumnos. De las muchas familias de técnicas propias dentro del campo de Aprendizaje Automático, la que ha atraído la atención de los investigadores recientemente debido a su elevada tasa de acierto está en los métodos *Ensemble*. Éstos, agrupan varios sistemas inteligentes (llamados clasificadores) en uno sólo. Destacan los sistemas *Gradient Boosting*, el funcionamiento de los cuales se basa en la mejora continua. Este capítulo resume cada resultado obtenido según los objetivos planteados y expone las conclusiones.

8.1. Resumen del Trabajo Final de Grado

Ser capaz de predecir el rendimiento de un alumno resulta una tarea muy compleja incluso para el personal docente más experimentado. Por este motivo cualquier intento de automatizar dicho proceso resulta un auténtico reto. El alumnado es muy difícil de

modelar debido a que su comportamiento dista de ser ideal, es decir, nunca se puede estar cien por cien seguro de que las métricas que se usan para evaluarlo sean precisas, por ejemplo que haya entregado y superado todos los trabajos de evaluación continua no significa que este alumno vaya a aprobar un examen final. Actualmente, debido a la elevada capacidad de proceso de los ordenadores actuales resulta más accesible el hecho de utilizar técnicas de Aprendizaje Automático para solucionar este problema. Estas técnicas permiten automatizar tareas y solucionar problemas con los que los métodos tradicionales de la ingeniería del software no son capaces, ya sea por las grandes cantidades de información a procesar o la dificultad de hallar algún patrón que permita converger (llegar a la solución) al algoritmo [Orriols-Puig, 2008]. Dentro del Aprendizaje Automático existe multitud de técnicas, este Trabajo Final de Grado se ha centrado en el aprendizaje supervisado. Este paradigma de aprendizaje consiste en obtener un modelo predictivo a partir de los datos del problema. Una de las características fundamentales es que dicho modelo generaliza. Los principales sistemas clasificadores que pertenecen a este paradigma son:

Aprendizaje Perezoso. Se caracteriza por no generar ningún modelo y se basa en una función de distancia para clasificar, un algoritmo muy preciso es *Instance Based Classifier (IBK)*.

Redes Neuronales. Inspiradas en el sistema nervioso, se interconectan neuronas o perceptrones entre sí para producir un estímulo de salida, *Back Propagation (BP)* es un algoritmo de entrenamiento común.

Modelos probabilísticos. Estos se fundamentan en el razonamiento estadístico, el más famoso es *Naive Bayes (NB)*.

Máquinas de Soporte Vectorial. Estos algoritmos se basan en el principio de minimización del riesgo estructural, el más preciso es *Sequential Minimal Optimization (SMO)*.

Árboles de decisión. Modelan los datos mediante estructuras recursivas en forma de árbol, el algoritmo más conocido es *C4.5* o *J48*.

No hay un clasificador que sea el mejor, cada uno de estos cinco presenta una serie de ventajas y desventajas, de este modo se adaptan mejor o peor según el problema a resolver. Existe un conjunto de métodos avanzados de clasificación denominados *Ensemble*, estos agrupan las predicciones de múltiples clasificadores como los ya mencionados obteniendo así una mayor precisión. Destacan dos familias:

Bagging. Se trata de generar *data sets* a partir del original con el mismo número de instancias elegidas al azar y con reemplazo, seguidamente se modela cada uno de estos mediante un sistema clasificador independiente y posteriormente se realiza

una votación dando como resultado el modelo final, *Random Forest* es uno de los algoritmos más precisos de la literatura.

Boosting. Este tipo de algoritmos se basan en la mejora continua: genera clasificadores simples o *weak learners* sucesivamente, de tal forma que la salida del primero será la entrada del segundo, la combinación de todos estos genera el modelo final o *strong learner*, *Gradient Boosting* es la técnica más precisa de su familia.

Debido a su naturaleza, *Gradient Boosting* resulta ideal para modelar al alumnado con una precisión muy elevada. Así pues, se ha construido (1) el sistema *GBM*: un algoritmo que implementa la técnica *Gradient Boosting* utilizando árboles de regresión, y (2) la plataforma *PGUI*, una interfaz gráfica que facilita el uso de *GBM* y permite realizar *stratified 10 fold cross validation*. Para comprobar su validez y que realmente una técnica que produce múltiples *weak learners* es más precisa que una que construye un solo *strong learner*, *GBM* se ha comparado con los sistemas clasificadores *IBK*, *SMO*, *NB* y *J48* y se ha realizado una validación estadística. Esta validación afirma que existe una diferencia estadísticamente significativa entre los resultados de *GBM* y *NB*, en las demás técnicas no se puede afirmar lo mismo aunque *GBM* da mejor resultado en promedio para los 13 *data sets* modelados.

Una vez se ha validado el sistema, se ha procedido a modelar al alumnado. Para ello se han realizado dos experimentos, el primero para probar la bondad del algoritmo y el segundo para automatizar la tarea del modelado del rendimiento del alumnado. Se ha partido de una base de datos que corresponde a las notas reales de los alumnos en una asignatura. Esta base de datos no se ha podido utilizar directamente ya que no tiene definido el atributo de salida o clase y está en formato *CSV*. Por lo tanto, en base a los criterios de evaluación y teniendo en cuenta todas las notas, se ha logrado generar la clase (en el primer experimento se trata de la nota final de la asignatura y en el segundo de la nota del examen final) y se ha podido convertir al formato *ARFF* mediante la plataforma *Weka*. En este punto se ha observado que este *data set* es muy complicado de modelar correctamente debido a los dos siguientes problemas:

1. El rango de valores mediante el que se representan las notas contempla los valores numéricos del cero al diez y *NP*. Este primer problema se ha solucionado añadiendo granularidad al sistema: se han añadido tantos atributos *dummy* como notas en el *data set*, estos mapean los valores *NP* y posteriormente se han substituido los *NP* por cero en las notas. Mediante esta solución se ha comprobado que los cinco algoritmos estudiados (*GBM*, *IBK*, *SMO*, *J48* y *NB*) aumentan en promedio un 21,14% su precisión.
2. Las clases no están balanceadas, así pues no hay un número similar de instancias por clase y los algoritmos en general asumen una distribución de clases uniforme (cada una tiene el mismo peso). Este problema se puede solucionar de dos

formas distintas, añadiendo (1) pesos a cada clase, aunque esta estrategia tiene el inconveniente de que favorece las clases más frecuentes, esto aumenta la precisión general a costa de no clasificar correctamente muchas de las instancias minoritarias [Orriols-Puig, 2008], y (2) datos sintéticos para equilibrar el número de instancias por clase por ejemplo con *SMOTE* [Orriols-Puig, 2008]. No se ha aplicado ninguna de las técnicas puesto que este estudio se ha dejado como líneas de futuro.

En el primer experimento, utilizando el *data set* docente se ha observado que *GBM* sigue dando la mejor precisión, esta es muy parecida a la precisión de *J48*, esto se debe a que ambas técnicas utilizan árboles de decisión y estos se adaptan muy bien al problema. Estos experimentos con la base de datos docente han servido para probar la validez del sistema.

En el segundo experimento *GBM* está muy por encima de las demás técnicas tradicionales. Estos resultados muestran que *GBM* es fiable y por consiguiente el cuerpo docente puede utilizarlo para predecir el comportamiento de sus alumnos en las distintas asignaturas y encarrilar a los alumnos tomando las medidas oportunas en caso de que *GBM* así lo detecte.

Al tratarse de un sistema supervisado se conocía previamente el atributo de salida. Para evaluar lo bien que se comporta el sistema, una vez se ha hallado una configuración que aumenta al máximo la precisión del sistema evaluada mediante una validación cruzada, se construiría un modelo utilizando todas las instancias del *data set* docente y se predeciría con un número mucho mayor de instancias, posteriormente el resultado se supervisaría por el personal docente.

8.2. Conclusiones

El objetivo general del presente Trabajo Final de Grado ha sido automatizar el proceso de modelar el rendimiento académico. Para alcanzarlo se han definido unas metas individuales, la consecución de las cuales ha permitido obtener, para cada uno de ellos, las siguientes conclusiones.

Revisar las principales técnicas del Aprendizaje Automático. Existe un número muy elevado de técnicas en el campo del Aprendizaje Automático y para cada una de ellas muchas variantes. En el capítulo 3 se han mostrado los cinco tipos de sistemas clasificadores supervisados más utilizados, estos son: Aprendizaje Perezoso, Redes Neuronales, Modelos probabilísticos, Máquinas de Soporte Vectorial y Árboles de decisión. Existen métodos que agrupan varios sistemas clasificadores para construir el modelo final generalmente con una mayor precisión, estas se denominan *Ensemble*. Las dos principales familias son *Bagging* y *Boosting* y funcionan mejor con árboles de decisión como base. No hay una técnica mejor

que otra, cada una presenta una serie de características que pueden ser útiles dependiendo del problema. Se ha optado por construir un *Gradient Boosting* con árboles de regresión como base, ya que es la técnica más precisa de la familia y al modelarse con árboles es relativamente interpretable, estas características hacen que sea la técnica más atractiva para modelar al alumnado de las presentadas en el documento.

Desarrollar un algoritmo *Gradient Boosting* y compararlo con otras técnicas.

Gradient Boosting es la técnica más apta para el problema, por ello se ha desarrollado el sistema *GBM*. Este recibe la configuración de entrada vía línea de comandos e implementa un algoritmo *Gradient Boosting* con árboles de regresión como base. Para obtener el mayor rendimiento posible, se ha implementado en *C++* y se ha hecho uso de *GDB*, *Valgrind* y *Gprof*. Con la finalidad de facilitar al usuario el uso de *GBM*, se ha construido con *Python* la interfaz gráfica *PGUI*. Para realizar la experimentación se (1) han particionado mediante *DCoL* 13 *data sets* de forma estratificada, cada uno de estos en diez pares *train-test*, (2) ha buscado una configuración óptima para el algoritmo, (3) se ha realizado una validación cruzada, se ha comparado su puntuación mediante la puntuación *F* (*F1*) con otras técnicas muy relevantes en la literatura (*IBK*, *SMO*, *J48* y *NB*) y por último (4) se ha realizado una validación estadística mediante la herramienta *SCI2S multiple test*, los resultados del experimento han sido los esperados: *GBM* se sitúa como la técnica con mayor precisión en promedio para los 13 *data sets* y existe una diferencia estadísticamente significativa entre *GBM* y *NB* aunque no se puede afirmar lo mismo para las demás técnicas. La experimentación realizada ha mostrado que efectivamente las técnicas de *Gradient Boosting* tienen en promedio una precisión más elevada que el resto de técnicas, con lo que las convierte en firmes candidatos para solucionar cualquier tipo de problema de Minería de Datos.

Aplicar *Gradient Boosting* a datos docentes con la finalidad de modelar al

alumnado. Modelar al alumnado es una tarea muy compleja debido al comportamiento poco predecible de los humanos. Modelar de forma precisa el comportamiento de los alumnos en las diversas asignaturas resulta un objetivo clave para el cuerpo docente. De esta forma el profesor podría predecir los resultados del examen final de cada alumno y en los casos necesarios aplicar acciones correctivas y preventivas para que éstos superen satisfactoriamente el curso. Mediante el estudio del estado del arte de las técnicas del Aprendizaje Automático se ha logrado modelar con suficiente precisión al alumnado. Así el personal docente dispone de una herramienta con la que automatizar el proceso de modelaje del rendimiento académico del alumnado. *GBM* es una herramienta que en general proporciona unos resultados significativamente superiores a los obtenidos mediante las técnicas tradicionales del Aprendizaje Automático como se ha visto en

la experimentación, además al tener un coste computacional relativamente bajo, permite procesar muchos más datos en un mismo espacio de tiempo. Esto significa que el profesor puede hacer uso de muchos más datos históricos (cursos de años pasados) y construir un modelo más exacto.

Como conclusiones generales, se ha logrado modelar el rendimiento académico con gran precisión mediante la técnica *Gradient Boosting*. A continuación se muestra el análisis *DAFO* (Debilidades, Amenazas, Fortalezas y Oportunidades) de la técnica implementada (ver tabla 8.1).

Fortalezas	Debilidades
<ul style="list-style-type: none"> - Precisión elevada. - Rapidez en generar modelos. - Rapidez en predecir. - Relativamente interpretable. - Pocos parámetros configurables. - Paralelizable (para cada clase). 	<ul style="list-style-type: none"> - Facilidad en obtener sobre ajuste. - Puede ocupar mucho espacio en memoria. debido a la gran cantidad de modelos que se pueden utilizar.
Oportunidades	Amenazas
<ul style="list-style-type: none"> - El algoritmo de particionado de variables se puede mejorar investigando otras técnicas - Se puede tener en cuenta el peso de cada atributo de manera simple. - La función <i>gamma</i> puede ser calculada con mayor precisión. 	<ul style="list-style-type: none"> - Es un algoritmo complejo que puede resultar difícil de implementar y esto puede ahuyentar a los practicantes. - Existen algoritmos más simples que dan un resultado similar.

TABLA 8.1: Análisis *DAFO* de *Gradient Boosting*.

El análisis *DAFO* de la tabla 8.1 resume las conclusiones acerca de las características del *Gradient Boosting* con árboles de decisión como base implementado. Las fortalezas representan las principales ventajas de la técnica, las debilidades muestran sus inconvenientes, las oportunidades sugieren líneas de investigación y las amenazas presentan debilidades de la técnica frente a otras técnicas.

Gradient Boosting cuenta con seis fortalezas principales, se ha observado que es la técnica más precisa de la literatura, es rápida modelando y rápida en predecir, al utilizar árboles de decisión como base resulta relativamente interpretable (sección 6.3.3), hallar una buena configuración no es muy problemático ya que tiene pocos parámetros configurables y por último es paralelizable, permite generar los árboles para cada clase de forma paralela incrementando así la rapidez de computo.

Referente a las debilidades de la técnica se han detectado dos, una es la facilidad por resultar en *overfitting*, por ello se debe llegar a un compromiso entre los parámetros configurables por medio de una validación cruzada y la otra es que según estos

parámetros el modelo final puede ocupar mucho espacio en memoria.

Por otro lado se han observado tres oportunidades principales, se puede (1) utilizar una técnica más sofisticada para el algoritmo de particionado de variables más eficaz, (2) tener en cuenta el peso de cada atributo de forma sencilla, esto es dar más peso a aquellas variables que resulten más importantes, para ello deberá realizarse un estudio previo e identificar dichos pesos y (3) calcular la función *gamma* de una forma más precisa como indica [Friedman \[2000\]](#).

Finalmente, se ha evidenciado que el algoritmo presenta una serie de amenazas debido a su complejidad a la hora de implementarlo y existen otros algoritmos más sencillos que dan un resultado similar según el problema a tratar, por eso hay que estudiar previamente el problema y valorar si vale la pena el esfuerzo en implementar un *Gradient Boosting* o por el contrario otra técnica más sencilla.

9

Líneas de futuro

Expuestas las conclusiones en el capítulo anterior, a continuación se proponen unas líneas de futuro o de continuación y ampliación del trabajo. Estas se fundamentan en la aplicación de mejoras y ampliaciones al algoritmo implementado en el presente trabajo.

Paralelizar el algoritmo. *Gradient Boosting* admite paralelización. Es posible aplicar la paralelización en la generación de árboles por clase, como cada clase requiere un modelo independiente, es posible tener un proceso paralelo para cada una de estas. Una posibilidad sería utilizar una librería específica como por ejemplo *OpenMP* ¹. Esta librería permite la paralelización de programas escritos en *C/C++* de manera transparente al programador. A continuación se muestra un pseudocódigo basado en *C* del funcionamiento de esta librería.

```
#include <omp.h>
...
#pragma omp parallel for private (i, numeroClases) shared(dataset)
for (i=0; i<numeroClases; i++) {
    generarArbol();
    evaluarArbol();
}
```

¹<http://openmp.org/wp/>

Como se puede apreciar en el código anterior es muy sencillo aplicarlo, pero es necesario hacer un estudio previo para valorar la ganancia en velocidad.

Utilizar una técnica más eficaz a la hora de particionar las variables. Hasta ahora se utiliza la ganancia en varianza para seleccionar la mejor partición de las variables (capítulo 3). Esta es la técnica estándar en la literatura, pero dependiendo del problema otras posibilidades serían más efectivas. Se propone hacer un estudio de las alternativas posibles de cara a la partición de las variables.

Tener en cuenta el peso de cada atributo. Hasta ahora no se ha tenido en cuenta la importancia de cada atributo en el modelado, es decir, todos tienen la misma importancia. No obstante en el mundo real esto puede no ser así, hay atributos que pueden ser más importantes que otros y por consiguiente tener esto en cuenta puede aumentar la precisión de manera significativa. Se propone el estudio de dos mecanismos de identificación de pesos distintos, el primero y más sencillo es otorgar menos peso a aquellos atributos con menos ganancia en varianza según el atributo de salida, el segundo es realizando un estudio de los atributos utilizando por ejemplo una técnica de análisis de componentes principales (*PCA*) [James et al., 2013]. *PCA* halla las dimensiones que más influyen en los datos, esto es, que describen más varianza en los atributos de entrada, se trata de una técnica no supervisada. Estas técnicas penalizan el rendimiento del algoritmo ya que requieren un cómputo adicional, especialmente *PCA*.

Calcular la función *gamma* de una forma más eficaz. La implementación actual sigue las recomendaciones dadas en [James et al., 2013]. Por lo tanto cada nodo hoja del árbol tiene una *gamma* (función que da la predicción de cada nodo hoja del árbol) constante. Pese a que este proceso es muy rápido, se pierde precisión. Para obtener una mayor precisión existen otras optimizaciones para calcular este valor [Friedman, 2000]. Se propone realizar un estudio del balance o *trade-off* entre tiempo de cálculo y precisión utilizando distintas funciones *gamma*.



Software externo: guía de referencia

A.1. Ficheros *ARFF*

El formato *Attribute-Relation File Format* (*ARFF*) con el paso del tiempo y debido a su uso masivo se ha convertido casi en un estándar [Orriols-Puig, 2008]. La estructura de este formato se divide en tres partes: (1) cabecera, (2) declaraciones de atributos y (3) sección de datos.

Cabecera. indica el nombre del *data set* mediante un *string*.

```
@relation <nombre relacion>
```

Declaraciones de atributos. Hay tantas líneas como atributos en el *data set*. Se indica el nombre del atributo con un *string* y el tipo puede ser *numeric* (número real), *integer* (número entero), *date* (fecha), *string* y enumerado (típicamente los atributos de salida).

```
@attribute <nombre atributo> <tipo>
```

Sección de datos. Se asignan los valores para cada uno de los atributos.

```
@data <valor1>,<valor2>,...,<valor clase>
```

A modo de ejemplo se muestra un fragmento del *data set* docente procesado utilizando atributos *dummy* presentado en el capítulo 6.

```
@relation NotasProcesadas-weka
@attribute Ejercicio-EC1 numeric
@attribute Ejercicio-EC2 numeric
@attribute Midterm numeric
@attribute Ejercicio-EC3 numeric
@attribute ExamenFebrero numeric
@attribute NP-EC1 numeric
@attribute NP-EC2 numeric
@attribute NP-Midterm numeric
@attribute NP-EC3 numeric
@attribute NP-EFebrero numeric
@attribute class SUFICIENTE,BIEN,SUSPENDIDO,NOTABLE,EXCELENTE
@data
7.9,0,6.5,2,5.5,0,1,0,0,0,SUFICIENTE
10,7.75,6.6,2,6,0,0,0,0,0,BIEN
0,0,4.6,0,0,1,1,0,1,1,SUSPENDIDO
```

A.2. Manual de *Weka*

Weka es una plataforma implementada en *Java* que proporciona una extensa colección de algoritmos de Máquinas de conocimiento desarrollados por la universidad de *Waikito* (Nueva Zelanda). La licencia es *GPL*, es decir, este programa es de libre distribución y difusión. En este apartado del anexo se muestra un breve manual de *Weka* con la intención de justificar como (1) se ha convertido el formato *CSV* a *ARFF* y (2) configurar y aplicar algoritmos. Para facilitar al usuario el uso del manual, las diferentes secciones del programa están marcadas en la figura A.1 mediante números.

Conversión de formato *CSV* a *ARFF*. *Weka* facilita mucho el proceso de conversión entre formatos, simplemente hay que navegar a la ventana de *Explorer*, en la pestaña *Preprocess* [1] cargar el *CSV* mediante *Open File*, seleccionar que atributos se quieren eliminar y guardar el *data set* en formato *ARFF* utilizando el botón *Save*.

Configurar y aplicar algoritmos. En la ventana de *Explorer*, una vez se ha cargado el *data set* hay que navegar a la pestaña *classify* [2]. Mediante el botón *Choose* [3]

podemos elegir la técnica a utilizar, estas están categorizadas por carpetas, por ejemplo la carpeta *functions* tiene entre otros el *Sequential Minimal Optimization* (*SMO*). Suponiendo que se ha elegido un algoritmo como el ya mencionado, si se hace *click* encima de este [4] (delante del botón *Choose*) se abre una ventana donde se puede configurar, mediante *OK* se aplican los cambios. El tipo de experimento también se puede configurar, en *Test options* [5] se puede seleccionar por ejemplo *10 fold Cross-validation*. Si se pulsa *Start* [6] se ejecuta el algoritmo con la configuración guardada y en la ventana *Classifier output* [7] se muestra el resultado obtenido así como las medidas del error.

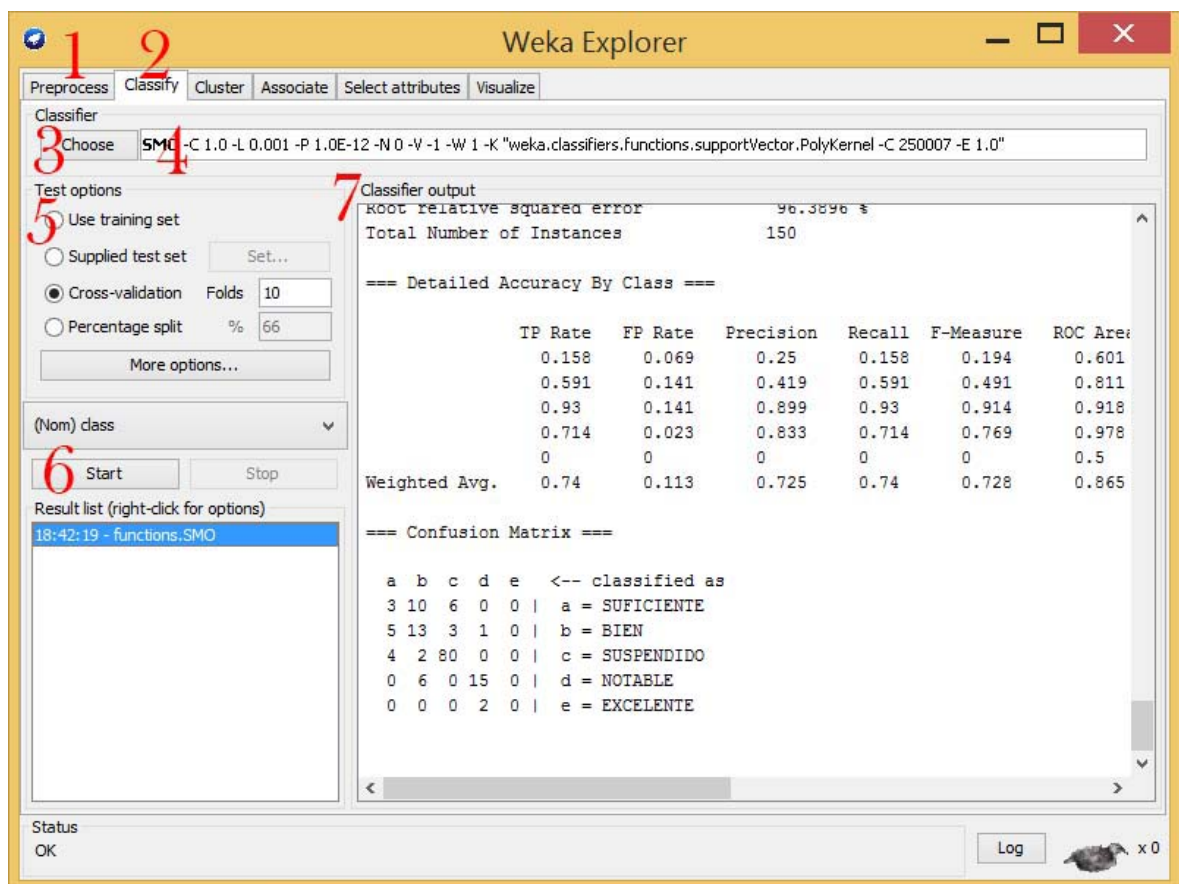


FIGURA A.1: Pestaña Explorer en *Weka*. Se ha aplicado un *10 fold cross-validation* con un *SMO* configurado con un *Kernel* polimómico de primer grado. El promedio de las medidas *F Measure* (*F1*) ha sido 72,80 %.

A.3. Manual de *PGUI* y *ModelToGpv*

A.3.1. *PGUI*

El sistema *GBM* implementa la técnica *Gradient Boosting* utilizando árboles de regresión como base. El usuario introduce vía comandos los parámetros de entrada del algoritmo (Capítulo 4), este se ejecuta y construye el modelo o predice según la configuración deseada. *PGUI* es una interfaz gráfica que se ha construido con la finalidad de facilitar al usuario la experimentación con *GBM* y poder realizar de forma muy sencilla una *stratified 10 fold cross validation*.

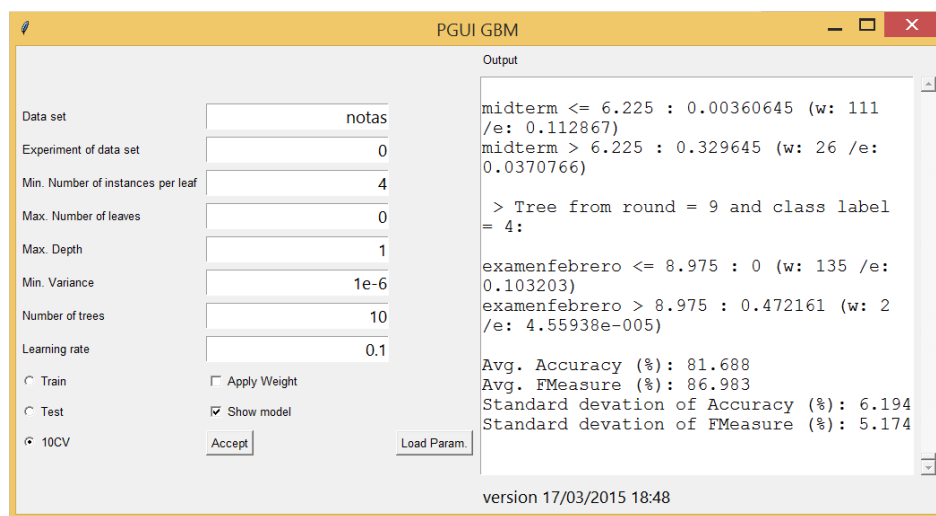


FIGURA A.2: Plataforma *PGUI*, muestra la configuración y el output de *GBM* en utilizado en la sección 6.3.2.

El funcionamiento de *PGUI* es muy similar al de *Weka*. La ventana principal está dividida en dos secciones distintas (figura A.2):

Sección izquierda. Donde se configura el experimento. El algoritmo está pensado para trabajar con bases de datos particionadas por *stratified 10 fold cross validation*, así pues el usuario introduce el nombre del *data set* y automáticamente el programa busca la carpeta con el mismo nombre, dentro de esta carpeta se hallan los diez pares *train-test*. Si se elige la opción *10 CV* el algoritmo entrena y predice con cada uno de los diez pares *train-test*, al finalizar calcula el promedio y la desviación típica de las medidas *ACC* y *F1* teniendo en cuenta cada experimento. Por el contrario si se selecciona la opción *Train* o *Test*, se realiza el experimento únicamente con el archivo *train* o *test* del número de experimento introducido por el usuario, por ejemplo *notas-10-0-tra.dat* corresponde al archivo número cero de entrenamiento del *data set* notas. El usuario puede configurar el

mínimo número de instancias por nodos hoja, el máximo número de nodos hoja, la máxima profundidad del árbol, la varianza mínima, el número de árboles y el ratio de aprendizaje. A parte de los parámetros de configuración del *Gradient Boosting*, se puede visualizar el modelo generado marcando la casilla *Show model*. Una vez se pulsa el botón *Accept* se ejecuta el experimento.

Sección derecha. Donde se muestra la respuesta del experimento.

A.3.2. ModelToGpv

Todos los árboles mostrados en el presente Trabajo Final de Grado se han construido mediante *Graphviz* (*Graph Visualization Software*). Este es un paquete de herramientas de código abierto construido principalmente por *AT&T Labs Research* que funciona bajo lenguaje *DOT* y permite generar grafos. Para facilitar la representación del modelo generado por los árboles, se ha construido un script *Python* denominado *ModelToGpv.py* que recibe un fichero llamado *input.txt* con el modelo (tal y como se muestra en la pantalla *output* de *PGUI*) y genera otro fichero *output.txt* con el código traducido a *DOT* (este código se puede ejecutar mediante *Graphviz*). A continuación se muestra un ejemplo de su funcionamiento.

Fichero de entrada *input.txt*

```
examenfebrero <= 3.475 : 0 (w: 62 /e: 0)
examenfebrero >3.475
| examenfebrero <= 5.7
| | ejercicio-ec1 <= 5.45 : 0 (w: 7 /e: 0)
| | ejercicio-ec1 >5.45 : 0.538462 (w: 26 /e: 0.248521)
| examenfebrero >5.7
| | midterm <= 1.75 : 1 (w: 2 /e: 0)
| | midterm >1.75 : 0.0285714 (w: 35 /e: 0.0277551)
```

Fichero de salida *output.txt*

```
digraph G {
graph [splines=line, nodesep=1]
'0-examenfebrero(0)''->'0(0)'' [label='<=3.475 (w:62/e:0)']
'0-examenfebrero(0)''->'1-examenfebrero(0)'' [label='>3.475']
'1-examenfebrero(0)''->'2-ejercicio-ec1(0)'' [label='<=5.7']
'2-ejercicio-ec1(0)''->'0(1)'' [label='<=5.45 (w:7/e:0)']
'2-ejercicio-ec1(0)''->'0.538462(0)'' [label='>5.45 (w:26/e:0.248521)']
'1-examenfebrero(0)''->'2-midterm(0)'' [label='>5.7']
'2-midterm(0)''->'1(0)'' [label='<=1.75 (w:2/e:0)']
```

```

    '2-midterm(0)'->'0.0285714(0)' [label='>1.75 (w:35/e:0.0277551)']
}

```

Modelo generado. En la figura A.3 se muestra el modelo generado en *Graphviz* por el código del fichero *output.txt*.

Se representan los árboles mediante nodos, estos pueden ser nodos internos y nodos hoja.

1. Los nodos internos se representan con la siguiente nomenclatura:

<nivel>-<atributo>(<repeticiones>)

2. Los nodos se representan con la siguiente nomenclatura:

<valor>-(<repeticiones>)

En el modelo generado, el valor de regresión 0 (en un nodo hoja) se puede interpretar como que no es la clase que se busca y si este valor es la unidad significa que es la clase que se busca, no se puede afirmar nada en caso contrario. Las relaciones se muestran con flechas, cada una de estas representa una partición. Estas pueden ser (1) mayor o (2) menor o igual a un número determinado. En el segundo caso irá acompañado del número de instancias w que se obtienen por esa rama y el error relativo cometido (la suma del error dividido por la suma de instancias) e .

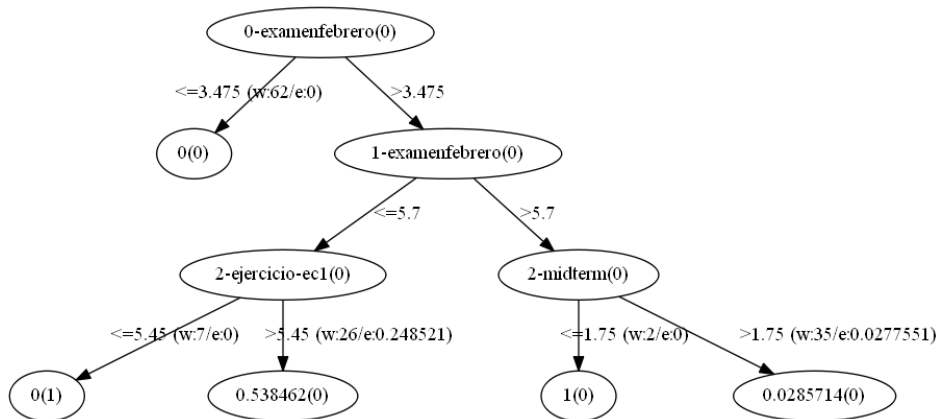
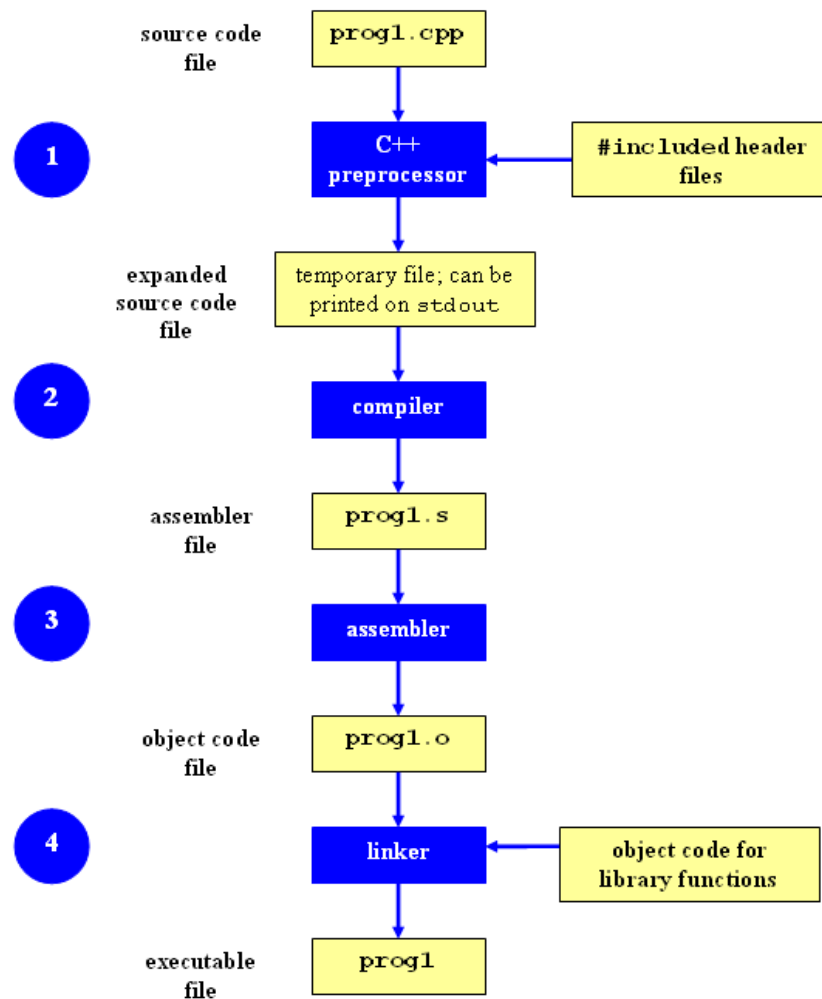


FIGURA A.3: Árbol generado por el código de *output.txt*.

A.4. Optimización y depuración del código

En el capítulo 4 se ha mencionado el uso de herramientas en un sistema *Unix* para depurar el código. Estas son *GDB*, *Valgrind* y *Gprof*. En el presente apartado del anexo se describe el proceso de compilación del lenguaje de programación *C++* y el funcionamiento de estas tres herramientas.

FIGURA A.4: Proceso de compilación en *C++* [McMahon, 2015].

En la figura A.4 se muestra el proceso de compilación en *C++*. El preprocesador copia los contenidos incluidos en la cabecera (*.h*) en el código fuente (*.cpp*). Este código producido por el preprocesador se compila en ensamblador para la plataforma (*.s*), posteriormente se convierte en código objeto (archivo binario no ejecutable) para la plataforma (*.o*) y finalmente se hace un ejecutable.

En un sistema operativo Microsoft utilizando por ejemplo *Visual Studio*, se programa el código fuente (*.cpp*) y las cabeceras (*.h*) y el propio programa utiliza su compilador para generar automáticamente el ejecutable (*.exe*) mediante el que se podrá acceder al programa. Si se utiliza un sistema *Unix* se debe generar un archivo *Makefile* el cual genera y une los binarios necesarios para crear el ejecutable.

```

OBS = Config.o DataSet.o BinaryTree.o GradientBoosting.o Main.o
#DEBUGGING FLAGS
CFLAGS = -Wall -Wextra -std=c++11 -c -O0 -ggdb
OFLAGS = -o
P_FLAGS = -g -c
#MEMORY PROFILING FLAGS #CFLAGS = -Wall -Wextra -std=c++11 -c -O3 -pg -ggdb
-fno-inline -mtune=native -march=native -msse -msse2
-mssse3 -msse4 -mfpmath=sse -finline-functions
-mfpmath=sse -m64 -DLEVEL1_DCACHE_LINESIZE='getconf LEVEL1_DCACHE_LINESIZE'
#OFLAGS = -g -pg -fno-inline -o
#P_FLAGS = -g -pg -c
#RUNNING TAGS #CFLAGS = -Wall -Wextra -std=c++11 -c -O3 -mtune=native
-march=native -msse -msse2 -mssse3 -msse4 -mfpmath=sse
-finline-functions -mfpmath=sse -m64
-DLEVEL1_DCACHE_LINESIZE='getconf LEVEL1_DCACHE_LINESIZE'
#OFLAGS = -DLEVEL1_DCACHE_LINESIZE='getconf LEVEL1_DCACHE_LINESIZE' -o
#P_FLAGS = -c
all: executable
Config.o: Config.h Config.cpp
    g++ $CFLAGS Config.cpp
DataSet.o: Config.o DataSet.h DataSet.cpp NominalFeature.h
    g++ $CFLAGS DataSet.cpp
BinaryTree.o: Config.o BinaryTree.h BinaryTree.cpp DataSet.o
    g++ $CFLAGS BinaryTree.cpp
GradientBoosting.o: GradientBoosting.cpp GradientBoosting.cpp Config.o DataSet.o
BinaryTree.o
    g++ $CFLAGS GradientBoosting.cpp
Main.o: Config.o DataSet.o BinaryTree.o GradientBoosting.o Main.cpp
    g++ $CFLAGS Main.cpp
executable: $OBS
    g++ $OBS $OFLAGS GBM -lm -pthread
clean:
    rm -f *.o GBM core* *~

```

En el código anterior muestra un *Makefile*. Este se encarga de unir las cabeceras con el código fuente para producir los ficheros objeto mediante el compilador *g++*. Los comandos comentados bajo los títulos *DEBUGGING FLAGS*, *MEMORY PROFILING FLAGS* y *RUNNING TAGS* sirven para optimizar el proceso según los intereses del usuario. Si se pasan ficheros de *Windows* a *Linux*, estos pueden contener caracteres de la codificación que utiliza Windows que no existe en *Linux*, este problema se soluciona con el siguiente comando:


```
$dos2unix fichero_original >fichero_redireccionado
```

Para asegurar un código bien construido, se ha instalado *Debian* en una máquina virtual con el siguiente software libre: *GDB*, *Valgrind* y *Gprof*.

***GDB* o *GNU Debugger*.** Es un depurador, permite trazar y modificar la ejecución de un programa. Una particularidad es que el usuario puede modificar el valor de las variables internas del programa (al vuelo). En lo que sigue se muestra un ejemplo de su funcionamiento.

```
$cd /mnt/hgfs/SharedFolder/GBM
$make
$gdb GBM
$run --train=../DataSet/iris.arff
--maximumdepth=2 --minimumvariance=0.003
```

El código anterior se ha ejecutado en una máquina virtual con un sistema *Linux*. Navega a la ruta donde está el proyecto *GBM*, lo compila (make ejecuta el *Makefile*) generando los objetos y llama al sistema *GBM* mediante *run* con los parámetros deseados, en este caso carga el *data set Iris* con árboles de máximo dos niveles de profundidad y una varianza mínima de 0.003.

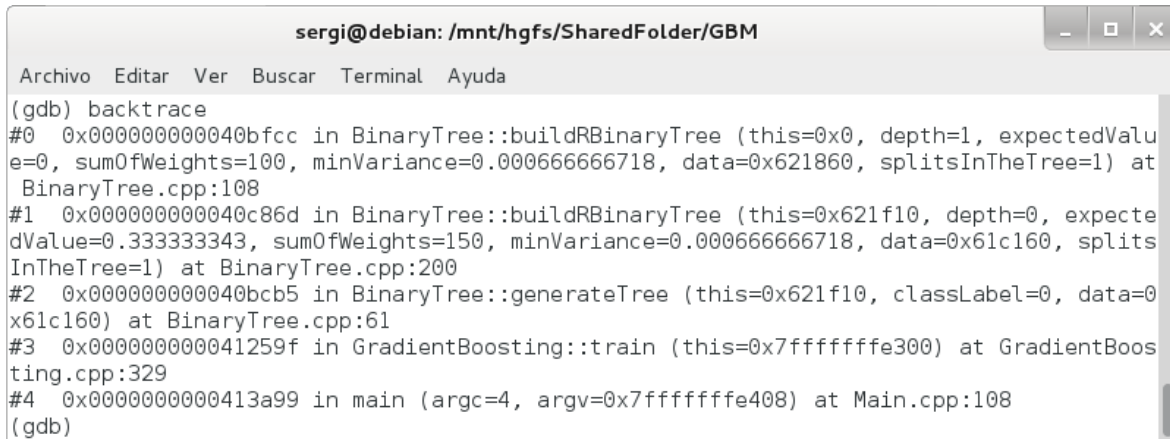
Al ejecutar el código se muestra un error (figura A.5) del tipo *Segmentation fault* en la línea 108 del fichero *BinaryTree.cpp* en la función *buildRBinaryTree*.



```
sergi@debian: /mnt/hgfs/SharedFolder/GBM
Archivo Editar Ver Buscar Terminal Ayuda
(gdb) run --train=../DataSet/iris.arff --maximumdepth=2 --minimumvariance=0.003
Starting program: /mnt/hgfs/SharedFolder/GBM/GBM --train=../DataSet/iris.arff --maximum
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
> Sergi Sancho Asensio <Binary> Regression tree version 2014.11.01-20:12
Configuration:
  Input file: ../dataset/iris.arff
  Model file: model.bin
  Min variance: 0.003
  Min number of instances per leaf: 2
  Maximum depth: 2
  Maximum number of leaves: 0
  Learning rate: 0.3
  Number of trees: 3
  Verbose: 0
Trying to open the file '../dataset/iris.arff'...
Number of features read = 5
Number of instances read = 150
I have read 0 unkown values.
Program received signal SIGSEGV, Segmentation fault.
0x000000000040bfcc in BinaryTree::buildRBinaryTree (this=0x0, depth=1, expectedValue=0,
108         totValueSum += data->getResidualAt(i, classLabel) * data->getWe
(gdb)
```

FIGURA A.5: Error al ejecutar el algoritmo.

Existe un comando de *GDB* llamado *\$backtrace* que traza la propagación del error. En la figura A.6 se muestra la ruta que ha seguido éste en el programa, se ha generado en la clase *BinaryTree* y se ha propagado en la clase *Gradient Boosting* y finalmente en el *Main*.



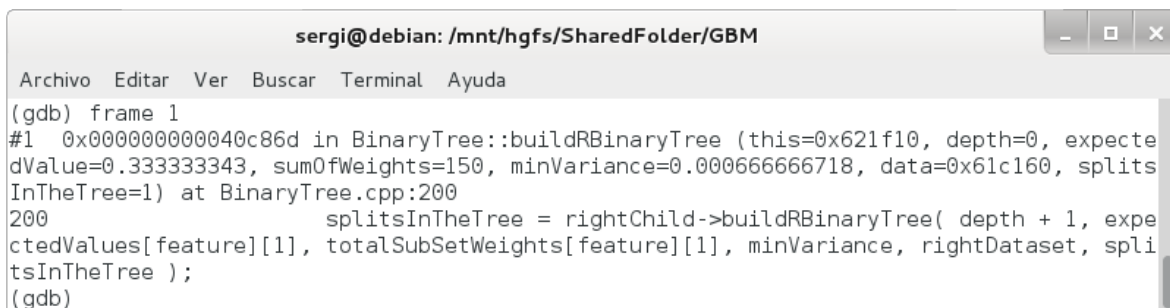
```

sergi@debian: /mnt/hgfs/SharedFolder/GBM
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
(gdb) backtrace
#0  0x000000000040bfcc in BinaryTree::buildRBinaryTree (this=0x0, depth=1, expectedValue=0, sumOfWeights=100, minVariance=0.000666666718, data=0x621860, splitsInTheTree=1) at BinaryTree.cpp:108
#1  0x000000000040c86d in BinaryTree::buildRBinaryTree (this=0x621f10, depth=0, expectedValue=0.333333343, sumOfWeights=150, minVariance=0.000666666718, data=0x61c160, splitsInTheTree=1) at BinaryTree.cpp:200
#2  0x000000000040bcb5 in BinaryTree::generateTree (this=0x621f10, classLabel=0, data=0x61c160) at BinaryTree.cpp:61
#3  0x000000000041259f in GradientBoosting::train (this=0x7fffffffe300) at GradientBoosting.cpp:329
#4  0x0000000000413a99 in main (argc=4, argv=0x7fffffffe408) at Main.cpp:108
(gdb)

```

FIGURA A.6: Uso de Backtrace.

Se puede acceder a cualquiera de estos cinco frames o rutas donde el error está presente, si accedemos al segundo frame mediante el comando *\$frame 1* obtendremos más detalles del error (figura A.7). El error está reconocido en la línea 200.



```

sergi@debian: /mnt/hgfs/SharedFolder/GBM
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
(gdb) frame 1
#1  0x000000000040c86d in BinaryTree::buildRBinaryTree (this=0x621f10, depth=0, expectedValue=0.333333343, sumOfWeights=150, minVariance=0.000666666718, data=0x61c160, splitsInTheTree=1) at BinaryTree.cpp:200
200      splitsInTheTree = rightChild->buildRBinaryTree( depth + 1, expectedValues[feature][1], totalSubSetWeights[feature][1], minVariance, rightDataset, splitsInTheTree );
(gdb)

```

FIGURA A.7: Frame 1.

Se puede parar la ejecución en la línea anterior mediante el comando *\$break 199* (figura A.8) y ejecutar mediante *run* el algoritmo con los mismos parámetros.



```
sergi@debian: /mnt/hgfs/SharedFolder/GBM
Archivo Editar Ver Buscar Terminal Ayuda
(gdb) break 199
Breakpoint 1 at 0x40c7cb: file BinaryTree.cpp, line 199.
(gdb)
```

FIGURA A.8: Comando *break 199*.

El error ahora no aparece, en cambio si se avanza la ejecución mediante el comando *\$n*, éste resurge. Si consultamos el valor de la variable *rightChild* mediante el comando *\$p rightChild*, *GDB* indica que no está declarada y por eso tiene asignada el valor *0x0* (figura A.9).



```
sergi@debian: /mnt/hgfs/SharedFolder/GBM
Archivo Editar Ver Buscar Terminal Ayuda
(gdb) p rightChild
$7 = (BinaryTree *) 0x0
(gdb)
```

FIGURA A.9: *GDB* indica que la variable *rightChild* no está declarada.

La solución de este error es declarar memoria en la *línea 199* para este puntero del tipo *BinaryTree* que se encargará de generar las ramas de la izquierda del árbol de decisión. Para salir del *GDB* se utiliza el comando *\$q*.

Valgrind. Se trata de un conjunto de herramientas que permite depurar problemas de memoria como fugas y rendimiento en programas. Se debe tener los *DEBUGGING FLAGS* activados en el *Makefile* para poder utilizar este programa. A continuación se muestra un ejemplo de uso.

```
$valgrind --track-origins=yes --leak-check=full
-v --show-reachable=yes ./GBM --train=./DataSet/iris.arff
--maximumdepth=2 --minimumvariance=0.003
```

Gprof. Es un *profiler* u optimizador de código fuente. Ayuda a identificar cuellos de botella (*bottlenecks*). Identifica los fragmentos de código que consumen más tiempo, rastrea el flujo del programa ayudando así a la depuración y a resolver problemas de rendimiento. Si hay alguna función en el código fuente que se llama muchas veces y no es excesivamente grande, se puede definir inline en *C++*, de

esta forma se consigue aumentar el rendimiento. En el *Makefile* se deben desactivar los *DEBUGGING FLAGS* y activar los *MEMORY PROFILING FLAGS* para poder utilizar esta funcionalidad.

```
$make $./GBM --train=../DataSet/iris.arff
--minimumnumberofinstancesperleaf=4 --maximumnumberofleaves=0
--maximumdepth=0 --minimumvariance=0.003 --numberoftrees=100
--learningrate=0.1 --verbose=0
$gprof GBM gmon.out >analysis.txt
```

En los comandos anteriores se ha compilado el proyecto, se ha ejecutado el algoritmo *Gradient Boosting* con una configuración determinada y posteriormente se ha utilizado el programa *Gprof* para generar un fichero *analysis.txt*.

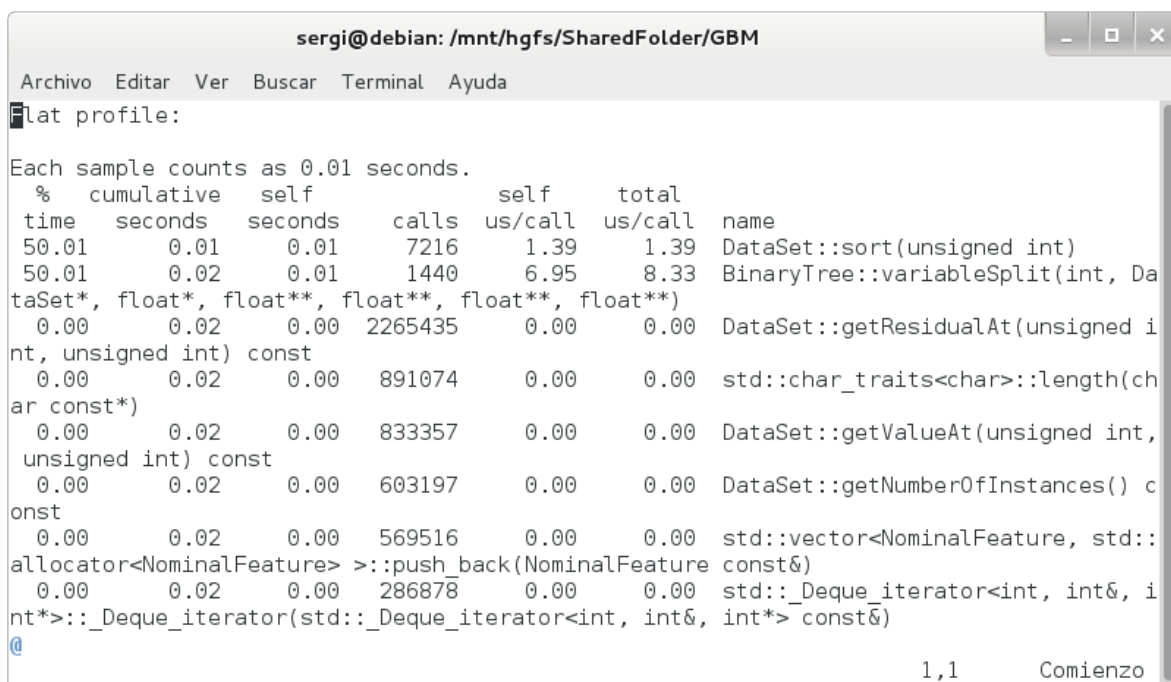


FIGURA A.10: Fichero *analysis.txt*.

El fichero *analysis.txt* (figura A.10) muestra una tabla con el tiempo consumido por cada función y el número de llamadas a estas. Como se puede apreciar en la imagen, la función que más tiempo consume es la de ordenar el *data set*, es lógico que así sea ya que es una operación costosa, depende directamente de la medida del *data set* y se llama 7216 veces.

A.5. *Doxygen*

Doxygen es el acrónimo de *dox* (document) y *gen* (generator). Es un programa bajo licencia *GPL* desarrollado para los sistemas *Unix*, *Windows* y *Mac OS X* que permite generar documentación para *C++*, *C*, *Java*, *C#*, *PHP*, etcétera. Esta documentación puede generarse en *HTML* y *Latex* entre otros. Para que *Doxygen* genere la documentación, se debe añadir un comentario a cada función del *header*. Este comentario debe estar estructurado en tres partes distintas: (1) explicación de la función, (2) parámetros de entrada y explicación de cada uno de estos y (3) parámetro de retorno y explicación. En lo que sigue se muestra un ejemplo aplicado al sistema *GBM* y su resultado (figura [A.11](#)), concretamente en la clase *BinaryTree*, función *computeVariance*.

```
/** * It computes the variance of a single instance.
 * @param value is the value for computing a sample mean.
 * @param sumSQValue is the sum of squared values.
 * @param weight is the normalization term.
 * @return the sample variance.
 */
float computeVariance( float value, float sumSQValue, float weight );
```

GBM
041215

Main Page
Classes
Files
Search

Class List
Class Index
Class Members

Public Member Functions | List of all members

BinaryTree Class Reference

Public Member Functions

BinaryTree ()
void generateTree (unsigned int classLabel, Data Set *data)
~BinaryTree ()
float computeVariance (float value, float sumSQValue, float weight)
int getNumberOfRegions ()
float makeAPrediction (const float *example) const
void saveToBinaryFile () const
void readFromBinaryFile ()
void readFromBinaryFile (std::ifstream &mFile)
void saveToBinaryFile (std::ofstream &oFile) const
void print (const Data Set *data)
void printToFile (const Data Set *data)
float getGamma (const float *example) const

Constructor & Destructor Documentation

BinaryTree::BinaryTree ()

The object constructor.

BinaryTree::~~BinaryTree ()

The object destructor.

Member Function Documentation

float BinaryTree::computeVariance (float value,
float sumSQValue,
float weight
)

It computes the variance of a single instance.

Parameters

value is the value for computing a sample mean.

sumSQValue is the sum of squared values.

weight is the normalization term.

Returns

the sample variance.

FIGURA A.11: Fragmento de la documentación generada por *Doxygen*.

Bibliografía

- Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22, pages 207–216, Washington D. C., USA, June 1993. ACM. 7
- David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991. ISSN 0885-6125. 60
- Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011. 59
- Jaume Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: Representations, generalisation, and run-time*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, October 2004. 11
- Stephen K. Bache and Moshe Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>. 27, 59, 82
- Birgi. Non-linear svm, 2014. URL <http://i.stack.imgur.com/1gvce.png>. 21
- Barry Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 4:14–24, 1986. xiv, 45, 46
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. ISBN 0534980538. 24, 27
- Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. 32
- Nicholas Chapman. Neuralbot, 1999. URL http://homepages.paradise.net.nz/nickamy/neuralbot/nb_about.htm. 17

- George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. ISSN 0932-4194. [14](#), [16](#)
- Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014. [32](#)
- Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. [56](#)
- Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, November 1996. ISSN 0001-0782. doi: 10.1145/240455.240464. [XIII](#), [5](#), [6](#)
- Albert Fornells. *Marc integrador de les capacitats de soft-computing i de knowledge discovery dels mapes autoorganitzatius en el raonament basat en casos*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, february 2006. [14](#)
- Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000. [2](#), [30](#), [33](#), [37](#), [39](#), [58](#), [62](#), [65](#), [89](#), [92](#)
- Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. ISSN 01621459. [56](#)
- João Gama, editor. *Knowledge Discovery from Data Streams*. Advances in Database Systems. Chapman and Hall/CRC, first edition, may 2010. ISBN 978-1439826119. [2](#)
- Salvador García and Francisco Herrera. An extension on ”Statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, December 2008. [56](#), [59](#), [61](#)
- Álvaro García-Piquer. *Facing-up challenges of multiobjective clustering based on evolutionary algorithms: Representations, scalability and retrieval solutions*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, december 2012. [2](#)
- Anthony Goldbloom. Kaggle, 2010. URL <http://www.kaggle.com>. [2](#), [39](#), [65](#), [79](#)
- Wolfgang Härdle and Heiko Lehmann. A neuron within a neural network., 2004. URL http://sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/tutorials/xlhtmlimg565.gif. [15](#)

- Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):289–300, 2002. doi: 10.1109/34.990132. 55
- Ignacio Icke. Diagram showing overfitting of a classifier, 2008. URL <http://upload.wikimedia.org/wikipedia/commons/1/19/Overfitting.svg>. XIII, 10
- IEEE-STD-830-1998. Especificaciones de los requisitos del software, 1998. URL http://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf. 39, 53
- G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer, 2013. ISBN 9781461471387. 3, 6, 7, 18, 23, 30, 34, 92
- George John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995. 17, 61
- Jose M. Leiva Murillo. *Extracción de características mediante criterios basados en teoría de la información*. PhD thesis, Universidad Carlos III de Madrid, Madrid, España, 2007. 9
- Xavier Llorà. *Aprenentatge artificial evolutiu emprant paral·lelisme de gra fi en el marc de la mineria de dades*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, february 2002. 11
- Xavier Llorà, Rohith Reddy, Brian Matesic, and Rohit Bhargava. Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In *GECCO*, pages 2098–2105, 2007. 8, 9
- Kurt McMahon. The c++ compilation process, 2015. URL <http://faculty.cs.niu.edu/~mcmahon/CS241/Images/compile.png>. 99
- Tom Mitchell. *Machine Learning*. Prentice Hall, Pittsburgh, 1997. ISBN 0070428077. 1, 14, 15, 20, 22, 23
- Indraneel Mukherjee, Cynthia Rudin, and Robert E. Schapire. The rate of convergence of adaboost. *Journal of Machine Learning Research*, 14(1):2315–2347, 2013. 33
- Oleg Okun and Giorgio Valentini, editors. *Applications of Supervised and Unsupervised Ensemble Methods*, volume 245 of *Studies in Computational Intelligence*. Springer, 2009. ISBN 978-3-642-03998-0. doi: 10.1007/978-3-642-03999-7. 30
- Genevieve Orr. Preventing overfitting, early stopping, 1999. URL <http://www.willamette.edu/~gorr/classes/cs449/figs/earlystop.gif>. 10

- Albert Orriols-Puig. *New challenges in learning classifier systems: mining rarities and evolving fuzzy models*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, November 2008. 2, 11, 44, 84, 86, 93
- Yifan Peng. Support vector machine hyperplane, 2013. URL <http://www.pengyifan.com/blog/wp-content/uploads/2013/09/svm.png>. 21
- John Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. 20, 22, 60
- John Platt. Fast training of support vector machines using sequential minimal optimization. Technical report, Microsoft Research, 1 Microsoft Way, Redmond, WA 98052, USA, 2000. 22
- Robi Polikar. Combining classifiers2, 2008. URL http://images.scholarpedia.org/w/images/8/82/Combining_classifiers2.jpg. XIII, 30
- John R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, 1993. ISBN 1-55860-238-0. 23, 61
- Andreu Sancho-Asensio, Albert Orriols-Puig, and Elisabet Golobardes. Robust on-line neural learning classifier system for data stream classification tasks. *Soft Computing*, 18(8):1441–1461, 2014. doi: 10.1007/s00500-014-1233-9. 16, 60
- David J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman & Hall, third edition, 2004. ISBN 1-58488-440-1. 57, 79
- Jamie Shotton, Toby Sharp, Alex Kipman, Andrew W. Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Commun. ACM*, 56(1):116–124, 2013. doi: 10.1145/2398356.2398381. XIV, XIV, 32
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, an Introduction*. The MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-19398-1. 11
- Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, USA, 1995. 20, 23
- Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, september 1990. ISSN 0018-9219. 16

- Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufmann, third edition, January 2011. ISBN 978-0-12-374856-0. [3](#), [41](#), [45](#), [58](#), [61](#)
- Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey McLachlan, Angus Ng, Bing Liu, Philip Yu, Zhi-Hua Zhou, Michael Steinbach, David Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, January 2007. ISSN 0219-1377. [13](#), [17](#), [20](#), [23](#), [60](#)