UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

APRENDIZAJE AUTOMÁTICO EN CONJUNTOS DE CLASIFICADORES HETEROGÉNEOS Y MODELADO DE AGENTES

TESIS DOCTORAL

Agapito Ismael Ledezma Espino
Leganés, 2004

Departamento de Informática

Escuela Politécnica Superior
Universidad Carlos III de Madrid

APRENDIZAJE AUTOMÁTICO EN CONJUNTOS DE CLASIFICADORES HETEROGÉNEOS Y MODELADO DE AGENTES

AUTOR: Agapito Ismael Ledezma Espino
DIRECTORES: Ricardo Aler Mur
Araceli Sanchis de Miguel

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..... de ..... de 2004.

Presidente: D. .....

Vocal: D. .....

Vocal: D. .....

Vocal: D. .....

Secretario: D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de ..... de 2004 en .....

Calificación: .....

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mi esposa, Yolanda
A mi tío, Víctor

Agradecimientos

Odisea, esa es la palabra con la cual podría describir lo que ha sido para mí la culminación de este trabajo. Nadie me dijo hace años, en aquel pequeño país llamado Panamá, lo que era una tesis doctoral. Quizás, si en aquel momento en que decidí salir del país hubiese sabido lo que ello implicaba, me lo hubiera pensado dos veces, ... creo que no.

Así como Ulises no hizo su recorrido solo, tampoco yo lo he realizado solo. Ahora es cuando tengo que agradecer a las personas que de alguna forma me han ayudado a terminar lo que en estas páginas se refleja.

En primer lugar a mis directores de tesis, Araceli Sanchis y Ricardo Aler, por el tiempo que me han dedicado, por los consejos que me han dado y, porque no decirlo, por soportar mi constante autocrítica que en algunas ocasiones rozaba el pesimismo más puro y por haberme ayudado a superarlo y de esa forma concluir este trabajo.

A Daniel, por sus consejos y por el valioso tiempo que me ha dedicado desde que decidí hacer la tesis doctoral en el Grupo de Sistemas Complejos Adaptativos. A mis compañeros de SCALAB por su apoyo. A mis amigos de dentro y fuera del Departamento de Informática de la Universidad Carlos III, por hacerme sentir como en casa, aun cuando me encontraba tan lejos.

A la Agencia Española de Cooperación Internacional, por haber financiado, aunque fuese en parte, mis estudios de doctorado, dándole así el toque trágico de que toda buena odisea hace alarde. Al Departamento de Informática de la Universidad Carlos III de Madrid, por permitirme formar parte de él y gracias a cuyos medios, experiencia y soporte ha sido posible la finalización de esta investigación.

A mi *mama Lety* y mi *viejo*, que a pesar de la distancia siempre han estado a mi lado, y porque sé lo que esto significa para ellos. A mi hermano, Leo, por ser como es.

A mi tío Víctor y a mi tía *May*, por ser mis segundos padres y por sus consejos y soporte a lo largo de todos estos años tan lejos de casa.

Al resto de mi familia, por estar ahí los días que llamé, y por hacerme sentir más cerca de casa.

Por último, mi agradecimiento más profundo y sabiendo que con palabras no puedo expresar lo que siento, a Yolanda. Gracias por haber creído en mí desde siempre, por querer ser parte de mi proyecto de vida, y por haber dejado todo en Panamá para apoyarme. A ella debo en gran medida el concluir esta tesis y espero tenerla siempre conmigo para poder compensarla por todo lo que ha hecho por mí.

Resumen

Una de las áreas que más auge ha tenido en los últimos años dentro del aprendizaje automático es aquélla en donde se combinan las decisiones de clasificadores individuales con la finalidad de que la decisión final de a qué clase pertenece un ejemplo sea realizada por un conjunto de clasificadores. Existen diversas técnicas para generar conjuntos de clasificadores, desde la manipulación de los datos de entrada a la utilización de meta-aprendizaje. Una de las maneras en las que se clasifican estas técnicas es por el número de algoritmos de aprendizaje diferentes que utilizan con el fin de generar los miembros del conjunto. Aquellas técnicas que utilizan un único algoritmo para generar todos los miembros del conjunto se dice que generan un conjunto homogéneo. Por otra parte, aquellas técnicas que utilizan más de un algoritmo para generar los clasificadores se considera que generan un conjunto de clasificadores heterogéneo. Entre los algoritmos de generación de conjuntos heterogéneos se encuentra *Stacking*, el cual, además de generar los clasificadores del conjunto a partir de distintos algoritmos de aprendizaje, utiliza dos niveles de aprendizaje. El primer nivel de aprendizaje o nivel-0 utiliza los datos del dominio de manera directa, mientras que el meta-nivel o nivel-1 utiliza datos generados a partir de los clasificadores del nivel-0.

Un problema inherente a *Stacking* es determinar la configuración de los parámetros de aprendizaje del algoritmo, entre ellos, qué y cuántos algoritmos deben ser utilizados en la generación de los clasificadores del conjunto. Trabajo previos han determinado que no hay un número exacto de algoritmos a utilizar que sea el óptimo para todos los dominios. Tampoco está perfectamente definido qué algoritmos se deberían utilizar, aunque existen trabajos que utilizan algoritmos representativos de cada tipo.

Uno de los objetivos de esta tesis doctoral es la utilización de algoritmos genéticos como técnica de optimización para determinar los algoritmos que deben ser utilizados para generar el conjunto de clasificadores, al igual que la configuración de los parámetros de aprendizaje de éstos. De esta manera el método que se propone es independiente del dominio, mientras que la configuración de los parámetros de *Stacking* encontrada, dependerá del dominio.

El crecimiento del comercio electrónico y las aplicaciones en la *World-Wide-Web* ha motivado el incremento de los entornos en donde intervienen agentes. Estos entornos incluyen situaciones competitivas y/o colaborativas en donde el conocimiento que se posea sobre los individuos involucrados en el entorno, proporciona

una clara ventaja a la hora de tomar una decisión sobre qué acción llevar a cabo. Existen diversas formas de adquirir este conocimiento. Una de ellas es a través del modelado del comportamiento de los agentes.

A su vez, existen diversas formas de construir el modelo de un agente. Algunas técnicas utilizan modelos previamente construidos y su objetivo es intentar emparejar el comportamiento observado con un modelo existente. Otras técnicas asumen un comportamiento óptimo del agente a modelar con el fin de crear un modelo de su comportamiento.

Un segundo objetivo de esta tesis doctoral es la creación de un marco general para el modelado de agentes basándose en la observación del comportamiento del agente a modelar. Para ello se propone la utilización de técnicas de aprendizaje automático con el propósito de llevar a cabo la tarea de modelado basándose en la relación existente entre la entrada y la salida del agente.

Abstract

In the last years, one of the most active research areas in Machine Learning is that of ensembles of classifiers. Their purpose is to combine the decisions of individual classifiers so that all classifiers in the ensemble are taken into account in order to classify new instances. There are many techniques that generate such ensembles. Some manipulate the input data, while others use meta-learning. In general, ensembles can be homogeneous or heterogeneous. Homogeneous ensembles consist of several classifiers generated by the same learning technique, whereas heterogeneous ensembles contain classifiers generated by different algorithms. A well-known approach to generate heterogeneous ensembles is Stacking. Stacking uses two levels of learning. The first learning level or level-0 uses direct data from the domain, whereas the meta-level or level-1 uses data generated by classifiers from level-0.

An inherent problem to Stacking is to determine the right configuration of the learning parameters, like how many classifiers, and which learning algorithms, must be used in the generation of the ensemble of classifiers. Previous work have shown that there is no optimal decision for all the domains, although there are works that use representative algorithms from each type.

One goal of this thesis is to use Genetic Algorithms as an optimization technique in order to determine the type and number of algorithms to be used to generate the ensemble of classifiers, as well as the configuration of the learning parameters of these algorithms. The proposed method is domain independent, and the Genetic Algorithm will be able to adapt to particular domains.

The growth of the e-commerce and applications over the World-Wide-Web has motivated the increase of environments where agents can interact. These environment include competitive and/or colaborative situations where the knowledge about other individuals involved in the environment, provides a clear advantage when making decision about actions to perform. There are several ways to acquire this knowledge. One of them is by modeling the behavior of other agents.

There are several ways to construct an agent's model. Some techniques use previously constructed models and its goal to match the observed behavior with an existing model. Other techniques assume that the agent to model carries out an optimal strategy in order to create a model of its behavior.

In this thesis, a second approach to model agents will be used based on the observation of other agents behavior. In order to do this, a general framework that uses machine learning techniques for agent modeling is proposed.

“Sólo los que construyen sobre ideas, construyen para la eternidad”
- Emerson -

Índice general

Índice de figuras

Índice de tablas

Parte I

Introducción

Capítulo 1

Introducción

La aplicación de la Inteligencia Artificial a la resolución de problemas es algo muy común en la actualidad en la mayor parte de los sectores, aunque puede estar oculto bajo otros términos como *Data Mining*, *Business Intelligence* o incluso Robótica. Hoy en día estas incursiones de la Inteligencia Artificial varían desde las aplicaciones industriales hasta las aplicaciones en el mundo de los negocios, pasando por el mundo académico y de investigación. Cabe señalar que la Inteligencia Artificial incluye una diversa gama de técnicas capaces de resolver problemas en entornos complejos y dinámicos. Es en estos entornos en donde los resolvedores de problemas[1] tienen que ser capaces de encontrar una solución a la situación planteada ya sea individualmente o mediante la colaboración con otros resolvedores de problemas. En la primera parte de esta tesis doctoral se busca definir formas de combinar las soluciones propuestas por diferentes resolvedores de problemas. Por otro lado, en la segunda parte se abordan situaciones en donde los resolvedores de problemas poseen la característica de ser autónomos y pueden competir y/o colaborar con otros resolvedores de problemas, razón por la cual la posesión de información acerca de los individuos presentes en el entorno es una ventaja con vistas a resolver la tarea asignada.

El primero de los temas que aborda esta tesis doctoral se concentra en un tipo concreto de resolvedores de problemas como los son los conjuntos de clasificadores. En más detalle, el objetivo perseguido en esta parte de la tesis doctoral es la obtención de la configuración óptima de los parámetros del algoritmo de generación de conjuntos conocido como *Stacked Generalization*, o *Stacking*. En *Stacking* se genera un conjunto de clasificadores a partir de distintos algoritmos de aprendizaje, por lo que se puede decir que es un conjunto de clasificadores heterogéneo. Además de generar el conjunto de clasificadores a partir de distintos algoritmos, *Stacking* utiliza el concepto de *meta-aprendizaje*; es decir, el objetivo es aprender a

[1] Se ha elegido el nombre *resolvedores de problemas* como traducción del término inglés *problem solvers*.

combinar las predicciones dadas por los clasificadores (resolvedores de problemas) para un ejemplo dado (problema). Un problema inherente a *Stacking* es determinar la configuración óptima de los parámetros que intervienen en la generación del conjunto de clasificadores; entre ellos, el número y tipo de algoritmos que han de utilizarse en los dos niveles de aprendizaje que posee *Stacking*. En esta tesis se propone una solución a este problema mediante la utilización de algoritmos genéticos.

El segundo tema de la tesis se enmarca dentro de los sistemas de agentes, más concretamente en la tarea de adquisición de información sobre los demás agentes presentes en el entorno de acción de un determinado agente. Uno de los dominios elegidos para validar la propuesta de modelado de agentes en esta tesis doctoral es el simulador de fútbol de la *RoboCup*, utilizado ampliamente como entorno de prueba para el desarrollo de software multiagente.

Se asume que cada agente puede ser visto como un sistema entrada/salida que puede ser modelado mediante técnicas de aprendizaje automático. Así, un agente cuyo rol es el de delantero de un equipo de fútbol, puede obtener el modelo de un agente que actúa como portero del equipo contrario basándose en observaciones de su comportamiento. Comportamiento que es descrito a través de la relación existente entre las entradas y salidas del agente a modelar. Posteriormente, el delantero puede utilizar el modelo adquirido con el propósito de predecir las acciones del portero y anticiparse a éstas.

En este documento se describen, para ambos temas, el estado de la cuestión, la propuesta para resolver el problema planteado y los resultados obtenidos. En la Parte II se detalla la aplicación de algoritmos genéticos para encontrar la combinación óptima de parámetros de *Stacking* (*GA-Stacking*). En la Parte III se presenta una propuesta para llevar a cabo la tarea del modelado de agentes mediante técnicas de aprendizaje automático (MABT y MABO).

A continuación se describen en más detalle ambos temas para una mejor comprensión del resto del documento.

1.1. *GA-Stacking*

Una de las áreas de investigación dentro del aprendizaje automático que más se ha desarrollado en los últimos años son los *conjuntos de clasificadores*. Un conjunto de clasificadores es un grupo de clasificadores cuyas decisiones individuales son combinadas de alguna manera para clasificar nuevos ejemplos [30]. El propósito de combinar clasificadores es mejorar la precisión que se puede obtener utilizando un único clasificador.

Existen muchas maneras de construir un conjunto de clasificadores, pero las más utilizadas son *Bagging* [12], *Boosting* [116] y *Stacking* [145]. *Bagging* construye un grupo de clasificadores por submuestreo de los ejemplos de entrenamien-

to para generar diferentes hipótesis. Después de que las diferentes hipótesis son generadas, éstas son combinadas mediante votos. *Boosting* también utiliza un sistema de votos para combinar los clasificadores pero en lugar de submuestrear los ejemplos de entrenamiento, genera las hipótesis dando un peso a cada instancia de entrenamiento y ajustando este peso de acuerdo a su importancia. De esta manera, en cada repetición se genera un clasificador enfocado en las instancias que han sido tratadas incorrectamente por el clasificador previo. Ambos, *Bagging* y *Boosting* generan clasificadores homogéneos, es decir, clasificadores generados a partir del mismo algoritmo de aprendizaje. Por otra parte, *Stacking* genera clasificadores a partir de diferentes algoritmos de aprendizaje siguiendo un proceso similar a la validación cruzada y utiliza un clasificador en un nivel superior (alto nivel) para combinar los clasificadores generados (bajo nivel). El conjunto de clasificadores formado por clasificadores heterogéneos en dos niveles, es el que lleva a cabo la clasificación de un nuevo ejemplo.

Se sabe que no existe un algoritmo de aprendizaje que sea el mejor en todas las tareas de aprendizaje. El enfoque utilizado por *Stacking* está basado en la teoría de que diferentes algoritmos de aprendizaje aplican diferentes heurísticas llevando a cabo búsquedas en distintas áreas del espacio de hipótesis para obtener resultados diversos.

Uno de los problemas de *Stacking*, es determinar la configuración de los parámetros de aprendizaje que utiliza éste, como por ejemplo, qué algoritmo puede ser utilizado para generar el clasificador de alto nivel y cuáles utilizar para los de bajo nivel. En el enfoque presentado en esta tesis doctoral se plantea esta tarea como un problema de optimización para la cual se propone la aplicación de algoritmos genéticos con la finalidad de obtener la configuración ideal de los parámetros de aprendizaje de *Stacking*.

1.2. Modelado de Agentes mediante Aprendizaje Automático

El desarrollo de software basado en agentes sufre problemas análogos a otras metodologías de desarrollo de software o paradigmas. En concreto, esta tesis doctoral se centra en el proceso de adquisición de conocimiento; esto es, dónde y cómo extraer conocimiento para construir un sistema. En el caso de la tecnología basada en agentes, existen muchos tipos de conocimientos que podrían ser adquiridos con el propósito de construir tales sistemas, como el modelo interno, modelo de otros agentes, estrategias de comunicación, o heurísticas de razonamiento. Una manera de resolver este problema es adquirir dicho conocimiento manualmente a partir de expertos u otras fuentes de conocimiento. Por otra parte, otras aproximaciones consisten en el aprendizaje automático de estos modelos mediante el análisis de casos pasados [85], experimentación con el entorno [15, 123], observación del compor-

tamiento de otros agentes [138], o compartiendo el conocimiento adquirido entre los agentes [51, 129].

En esta tesis doctoral se propone un esquema para la adquisición de uno de los tipos de conocimiento que se ha mencionado con anterioridad: la descripción y codificación del modelo de otros agentes. Según Kitano et al. [81] el modelado de agentes se puede definir como el modelado y razonamiento acerca de las metas, planes, conocimientos, capacidades o emociones de otro agente.

El enfoque propuesto se basa en el aprendizaje de estos modelos a partir de la observación del comportamiento de otros agentes sin conocimiento de su estructura interna. Como primera aproximación, se han seleccionado tareas de razonamiento de un-paso. En estas tareas, el proceso de razonamiento puede ser complicado, pero la solución (salida) dada para unos valores dados correspondientes a un grupo de atributos (entradas) es una única alternativa de un conjunto predeterminado de alternativas. Por ejemplo, tareas de clasificación, tareas de predicción, juegos de dos oponentes de suma-cero con conocimiento perfecto (e.g. ajedrez o damas) o tareas de diagnóstico.

1.3. Objetivos de la Tesis

Tal y como se planteó en la sección anterior, esta tesis doctoral abarca dos áreas de evidente interés dentro del aprendizaje automático. Por esta razón se detallan los objetivos relacionados con cada uno de los temas por separado.

Stacking es uno de los algoritmos de generación de conjuntos de clasificadores heterogéneos más utilizados. Este algoritmo combina clasificadores generados a partir de distintos algoritmos de aprendizaje con la finalidad de aprovechar las heurísticas de cada algoritmo al buscar en el espacio de hipótesis. Por otra parte, los algoritmos genéticos han demostrado su utilidad como técnica de optimización en una amplia gama de dominios.

Uno de los objetivos generales que persigue esta tesis es diseñar un método capaz de determinar la configuración ideal de los parámetros de aprendizaje involucrados en la utilización de *Stacking* mediante algoritmos genéticos. Para lograr este objetivo general, se citan a continuación los objetivos específicos que se deben cumplir:

- Desarrollo de una codificación del problema para la aplicación de los algoritmos genéticos.
- Determinación de una función de evaluación de las soluciones generadas por los algoritmos genéticos.
- Evaluación de la utilización de algoritmos genéticos en la configuración de los parámetros de *Stacking*.

La capacidad de poder crear un modelo del agente o agentes con los que se está interactuando proporciona información que resulta de mucha importancia a la hora de tomar una decisión sobre que acción llevar a cabo. Por otra parte, las técnicas de aprendizaje automático son ampliamente utilizadas en entornos en donde intervienen agentes.

El objetivo general, en cuanto al modelado de agentes se refiere, es el desarrollo de un marco que permita a un agente tomar decisiones en presencia de otros agentes, utilizando para ello, entre otras cosas, el modelo del comportamiento de otro agente. La adquisición de este modelo está basada en la observación de las acciones que realiza el agente que es objeto del modelado. Para ello se propone la utilización de técnicas de aprendizaje automático, tanto en el proceso realizado para adquirir el modelo, como en la fase de utilización del mismo.

Con el propósito de alcanzar el objetivo general propuesto, se plantean una serie de objetivos específicos que se detallan a continuación:

- Desarrollo de un marco conceptual que permita llevar a cabo la tarea del modelado de un agente y la utilización de este modelo en el proceso de toma de decisiones.
- Desarrollo de un método de adquisición de datos para el modelado de agentes en entornos dinámicos. Estos datos son necesarios para construir el modelo propiamente dicho. Los subobjetivos de este objetivo son:
  - Definición de técnicas para la adquisición de datos relacionados con las acciones llevadas a cabo por los agentes a modelar.
  - Definición de técnicas de preprocesado de los datos.
  - Definición de técnicas de almacenamiento de los datos.
- Desarrollo de un marco basado en aprendizaje automático para crear un modelo de otros agentes basado en la información obtenida por el método de adquisición de datos. Los subobjetivos son:
  - Definición de las distintas técnicas de aprendizaje automático y parámetros que se utilizarán para la construcción del modelo de otros agentes.
  - Definición de técnicas de representación y almacenamiento de los modelos adquiridos de tal forma que puedan ser utilizados con posterioridad.
- Definición de un método de razonamiento que utilice los modelos de los agentes adquiridos para tomar una decisión sobre qué acción realizar. Para ello, se deberán definir técnicas capaces de utilizar la salida dada por los modelos de los agentes. Sin estas técnicas la decisión habría que tomarla sin tener en cuenta la predicción sobre lo que va a hacer el otro agente.

Parte II

GA-Stacking

Capítulo 2

Introducción

En una reunión de la junta de una gran empresa directiva se debe tomar una de las decisiones más importantes de los últimos años para la empresa: se está debatiendo la posible ampliación de capital de acuerdo a los últimos datos de mercado y al comportamiento de la economía global. Es necesario que esta decisión sea meditada lo suficiente porque una decisión errónea conllevaría importantes pérdidas. Dada la importancia de esta medida, la decisión no puede recaer sólo en el presidente de la empresa, sino que es importante que todos los miembros de la junta directiva, basándose en su experiencia y área de trabajo, den su opinión al respecto, con el propósito de tomar en *conjunto* la decisión adecuada. Este ejemplo hila con la necesidad que surge dentro del aprendizaje automático de sistemas que sean capaces de tomar decisiones *en conjunto* valorando la aportación de distintos sistemas de aprendizaje, incluso decidiendo cuál de ellos resulta más adecuado para cada caso. Así una de las áreas que despierta mayor interés dentro del aprendizaje automático es la combinación de clasificadores con la finalidad de incrementar la precisión en la clasificación [30]. Este enfoque es conocido como *conjuntos de clasificadores*. La idea principal detrás de los conjuntos, es que a menudo, estos son mucho más precisos que cualquiera de los clasificadores que forman parte de éste.

La mayoría de las investigaciones dentro del área de conjunto de clasificadores se centra, generalmente, en la generación de clasificadores a partir de la aplicación de un único algoritmo de aprendizaje [31]. Estos clasificadores son denominados clasificadores homogéneos. Existen varios métodos utilizados en la generación de los clasificadores homogéneos que forman el conjunto. Estos métodos se pueden agrupar en aquéllos que llevan a cabo un submuestreo de los ejemplos de entrenamiento (e.g. Bagging [12] y Boosting [48]), los que manipulan los atributos de entrada para generar distintos datos de entrenamiento [20], los que manipulan la salida esperada (e.g. *ECOC* [33]) y aquéllos que mediante la incorporación de aleatoriedad en el algoritmo de aprendizaje [84] generan los clasificadores del conjunto. Una vez que han sido generados los clasificadores, estos son combinados, en

la mayoría de los casos por un mecanismo de voto mayoritario o mediante votos con peso asignado.

Otras investigaciones dentro del área de conjuntos de clasificadores aplican diferentes algoritmos de aprendizaje sobre el conjunto de datos para generar los clasificadores (heterogéneos) que forman parte del conjunto. Un ejemplo representativo de este tipo de técnicas es la conocida como *Stacked Generalization* o *Stacking* [145]. Una vez que han sido generados los clasificadores del conjunto a partir de algoritmos de aprendizaje distintos, *Stacking* utiliza lo que se conoce como *meta-aprendizaje* para combinar las decisiones de éstos. En otras palabras, *Stacking* plantea una tarea de aprendizaje de nivel superior, en donde se utiliza un algoritmo para aprender de qué manera combinar las salidas de los clasificadores de nivel base. El trabajo realizado en esta tesis doctoral se centra en este método de generación de conjuntos.

Uno de los problemas inherentes a *Stacking* es la obtención de la combinación adecuada de los parámetros del algoritmo (i.e. los clasificadores de nivel-base, tipo de datos del meta-nivel y el meta-clasificador) dado a un conjunto de datos específico. Si el número de clasificadores y algoritmos que se pueden utilizar para generar estos es pequeño, este problema se puede resolver por un método simple en tiempo razonable (i.e. búsqueda exhaustiva). Pero, cuando el espacio de búsqueda es grande, la búsqueda de la configuración óptima de *Stacking* puede resultar difícil. En el enfoque que se presenta en esta tesis doctoral, se plantea este problema como una tarea de optimización. Se propone la utilización de técnicas de optimización basadas en búsqueda heurística para resolverla. Más precisamente, se propone la aplicación de *algoritmos genéticos* [63] para obtener automáticamente la configuración ideal de los parámetros de *Stacking*.

Con la finalidad de dar una visión general del tema que se trata en esta tesis doctoral, en el capítulo 3 se da una panorámica general del estado del arte. En el capítulo 4 se presenta la propuesta de esta tesis doctoral en lo referente a los conjuntos de clasificadores. En el capítulo 5 se muestra la evaluación de la propuesta presentada. Finalmente, en el capítulo 6 se presentan las conclusiones y trabajos futuros.

Capítulo 3

Estado del Arte

El objetivo de este capítulo es brindar una visión general del área en la cual se enmarca esta parte de la tesis doctoral. En primera instancia y a manera de introducción en la sección 3.1 se hace una breve descripción del concepto de *Aprendizaje Automático* y su taxonomía. Posteriormente, en la sección 3.2 se desarrolla el concepto de *conjuntos de clasificadores* tomando en cuenta aspectos como la definición, razones de su eficiencia y las técnicas de construcción. En la sección 3.3 se presenta la técnica de generación de conjuntos heterogéneos conocida como *Stacked Generalization* o *Stacking* la cual sirve de base del enfoque propuesto en esta tesis doctoral. Además, se muestran los últimos avances relacionados con esta técnica. En la sección 3.4 se presentan los algoritmos genéticos, técnica de optimización utilizada en el planteamiento propuesto en esta tesis doctoral. Por último, en la sección 3.5, se presentan algunas conclusiones derivada de los puntos abordados.

3.1. Aprendizaje Automático

En general, el aprendizaje automático trata de la construcción de programas que, utilizando la experiencia sean capaces de mejorar automáticamente su rendimiento. Este campo ha recibido la influencia de otros muchos campos como la estadística, la inteligencia artificial, la biología y la teoría de la información, entre otros.

Mitchell [97] señala que "un programa de ordenador se dice que aprende de la experiencia E con respecto a una cierta clase de tarea T y medida de funcionamiento P, si su funcionamiento en la tarea T según lo medido por P, mejora con la experiencia E."

Debido a la amplitud y variedad de este campo de investigación, en esta tesis doctoral no se pretenden detallar todas las técnicas y enfoques que abarca. En cambio, se da una idea general de las técnicas utilizadas. Dependiendo del tipo de

realimentación, se puede clasificar el tipo de aprendizaje en tres grupos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

3.1.1. Aprendizaje Supervisado

El aprendizaje supervisado consiste en un tipo de aprendizaje automático en donde al algoritmo que se utiliza se le proporcionan una serie de ejemplos con sus correspondientes etiquetas, es decir, que todos los ejemplos han sido clasificados "a priori". De esta forma en el proceso de aprendizaje, el algoritmo compara su salida actual con la etiqueta del ejemplo para luego realizar los cambios que sean necesarios.

En el caso del aprendizaje supervisado cada ejemplo (a menudo llamado instancia) dentro del conjunto de aprendizaje se puede expresar mediante la forma atributo-valor o mediante relaciones. Cuando a un programa de aprendizaje se le pasan un conjunto de ejemplos $\{(x_1, y_m), \ldots, (x_m, y_m)\}$ para descubrir una función desconocida $y = f(x)$, los valores de x_i son vectores de la forma

$$(x_{i,1}\,, x_{i,2}\,, \ldots, x_{i,n}\,)$$

donde $(x_{i,j}\,)$ se refiere a cada característica (atributo) de (x_i) y n es el número total de atributos de la instancia. Los atributos que forman parte de la instancia pueden ser categóricos o nominales y numéricos. Por ejemplo, el atributo Sexo es categórico con sus posible valores (Masculino o Femenino). Por otro lado, los valores de los atributos Tamaño, Peso y Edad pueden ser numéricos y, en consecuencia, pueden ser llamados continuos.

Los valores de y pueden ser también nominales o continuos. Si estos valores pertenecen a un número definido de clases $\{1, \ldots, K\}$ se dice que es una tarea de *clasificación* y si el valor de y es continuo la tarea es una *regresión*.

El conjunto de todos los posibles valores que pueden tomar los atributos de x se conoce como espacio de instancias o espacio de entrada. El conjunto de los posibles valores de y se conoce como espacio de salida.

Generalmente, cuando se lleva a cabo una tarea de clasificación o de regresión, se utiliza un conjunto de instancias para que el algoritmo de aprendizaje construya un *clasificador*. Este conjunto de ejemplos es llamado conjunto de *entrenamiento* o *aprendizaje*. Un clasificador es una hipótesis sobre la función real f. Para validar este clasificador, generalmente, se utiliza un conjunto de instancias que no se ha utilizado para construir el clasificador. Este conjunto recibe el nombre de conjunto de *prueba* o *test*.

Ejemplos de una tarea de clasificación pueden ser: predecir si un paciente puede tener cáncer o no, predecir el tiempo, secuencias en la cadena de ADN, etc.

A la hora de evaluar la precisión de un clasificador se utiliza el conjunto de test sobre el cual se obtiene una *precisión de clasificación* que es calculada basándose

en los ejemplos del conjunto de test que el clasificador ha clasificado correctamente. Al medir la precisión de un clasificador se puede utilizar también la *tasa de error* que es el complemento de la precisión de clasificación.

La tarea de clasificación se puede definir como tomar como entrada un ejemplo nuevo, con clase desconocida, y utilizar un clasificador o método para obtener la clase de dicho ejemplo.

Dentro del aprendizaje supervisado, de acuerdo al tipo de representación de los datos de entrada, se puede hacer una clasificación en dos grupos: los representados en la forma atributo-valor y los que están representados en forma de relaciones (utilizan lógica de primer orden).

Dentro del grupo de algoritmos que utilizan la representación de atributo-valor existen a su vez dos grupos: algoritmos simbólicos y subsimbólicos. Entre los algoritmos simbólicos se pueden destacar los árboles de decisión (e.g. ID3 [105], C4.5 [110]), y los sistemas basados en reglas (*AQ* [94] y PART [47]). Ejemplos de aprendizaje subsimbólico son las *redes de neuronas* [62] y los *algoritmos genéticos* [63] (cuando éstos se utilizan como técnica de clasificación). Además de los ejemplos mencionados anteriormente existen algoritmos del tipo numérico, como los basados en regresión (*M5* [106]) y en probabilidades (*Naive Bayes* [77]).

FOIL [109] y *PROGOL* [98] son ejemplos de sistemas que utilizan la representación relacional.

3.1.2. Aprendizaje No Supervisado

A diferencia del aprendizaje supervisado en este tipo de aprendizaje no se conoce “a priori” el atributo dependiente. Ejemplos de este tipo de aprendizaje son los algoritmos de agrupamiento o *clustering* entre los que se pueden señalar *COBWEB* [45], *EM* [28], *K-Medias* [91] y los *Sistemas Clasificadores* [89].

3.1.3. Aprendizaje por Refuerzo

En el aprendizaje por refuerzo, el algoritmo utilizado recibe las entradas y una evaluación (en forma de recompensa, que puede venir retardada en el tiempo) de tal manera que el algoritmo debe aprender qué acción es la que brinda mayor rendimiento a largo plazo. Entre los algoritmos de aprendizaje por refuerzo se pueden mencionar *Q-learning* [140, 141] y *ARTDP* (*Adaptative Real Time Dynamic Programming*) [3].

3.1.4. Taxonomía Basada en Otros Criterios

Otro tipo de clasificación de las técnicas de aprendizaje automático es la que clasifica dicho aprendizaje como aprendizaje inductivo y aprendizaje deductivo.

La idea principal del aprendizaje inductivo es que a partir de un número elevado de ejemplos, asumiendo que existe un concepto o conceptos en los que se encuadran dichos ejemplos, se puede construir una representación de dichos conceptos. Una vez construida la representación, ésta puede ser utilizada para realizar predicciones sobre nuevas instancias, utilizando esencialmente el conocimiento obtenido a partir de los ejemplos disponibles. Algoritmos que se clasifican como inductivos son los que se han mencionado hasta este punto dentro de la clasificación de aprendizaje automático.

Por el contrario, el aprendizaje deductivo utiliza, principalmente, conocimiento del dominio y algún ejemplo, de tal forma que su objetivo principal es hacer operativo el conocimiento que posee el sistema y no generar conocimiento nuevo. El tipo de aprendizaje deductivo más estudiado es el *Aprendizaje Basado en la Explicación* (*EBL*) [95]

Existen otro grupo de algoritmos capaces de combinar el aprendizaje inductivo con el deductivo como son *KBANN* [134] y *HAMLET* [8].

3.2. Conjuntos de Clasificadores

En este punto se presentan los sistemas en los que se enmarca parte de la presente tesis doctoral. Cabe señalar que *Stacking* se apartado específico.

Según Dietterich [30], un conjunto de clasificadores es un grupo de clasificadores cuyas decisiones individuales se combinan de alguna manera (generalmente mediante votos) con la finalidad de clasificar nuevos ejemplos. Son diversos y numerosos los estudios realizados en el área, que demuestran que, habitualmente, los conjuntos de clasificadores mejoran la precisión de cualquiera de los clasificadores individuales que forme parte de éste [4, 12, 32, 49, 108].

Para que un conjunto de clasificadores mejore la precisión de cualquiera de los miembros que lo componen, es una condición necesaria y suficiente que los clasificadores sean a su vez precisos y diversos [61]. Se considera que un clasificador es preciso si el error que comete es menor que el que se podría obtener eligiendo aleatoriamente una clase de entre las clases disponibles. Por otra parte, se considera que dos clasificadores son diversos, si los errores que cometen sobre los datos de entrada no están correlados, es decir, que no cometen los mismos errores.

Con el propósito de apreciar la necesidad de que los clasificadores que formen parte del conjunto sean precisos y diversos, considérese un ejemplo en donde se tienen tres clasificadores: h_1, h_2, h_3 y una nueva instancia a clasificar x. En una

situación en la que los tres clasificadores no sean diversos, es decir, sean idénticos, se da el caso de que si $h_1(x)$ es erróneo, $h_2(x)$ y $h_3(x)$ también estarán errados. Por otra parte, si los errores que cometen los clasificadores no están correlados, en el caso que $h_1(x)$ esté errado, $h_2(x)$ y $h_3(x)$ podrían ser correctos, en cuyo caso utilizando el voto mayoritario, la instancia x sería clasificada correctamente. Precisando más, si la tasa de error de L hipótesis h_l son todas iguales a $p < 1/2$ y los errores que comenten son independientes, la probabilidad de que utilizando el voto mayoritario para combinar las decisiones de los clasificadores esté errada, viene dada por el área bajo la curva de una distribución binomial en donde más de $L/2$ hipótesis estén erradas. En la Figura 3.1 se muestra un hipotético conjunto formado por 21 hipótesis, cada una de ellas con una tasa de error del 0.3. El área bajo la curva en donde 11 o más hipótesis estén simultáneamente erradas es 0.026, que es mucho menor que la tasa de error individual de las hipótesis [30].

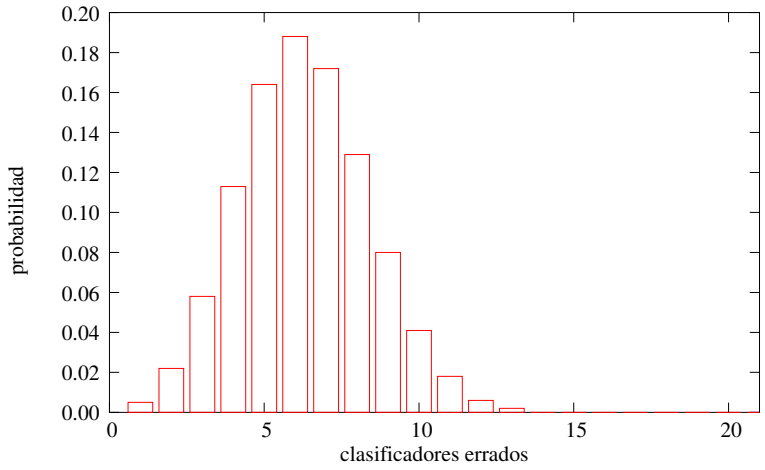


Figura 3.1: Probabilidad de que exactamente l (de 21) hipótesis cometan un error, asumiendo que cada hipótesis tiene una tasa de error de $0{,}3$ y comenten sus errores independientemente de las demás hipótesis [30].

3.2.1. ¿Por qué Funcionan los Conjuntos de Clasificadores?

Existen, según Dietterich [31], tres razones fundamentales por las cuales se pueden encontrar buenos conjuntos de clasificadores. Estas razones son:

- *Estadística*: un algoritmo de aprendizaje se puede ver como una búsqueda en un espacio de hipótesis, $\mathcal{H}$, con la finalidad de identificar la mejor hipótesis en dicho espacio. Si el conjunto de datos que se posee es demasiado pequeño en comparación con el espacio de hipótesis, surge el problema estadístico. Si los datos disponibles no son suficientes, el algoritmo de aprendizaje puede

encontrar una gran cantidad hipótesis dentro de $\mathcal{H}$ con igual precisión sobre los datos disponibles. Si estos clasificadores se combinan, disminuye el riesgo de seleccionar un clasificador que devuelva una hipótesis errónea. En la figura 3.2 (parte superior izquierda) se representa gráficamente esta situación. El espacio de hipótesis está representado por la curva exterior, la curva interior representa las hipótesis con una precisión buena, y f representa la hipótesis real. Como se puede apreciar, si se promedian las hipótesis dentro de la curva interior, se puede obtener una buena aproximación de f.

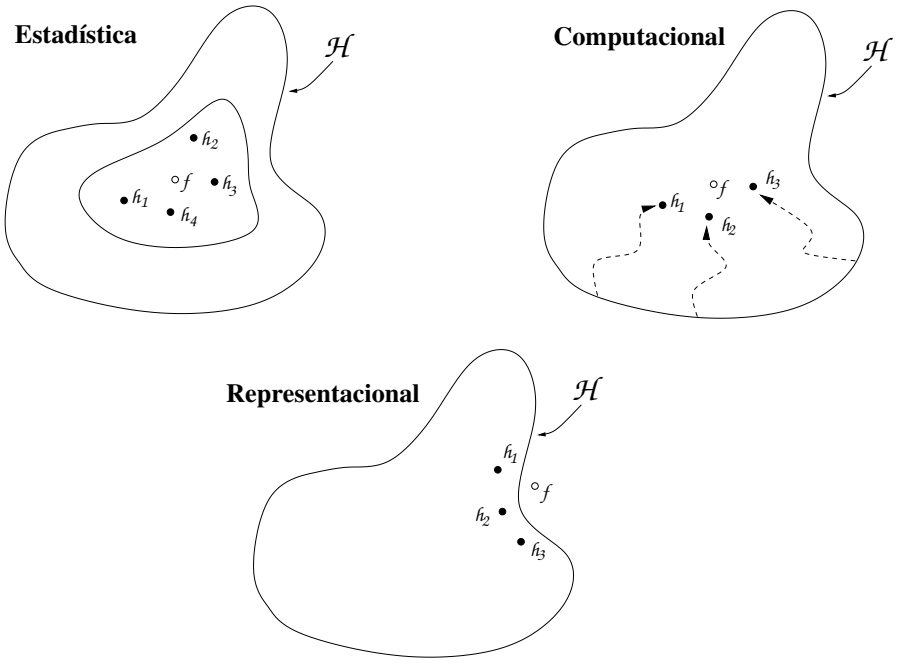


Figura 3.2: Razones fundamentales por las que un conjunto de clasificadores puede funcionar [31].

- *Computacional*: en los casos en los que los datos de entrenamiento son suficientes, y por ende no existe el problema estadístico, se puede presentar otro problema de tipo computacional. Este problema se da debido a que muchos algoritmos de aprendizaje funcionan llevando a cabo búsquedas locales que pueden quedar atrapadas en máximos locales. Por esta razón puede ser computacionalmente difícil que el algoritmo encuentre la mejor hipótesis para un conjunto de datos dado. Por ejemplo, el entrenamiento óptimo para las redes de neuronas y los árboles de decisión es un problema *NP-completo* [7, 71]. Si se llevan a cabo una serie de búsquedas locales con puntos de partida diferentes para obtener hipótesis que luego se combinan, se puede obtener una mejor aproximación a la hipótesis real, en vez de utilizar cualquiera de las hipótesis generadas (Figura 3.2 parte superior derecha).

- *Representacional*: en la mayoría de las aplicaciones de aprendizaje automático la función real, f, puede no ser representada por ninguna de las hipótesis en $\mathcal{H}$. Al combinar varias hipótesis, es posible que se aumente el espacio de posibles funciones representables y con ellas las hipótesis que se pueden representar y, de esta manera se podría aproximar mejor la función real (f) (Figura 3.2 parte inferior).

3.2.2. Construcción de Conjuntos de Clasificadores

En la actualidad existen muchos métodos para generar conjuntos de clasificadores. Tomando en consideración aquellos métodos que se pueden aplicar a una gran cantidad de algoritmos, Dietterich [31] los clasifica en:

- *Voto bayesiano: enumeración de hipótesis.* Basándose en el teorema de Bayes se consideran todas las hipótesis en $\mathcal{H}$ como parte del conjunto, cada una de éstas con un peso asignado equivalente a su probabilidad posterior. Este método es aplicable en tareas de aprendizaje en donde se pueden enumerar todas las hipótesis h_l y calcular su probabilidad "a posteriori". El voto bayesiano trata sobre todo el componente estadístico de conjuntos.

- *Manipulación de los ejemplos de entrenamiento*. Una manera de generar conjuntos de clasificadores es a partir de la manipulación de los ejemplos de entrenamiento con la finalidad de obtener diferentes hipótesis. El algoritmo de aprendizaje, cualquiera que sea, se ejecuta repetidamente utilizando un conjunto distinto de instancias de entrenamiento cada vez, generando así los clasificadores que forman parte del conjunto. Esta técnica funciona mejor con algoritmos de aprendizaje *inestables*, es decir aquellos cuyo modelo resultante puede variar mucho al cambiar en menor grado los ejemplos de entrenamiento. Por ejemplo, los árboles de decisión, las redes de neuronas artificiales y los algoritmos de inducción de reglas, son algoritmos inestables. En cambio, los métodos de regresión lineal y el vecino más cercano, suelen ser muy estables.

  Dentro de los métodos que manipulan los ejemplos de entrenamiento para generar los clasificadores, el más sencillo es conocido como *Bagging* (derivado de *bootstrap aggregation*) [12]. Esta técnica se basa en el submuestreo con reemplazo del conjunto de entrenamiento para generar un grupo diferente de hipótesis, utilizando cada muestra obtenida como conjunto de entrenamiento. Cada una de las muestras (*bootstrap replicates*) contiene el mismo número de instancias que el conjunto original. Además poseen, en promedio, un 63.2 % de las instancias del conjunto original de las cuales existen instancias que se repiten múltiples veces. *Bagging* combina las decisiones de los clasificadores en una sola predicción por medio del mecanismo de voto

mayoritario (la clase que obtiene más votos por parte de los clasificadores, es la clase ganadora). En el Anexo A se detalla el algoritmo *Bagging*.

Otro método que se basa en la manipulación de los ejemplos de entrenamiento es conocido como *Boosting* [116]. Existen diferentes versiones de este método, siendo el más representativo el algoritmo *AdaBoost* (*Ada*ptative *Boost*ing) [48, 49]. *AdaBoost* genera los clasificadores de manera secuencial, dándole más importancia a los ejemplos que fueron clasificados de manera errónea por el clasificador anterior mediante la asignación de un peso a cada instancia del conjunto de entrenamiento. Este peso es actualizado en cada interacción. Una vez generados los clasificadores, las decisiones de éstos se combinan mediante un mecanismo de voto con peso. El peso correspondiente a cada clasificador varía de acuerdo a la precisión de éste sobre el conjunto de entrenamiento (con instancias con peso) utilizado para construirlo. En el Anexo A se muestra el algoritmo *AdaBoost.M1*.

Existen otros métodos que manipulan los ejemplos de entrenamiento para generar los clasificadores. Entre estos métodos se pueden señalar los *cross-validated committees* [102] que siguen un proceso similar a la validación cruzada para generar los clasificadores.

- *Manipulación de los atributos de entrada*. Una tercera técnica general para generar conjuntos, según Dietterich, es mediante la manipulación de los atributos de entrada disponibles a la hora de utilizar el algoritmo de aprendizaje. Un ejemplo de este tipo de técnica es el aplicado por Cherkauer [20] en donde lleva a cabo diferentes agrupaciones de los atributos de entrada para generar los clasificadores que forman parte del conjunto. Tumer y Ghosh [135] aplican una técnica similar a una base de datos de sonar. Una debilidad de esta técnica es que sólo funciona cuando los atributos de entrada son altamente redundantes.

- *Manipulación de las salidas*. Otra técnica para generar conjuntos de clasificadores es la manipulación de la salida esperada, es decir, la clase de la instancia (y). Un método representativo de estas técnicas es conocido como *ECOC* (*error correcting output code*) [33]. *ECOC* asume que el número de clases, K, es grande. De esta forma se crean nuevas tareas de aprendizaje dividiendo aleatoriamente las K clases en dos subconjuntos A_ℓ y B_ℓ. Los datos de entrada son entonces re-etiquetados de forma tal que todas la instancias en el conjunto A_ℓ de cualquiera de las clases originales son re-etiquetadas con 0 y todas las instancias de cualquier clase en B_ℓ son re-etiquetadas con 1. Con estos datos re-etiquetados se entrena el algoritmo de aprendizaje, generando así un clasificador h_ℓ. Al repetir este proceso L veces se obtiene un conjunto formado por L clasificadores ($h_1, ..., h_\ell$).

  Una vez creado el conjunto, ECOC clasifica una nueva instancia x aplicando cada clasificador h_ℓ a ésta. Si $h_\ell(x) = 0$, entonces cada clase en A_ℓ recibe un voto, si $h_\ell(x) = 1$, cada clase en B_ℓ recibe un voto. Una vez que los L

clasificadores han votado, la clase con el mayor número de votos será seleccionada como la predicción del conjunto.

- *Introducción de aleatoriedad*. La incorporación de aleatoriedad dentro del algoritmo de aprendizaje es otra técnica utilizada para la generación de conjuntos. Por ejemplo, Kolen & Pollack [84] muestran que entrenando una red de neuronas con el mismo conjunto de entrenamiento, pero con diferentes pesos iniciales aleatoriamente seleccionados, se pueden obtener clasificadores bastante distintos.

  El algoritmo de generación de árboles de decisión, C4.5, ha sido utilizado para generar conjuntos introduciendo aleatoriedad a la hora de evaluar la ganancia de los atributos [32, 87]. Ali y Pazzani [2] incorporan aleatoriedad en el algoritmo de generación de reglas estilo *Prolog*, *FOIL*, para generar conjuntos.

La clasificación propuesta por Dietterich [31] sólo considera los conjuntos de clasificadores que se forman a partir de un único algoritmo de aprendizaje, es decir que los clasificadores generados son homogéneos. Existe otra manera de generar conjuntos de clasificadores mediante la aplicación de distintos algoritmos de aprendizaje al momento de generar los miembros del conjunto. De esta forma, el conjunto que se genera está formado por clasificadores heterogéneos, aprovechando así los distintos *bias* de los algoritmos utilizados. Entre estos métodos de generación de conjuntos heterogéneos destaca el conocido como *Stacking* [145]. Además de utilizar distintos algoritmos de aprendizaje para generar los clasificadores que forman parte del conjunto, *Stacking* utiliza lo que se conoce como meta-aprendizaje en la etapa de combinación de las predicciones de los clasificadores generados. En otras palabras *Stacking* utiliza un algoritmo de aprendizaje para generar un clasificador que combine las predicciones de los demás clasificadores. En la sección 3.3 se detallarán más aspectos de este método.

3.3. *Stacked Generalization*

3.3.1. Definición

Stacking es quizás, junto con *Bagging* y *Boosting* la técnica de generación de conjuntos de clasificadores más utilizada. *Stacking* es la abreviación de *Stacked Generalization* [145]. A diferencia de otras técnicas de generación de conjuntos como *Bagging* y *Boosting*, *Stacking* utiliza diferentes algoritmos de aprendizaje para generar el conjunto de clasificadores. La idea principal que subyace tras *Stacking*, es la combinación de clasificadores generados a partir de diferentes algoritmos de aprendizaje como son: los árboles de decisión, los algoritmos basados en instancias, las redes de neuronas, etc. Puesto que cada uno de estos algoritmos utiliza

maneras diferentes de representar el conocimiento y diferentes *bias* o sesgos de aprendizaje, el espacio de hipótesis será explorado de manera distinta, generando así clasificadores diferentes. De este modo, se espera que los clasificadores generados no estén correlados.

Una vez que los clasificadores han sido generados, éstos han de ser combinados. *Stacking* a diferencia de *Bagging* y *Boosting*, no utiliza un mecanismo de votos porque, por ejemplo, si la mayoría de los clasificadores llevan a cabo malas predicciones, esto conduciría a una clasificación final errónea. Para tratar este problema, *Stacking* utiliza el concepto de *meta-clasificador*. El meta-clasificador (o modelo de nivel-1) generado utilizando un algoritmo de aprendizaje, intenta modelizar el cómo se deben combinar las decisiones de los clasificadores base (o modelos de nivel-0). En la Figura 3.3 se muestra el funcionamiento general de *Stacking*.

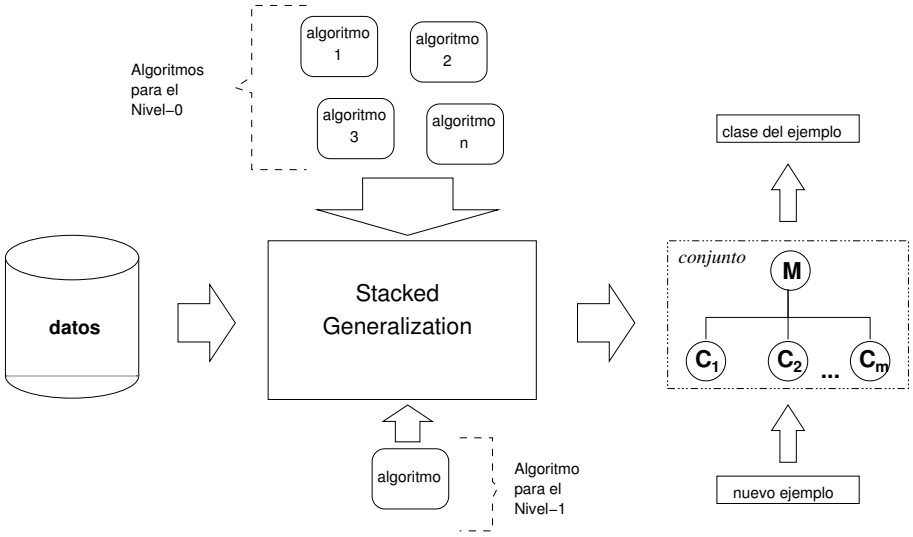


Figura 3.3: Funcionamiento general de *Stacking*.

Formalmente, dado un conjunto de datos S, *Stacking* genera, en primer lugar, un subgrupo de conjuntos de entrenamiento $S_1, ...S_T$ para luego seguir un proceso similar al de la validación cruzada: se deja uno de los subconjuntos fuera (e.g. S_j) para utilizarlo posteriormente. El resto de las instancias $S^{(-j)} = S - S_j$ son utilizadas para generar los clasificadores de nivel-0 mediante la aplicación de K algoritmos de aprendizaje distintos, $k = 1, ..., K$, para obtener K clasificadores. Después de que los modelos de nivel-0 han sido generados, el conjunto S_j es utilizado para entrenar el meta-clasificador (clasificador de nivel-1). Los datos de entrenamiento de nivel-1 se forman a partir de las predicciones de los modelos de nivel-0 sobre las instancias en S_j, las cuales han sido reservadas para este propósito (Figura 3.4 *a*). Los datos de nivel-1 tienen K atributos cuyos valores son las predicciones de cada uno de los K clasificadores de nivel-0 para cada instancia en S_j. De este mo-

do, una instancia de entrenamiento de nivel-1 está constituida por K atributos (las K predicciones) y la clase objetivo, la cual es la clase real para cada instancia en particular en S_j. Una vez que los datos de nivel-1 han sido construidos a partir de todas la instancias en S_j, cualquier algoritmo de aprendizaje puede ser utilizado para generar el modelo de nivel-1 (Figura 3.4 *b*). Para completar el proceso, los modelos de nivel-0 son regenerados a partir del conjunto S completo (de esta manera se espera que los clasificadores sean ligeramente más precisos) (Figura 3.4 *c*). En la Figura 3.4 *d* se muestra la estructura final del conjunto de clasificadores generados mediante *Stacking*. Para clasificar una nueva instancia, los modelos de nivel-0 producen un vector de predicciones que es la entrada al modelo de nivel-1, el cual genera la predicción final del conjunto (Figura 3.5).

3.3.2. Trabajos Relacionados

Dentro del grupo de técnicas capaces de construir conjuntos utilizando el *Meta-Aprendizaje* existen las que se centran en la predicción del algoritmo adecuado para un problema específico, basándose en las cualidades del conjunto de datos [11] o basados en el rendimiento de otro algoritmos de aprendizaje más simples [103]. Sin embargo el propósito de esta sección es dar una visión general de los trabajos realizados basándose en *Stacking* u otros algoritmos similares relacionados con éste.

Los trabajos basados en *Stacking*, o que mantienen similitudes con éste, se pueden agrupar en dos grupos: aquéllos que abordan la selección de parámetros de *Stacking* y aquéllos que de alguna manera son o se pueden considerar variantes de *Stacking*. A continuación se considerarán los trabajos relevantes de estos dos grupos.

Selección de Parámetros y Enfoques Relacionados

Como señalara inicialmente Wolpert [145], algunos aspectos en *Stacking*, como la selección de los clasificadores base, el tipo de meta-datos y el clasificador que se debe utilizar en el nivel-1 son considerados como *magia negra* (del inglés: *black art*). Algunos trabajos que tratan sobre la configuración de estos parámetros y otros temas relacionados se detallan a continuación.

Skalak [120] presenta una perspectiva general sobre los métodos de construcción de conjuntos. Otra contribución significativa del trabajo de Skalak es el estudio de los conjuntos de clasificadores basados en *Stacking*, en donde se considera la utilización de clasificadores basados-en-instancias (almacenando solo unos pocos prototipos por clase) como clasificadores de nivel-base (nivel-0) y como meta-clasificador (nivel-1) un árbol de decisión.

Fan et al. [42] proponen determinar la precisión total del conjunto generado

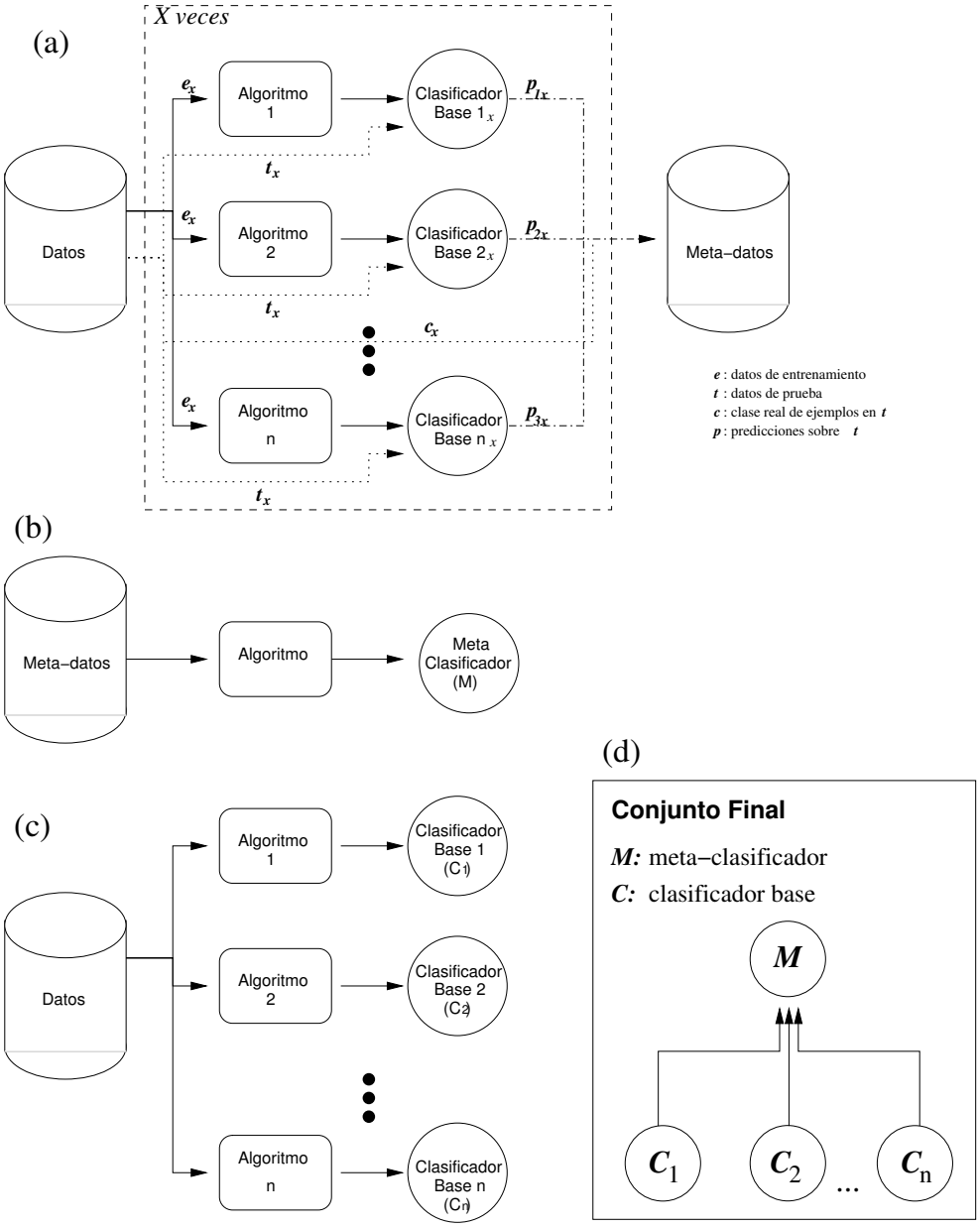


Figura 3.4: Proceso de generación del conjunto mediante *Stacking*.

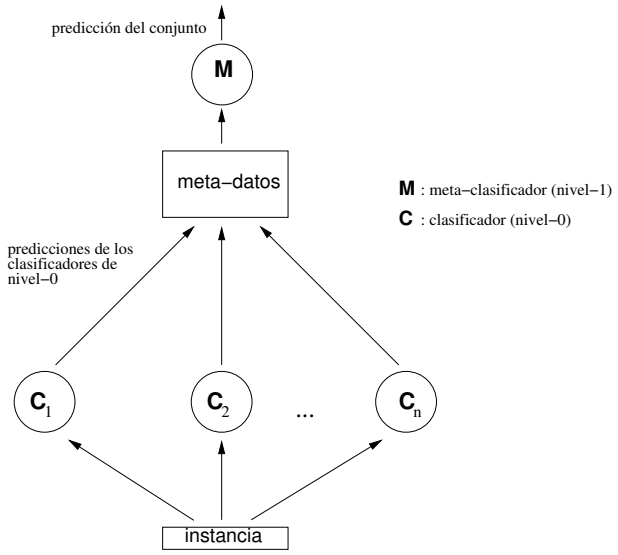


Figura 3.5: Proceso de clasificación de una nueva instancia en un conjunto generado mediante *Stacking*.

mediante *Stacking* utilizando una *estimación de la precisión basada-en-conflicto*. Los clasificadores de nivel-base utilizados son dos basados en árboles y uno basado en reglas; en cuanto al meta-clasificador utilizan un árbol de decisión sin podar. Esta configuración es evaluada utilizado cuatro conjuntos de datos (dos de ellos artificiales). A pesar de que los autores afirman que la medida que proponen es mejor que cualquiera de las otras medidas propuestas previamente, no queda claro en sus resultados que se pueda generalizar esta estimación a un número mayor de conjuntos de datos o en la aplicación de otros meta-clasificadores.

Ting y Witten [132], abordan dos problemas de configuración de *Stacking*: el tipo de clasificador que se debe utilizar en el nivel-1 y el tipo de datos del meta-nivel. Proponen que las salidas de los clasificadores de nivel-0 no sean la predicción de la clase dada por el clasificador, sino una distribución de probabilidad de clase. De esta manera, los atributos del meta-nivel están formados por la concatenación de las distribuciones de probabilidad de todos los clasificadores de nivel-0 seguidas del valor real de la clase. Los autores argumentan que utilizando distribuciones de probabilidad como meta-datos, son utilizados, tanto la predicción, como la confianza de los clasificadores de nivel-base. En cuanto al tipo de meta-clasificador que se debe utilizar, los autores concluyen que una técnica de *regresión lineal de multi-respuesta* (*MLR* por sus siglas en inglés *multi-response linear regression*) resulta la más adecuada como algoritmo de meta-nivel, al menos cuando se utiliza distribuciones de probabilidad. Por otra parte, Ting y Witten investigan la necesidad de utilizar restricciones *no-negativas* en los pesos de los atributos dentro de los modelos lineales, ya que tanto Breiman [13] como LeBlanc y Tibshirani [90]

informan de la necesidad de utilizar las restricciones no-negativas al utilizar *Stacking* en una tarea de regresión. Ting y Witten [132] concluyen que las restricciones no-negativas no son necesarias en *Stacking* cuando se está llevando a cabo una tarea de clasificación y, en el momento de buscar una mejora en la precisión del conjunto. Sin embargo, éstas resultan útiles si se desea mejorar la interpretabilidad del modelo de nivel-1.

Basado en el trabajo de Ting y Witten [132], Seewald [117] propone que *MLR* utilice un grupo distinto de atributos en el meta-nivel, con el propósito de superar una debilidad de *Stacking con MLR* en dominios de más de dos clases, debilidad que no estaba presente en la versión original de *Stacking*. Seewald argumenta que la dimensionalidad de los meta-datos puede ser la causa probable de esta debilidad. Este método, conocido como *StackingC*, propone utilizar sólo la probabilidad de la clase con la más alta probabilidad dentro de todas la posibles clases, reduciendo así la dimensionalidad de los atributos del meta-nivel en un factor correspondiente al número de clases. Los resultados de esta investigación muestran una mejora sobre *Stacking con MLR* con el conjunto completo de distribuciones de probabilidad. Adicionalmente, el autor argumenta que la mejora observada no solo se debe a la reducción de la dimensionalidad de los atributos del meta-nivel, sino también a la alta diversidad de los modelos lineales de clase generados en el meta-nivel. Esta diversidad es notable al comparar *StackingC* con *Stacking con MLR*, en donde todos los modelos lineales están basados, exactamente, en los mismos atributos del meta-nivel.

Recientemente Džeroski y Ženko [41] proponen dos nuevas versiones de *Stacking*. La primera de estas versiones aborda el problema del tipo de datos del meta-nivel. En su trabajo proponen extender el conjunto de atributos del meta-nivel incluyendo la distribución de probabilidad multiplicada por la probabilidad máxima y las entropías de las distribuciones de probabilidad. Por otra parte, proponen otra extensión de *Stacking* en donde utilizan un modelo de inducción de árboles en vez de una regresión lineal como algoritmo del meta-nivel. Este método es llamado *Stacking con modelo de árboles multi-respuesta* (del inglés: *Stacking with multi-response model trees*). Una de las principales conclusiones de su trabajo es que, al comparar *Stacking con MLR* como meta-clasificador es, en el mejor de los casos, competitivo con la selección del mejor clasificador por validación cruzada (*X-val*) y no es significativamente mejor como señalan algunos trabajos. Por otra parte, Džeroski y Ženko llevan a cabo una amplia comparación de enfoques relativos al estado del arte en cuanto a *Stacking* se refiere. En su trabajo, comparan *Stacking con meta árboles-de-decisión* [133], *Stacking con MLR* [132], *StackingC* [117], *Stacking con un número ampliado de atributos en el meta-nivel* y *Stacking con modelo de árboles multi-respuesta*. Además utilizan como base otros enfoques como *X-val* (selección del mejor clasificador mediante validación cruzada) y un sistema simple de selección por votos. Derivado de esta comparación, concluyen que *Stacking con modelo de árboles multi-respuesta* posee un rendimiento mejor que cualquiera de los enfoques de *Stacking* existentes, incluyendo *StackingC* y la se-

lección del mejor clasificador del conjunto por validación cruzada (*X-val*).

Variantes de *Stacking* y Esquemas Alternativos

Existe una serie de trabajos relacionados con *Stacking* que se pueden considerar implementaciones basadas en este enfoque o bien mantienen muchas similitudes con éste.

Chan y Stolfo [18] formulan un algoritmo muy similar a *Stacking* al cual llaman *combinador*. Además proponen una variante de éste a la que llaman *combinador-de-atributos*, en el cual, los atributos del meta-nivel están formados, no solo por las predicciones de clase sino que mantienen también los atributos originales de la instancia. Como muestran los resultados presentados por Schaffer [115] en su estudio del bi-nivel de *Stacking*, esto puede llevar a un peor rendimiento del conjunto.

Por otra parte Chan y Stolfo [18] proponen un enfoque que utiliza lo que denominan un "árbitro". El árbitro es un clasificador individual independiente del resto de los clasificadores base, que es entrenado sobre un subconjunto del conjunto original de datos. Este subconjunto está formado por las instancias en las que los clasificadores base están en desacuerdo. El propósito de un árbitro es brindar una predicción alternativa y más elaborada cuando los clasificadores base presentan contradicciones. Adicionalmente, Chan y Stolfo proponen lo que llaman "árbol-árbitro" en el cual los árbitros que se especializan en resolver conflictos entre pares de clasificadores son organizados en un árbol de decisión binario. Para llevar a cabo la clasificación de una instancia se parte de los nodos hoja formados por los clasificadores base y se va subiendo por el árbol hasta llegar al nodo raíz que proporciona la clasificación final.

Ting [131] propone la utilización de las predicciones de los clasificadores base para aprender una función que refleje la medida interna de confianza del algoritmo en una estimación de su precisión sobre la salida. Esta función puede ser utilizada para combinar los conocimientos del clasificador.

Una condición necesaria para crear un buen conjunto de clasificadores es que los errores de los clasificadores de nivel-base no estén correlados [61]. Partiendo de este hecho, Merz [93] propone una variante de *Stacking* que utiliza análisis de correspondencia con el propósito de detectar correlaciones entre los clasificadores de nivel-base. Una vez que las dependencias del espacio original de datos del meta-nivel se hayan eliminado, un método del vecino más cercano (algoritmo de meta-nivel) se aplica sobre el espacio de atributos resultante. De acuerdo a sus resultados sobre datos sintéticos, su enfoque, denominado *SCANN*, es equivalente al voto plural si los modelos cometen errores no-correlados.

Gama y Brazdil [50] propone un método muy relacionado con *Stacking* al que denominan *Cascade Generalization*. En este método los clasificadores son aplicados secuencialmente y no existe un meta-clasificador como tal. Cuando cada

clasificador base es aplicado sobre los datos, éste incrementa el conjunto de datos añadiendo su distribución de probabilidad de clase. Este nuevo conjunto de datos es utilizado por el siguiente clasificador, de esta manera, el orden en que se empleen los clasificadores se convierte en un factor a tener en cuenta.

Una variante de *Stacking* que predice qué clasificador es el más indicado dado un ejemplo específico, es descrita por Todorovski y Džeroski [133]. Esta variante utiliza un nuevo método de aprendizaje en el meta-nivel. Este método, conocido como *meta árboles de decisión* (*MDT* por sus siglas en inglés - *meta decision trees*), sustituye las predicciones clase-valor en sus nodos hoja por los clasificadores de nivel-base. Los meta-datos están compuestos de propiedades de las distribuciones de probabilidades que reflejan la confianza de los clasificadores de nivel-base, como la entropía y la probabilidad máxima, en vez de las propias distribuciones. Basados en estas propiedades, se generan pequeños *MDT's*.

Seewald y Fürnkranz [118] proponen el esquema conocido como *Grading*. Este esquema crea un clasificador de meta-nivel por cada clasificador de nivel-0. La tarea de aprendizaje para cada clasificador de nivel-1 es predecir si la predicción del clasificador de nivel-0 es incorrecta. Los meta-datos están compuestos por los atributos de nivel-base y los valores de la clase (i.e. *correcto* o *incorrecto*). Un mecanismo de votos con pesos sobre las predicciones de los clasificadores base produce como resultado la predicción final de la clase. El peso asignado al voto de cada clasificador base es la confianza en que su predicción sea correcta. Este peso es estimado por el meta-clasificador asociado al clasificador base. Este trabajo mantiene algunas similitudes con el trabajo realizado por Ting [130].

3.4. Algoritmos Genéticos

3.4.1. Definición

Los Algoritmos Genéticos (AG's) son procedimientos de búsqueda muy ligados a la teoría de la evolución mediante la selección artificial [63]. En términos de búsqueda clásica, los AG's se pueden ver como un tipo de búsqueda en haz [36]. Los tres principales componentes son:

- El haz, denominado población en este tipo de técnica. Contiene el conjunto de puntos (soluciones candidatas llamadas individuos) en el espacio de búsqueda que el algoritmo está actualmente explorando. Todos los puntos son representados usualmente mediante cadenas de bits. Esta representación, independiente del dominio de las posibles soluciones, hace que los AG's sean muy flexibles.
- Operadores de búsqueda. Éstos transforman las soluciones candidatas actuales en nuevas soluciones candidatas. Su principal característica es que, como

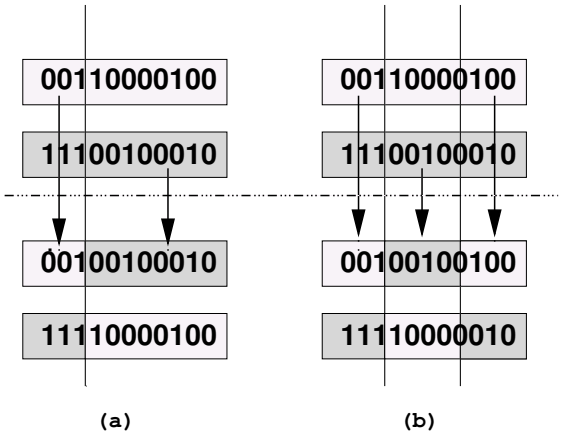


Figura 3.6: Algoritmos Genéticos: sobrecruzamiento de uno (a) y dos puntos (b).

operan sobre cadenas de bits, son independientes del dominio. Los operadores de los AG's están también basados en analogías biológicas [63, 64]. Los tres operadores más utilizados son:

- Reproducción: copia una solución candidata sin modificación.
- Cruzamiento: toma dos soluciones candidatas, las mezcla y genera dos nuevas soluciones candidatas. Existen muchas variaciones de este operador (principalmente, de un punto y de dos puntos). Ver figura 3.6.
- Mutación: invierte un bit de una solución candidata (puede mutar de 0 a 1 ó de 1 a 0). El bit mutado es seleccionado aleatoriamente de entre los bits del individuo, con una cierta probabilidad que es parámetro del método, aunque en la bibliografía son habituales tasas de mutación del 1 %.

- La función heurística (o función de *fitness*). Esta función mide el valor de una solución candidata. La meta de un AG es encontrar soluciones candidatas que maximicen esta función.

Los AG's parten de una población creada aleatoriamente. A continuación se aplican los operadores genéticos para evaluar las soluciones candidatas (de acuerdo con la función heurística) hasta que es obtenida una nueva población (nueva generación). Un AG continúa produciendo nuevas generaciones hasta que se encuentra un individuo que se considera lo suficientemente bueno, o cuando el algoritmo llega a un punto donde es incapaz de encontrar mejores individuos (o hasta que el número de generaciones llega a un limite predefinido).

Los AG's se basan en el siguiente pseudocódigo:

1. Generación aleatoria de población inicial $G(0)$.

2. Evaluación de los individuos en $G(0)$ con la función heurística.

3. Repetir hasta que se encuentre una solución o la población converja:

   3.1 Aplicar selección-reproducción: $G(i) \longrightarrow G_a(0)$
   3.2 Aplicar sobrecruzamiento: $G_a(i) \longrightarrow G_b(i)$
   3.3 Aplicar mutación: $G_b(i) \longrightarrow G_c(i)$
   3.4 Obtener una nueva generación $G(i+1) = G_c(i)$
   3.5 Evaluar la nueva generación $G(i+1)$
   3.6 $i = i + 1$

La producción de una nueva generación $G(i+1)$ a partir de $G(i)$ (pasos 3.1, 3.2, 3.3) se describe a continuación. Primero, una nueva población $G_a(i)$, también denominada población auxiliar, es generada por medio de la selección. Con el propósito de completar la población de n individuos para $G_a(0)$, las soluciones candidatas son estocásticamente seleccionadas con reemplazo de $G(i)$ n veces. La probabilidad de seleccionar un individuo es, en la mayoría de los casos, el cociente entre su *fitness* y el *fitness* total de la población. Esto quiere decir que habrá varias copias de individuos muy buenos en $G_a(0)$, mientras que probablemente no se encuentren individuos cuya función de *fitness* sea pobre. Sin embargo, debido a la aleatoriedad, aún los malos individuos tienen una probabilidad de aparecer en $G_a(0)$. Este método es conocido como "selección proporcional al fitness", pero existen otros varios métodos más, como el torneo y el ranking [59, 96].

Para obtener $G_b(i)$ se aplica el sobrecruzamiento a un porcentaje fijo, p_c, de individuos seleccionados aleatoriamente en $G_a(i)$. Como en cada sobrecruzamiento se toman dos padres y se producen dos descendientes, el sobrecruzamiento se lleva a cabo $p_c/2$ veces. De igual forma, $G_c(i)$ es generada a partir de $G_b(0)$ aplicando la mutación a un porcentaje p_m de los individuos.

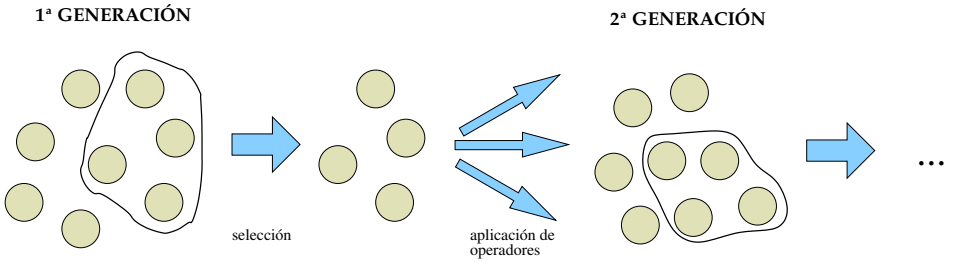


Figura 3.7: Proceso general de los Algoritmos Genéticos.

En la Figura 3.7 se muestra gráficamente el proceso seguido por los Algoritmos Genéticos.

3.4.2. Optimización mediante AG's

En los últimos años han ido apareciendo un conjunto de técnicas que comparten un principio común, el de emular los principios de la evolución natural, para la optimización de problemas. De todas estas técnicas, que han mostrado ser útiles y potentes, las más asentadas y aceptadas por la comunidad científica son los *Algoritmos Genéticos*, AG's. Esta mayor aceptación se traduce en que son las técnicas más empleadas en la industria: diseño de circuitos, distribución de componentes en la superficie de una antena, optimización del recorrido de tuberías en un edificio, etc, proporcionando resultados tan buenos, que han sorprendido a los propios expertos.

Desde el punto de vista más formal, son la única de estas técnicas que posee una base matemática, que aunque ha sido discutida en numerosas ocasiones, no ha sido rebatida y es aceptada por todos [64]. Se trata del problema conocido como *two-armed-bandit*, descrito ampliamente en [64] y [59].

Desde el punto de vista de la utilidad de los AG's en problemas de optimización, han sido empleados también con éxito en la optimización de funciones con restricciones, abordado por una parte, mediante el diseño de operadores genéticos que incorporaban esas restricciones, por ejemplo en la resolución del problema del TSP con AG's, o para evitar la pérdida de generalidad en el método que esto supone, mediante la transformación en un problema sin restricciones, donde se produce la penalización de las soluciones que incumplen la restricción, incluyéndolo en la función de fitness.

En cuanto a la utilización de AG's en la optimización de conjuntos de clasificadores, recientemente Zhou et al. [150] determinan el número correcto clasificadores que forman parte del conjunto utilizando AG's. Es importante señalar que todos los clasificadores son homogéneos puesto que todos son redes de neuronas.

3.5. Conclusiones

Una de las principales conclusiones de todos los trabajos vistos sobre *Stacking* es que existen resultados contradictorios en la literatura y que no existe un consenso sobre cuál configuración de clasificadores es la mejor.

Además, de todas ellas emana la sensación de que es necesario un conocimiento muy profundo del sistema y del problema para la determinación de los valores de cada uno de los parámetros de *Stacking*.

Por ello, se propone emplear una técnica de optimización que ha sido ampliamente probada en otros campos, los AG's, a la optimización de configuraciones de *Stacking*.

Capítulo 4

GA-Stacking

Dado que un conjunto de clasificadores generado a partir de *Stacking* está compuesto por un grupo de modelos creados a partir de distintos algoritmos de aprendizaje, surge la pregunta: ¿Qué algoritmos deben ser utilizados para generar los modelos de nivel-0 y que algoritmo debe utilizarse para generar el modelo de nivel-1?. Wolpert [145] originalmente señala que muchos aspectos sobre los parámetros de *Stacking*, incluidos los algoritmos que generan los clasificadores, se pueden considerar como *magia negra*. En principio, cualquier algoritmo puede ser utilizado para generar los clasificadores de ambos niveles. En el capítulo anterior se han mostrado una serie de trabajos encaminados a dar respuestas a estas preguntas. Por ejemplo, Ting y Witten [130] mostraron que un modelo lineal es útil para generar el clasificador de nivel-1 cuando se utilizan salidas probabilísticas de los modelos de nivel-0, Seewald [117] propone una variación en el espacio de atributos del nivel-1, Džeroski y Ženko [41] proponen la utilización de un árbol de regresión como meta-clasificador en vez del modelo lineal propuesto por Ting y Witten [130].

Tomando como base la idea de utilizar de distribuciones de probabilidades de clase como datos del meta-nivel propuesta por Ting y Witten [130], en este capítulo se describe un nuevo enfoque, basado en algoritmos genéticos, que busca obtener la configuración óptima de los parámetros de *Stacking* para un problema dado. En la sección 4.1 se describe el marco general propuesto en donde se aplican algoritmos genéticos en la búsqueda de la configuración óptima de *Stacking*. En la sección 4.2 se presenta la codificación propuesta para la utilización de los AG's. Por último en la sección 4.3 se detalla el método de evaluación de las posibles soluciones encontradas por los AG's.

4.1. Marco General: *GA-Stacking*

El término *GA-Stacking* es el acrónimo en inglés de *Genetic Algorithms for Stacking*. *GA-Stacking* plantea las respuestas a las preguntas de, ¿qué y cuántos algoritmos de aprendizaje utilizar para generar los clasificadores de nivel-base? y ¿qué algoritmo utilizar para genera el clasificador del meta-nivel?, como un problema de optimización, el cual puede ser resuelto mediante la aplicación de algoritmos genéticos.

En la Figura 4.1 se muestra el esquema propuesto, en donde se puede apreciar que, tomando como entrada los datos del dominio de aplicación, se aplican los algoritmos genéticos con la finalidad de obtener como salida del sistema la configuración óptima de los parámetros de *Stacking*.

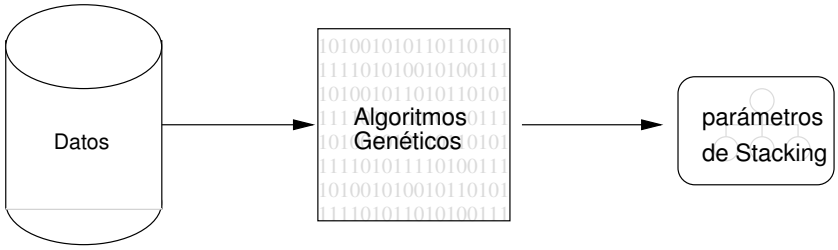


Figura 4.1: Esquema General de *GA-Stacking*.

La aplicación de los algoritmos genéticos a un problema de optimización dado requiere, principalmente, el estudio de dos aspectos: la especificación de la codificación de las soluciones y la definición de la función de *fitness*. El proceso de codificación de las soluciones se produce en lo que se puede considerar una fase previa (Figura 4.2, Fase I) a la ejecución de los algoritmos genéticos en sí. En cuanto a la evaluación del *fitness*, es un proceso iterativo que se lleva a cabo en cada generación de los algoritmos genéticos (Figura 4.2, Fase II) sobre todos los individuos (p) de la población (P). La fase de codificación de las soluciones se detalla en la sección 4.2. La evaluación de la función de *fitness* de cada individuo se detalla en la sección 4.3

4.2. Codificación de las Soluciones

Existen diversas maneras de representar las soluciones de un problema para que éste pueda ser tratado mediante la aplicación de algoritmos genéticos (e.g. codificación binaria, decimal, hexadecimal, etc). Para representar las posibles soluciones o individuos en el enfoque que se propone, se ha optado por una representación binaria ya que ésta permite el empleo de los AG's canónicos. Son estos AG's la forma original propuesta por Holland [63, 64], donde los operadores genéticos tienen un carácter completamente general y la base matemática es más

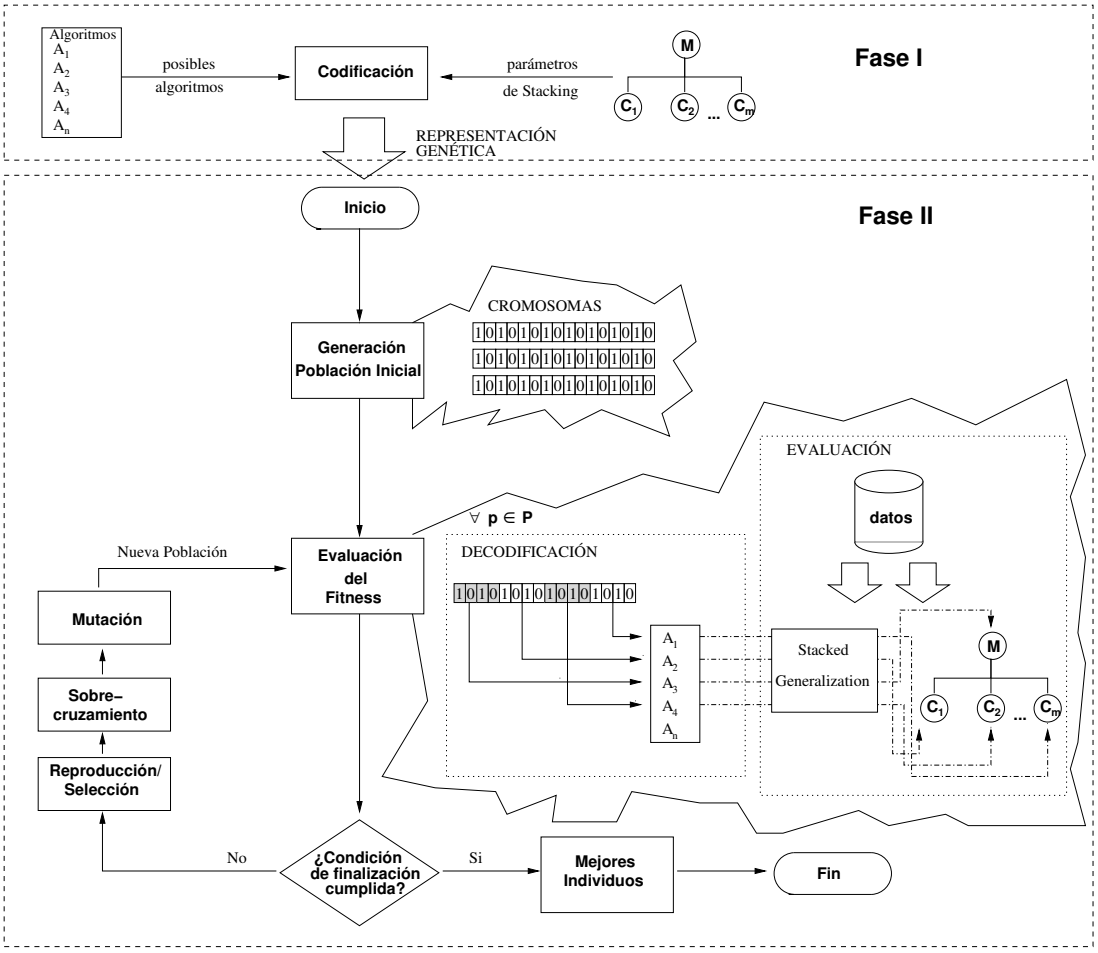


Figura 4.2: Marco Propuesto: *GA-Stacking*.

rigurosa. Además, existe un amplio estudio de la capacidad de barrido del espacio de búsqueda (hipótesis de los bloques constructivos [59, 96]) que no está contrastado cuando la base de codificación deja de ser binaria. Sin embargo, esquemas de codificación no binarias, con sus correspondientes modificaciones en los operadores genéticos, han sido muy empleadas. Una interesante recopilación se puede encontrar en [36]. Además, en este caso concreto, el empleo de codificaciones en bases distintas de la binaria, no serían imprescindibles, aunque pudieran reducir la longitud del cromosoma en algún caso.

En cuanto al tamaño del cromosoma que representa al individuo, éste está dado en función de dos factores:

- el número de algoritmos, m, que pueden ser seleccionados para generar los clasificadores, tanto los del nivel-base como el del meta-nivel.

- el número máximo de clasificadores de nivel-base, n, que pueden formar parte del conjunto.

Si en la codificación de los individuos sólo se considera el nombre del algoritmo que se utiliza para generar un clasificador, el tamaño del gen que representa al algoritmo dependerá del número de algoritmos disponibles. Por ejemplo, si existen 7 algoritmos de aprendizaje que se pueden seleccionar para generar los clasificadores del conjunto, se puede utilizar un gen con una longitud de tres bits para representar cualquiera de los algoritmos de aprendizaje posibles y representar también la opción de no seleccionar ninguno de éstos.

En la Figura 4.3 se muestra la codificación de un individuo en donde los primeros cuatro genes del cromosoma representan los cuatros algoritmos de aprendizaje a partir de los cuales se construirán los clasificadores de nivel-0 y el último gen representa el algoritmo a partir del cual se construirá el clasificador de nivel-1. La longitud de cada gen es de tres bits, razón por la cual se pueden seleccionar de entre 7 algoritmos de aprendizaje y la no presencia de ninguno.

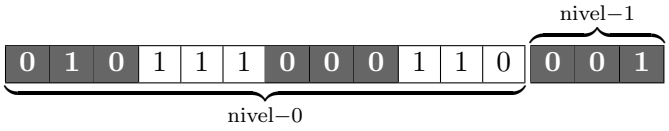


Figura 4.3: Descripción de la codificación binaria del individuo.

Por otra parte, si se considera que los parámetros de aprendizaje de los algoritmos deben formar parte de la tarea de optimización, se utilizan, además del gen que representa el nombre del algoritmo, una serie de genes que representan los parámetros de aprendizaje de éste. El tamaño de cada uno de estos genes depende de los parámetros de aprendizaje que representen.

De esta manera, el tamaño del cromosoma dependerá del número de genes que se utilicen para representar un algoritmo (G). En otras palabras, el tamaño del cromosoma en genes, T_c, está dado por:

$$T_c = G(n+1)$$

La longitud en bits del cromosoma se deriva del número y tamaño de los genes utilizados para representar los algoritmos capaces de generar un clasificador. En otras palabras la longitud en bits, T_b, del cromosoma está dada por:

$$T_b = (n+1)\left(\sum_{i=1}^{G} x_i\right)$$

en donde x_i representa el número de bits utilizados para codificar el gen i. En el caso del ejemplo mostrado en la Figura 4.3, $G = 1$ y $x_i = 3$.

4.3. Evaluación del *Fitness*

El proceso de evaluación u obtención del *fitness* de los individuos que conforman la población se lleva a cabo en dos etapas. La primera de éstas (Figura 4.4 [a]) implica la decodificación del individuo y, basándose en esta representación, la selección de los algoritmos que serán utilizados para generar los clasificadores del conjunto. Una vez que los clasificadores han sido seleccionados, se procede a la generación del conjunto de clasificadores, utilizando una parte del conjunto de datos disponibles (datos de entrenamiento). Al finalizar esta etapa, se tiene como resultado el conjunto de clasificadores.

La segunda etapa (Figura 4.4 [b]) en el proceso de evaluación del *fitness* de los individuos consiste en estimar la precisión del conjunto de clasificadores sobre un conjunto de datos que no ha sido utilizado en la construcción del conjunto (datos de validación). De esta forma, el *fitness* de cada individuo es el porcentaje de aciertos que obtenga el conjunto de clasificadores sobre las instancias del conjunto de datos de validación.

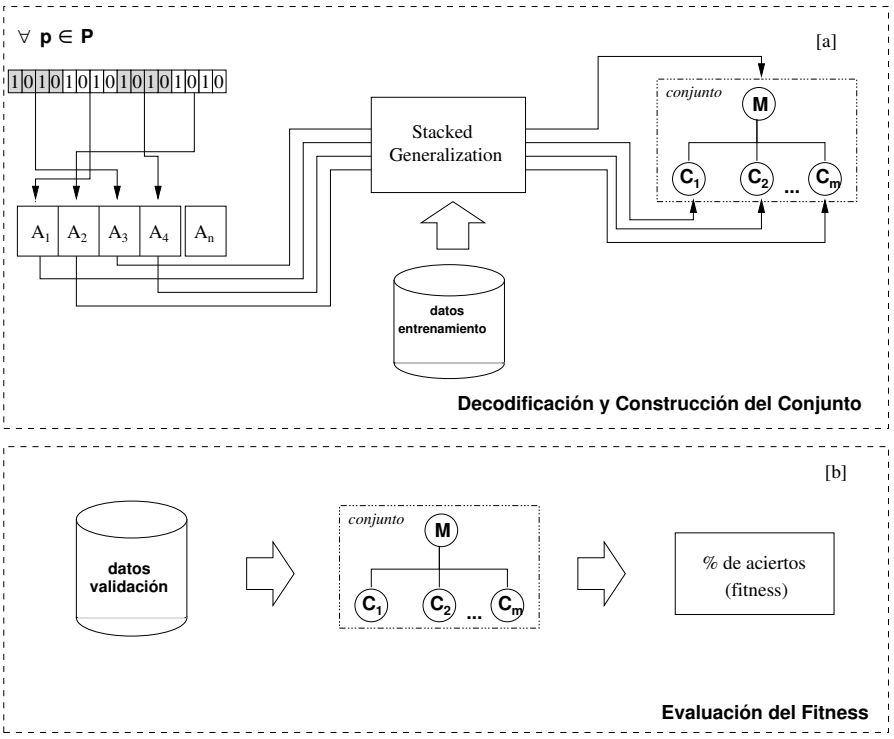


Figura 4.4: Evaluación del *fitness* en *GA-Stacking*.

Una alternativa a dividir el conjunto de datos en entrenamiento y validación una única vez, es realizar una validación cruzada. De esta manera, el *fitness* es la media del porcentaje de aciertos de la validación cruzada, realizando así, una mejor

estimación de la precisión de la solución.

4.4. Otros Parámetros de los AG's

Además del tipo de codificación y del número de genes que componen el cromosoma, los AG's poseen una serie de parámetros que deben ser configurados antes para llevar a cabo la búsqueda en el espacio de hipótesis. Parámetros de tipo estructural, como el tamaño de la población, y parámetros de ejecución como la tasa y el tipo de mutación, tipo de selección, etc, han de ser configurados para ejecutar los AG's. Sin embargo, se han empleado los valores utilizados con mayor frecuencia en estos sistemas [59]

Los valores asociados a estos parámetros se detallan en el capítulo de evaluación (capítulo 5).

Capítulo 5

Evaluación

Hasta este punto se ha presentado el sistema *GA-Stacking* para la obtención de la configuración óptima del algoritmo de generación de conjuntos, *Stacking*. Cómo cualquier otro sistema de aprendizaje, éste debe ser evaluado con la finalidad de comprobar si la configuración obtenida es la adecuada. Con el propósito de llevar a cabo esta evaluación, se han realizado una serie de experimentos que van desde los realizados con la finalidad de evaluar la viabilidad de la propuesta hasta los experimentos comparativos con los enfoques más actuales en cuanto a conjunto de clasificadores se refiere.

En la sección 5.1 se reflejan los resultados obtenidos en la primera fase de formulación de la propuesta. En la sección 5.2 se muestran los experimentos realizados con el propósito de evaluar los diferentes parámetros involucrados en *GA-Stacking*. Una comparativa de los resultados obtenidos mediante *GA-Stacking* con los bien conocidos métodos de generación de conjuntos de clasificadores, *Bagging* y *Boosting*, al igual que con los más recientes métodos de generación de conjuntos de clasificadores basados en *Stacking*, se presenta en la sección 5.3.

5.1. Viabilidad de *GA-Stacking*

En esta sección se muestran los resultados obtenidos en el proceso de evaluación de la viabilidad de utilizar *GA-Stacking* con el propósito de obtener la configuración óptima de los parámetros de *Stacking*. Para llevar a cabo esta evaluación se han utilizado dominios del conocido repositorio de datos del UCI [6]. En una primera serie de experimentos, se obtienen resultados prometedores, pero al llevar a cabo un análisis de los mismos, se muestran ciertos signos de sobreadaptación de los AG's a los datos utilizados para obtener el *fitness* de los individuos. Con el propósito de evitar esta sobreadaptación observada en los primeros experimentos, se lleva a cabo una segunda serie de experimentos en donde varía la forma de eva-

luar al individuo, de esta manera el cálculo del fitness de los individuos es distinto al método utilizado en los experimentos preliminares.

Para la realización de los experimentos que se muestran en esta sección, se han utilizado las implementaciones de los distintos algoritmos de aprendizaje que están disponibles en la herramienta conocida como WEKA [144] (versión 3.1.7). Esta herramienta incluye todos los algoritmos de aprendizaje utilizados, desde los algoritmos individuales hasta los algoritmos de construcción de conjuntos utilizados (i.e. *Bagging*, *Boosting* y *Stacking*).

La implementación de *GA-Stacking* combina dos partes: la primera de éstas se basa en los algoritmos de aprendizaje implementados en WEKA mientras que la implementación de los Algoritmos Genéticos esta basada en la librería GAJIT (*Genetic Algorithm Java Implementation Toolkit*) [43].

5.1.1. Resultados Preliminares

Los resultados que se muestran en esta sección corresponden a la evaluación inicial de *GA-Stacking* llevada a cabo sobre dos dominios (*ionosphere* y *dermatology*) del conocido repositorio de datos del UCI.

En la Tabla 5.1 se muestran los valores de los parámetros de los AG's utilizados en estos experimentos.

Tabla 5.1: Parámetros de los algoritmos genéticos.

Parámetros	Valores
Población	10
Generaciones	10
Tasa de élite	0.10
Tasa de desecho	0.40
Tasa de mutación	0.067

Tanto el tamaño de la población como el número de generaciones utilizados en estos experimentos, se configuraron basándose en el tamaño del espacio de búsqueda de los AG's. El espacio de búsqueda está determinado por el número de posibles combinaciones de algoritmos a partir de los cuales se generarán los miembros del conjunto siguiendo el algoritmo *Stacking*. Existen 490 posibles combinaciones de algoritmos si se considera que un algoritmo dado sólo puede aparecer una vez en cada combinación. Por otra parte, si se permite que un algoritmo pueda aparecer una o más veces dentro de una combinación, existe 2310 combinaciones posibles. En estos experimentos, el espacio de búsqueda utilizado es el que contempla la posibilidad de la presencia de un algoritmo una o más veces dentro de una combinación. En cuanto a las tasas de élite, desecho y mutación, se utilizaron los valores

que aparecen en los ejemplos de funcionamiento que proporciona GAJIT.

Ocho algoritmos de aprendizaje forman el grupo de algoritmos que pueden ser utilizados para generar, tanto los clasificadores de nivel-0 como el clasificador de nivel-1, dentro del conjunto de clasificadores generado mediante *Stacking*. A continuación se detallan los algoritmos de aprendizaje utilizados:

- C4.5 [110]. Genera árboles de decisión - (C4.5).
- Un clasificador *Naive Bayes* probabilístico [76] - (NB).
- IB1 [1]. Este es un algoritmo basado en instancias - (IB1).
- PART [47]. Forma listas de decisión a partir de árboles de decisión parcialmente podados, generados utilizando la heurística de C4.5 - (PART).
- *Decision Table* [83]. Es un clasificador simple que utiliza la clase mayoritaria. - (DT).
- *Decision Stump* [72]. Genera árboles de decisión de un solo nivel - (DS).
- IB*k* [1]. Algoritmo de K-vecinos más cercanos - (IB*k*).
- Un clasificador *Naive Bayes* simple en donde los atributos numéricos son modelados por una distribución normal [35] - (NBS).

Es importante señalar que en IB*k* por defecto el valor de K es 1, por esta razón su comportamiento es similar a IB1. De igual forma, los resultados que obtiene NBS son similares a NB y, en algunos dominios, por su forma de generar el clasificador, no es capaz de construir éste. No obstante, cabe mencionar que estos algoritmos han sido utilizado con el fin de redondear el número de algoritmos disponibles a ocho con la finalidad de poder representar los mismos mediante un gen de tres bits. Por estas razones, en estos experimentos, se hace referencia sólo a los resultados obtenidos por IB1 y NB.

Por otra parte, se han utilizado dos métodos de generación de conjuntos con la finalidad de comparar los resultados obtenidos mediante *GA-Stacking*. Estos métodos son:

- *Bagging*: Método de construcción de conjuntos homogéneos basado en el submuestreo del conjunto de datos. (ver sección 3.2.2). En este caso el algoritmo base utilizado es C4.5.
- *Boosting*: Método de construcción de conjunto homogéneos basado en la asignación de pesos a las instancias del conjunto de datos (ver sección 3.2.2). La implementación utilizada es la del algoritmo *AdaBoostM1*. El algoritmo base utilizado es C4.5.

Para evaluar el conjunto de clasificadores generado a partir de la configuración de *Stacking* encontrada por *GA-Stacking*, se realizó el proceso que se detalla a continuación:

- cada conjunto de datos fue dividido aleatoriamente en dos partes, *A* y *B*
- la parte *A*, que posee alrededor de un 85 % del total de instancias disponibles en el dominio, fue utilizada como conjunto de entrenamiento y a su vez como conjunto de evaluación de la función de *fitness*.
- para calcular el *fitness* de cada individuo de la población, se genera un conjunto de clasificadores mediante *Stacking*. Los parámetros de *Stacking* (en este caso los algoritmos de aprendizaje) están codificados en el individuo. Una vez generado el conjunto de clasificadores, se utiliza como función de *fitness* la precisión del conjunto de clasificadores a la hora de clasificar las instancias que forman el conjunto de datos *A*
- el conjunto de datos *B* es utilizado como conjunto de test para estimar la precisión de las hipótesis obtenidas mediante *GA-Stacking* sobre un conjunto de datos que no ha sido utilizado en su construcción. Cabe señalar que por la condición de preliminares de estos experimentos, la evaluación del *fitness* no se lleva a cabo como se detalla en la sección 4.3, en donde se propone dividir el conjunto de entrenamiento en dos o realizar una validación cruzada.

En esta serie de experimentos se seleccionaron dos dominios del repositorio de datos del UCI, *ionosphere* y *dermatology*. Estos dominios han sido ampliamente utilizados en estudios previos en cuanto a conjunto de clasificadores se refiere.

Los resultados obtenidos en esta serie de experimentos se reflejan en la Tabla 5.2. Las columnas dos y cuatro muestran la precisión en la tarea de clasificación del conjunto de clasificadores sobre el conjunto de entrenamiento en los dominios de *ionosphere* y *dermatology* respectivamente. Por otro lado, las columnas tres y cinco reflejan los resultados obtenidos sobre los conjuntos de test.

La parte superior de la tabla muestra los resultados obtenidos utilizando individualmente los algoritmos de aprendizaje disponibles. En la parte central de la tabla se refleja los resultados obtenidos por los algoritmos de generación de conjuntos homogéneos, *Bagging* y *Boosting*, y por la hipótesis encontrada por *GA-Stacking*. La hipótesis evaluada corresponde al mejor individuo de la última generación de los AG's. Esta hipótesis obtiene un 100 % de precisión sobre el conjunto de entrenamiento en ambos dominios, igualado únicamente por IB1[1]. Sin embargo, los resultados sobre los conjuntos de test (85.71 % en *ionosphere* y 95.45 % en *dermatology*) reflejan un rendimiento inferior sobre los datos que no han sido utilizados

[1]Dado que las instancias de test se evalúan basándose en las instancias de entrenamiento almacenadas

en la fase de entrenamiento. En el dominio de *ionosphere* el resultado obtenido mediante *GA-Stacking* es superado por PART (88.57 %), la tabla de decisión (88.57 %) y *Boosting* (91.43 %), siendo este último el mejor de todos de los algoritmos utilizados. En tanto que en el dominio *dermatology*, el resultado sobre el conjunto de test obtenido por la hipótesis encontrada por *GA-Stacking* es superada por los clasificadores generados a partir de *Naive Bayes* y *Boosting*, obteniendo ambos los mejores resultados en este dominio (96.97 %).

Sin embargo, si se analiza la evolución de los individuos, se pueden encontrar hipótesis en generaciones previas de los AG's que obtienen mejores resultados que cualquiera de los clasificadores individuales y los de generación de conjuntos, sobre los conjuntos de test de ambos dominios (parte inferior de la Tabla 5.2). Estos resultados y el hecho de que en ambos dominios el porcentaje de acierto sobre el conjunto de entrenamiento es del 100 %, indican que *GA-Stacking* se está sobreadaptando a los datos de entrenamiento a medida que pasa cada generación.

Tabla 5.2: Resultados preliminares de la evaluación de *GA-Stacking*.

Algoritmo	*Ionosphere*		*Dermatology*	
	Entrenamiento	Test	Entrenamiento	Test
Individuales				
C4.5	98.42	82.86	96.67	92.42
Naive Bayes	85.13	82.86	98.67	96.97
PART	98.73	88.57	96.67	93.94
IB1	100.00	82.86	100.00	92.42
Decision Stump	84.18	80.00	51.33	45.45
Decision Table	94.94	88.57	96.67	87.88
Conjuntos				
Bagging con C4.5	97.78	85.71	97.00	93.94
Boosting con C4.5	100.00	91.43	100.00	96.97
GA-Stacking	100.00	85.71	100.00	95.45
Generaciones previas (AG's)				
GA-Stacking	97.78	94.29	98.67	98.48

En la Figura 5.1 se muestra la evolución del *fitness* en el dominio *dermatology*. Como se puede apreciar en la tercera generación existe un individuo que obtiene el máximo *fitness* (100 % de acierto), pero los resultados sobre el conjunto de test empeoran a partir de esta generación. Por otra parte, también se encuentran reflejadas la media de aciertos de los tres mejores individuos de cada generación, tanto en entrenamiento/*fitness* como los resultados sobre el conjunto de test. Cabe destacar que el *fitness* promedio llega a alcanzar el 100 % de acierto en la sexta generación, generación hasta la cual el porcentaje de acierto sobre el conjunto de test ha ido decrementándose generación tras generación.

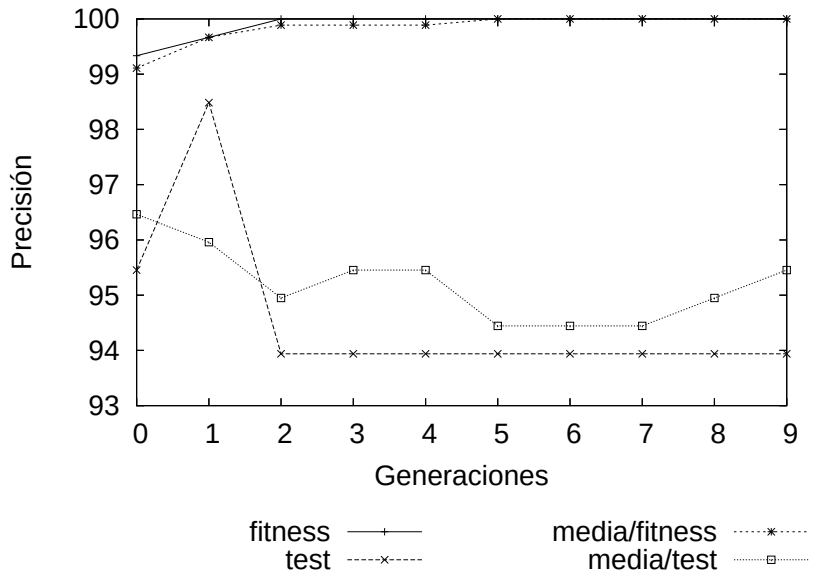


Figura 5.1: Evolución del *fitness* en el dominio *dermatology* (mejor individuo y promedio de los tres mejores individuos en cada generación).

5.1.2. Evitando la Sobreadaptación

En los experimentos preliminares de la sección anterior, los individuos se sobreadaptan porque el valor de la función de *fitness* se obtiene a partir de las mismas instancias de entrenamiento que fueron utilizadas para construir el conjunto de clasificadores mediante la configuración de *Stacking* asociada al individuo. Obviamente, la precisión del individuo sobre las instancias que fueron utilizadas en su construcción es alta. Por esta razón, con el propósito de evitar la sobreadaptación, en esta nueva serie de experimentos, el valor de la función de *fitness* fue calculada a partir de un conjunto de datos denominado conjunto de *validación*. En otras palabras, el conjunto de instancias de entrenamiento se dividió, a su vez, en dos partes aleatoriamente. El 80 % de las instancias de entrenamiento se utilizan para construir el conjunto de clasificadores a partir de la configuración de *Stacking* asociada a cada individuo, y el restante 20 % - el conjunto de validación - se utiliza para dar una estimación sin sesgo de la precisión del individuo. Es decir, el segundo conjunto se utiliza para evaluar el *fitness* del individuo. Cabe destacar que al utilizar un 20 % del conjunto de entrenamiento como conjunto de evaluación del *fitness*, el conjunto de clasificadores se genera a partir de menos instancias de entrenamiento que en los experimentos previos.

Con vistas a evaluar experimentalmente *GA-Stacking* con un número mayor de dominios que en los experimentos anteriores, se amplió la cantidad de dominios utilizados a seis. Todos los dominios han sido utilizados previamente en otros estudios relacionados con conjuntos de clasificadores y están disponibles en el repositorio de bases de datos para aprendizaje automático del UCI. Las características de estos conjuntos de datos se describen en la Tabla 5.3.

Tabla 5.3: Descripción de los dominios utilizados en la evaluación de *GA-Stacking*.

Dominio	Atributos	Tipo de Atributos	Instancias	Clases
dermatology	34	numérico-nominal	366	6
dna-splice	60	nominal	3190	3
heart	13	numérico-nominal	303	2
ionosphere	34	numérico	351	2
musk	166	numérico	476	2
sonar	60	numérico	208	2

Los parámetros de los AG's utilizados en estos experimentos son los mismos que se utilizaron en la primera serie de pruebas y que se muestran en la Tabla 5.1. Con el propósito de estimar la tasa de acierto de los algoritmos evaluados, se realizó una validación cruzada de 5 *carpetas* (del inglés: *folders*). Por razones experimentales *GA-Stacking* fue ejecutado una única vez en cada ciclo de la validación cruzada. Es decir, se exploró el espacio de búsqueda partiendo de un único punto con cada conjunto de datos.

La precisión de clasificación de cada clasificador/conjunto, C, para un dominio dado es estimada mediante la media de la validación cruzada realizada, denominada $prec(C)$. Para comparaciones entre dos algoritmos de aprendizaje, se calculan las mejoras relativas y *t-test* pareados, como se describe a continuación.

Para medir la mejora en la tarea de clasificación en un dominio dado utilizando un clasificador/conjunto C_1 en comparación a utilizar C_2 se calcula la mejora relativa mediante: $1 - error(C_1)/error(C_2)$. El error de un clasificador/conjunto C viene dado por $1 - prec(C)$. La media de la mejora relativa (MMR) sobre todos los dominios es calculada utilizando la media geométrica de la reducción del error en dominios individuales: $1 - media\_geométrica(error(C_1)/error(C_2))$ [41].

La significación estadística de la diferencia en el rendimiento de los algoritmos es llevada a cabo mediante un *t-test* pareado (C_1 y C_2 utilizan exactamente los mismos conjuntos de datos) con un nivel de significación del 95 %: $+/-$ a la derecha de un dato en la tablas que reflejan los resultados de los clasificadores indican que C_1 es significativamente mejor/peor que C_2.

La Tabla 5.4 muestra los resultados de los clasificadores individuales sobre

los dominios de prueba y en la Tabla 5.5 se muestra la precisión obtenida por los tres métodos de construcción de conjuntos, incluyendo la hipótesis encontrada por *GA-Stacking*. Los mejores resultados están resaltados.

Tabla 5.4: Tasa de precisión de los algoritmos individuales.

Dominio	C4.5	PART	NB	IB1	DT	DS
dermatology	94.33	94.59	97.03	94.59	87.83	50.40
dna-splice	94.12	92.11	95.63	75.62	92.49	62.42
heart	74.00	82.00	**83.00**	78.00	76.33	70.67
ionosphere	90.14	89.30	82.82	87.61	89.30	82.82
musk	83.33	85.21	74.38	86.46	82.08	71.46
sonar	72.38	73.81	67.62	79.52	71.90	73.93

Tabla 5.5: Tasa de precisión de los algoritmos de generación de conjuntos.

Dominio	*Bagging*	*Boosting*	*GA-Stacking*
dermatology	94.59	97.03	**97.30**
dna-splice	94.56	94.43	**95.72**
heart	76.33	79.67	80.67
ionosphere	**92.11**	91.83	90.42
musk	87.29	**88.96**	83.96
sonar	80.00	79.05	**80.48**

A excepción de un dominio (*heart*), los algoritmos de generación de conjuntos obtienen mayor precisión que cualquiera de los clasificadores individuales. El conjunto de clasificadores generado a partir de la configuración de los parámetros de *Stacking* encontrada por los algoritmos genéticos obtiene mayor precisión en tres de los seis dominios utilizados en los experimentos en comparación con los otras técnicas de generación de conjuntos.

En la Tabla 5.6 se muestra la comparación de los conjuntos de clasificadores generados a partir de la configuración de *Stacking* encontrada por *GA-Stacking* con los clasificadores individuales y las otras técnicas de generación de conjuntos. Como se puede apreciar, las soluciones generadas a partir de *GA-Stacking* mejoran, en promedio, a todos los clasificadores individuales y a *Bagging*, no así a *Boosting*. Sin embargo, si se analiza la significación estadística de los resultados, las soluciones de *GA-Stacking* no son significativamente peores que ninguno de los clasificadores individuales ni que ninguno los métodos de generación de conjuntos en ninguno de los dominios. Por otra parte, *GA-Stacking* es mejor, significativamente, que cualquiera de los clasificadores individuales o las técnicas de generación de conjuntos al menos una vez, como se puede apreciar en la última fila de la

Tabla 5.6: Mejora relativa en precisión (en %) de las hipótesis encontradas por *GA-Stacking* comparándolas con los distintos clasificadores individuales y las técnicas de clasificación de conjuntos homogéneos *Bagging* y *Boosting* y su significación estadística ($+/-$ es mejor/peor, '.' es no significativa.)

Dominio	C4.5	PART	NB	IB1	DT	DS	*Bagging*	*Boosting*
heart	25.64 .	-7.41 .	-13.73 .	12.12 .	18.31 .	34.09 +	18.31 +	4.92 .
sonar	29.31 .	25.45 .	39.71 +	4.65 .	30.51 .	26.79 .	2.38 .	6.82 .
musk	3.75 .	-8.45 .	37.40 +	-18.46 .	10.47 .	43.80 +	-26.23 .	-45.28 .
ionosphere	5.77 .	7.92 .	42.64 +	20.48 .	7.92 .	42.64 +	-24.96 .	-20.65 .
dermatology	52.39 .	50.01 +	9.12 .	50.01 +	77.78 +	94.54 +	50.01 .	9.11 .
DNA splice	27.27 +	45.82 +	2.16 .	82.45 +	43.10 +	88.62 +	21.39 +	23.16 +
Promedio	25.94	22.51	22.37	35.92	37.58	68.55	11.08	-1.36
Gana/pierde	1+/0-	2+/0-	3+/0-	2+/0-	2+/0-	5+/0-	2+/0-	1+/0-

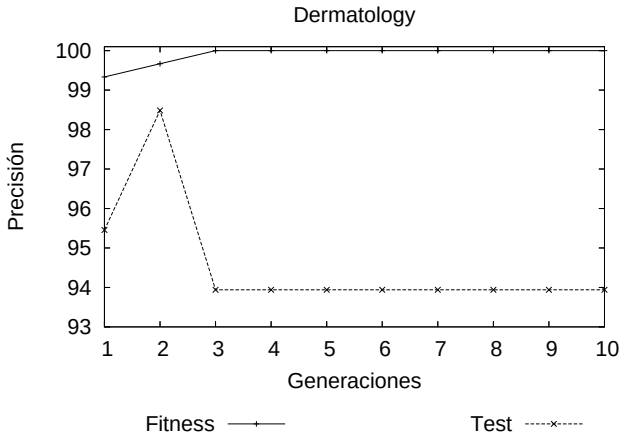
Tabla 5.6.

En cuanto a la evolución del *fitness*, en la Figura 5.2 se muestra una comparativa entre la utilización de un mismo conjunto de datos, tanto como para generar el conjunto de clasificadores como para evaluar la función de *fitness* (**a**), y la utilización de un conjunto de datos de validación reservados para evaluar la función de *fitness* (**b**). Como se puede apreciar, al utilizar dos conjuntos de datos en el proceso de obtención del conjunto de clasificadores, el porcentaje de aciertos sobre el conjunto de datos de test es superior al obtenido al utilizar un único conjunto para entrenamiento y *fitness*. Por otra parte, si se analiza la evolución de las curvas que reflejan el *fitness*, es evidente que la utilización de dos conjuntos de datos evita, en cierta medida, la sobreadaptación de las soluciones.

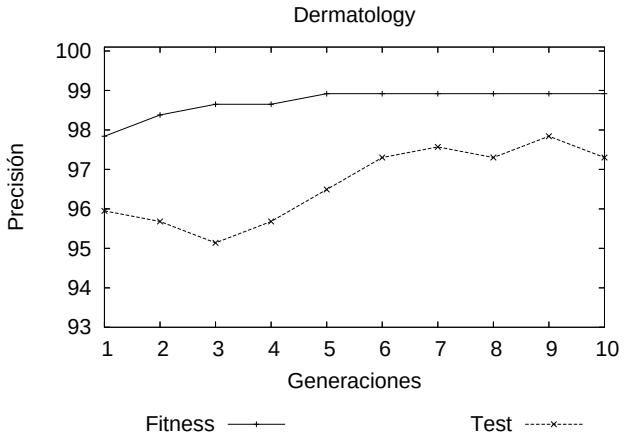
En las Figuras 5.3 y 5.4 se muestra la evolución del *fitness* y la evolución de la precisión sobre los conjuntos de entrenamiento y test para los distintos dominios utilizados en estos experimentos. En ninguno de los dominios se alcanza un 100 % de acierto sobre el conjunto de entrenamiento o sobre el conjunto de *fitness*. Por otro lado los resultados sobre el conjunto de test, a pesar que en algunos dominios experimenta un leve decremento, se pueden considerar como buenos. A partir de este análisis se puede señalar que el utilizar conjuntos distintos para entrenamiento y *fitness* es adecuado para prevenir la sobreadaptación.

5.2. Parámetros de *GA-Stacking*

Una vez establecida la viabilidad de la aplicación de AG's a la tarea de configuración de *Stacking*, se han llevado a cabo una serie de experimentos con la finalidad de determinar la configuración adecuada de los parámetros asociados a *GA-Stacking*. Estos parámetros van, desde los algoritmos de aprendizaje que pueden ser utilizados por *GA-Stacking*, hasta la configuración de los parámetros de aprendizaje de cada uno de estos algoritmos. En las secciones 5.2.1, 5.2.2 y 5.2.3,



(a)



(b)

Figura 5.2: Comparación de la evolución del *fitness* utilizando el mismo conjunto de datos para entrenar y calcular el *fitness (a)* o distintos conjuntos *(b)* en el dominio *Dermatology*.

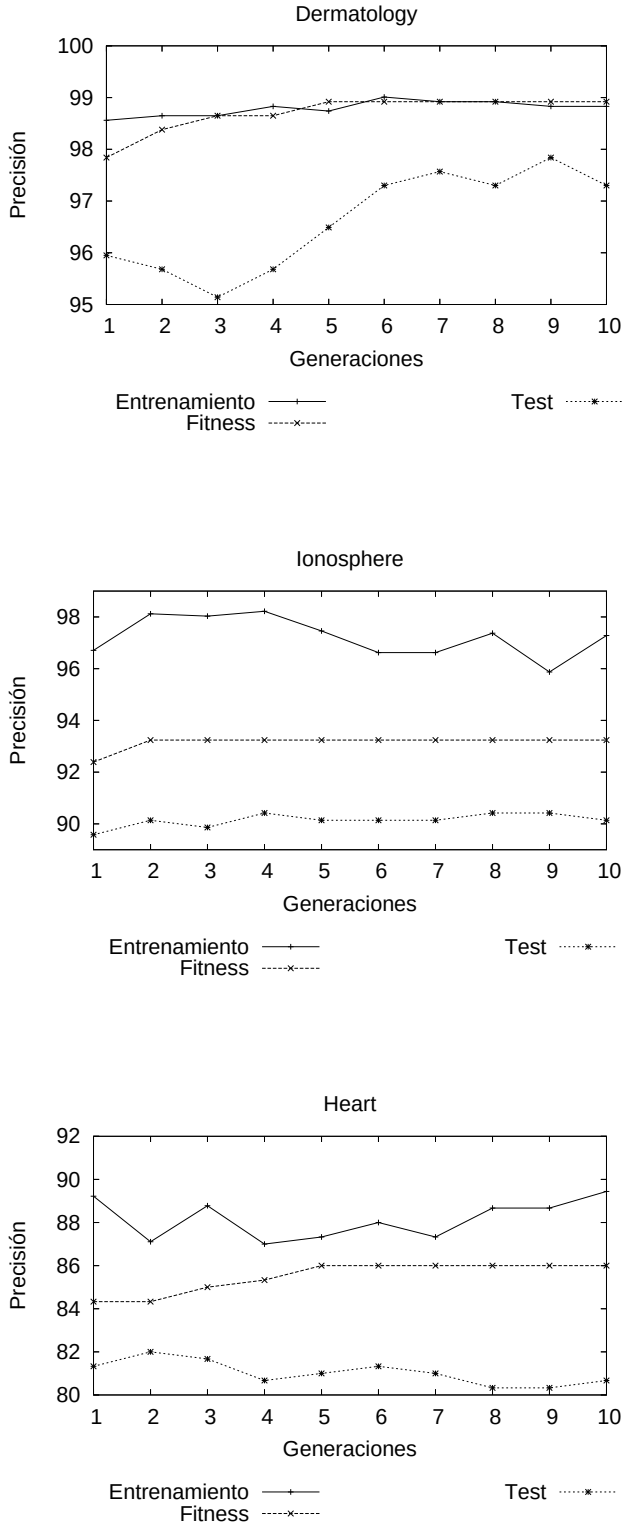


Figura 5.3: Evolución del *fitness* de las soluciones comparado con la precisión sobre el conjunto de entrenamiento y el conjunto de test para los dominios de *dermatology*, *ionosphere* y *heart*.

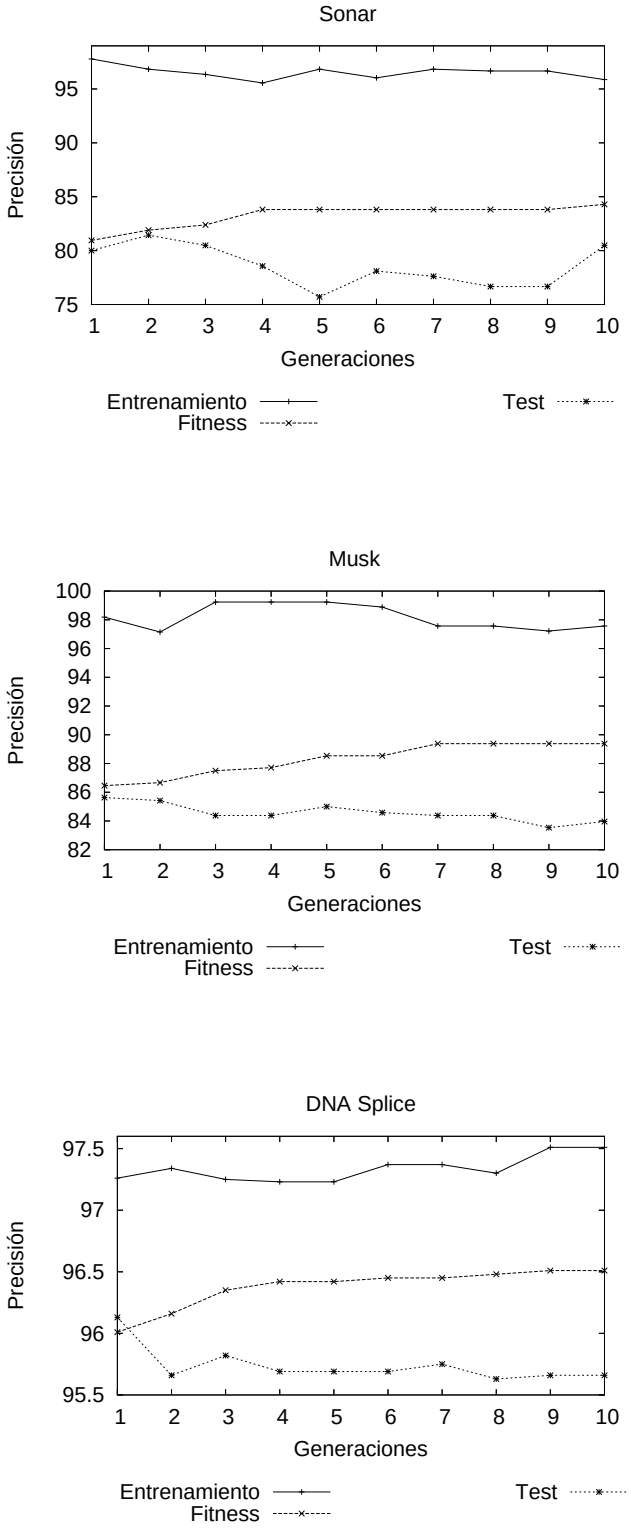


Figura 5.4: Evolución del *fitness* de las soluciones comparado con la precisión sobre el conjunto de entrenamiento y el conjunto de test para los dominios de *sonar*, *musk* y *DNA splice*.

se detallan los parámetros involucrados en la configuración de *GA-Stacking*. En la sección 5.2.4 se explica la configuración de los experimentos realizados. Por último en la sección 5.2.5 se muestran los resultados obtenidos en la comparación de las diferentes configuraciones.

5.2.1. Algoritmos de Aprendizaje

Con el propósito de ampliar el espacio de configuraciones en el cual los AG's llevan a cabo la búsqueda de la configuración óptima de los parámetros de *Stacking*, se amplió el número de posibles algoritmos de aprendizaje que pueden ser utilizados para generar los miembros del conjunto. Estos algoritmos pueden ser utilizados para generar tanto los clasificadores base como el meta clasificador.

En adición a los algoritmos utilizados en los primeros experimentos, C4.5, *Naive Bayes*, IB*k*, PART, *Decision Stump* y *Decision Table*, se han incorporado los siguientes algoritmos:

- *Random Forest* [14]. Este algoritmo construye un *Bosque Aleatorio* que se forma combinando una gran cantidad de árboles de decisión no podados - (RF).
- *Random Tree* [144]. Este algoritmo construye un árbol que considera K atributos al azar en cada nodo. No lleva a cabo ninguna poda - (RT).
- MLR [132]. Una regresión lineal de multirespuesta (*multi-response linear regression* - (MLR).
- MRMT [40]. Un árbol de regresión multirespuesta (*multi-response model tree*) - (MRMT).
- K* [22]. Este es un algoritmo basado en instancias que utiliza una medida de la distancia basada en la entropía - (K*).
- VFI [27]. Es un algoritmo que genera un clasificador que lleva a cabo la tarea de clasificación mediante votos, basado en intervalos de valores de los atributos. - (VFI)
- *Conjunctive Rule*. Es un algoritmo que genera un clasificador simple de reglas conjuntivas - (CR).
- *JRip* [25]. Un algoritmo de generación de reglas proposicionales - (JRIP).
- *Nnge* [92]. Algoritmo del tipo vecino más cercano el cual utiliza ejemplos generalizados no jerarquizados - (NNGE).
- *Hyper Pipes* [144]. Genera un clasificador que construye un *Hyper Pipe* para cada categoría, el cual contiene todos los puntos de esa categoría - (HP).

Para utilizar los clasificadores MLR y MRMT se utiliza un método de clasificación denominado *"clasificación por regresión"* (CPR) implementado en WEKA. De esta forma, la selección de MLR o MRMT es un parámetro de aprendizaje del método mencionado. Por omisión, el algoritmo utilizado es MLR.

5.2.2. Parámetros de Aprendizaje de los Algoritmos Utilizados

En los todos los trabajos relacionados con la configuración de *Stacking*, los algoritmos que se utilizan tanto, para construir los clasificadores de nivel-0 como el clasificador de nivel-1, se utilizan los parámetros de aprendizaje por omisión. Dado que estos parámetros pueden influir en los resultados que obtiene cada clasificador, *GA-Stacking* además de seleccionar entre los algoritmos de aprendizaje disponibles, puede realizar una búsqueda en el espacio de parámetros de cada uno de los algoritmos utilizados. En la Tabla 5.7 se detallan los parámetros de aprendizaje de los algoritmos utilizados que han sido seleccionados para ampliar el espacio de búsqueda.

Tabla 5.7: Parámetros de aprendizaje los algoritmos utilizados por *GA-Stacking* para generar el conjunto de clasificadores.

Algoritmo	Opción	Descripción
Naive Bayes	-K	Utilizar estimación del núcleo para atributos numéricos en vez de una simple distribución normal.
PART	-B	Utilizar división binaria para atributos nominales.
	-C	Establece el umbral de confianza para llevar a cabo la poda (por omisión: 0.25).
C4.5	-R	Utiliza la poda para reducir el error. No se desarrolla ningún sub-árbol.
	-U	Utiliza árbol sin podar.
	-B	Utilizar división binaria para atributos nominales.
	-S	No desarrollar sub-árboles.
	-C	Establece el umbral de confianzar para llevar a cabo la poda (por omisión: 0.25).
	-A	Si está fijado, se utiliza un suavizado Laplace para la predicción de probabilidades.

Continúa ...

Algoritmo	Opción	Descripción
IB*k*	-F	Se asigna un peso a los vecinos equivalente su similitud cuando se vota (por omisión: igual peso).
	-D	Se asigna un peso a los vecinos equivalente al inverso de su distancia cuando se vota (por omisión: igual peso).
	-N	No se utiliza normalización.
	-S	cuando K es seleccionado por validación cruzada para atributos numéricos, minimiza el error cuadrático medio (por omisión: error medio absoluto).
	-K	Fija el número de los vecinos más cercanos que se utilizarán para llevar a cabo la predicción (por omisión: 1).
Decision Stump	-	No posee parámetros configurables.
Decision Table	-I	Utiliza el vecino más cercano en vez de mayoría global de la tabla.
Clasificación por *Regresión*	-W	Especifica el nombre del algoritmo de predicción numérica que será utilizado como base del clasificador.
Random Tree	-S	Fija la semilla para el generador de números aleatorios (por omisión:1).
	-K	Fija el número de atributos a tener en cuenta en cada nodo.
	-M	Fija el número mínimo de instancias por nodo.
Random Forest	-I	Fija el número de árboles en el bosque (por omisión: 10).
	-K	Fija el número de atributos a considerar. (por omisión: $logM + 1$. M es el número de entradas.
	-S	Fija la semilla para el generador de número aleatorios (por omisión:1).
K*	-B	Fija la mezcla global (por omisión:20).
	-E	Establece si se utiliza mezcla entrópica automática.
	-M	Establece el método para manejar valores desconocidos (por omisión: curva media de la entropía de la columna).
VFI	-B	Fija el bias exponencial hacia intervalos confidentes (por omisión: 1.0).

Continúa ...

Algoritmo	Opción	Descripción
	-C	No pesa los intervalos de votación por confianza.
Conjunctive Rule	-S	Fija la semilla para la aleatorización (por omisión:1).
	-R	Fija si *no* se usa aleatorización.
	-E	Fija si se considera expresiones exclusivas para atributos nominales.
	-N	Fija el número de carpetas. Una carpeta es utilizado para llevar a cabo la poda (por omisión: 3).
	-M	Fija los pesos mínimos de las instancias en el proceso de división (por omisión: 2.0).
JRIP	-O	Fija el número de ejecuciones en el proceso de optimización (por omisión: 2).
	-P	Si *no* se utiliza poda.
	-E	Si *no* se verifica que la tasa error $>=$ $0{,}5$ dentro del criterio de parada (por omisión: verificar).
	-F	Fija el número de carpetas. Una carpeta es utilizada para llevar a cabo la poda (por omisión: 3).
	-N	Fija los pesos mínimos de las instancias en el proceso de división (por omisión: 2.0).
NNGE	-G	Fija el número de intentos de la generalización (por omisión: 5).
	-I	Fija el número de carpetas para calcular la información mutua.
Hyper Pipes	-	No posee parámetros configurables.

Es importante señalar que en algunos casos no se han utilizado todos los parámetros configurables de un algoritmo dado, puesto que se han utilizado los que se consideran de mayor relevancia en la construcción del clasificador.

5.2.3. Otros Parámetros

Además de los algoritmos de aprendizaje y sus parámetros configurables, existen otros aspectos ligados a *GA-Stacking*, como son el tamaño del conjunto de

clasificadores, la representación de las soluciones y los parámetros ligados a los AG's. En esta sección se abordan en detalle estos parámetros.

Tamaño del Conjunto de Clasificadores

El número de algoritmos que deben ser utilizados para generar los clasificadores de nivel-0 varía de un estudio a otro según la literatura. Por ejemplo, Ting y Witten [132] utilizan tres algoritmos, mientras que Seewald [117] utiliza seis algoritmos. Recientemente Džeroski y Ženko [41] utilizan tres y siete clasificadores base para llevar a cabo la comparación entre los distintos métodos de construcción de conjuntos.

Como se puede apreciar no hay un consenso en cuanto al número de clasificadores base que deben formar parte del conjunto de clasificadores. En vista de tal situación, se estableció como parámetro de *GA-Stacking* la determinación del tamaño del conjunto en donde sólo se fija el número máximo de posibles clasificadores base. Para que el número de clasificadores base fuese variable, se consideró la posibilidad de no utilizar ningún algoritmo de aprendizaje como una alternativa más a los algoritmos disponibles. De esta manera se reduce el número de clasificadores base generados. Es decir, dado un número máximo de clasificadores base, n, y el número de algoritmos disponibles, m, la probabilidad de que un algoritmo o la ausencia de algoritmos se encuentre codificada en la solución encontrada por *GA-Stacking*, E, está dada por:

$$p(E) = \frac{C\binom{m+1}{n} - C\binom{m}{n}}{C\binom{m+1}{n}}$$

en donde C es la combinación de m elementos en grupos de n elementos.

Para llevar a cabo un estudio detallado de la influencia del número de clasificadores base que forma el conjunto de clasificadores, el número máximo de clasificadores base se ha fijado en cuatro y diez clasificadores. De esta forma, se incluyen el número de clasificadores utilizado por los estudios previos reflejados en la literatura.

Representación de las Soluciones

Como se detalla en la sección 4.2, la codificación binaria es la codificación utilizada para representar las soluciones dentro de *GA-Stacking*. La representación de las soluciones depende esencialmente de tres factores previamente mencionados: el número de algoritmos disponibles, la utilización de los parámetros de aprendizaje de dichos algoritmos y el número máximo de clasificadores base dentro del conjunto de clasificadores. Tomando en cuenta estos factores, se han desarrollado

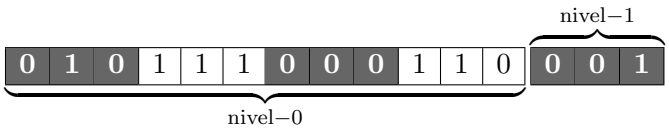


Figura 5.5: Codificación binaria de la configuración GAS5SPI.

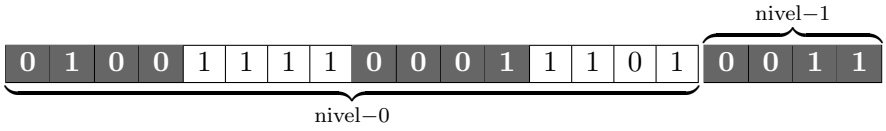


Figura 5.6: Codificación binaria de la configuración GAS5SPII.

seis configuraciones de *GA-Stacking* con la finalidad de determinar la mejor de éstas. A continuación se detallan las codificaciones utilizadas para cada una de las configuraciones.

- GAS5SPI[2]. La primera de las configuraciones de *GA-Stacking* en estos experimentos es similar a la utilizada en los experimentos previos (sección 5.1). La única diferencia es que en estos experimentos se considera la *no* presencia de algoritmos en una posición dada del cromosoma. El número máximo de posibles clasificadores base es cuatro ($n = 4$) y no se incluyen en la solución los parámetros de aprendizaje de los algoritmos. El número de algoritmos disponibles es siete ($m = 7$). En la Figura 5.5 se puede apreciar la codificación de las soluciones utilizada en estos experimentos.

  El número de genes del cromosoma es $T_c = 5$ y el tamaño en bits es $T_b = 15$.

- GAS5SPII. Esta configuración es similar a la anterior, con la única diferencia de que, en este caso, el número de algoritmos disponibles es 15 ($m = 15$). En la Figura 5.6 se puede apreciar la codificación de las soluciones utilizada en estos experimentos.

  El número de genes del cromosoma es $T_c = 5$ y el tamaño en bits es $T_b = 20$.

- GAS5CPI. En esta configuración de *GA-Stacking* se incorpora a la tarea de optimización la búsqueda de los parámetros de aprendizaje de cada uno de los algoritmos con la finalidad de encontrar la combinación adecuada para un dominio dado. En estos experimentos $m = 7$ y $n = 5$. Debido a la cantidad

[2]El nombre de las configuraciones de *GA-Stacking* deriva de, *GA-Stacking*, "GAS", el número de clasificadores que pueden formar el conjunto, "5" y "11" (incluyendo el meta-clasificador), la utilización o no de los parámetros de aprendizaje de los algoritmos, "CP" y "SP", y la versión del experimento dependiendo del número de algoritmos disponibles, "I" o "II" (7 y 15 respectivamente).

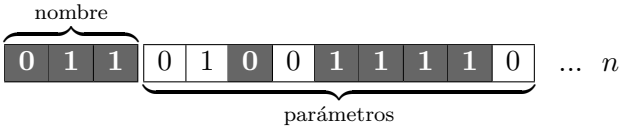


Figura 5.7: Codificación binaria de un clasificador dentro de la configuración GAS5CPI.

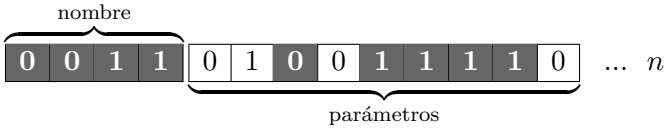


Figura 5.8: Codificación binaria de un clasificador dentro de la configuración GAS5CPII.

de algoritmos que se pueden utilizar y la diversidad de sus parámetros de aprendizaje, se optó por crear una representación general que incorporase la mayoría de los parámetros disponibles. En la Tabla 5.8 se puede apreciar la relación de los parámetros de aprendizaje de cada algoritmo y el gen que los representa. Dado que el número y tipo de parámetros de aprendizaje varía de un algoritmo a otro y, que para un mismo algoritmo existen parámetros mutuamente excluyentes, se diseño una codificación lo más general posible, aunque esto implique que en algunos casos los genes no representen ningún parámetro debido al algoritmo de aprendizaje.

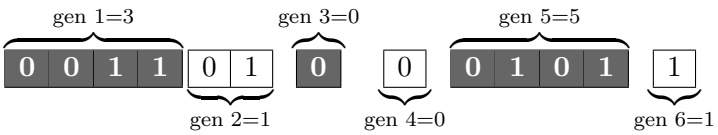
En la Figura 5.7 se muestra la codificación de un clasificador dentro del cromosoma que representa las soluciones en estos experimentos. Como se puede apreciar, se utilizan seis genes para representar un clasificador, uno para el nombre y cinco para codificar los parámetros de aprendizaje del mismo.

El número de genes del cromosoma es $T_c = 30$ y el tamaño en bits es $T_b = 60$.

- GAS5CPII. En esta configuración el valor de $m = 15$. Por esta razón el tamaño en bits del cromosoma es $T_b = 65$ mientras que el número de genes del cromosoma sigue siendo $T_c = 30$. En la Figura 5.8 se puede observar la codificación utilizada. Por otra parte, en la Figura 5.9 se muestra un ejemplo de la codificación de un clasificador que se generará a partir de C4.5.

- GAS11SP. En esta configuración de *GA-Stacking* se amplia el número máximo de clasificadores base que pueden formar parte del conjunto. En este caso $m = 10$ y la codificación es similar a la utilizada por GAS5SPII. En cuanto a la longitud del cromosoma, ésta está dada por $T_c = 11$ y $T_b = 44$.

- GAS11CP. En la última de las configuraciones de *GA-Stacking* evaluadas el número máximo de clasificadores base que pueden formar parte del conjunto



Gen	Valor	Opción
1	3	c4.5
2	1	-R
3	0	-
4	0	-
5	5	-C 0.35
6	1	-A

Figura 5.9: Ejemplo de la representación de C4.5 y sus parámetros de aprendizaje mediante una codificación binaria.

es 10 al igual que la configuración anterior, pero en este caso se incluyen los parámetros de aprendizaje de los algoritmos. Para estos experimentos la codificación es similar a la utilizada por GAS5CPII, tomando en cuenta el nuevo valor de m. En cuanto a la longitud del cromosoma, ésta está dada por $T_c = 66$ y su longitud en bits, $T_b = 143$.

Tabla 5.8: Correspondencia entre los parámetros de aprendizaje de los algoritmos utilizados por *GA-Stacking* y el gen que los representa dentro de la codificación binaria.

Genes \ Algoritmo[3]	Valor	NA[4]	NB	PART	C4.5	IB*k*	DS	DT	CPR[5]	RT	RF	K*	VFI	CR	JRIP	NNGE	HP
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Gen # 2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	1	-	-	-	-R	-F	-	-	-	-S 1	-S 1	-B 10	-B 2.0	-S 2	-O 3	-G 2	-
	2	-	-	-	-U	-D	-	-	-	-S 2	-S 2	-B 30	-B 3.0	-S 3	-O 4	-G 3	-
	3	-	-	-	-	-	-	-	-	-S 3	-S 3	-B 40	-B 4.0	-S 4	-O 5	-G 4	-
Gen # 3	0	-	-	-	-	-	-	-	-W A	-	-	-	-	-	-	-	-
	1	-	-K	-B	-B	-N	-	-I	-W B	-K 10	-K 10	-E	-C	-R	-P	-	-
Gen # 4	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	1	-	-	-	-S	-S	-	-	-	-	-	-	-	-E	-E	-	-
Gen # 5	0	-	-	-C 0.10	-C 0.10	-K 1	-	-	-	-	-	-	-	-	-	-	-
	1	-	-	-C 0.15	-C 0.15	-K 2	-	-	-	-K 5	-I 2	-M m	-	-N 4	-F 4	-I 2	-
	2	-	-	-C 0.20	-C 0.20	-K 3	-	-	-	-K 10	-I 4	-M n	-	-N 5	-F 5	-I 3	-
	3	-	-	-C 0.25	-C 0.25	-K 4	-	-	-	-K 15	-I 6	-M a	-	-N 6	-F 6	-I 4	-
	4	-	-	-C 0.30	-C 0.30	-K 5	-	-	-	-K 20	-I 8	-M d	-	-N 7	-F 7	-I 5	-
	5	-	-	-C 0.35	-C 0.35	-K 6	-	-	-	-K 25	-I 10	-	-	-N 8	-F 8	-I 6	-
	6	-	-	-C 0.40	-C 0.40	-K 7	-	-	-	-K 30	-I 12	-	-	-N 9	-F 9	-I 7	-
	7	-	-	-C 0.45	-C 0.45	-K 8	-	-	-	-K 35	-I 14	-	-	-N 10	-F 10	-I 8	-
	8	-	-	-C 0.50	-C 0.50	-K 9	-	-	-	-K 40	-I 16	-	-	-	-	-I 9	-
	9	-	-	-C 0.50	-C 0.50	-K 10	-	-	-	-K 45	-I 18	-	-	-	-	-I 10	-
	10	-	-	-C 0.50	-C 0.50	-K 11	-	-	-	-K 50	-I 20	-	-	-	-	-	-
	11	-	-	-C 0.50	-C 0.50	-K 12	-	-	-	-K 55	-I 22	-	-	-	-	-	-
	12	-	-	-C 0.50	-C 0.50	-K 13	-	-	-	-K 60	-I 24	-	-	-	-	-	-
	13	-	-	-C 0.50	-C 0.50	-K 14	-	-	-	-K 65	-I 26	-	-	-	-	-	-
	14	-	-	-C 0.50	-C 0.50	-K 15	-	-	-	-K 70	-I 28	-	-	-	-	-	-
	15	-	-	-C 0.50	-C 0.50	-K 16	-	-	-	-K 75	-I 30	-	-	-	-	-	-
Gen # 6	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	1	-	-	-	-A	-	-	-	-	-M 5	-	-	-	-M 3	-N 3	-	-

[3]Es representado por el Gen #1.

[4]Ningún Algoritmo.

[5]Las opciones $-A$ y $-B$ del gen #3 corresponden a utilizar MLR y MRMT respectivamente.

5.2.4. Configuración Experimental

En esta sección se describe en detalle la configuración de los experimentos llevados a cabo con la finalidad de evaluar las diferentes versiones de *GA-Stacking*.

Dominios

Para la prueba experimental de las distintas configuraciones de *GA-Stacking* se han utilizado 18 dominios del repositorio del UCI. Estos dominios han sido ampliamente utilizados en otros estudios sobre *Stacking*. Con el propósito de evaluar *GA-Stacking* con un grupo de instancias distintas a las que utiliza como conjunto de entrenamiento, se ha dividido cada dominio en dos partes: la parte A utilizada para evaluar las configuraciones encontradas por cada una de las versiones de *GA-Stacking*, y la parte B que es utilizada como conjunto de entrenamiento para encontrar la configuración óptima. En la Tabla 5.9 se muestran las características de los dominios utilizados.

Tabla 5.9: Descripción de dominios utilizados.

Dominio	Atributos	Instancias	Parte A	Parte B	Clases
australian	14	690	345	345	2
balance	4	625	312	313	3
breast-w	9	699	349	350	2
car	6	1728	1382	346	4
chess	36	3196	2876	320	2
diabetes	8	768	384	384	2
echo	6	132	66	66	2
german	20	1000	500	500	2
glass	9	214	100	100	6
heart	13	270	135	135	2
hepatitis	19	155	77	78	2
hypo	25	3163	2846	317	2
image	19	2310	1848	462	7
ionosphere	34	351	175	176	2
iris	4	150	75	75	3
soya	35	683	341	342	19
vote	16	435	217	218	2
wine	13	178	89	89	3

Parámetros de los AG's

En vista que el espacio de búsqueda de configuraciones se ve incrementado con la incorporación de más algoritmos y los parámetros de aprendizaje de éstos, se incrementó el número de individuos de la población y el número de generaciones. Otra diferencia con respecto a los primeros experimentos (sección 5.1) es la tasa de mutación que se vio incrementada con la finalidad de generar cambios en los individuos en menos generaciones. Los valores de los parámetros de los AG's para estos experimentos se muestran en la Tabla 5.10.

Tabla 5.10: Parámetros de los algoritmos genéticos.

Parámetro	Valores
Población	50
Generaciones	50
Tasa de élite	0.10
Tasa de desecho	0.40
Tasa de mutación	0.10

Comparación de las Versiones de *GA-Stacking*

Con el propósito de evaluar las distintas versiones de *GA-Stacking*, cada una de las configuraciones de *GA-Stacking* fue ejecutada tres veces en cada dominio con el conjunto de datos B. El mejor individuo de estas ejecuciones es tomado como la configuración óptima de *Stacking* encontrada por una versión dada de *GA-Stacking*. Este individuo no es un conjunto de clasificadores construido a partir de *Stacking*, sino una configuración de los algoritmos que se deben utilizar para generar los clasificadores nivel-base y el clasificador del meta-nivel. Todas las configuraciones encontradas por las versiones de *GA-Stacking* son comparadas entre sí mediante una validación cruzada estratificada de 10 carpetas sobre el conjunto de datos A. Un *t*-test pareado se utiliza para medir la significación estadística con un nivel del 95 %.

Con el propósito de calcular la mejora relativa obtenida por las configuraciones de *Stacking* encontradas por una versión de *GA-Stacking* dada sobre el resto de versiones, se calcula la *mejora relativa media* (MRM) sobre todos los dominios utilizados.

Otros

Tanto las implementaciones de los algoritmos de aprendizaje utilizados, como la prueba estadística utilizada (*t*-test), corresponden a la versión 3.4 de WEKA.

Analizando los resultados obtenidos en los experimentos para evitar la sobreadaptación a los datos de entrenamiento (sección 5.1.2) se puede apreciar que se evita que los AG's se sobreadapten tanto a los datos de *fitness* como a los datos de entrenamiento. Sin embargo, *Stacking* cuenta con menos datos para generar los clasificadores que formarán parte del conjunto. Además puede existir algún sesgo hacia el conjunto de *fitness*. Por esta razón la función de *fitness* en estos experimentos se calcula mediante un proceso de validación cruzada de 2 carpetas. Es decir, el conjunto de datos disponibles se divide en dos partes, primero se entrena con una de ellas y se obtiene el porcentaje de aciertos sobre la otra parte y viceversa. El valor del *fitness* es el promedio de aciertos de la validación cruzada. En la Figura 5.10 se muestra el proceso llevado a cabo para obtener el *fitness* de todos los individuos de la población.

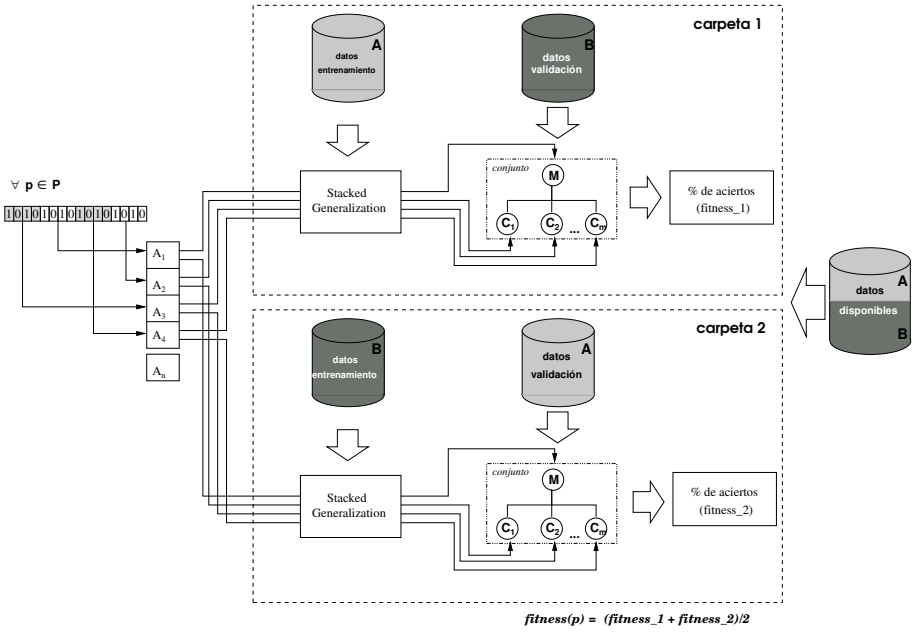


Figura 5.10: Cálculo del *fitness* mediante una validación cruzada de 2 carpetas.

Por otra parte, *Stacking* lleva a cabo un proceso de validación cruzada interna para construir los datos del meta-nivel. El número de carpetas utilizadas en este proceso, por omisión, es 10 y al igual que en los experimentos previos, no se ha incluido como parámetro en la evolución de las soluciones.

5.2.5. Resultados Experimentales

En esta sección se muestran los resultados obtenidos en la evaluación de las diferentes configuraciones de *GA-Stacking*.

El porcentaje de aciertos de los conjuntos de clasificadores formados a partir de las configuraciones de *Stacking* encontradas por las distintas versiones de *GA-Stacking* se muestran en la Tabla 5.11[6]. Sin embargo, con la finalidad de comparar el rendimiento de las soluciones encontradas por las distintas versiones de *GA-Stacking*, la Tabla 5.12 es más interesante: esta tabla refleja la mejora relativa media de X sobre Y para cada par de soluciones X e Y, al igual que el número de ganados:perdidos calculado a partir de un *t*-test realizado sobre los resultados de una validación cruzada de 10 carpetas (1×10 *t*-test).

Tabla 5.11: Resultados de la validación cruzada de 10 carpetas de las configuraciones de *Stacking* encontradas por las distintas versiones de *GA-Stacking*.

Dominios	GAS5SPI	GAS5SPII	GAS5CPI	GAS5CPII	GAS11SP	GAS11CP
australian	86.93	87.21	**88.11**	87.22	88.09	88.08
balance	90.08	94.23	**94.54**	92.95	94.22	94.23
breast	95.42	95.71	97.14	**95.99**	96.57	96.28
car	96.02	95.15	97.03	97.18	95.88	**97.47**
chess	99.27	99.17	99.23	99.24	**99.34**	99.27
diabetes	73.41	73.93	**75.24**	73.93	74.99	**75.24**
echo	86.67	84.17	87.50	**90.00**	84.17	**90.00**
german	74.00	**75.40**	71.00	75.00	71.00	73.60
glass	62.45	63.45	67.27	**76.00**	66.27	74.55
heart	82.75	**83.57**	82.03	82.80	75.44	82.75
hepatitis	**80.89**	**80.89**	79.64	78.57	80.71	79.64
hypho	**99.30**	99.12	99.26	98.98	98.95	99.23
images	97.94	97.89	97.56	**98.16**	97.62	98.10
ionosphere	89.02	90.23	**91.93**	89.61	90.26	90.23
iris	93.57	92.32	93.57	92.32	94.82	**97.32**
sonar	**97.18**	94.27	91.36	93.36	**97.18**	95.27
vote	**95.87**	93.07	95.84	**95.87**	91.75	94.00
wine	94.44	96.67	**98.89**	97.78	94.31	92.22

Examinando en detalle la Tabla 5.12 se puede ver que la diferencia en la media de la mejora relativa en todos los dominios entre la distintas configuraciones de *Stacking* es baja (entre -1.86 % y 1.82 %). Sin embargo, si se analiza la última columna de la tabla en donde se refleja la cantidad de veces en que las soluciones encontradas por las versiones de *GA-Stacking* son significativamente mejores o peores, en diferencia global, las versiones *GA-Stacking* que incorporan los parámetros de aprendizaje de los algoritmos utilizados, están por encima de las que no incor-

[6]En el Apéndice B se muestran las comparaciones individuales de cada una de las versiones de *GA-Stacking* con el resto.

Tabla 5.12: Mejora relativa de las soluciones encontradas por las distintas configuraciones de *GA-Stacking*. Las entrenadas en la fila X y columna Y muestran la mejora relativa de X sobre Y en % y en número de ganados:perdidos (de acuerdo a un 1×10 *t*-test).

	GAS5SPI	GAS5SPII	GAS5CPI	GAS5CPII	GAS11SP	GAS11CP	Total
GAS5SPI		0.12 1:1	0.80 1:3	1.46 2:3	-0.22 3:1	1.60 1:3	8:11
GAS5SPII	-0.14 1:1		0.66 0:2	1.33 0:2	-0.36 2:0	1.47 0:2	3:7
GAS5CPI	-0.81 3:1	-0.67 2:0		-0.67 2:0	-1.03 4:1	0.81 0:1	11:3
GAS5CPII	-1.48 3:2	-1.35 2:0	-0.67 0:2		-1.71 4:1	0.14 2:1	11:6
GAS11SP	0.22 1:3	0.36 0:2	1.02 1:4	1.68 1:4		1.82 0:3	3:16
GAS11CP	-1.63 3:1	-1.49 2:0	-0.82 1:0	-0.14 1:2	-1.86 3:0		10:3

poran éstos. En otras palabras, GAS5CPI (+8)[7], GAS11CP (+7) y GAS5CPII (+5), están por encima de GAS5SPI (-3), GAS5SPII (-4) y GAS11SP (-13).

Como se puede apreciar, la utilización de los parámetros de aprendizaje de los algoritmos con el propósito de ampliar el espacio de búsqueda de los AG's obtienen mejores resultados en comparación a no utilizarlos. Sin embargo, el incremento del número de algoritmos disponibles y el número de posibles clasificadores en el conjunto no parece que mejoren el *fitness* de las soluciones por sí solos. Esto puede deberse a que el espacio de búsqueda se incrementa significativamente y dado que se llevan a cabo 50 generaciones de 50 individuos, el número de configuraciones exploradas es relativamente pequeño en comparación con el espacio de búsqueda. Por otra parte, si se combinan la utilización de los parámetros de aprendizaje de los algoritmos, con el incremento del número de algoritmos y el número máximo de miembros del conjunto (GAS11CP), los resultados son comparables con la mejor de las versiones de *GA-Stacking*. En otras palabras, si se compara GAS5CPI y GAS11CP, ambos son tres veces significativamente peores que otra configuración de *GA-Stacking*, sin embargo GAS5CPI es, en un caso más, significativamente mejor que otra configuración de *GA-Stacking*. Pero, si se comparan las diferencias significativas entre uno y otro, GAS11CP es la mejor de las dos versiones (+1).

En cuanto a la evolución del *fitness*, en todas las versiones de *GA-Stacking* se nota un incremento constante en casi todos los dominios (ver Apéndice B) a excepción de los dominios *echo* y *sonar* en donde se alcanza el *fitness* máximo en la primera generación.

5.3. Rendimiento de *GA-Stacking*

Con la finalidad de medir el rendimiento de *GA-Stacking*, se ha llevado a cabo una comparativa de los resultados obtenidos mediante la aplicación de *GA-Stacking*

[7]El número entre paréntesis indica la diferencia absoluta entre ganados y perdidos

(versión GAS11CP) y los resultados obtenidos aplicando los conocidos algoritmos de generación de conjuntos homogéneos, *Boosting* y *Bagging*, además de otros algoritmos de generación de conjuntos heterogéneos basados en *Stacking*.

5.3.1. Configuración Experimental

En esta sección se muestran en detalle la configuración de los experimentos llevados a cabo con la finalidad de comparar *GA-Stacking* con otros métodos de generación de conjuntos y combinación de clasificadores.

Dominios

Con el propósito de comparar *GA-Stacking* con los diferentes métodos de generación de conjuntos, se ha utilizado un subconjunto de los dominios utilizados en la evaluación de las diferentes versiones de *GA-Stacking* (sección 5.2.4). La selección de estos dominios se llevó a cabo tomando en cuenta el número de clases, instancias y atributos con la finalidad de utilizar dominios representativos. Los dominios seleccionados son: *australian*, *balance*, *car*, *diabetes* y *glass*.

Parámetros de los AG's

Como se ha mencionado al principio de esta sección, la configuración de *GA-Stacking* utilizada para comparar el rendimiento de este método con el resto de algoritmos de generación de conjuntos es GAS11CP. Esta versión tiene como máximo 10 clasificadores base e incluye la configuración de los parámetros de aprendizaje de cada uno de los algoritmos utilizados para generar, tanto los clasificadores base como el clasificador del meta-nivel. Los parámetros de los AG's utilizados son los mismos utilizados en los experimentos de comparación de las distintas configuraciones de *GA-Stacking* y que están reflejados en la Tabla 5.10.

Algoritmos de Aprendizaje

Dentro de los algoritmos utilizados en estos experimentos existen dos categorías. La primera de estas categorías incluye los algoritmos de generación de clasificadores individuales. Estos algoritmos son los utilizados por *GA-Stacking* y los otros sistemas de construcción de conjuntos basados en *Stacking* para generar los miembros del conjunto. La otra categoría incluye los algoritmos de generación de conjuntos y métodos de combinación de clasificadores.

Los algoritmos de generación de clasificadores individuales utilizados son los que aparecen detallados en los experimentos de comparación de las distintas versiones de *GA-Stacking* (sección 5.2.1). Por otra parte, los algoritmos de generación

de conjuntos y combinación de clasificadores utilizados se describen a continuación:

- VOTO: Sistema de combinación de predicciones mediante votos.
- MEJORCV: Sistema que selecciona al mejor clasificador basado en una validación cruzada del conjunto de entrenamiento.
- *Bagging*. El algoritmo base utilizado es C4.5.
- *Boosting (AdaBoostM1)* El algoritmo base utilizado es C4.5.
- SMRMT: *Stacking* con modelo de árboles multi-respuesta como se describe en la sección 3.3.2. Dado que un estudio reciente utiliza grupos de tres y siete algoritmos para generar los clasificadores base [41], se han utilizado dos versiones de este algoritmo con dos grupos de clasificadores base. El primer grupo esta compuesto por C4.5, IB*k* y *Naive Bayes*, mientras que en el segundo grupo se agregan K*, *Decision Table*, MLR y *Kernel Density*.
- SCMLR (*StackingC*): *Stacking* con un número reducido de atributos del metanivel como se describe en la sección 3.3.2. Se han utilizado tres grupos de algoritmos para generar los clasificadores del nivel-base. En primera instancia, se ha usado el grupo utilizado por Seewald [117] (*Decision Table*, C4.5, *Naive Bayes*, *Kernel Density*, *MLR* y *K**). Además, se han utilizado los grupos de tres y siete algoritmos utilizados por Džeroski y Ženko [41] descritos en el punto anterior.
- SMLR: *Stacking* con regresión lineal multi-respuesta (MLR) propuesta por Ting y Witten [132]. Ver sección 3.3.2. Al igual que en los otros algoritmos basado en *Stacking*, se han utilizado los grupos de tres y siete algoritmos descritos anteriormente.

En un estudio reciente realizado por Džeroski y Ženko [41] se lleva a cabo una comparativa entre los métodos de construcción de conjuntos basados en *Stacking*. En este estudio se concluye que SMRMT tiene un mayor rendimiento que cualquiera de los otros métodos de construcción de conjuntos basados en *Stacking*.

Comparación de los Resultados

Con el propósito de estimar la tasa de precisión de los algoritmos usados se ha utilizado una validación cruzada estratificada de 10 carpetas. A fin de comparar los resultados obtenidos por *GA-Stacking* con cada uno de los algoritmos utilizados, se ha llevado a cabo el calculo de la mejora relativa que obtiene *GA-Stacking* sobre otro algoritmo en cada dominio. Además, se ha llevado a cabo un *t*-test pareado (1×10) con la finalidad de estimar si la mejora obtenida por *GA-Stacking* es estadísticamente significativa.

5.3.2. Resultados Experimentales

El porcentaje de aciertos que obtienen los diferentes algoritmos utilizados en estos experimentos se encuentra reflejado en la Tabla 5.13. Como se puede apreciar los resultados obtenidos por *GA-Stacking* son mejor en 4 de los 5 dominios utilizados. Por otra parte, con el propósito de comparar el rendimiento de *GA-Stacking* con cada uno de los algoritmos utilizados, en la Tabla 5.14 se muestra la mejora relativa en porcentaje que obtiene *GA-Stacking* al compararlo con los otros algoritmos de combinación y su significación estadística basado en un *t*-test de 1×10. Los resultados serán analizados en más detalle a continuación.

Comparaciones

En primera instancia se analizarán los resultados de *GA-Stacking* en comparación con los métodos de construcción de conjuntos homogéneos *Bagging* y *Boosting*. Como se puede apreciar en la Tabla 5.14, en cuatro de los cinco dominios utilizados, *GA-Stacking* es significativamente mejor que *Boosting*. En el caso de *Bagging*, *GA-Stacking* es mejor significativamente en tres de los cinco dominios. Por otra parte, la mejora relativa en ambos casos, supera el 58 %.

Al comparar *GA-Stacking* con los métodos de combinación por votos (VOTO) y la selección del mejor por validación cruzada (MEJORCV), se obtienen resultados distintos. Es decir, si se compara *GA-Stacking* con MEJORCV, en los cinco dominios utilizados, éste es mejor significativamente en todos los dominios, con una mejora relativa que ronda el 60 %. Sin embargo, al compararlo con el esquema de votación, las mejoras significativas se reducen a dos de los cinco dominios, pero la mejora relativa media sigue estando por arriba del 52 %.

Una vez analizados los resultados comparativos de los métodos de construcción de conjuntos homogéneos y otros métodos de combinación de clasificadores, se compara *GA-Stacking* con los otros métodos de construcción de conjuntos basados en *Stacking* (SMRMT, SCMLR, SMLR). En primera instancia, al comparar *GA-Stacking* con las dos versiones de SMRMT (tres y siete clasificadores base), se observa que el número de veces en que *GA-Stacking* es significativamente mejor varía de acuerdo al número de clasificadores base que formen el conjunto de clasificadores generados por SMRMT. *GA-Stacking* es significativamente mejor en un dominio más si compara con SMRMT7 en lugar de con SMRMT3. Por otra parte, la mejora relativa en comparación con ambas versiones ronda el 29 %. En cuanto a las comparaciones con las tres versiones de SCMLR (tres, cinco y siete clasificadores base), los dominios en que *GA-Stacking* es mejor significativamente, es similar en las versiones SCMLR5 y SCMLR7 (2+). Sin embargo, el número de dominios en que *GA-Stacking* es significativamente mejor se ve duplicado si se compara con la versión de sólo tres clasificadores base. Por otra parte, la mejora relativa media en todos los casos es superior al 50 %.

Por otra parte, si se comparan los resultados de *GA-Stacking* con los obtenidos por SMLR en sus dos versiones (tres y siete clasificadores base) la mejora relativa media al igual que en los resultados de SCMLR es superior al 50 %, pero además en tres de los cinco dominios utilizados, *GA-Stacking* es significativamente mejor que SMLR.

Estos resultados demuestran que el número de clasificadores base que formen parte del conjunto influye en el error que puede cometer éste al igual que el tipo de meta-clasificador que se utilice.

Cabe señalar, como se aprecia en la última fila de la Tabla 5.14, que en ninguno de los dominios *GA-Stacking* es significativamente peor que cualquiera de los algoritmos con los que se compara.

Tabla 5.13: Tasa de acierto (en %) de los métodos de construcción de conjuntos y combinación de clasificadores.

Dominio	GAS11CP	*Boosting*	*Bagging*	MEJORCV	VOTO	SMRMT3	SMRMT7	SCMLR5	SCMLR3	SCMLR7	SMLR3	SMLR7
australian	**87.10**	83.91	85.80	82.32	86.96	85.65	85.80	85.94	85.51	85.65	85,94	85,51
balance	**96.96**	79.21	82.25	86.88	86.88	92.97	92.50	89.92	90.56	89.92	89,92	89,28
car	**99.42**	96.18	93.29	93.75	94.85	99.13	98.96	95.02	93.92	95.02	94,73	95,49
diabetes	75.25	71.87	76.30	70.94	76.56	76.02	76.29	**77.07**	76.15	76.94	76,16	76,94
glass	**78.94**	73.33	72.90	70.50	76.62	68.18	73.74	76.08	70.11	76.56	66,75	76,06

Tabla 5.14: Mejora relativa en la precisión (en %) de *GA-Stacking* al compararlo con los otros métodos de generación y combinación de clasificadores. ($+/-$ significa mejor/peor,'.' significa que no hay diferencia significativa).

Dominio	*Boosting*	*Bagging*	MEJORCV	VOTO	SMRMT3	SMRMT7	SCMLR5	SCMLR3	SCMLR7	SMLR3	SMLR7
australian	19.82 +	9.18 .	27.05 +	1.11 .	10.10 .	9.18 +	8.25 .	11.00 +	10.10 .	8.25 .	11.00 +
balance	85.40 +	82.90 +	76.85 +	76.86 +	56.83 +	59.51 +	69.88 +	67.82 +	69.88 +	69.87 +	71.67 +
car	84.84 +	91.38 +	90.74 +	88.76 +	33.35 .	44.45 .	88.37 +	90.47 +	88.37 +	89.01 +	87.17 +
diabetes	12.00 +	-4.43 .	14.82 +	-5.58 .	-3.22 .	-4.37 .	-7.95 .	-3.77 .	-7.31 .	-3.80 .	-7.34 .
glass	21.02 .	22.28 +	28.61 +	9.91 .	33.81 +	19.80 +	11.95 .	29.54 +	10.16 .	36.65 +	12.03 .
MRM	58.48	59.52	60.58	52.39	29.29	29.76	50.22	54.29	50.28	54.28	50.23
gana/pierde	$4+/0-$	$3+/0-$	$5+/0-$	$2+/0-$	$2+/0-$	$3+/0-$	$2+/0-$	$4+0-$	$2+/0-$	$3+/0-$	$3+/0-$

Tabla 5.15: Número medio de clasificadores base en las soluciones encontradas por *GA-Stacking*.

Dominio	# de bases
australian	9.3
balance	9.4
car	9.5
diabetes	9.4
glass	9.6

Descripción de las Soluciones (Individuos)

Con la finalidad de observar la estructura de las configuraciones de *Stacking* obtenidas mediante *GA-Stacking*, se han analizados los mejores individuos de cada una de las carpetas de la validación cruzada estratificada. Es decir, de las tres ejecuciones de *GA-Stacking* llevadas a cabo en cada una de las carpetas de la validación cruzada, se ha analizado el mejor de los individuos de la ejecución con mayor valor en la función de *fitness*.

En la Tabla 5.15 se muestra la media del número de clasificadores base que poseen las soluciones. Como se puede ver, el número de clasificadores está entre 9 y 10 clasificadores base que es el número máximo que permite la configuración GAS11CP de *GA-Stacking*.

En cuanto a los algoritmos utilizados para generar los clasificadores base, en la Figura 5.11 se muestran el número de carpetas (superiores a seis) en las que aparece un algoritmo. Como se puede ver, en cada dominio existen entre tres y cuatro algoritmos que están presentes por lo menos en 7 de las 10 carpetas de la validación cruzada.

Por otra parte, en cuanto al clasificador del meta-nivel, en la Figura 5.12 se muestran los algoritmos utilizados para generar los clasificadores del meta-nivel en cada uno de los dominios utilizados. Como se puede apreciar, los mejores individuos de cada carpeta, tienden a utilizar el mismo algoritmo para generar el clasificador de nivel-1. Por ejemplo, tanto en el dominio de *balance* como en el de *car*, el algoritmo utilizado para generar el clasificador del meta-nivel en todas las carpetas es *Random Forest*. Esto indica que de acuerdo al dominio, la búsqueda realizada por los AG's tiende a converger en un mismo algoritmo para generar el clasificador de nivel-1.

Otros investigadores señalan que utilizar MLR [132, 117] o MRMT [40] para generar el clasificador del meta-nivel proporciona buenos resultados sin importar los clasificadores base. Sin embargo, los algoritmos utilizados por *GA-Stacking* para generar el clasificador del meta-nivel varían de acuerdo al dominio. Por ejemplo, *Naive Bayes* (8/10) para *australian*, *Random Forest* (10/10) para los dominios *car* y *balance*, MLR (8/10) en *diabetes* y IB*k* (7/10) en *glass*. Estos resultados muestran

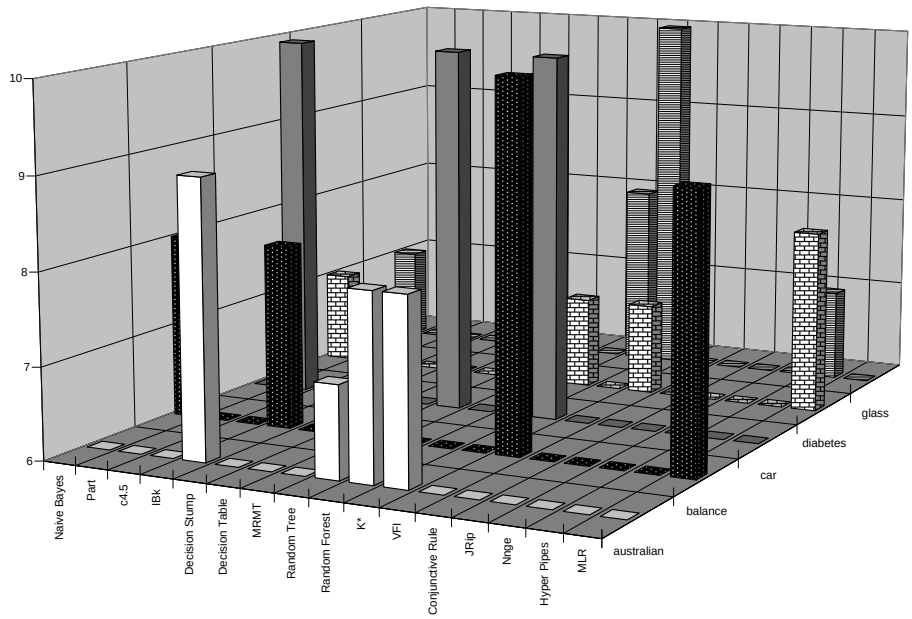


Figura 5.11: Número de carpetas (seis o más) en la validación cruzada en la que se utilizan los algoritmos para generar los clasificadores de nivel-base en cada uno de los dominios utilizados.

que además de las soluciones propuestas por otros investigadores, existen alternativas a los algoritmos propuestos, para el meta-nivel, que dependen del dominio.

Evolución del *Fitness*

Otro aspecto a tomar en cuenta en estos experimentos es el comportamiento de la función de *fitness*. En la Figura 5.13 se puede apreciar la evolución de la función de *fitness* para cada uno de los dominios utilizados. Los datos utilizados corresponden a la media del *fitness* observado en la validación cruzada. El *fitness* de cada carpeta es el promedio de las tres ejecuciones llevadas a cabo en dicha carpeta. El comportamiento del *fitness* es muy parecido en todos los dominios: se produce el mayor incremento en las primeras generaciones y luego se mantiene en constante crecimiento, incluso al llegar a la última generación. Esto indica que existe una evolución en las soluciones encontradas por *GA-Stacking*. Además, dado el constante incremento del *fitness*, es viable incrementar el número de generaciones con la finalidad de encontrar mejores individuos.

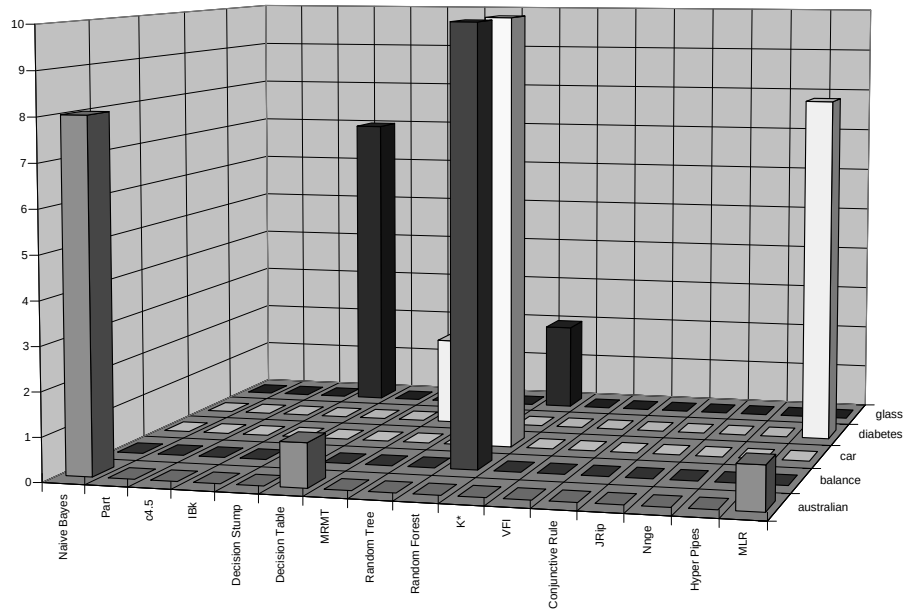


Figura 5.12: Número de carpetas en la validación cruzada en la que se utilizan los algoritmos para generar el clasificador del meta-nivel en cada uno de los dominios utilizados.

Discusión de los Resultados

Al igual que los algoritmos basados en *Stacking* utilizados son variantes de SMLR, *GA-Stacking* está basado en *Stacking* con distribuciones de probabilidades como datos del meta-nivel, lo mismo que SMLR. Seewald [117] presenta resultados experimentales en donde argumenta que SMLR se comporta peor en dominios multiclase (si se compara con dominios de dos clases). Seewald argumenta que este decremento puede ser causado por la dimensionalidad de los datos del meta-nivel y sugiere como mejora la reducción de esta dimensionalidad (lo que lleva a cabo SCMLR).

Recientemente, Džeroski y Ženko [41], argumentan que en situaciones con pocos clasificadores base, la utilización de la certeza de las predicciones prevalece. Esto es llevado a cabo mediante un algoritmo denominado SMLRE (ver sección 3.3.2). En sus resultados Džeroski y Ženko muestran que al incrementar el número de clasificadores base, la ventaja comparativa de SMLRE sobre SMLR, se ve reducida. Por último, y basado en el trabajo de Frank et al. [46] sobre la clasificación por regresión, señalan que *Stacking* con árboles de decisión multi-respuesta (SMRMT) presenta un mejor rendimiento que *Stacking* con regresión lineal multi-respuesta dado el uso de distribuciones de probabilidades en el meta-nivel.

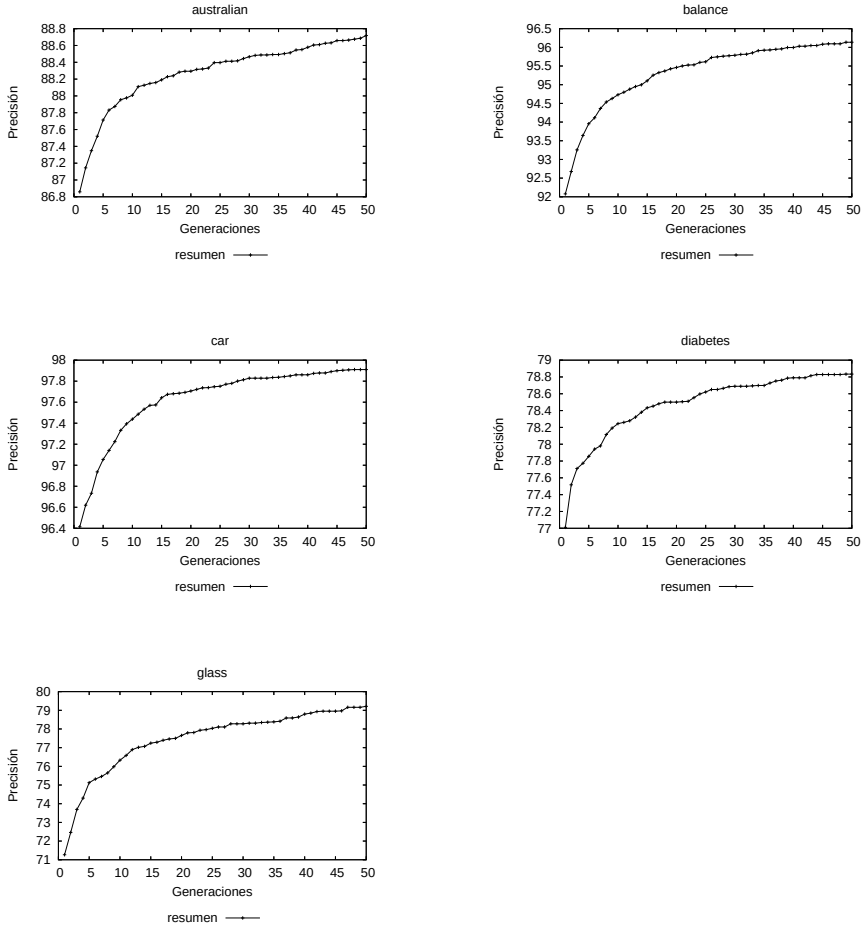


Figura 5.13: Evolución del *fitness* en los dominios utilizados.

Džeroski y Ženko [41] concluyen que SMRMT posee un mejor rendimiento que SMLR y que SCMLR. Señalan que la ventaja de SCMLR sobre SMLR es la reducción de la dimensionalidad de los datos del meta-nivel, pero que se deja a un lado información importante. Por esta razón, argumentan que, aparentemente, SMRMT es capaz de manejar el problema de la dimensionalidad sin dejar datos a un lado y haciendo uso de estos.

Como se ha visto en las secciones anteriores, *GA-Stacking* es capaz de encontrar configuraciones de *Stacking* con distribuciones de probabilidades en el meta-nivel, que en el 80 % de los dominios utilizados logra una mejora relativa sobre todos los métodos de construcción de conjuntos y combinación de clasificadores superior al 29 %. Y en un 40 % de los dominios la mejora es significativamente mejor si se compara con SCMLR y SMRMT. En ninguno de los dominios utiliza-

dos, cualquiera de los métodos utilizados son significativamente mejores que las soluciones encontradas por *GA-Stacking*.

Por otra parte, las soluciones propuestas por *GA-Stacking* tienden a converger en un mismo algoritmo para generar el clasificador del meta-nivel el cual puede variar de acuerdo al dominio. En resumen, qué y cuántos algoritmos tienen que ser utilizados para generar el conjunto de clasificadores depende en gran medida de los datos del dominio, por lo que fijar una configuración “a priori” puede llevar a obtener conjuntos de clasificadores sub-óptimos.

Capítulo 6

Conclusiones y Trabajos Futuros

En esta primera parte de la tesis se ha presentado un método que combina algoritmos genéticos y un algoritmo de generación de conjuntos de clasificadores heterogéneos con la finalidad de obtener la mejor configuración de clasificadores para un dominio dado (*GA-Stacking*). Como se ha visto, *Stacking* es una técnica de generación de conjuntos de clasificadores heterogéneos que utiliza dos niveles de aprendizaje. En el primer nivel de aprendizaje o nivel base se utilizan datos del dominio como entrada. Por otra parte, en el segundo nivel o meta-nivel, los datos son generados a partir de las predicciones de los clasificadores del nivel base. Al utilizar el conjunto generado por *Stacking* para clasificar un nuevo ejemplo, este es dado a los clasificadores del nivel base y basándose en las predicciones de éstos, el clasificador del meta-nivel determina la clase a la que pertenece el ejemplo. Un problema inherente a *Stacking* es determinar cuáles algoritmos deben ser utilizados para generar los clasificadores del primer nivel y qué algoritmo debe ser utilizado para generar el clasificador del segundo nivel. A pesar de que existen diversos estudios relacionados con la configuración de estos parámetros, no existe un consenso sobre los valores que deben tomar los mismos.

En esta tesis se propone un método basado en algoritmos genéticos para determinar la configuración óptima de los parámetros de *Stacking* para un dominio dado. Basándose en la variante de *Stacking* que utiliza distribuciones de probabilidades como datos del segundo nivel, *GA-Stacking* lleva a cabo una búsqueda en el espacio de combinaciones de algoritmos y sus parámetros de aprendizaje, con el propósito de determinar la configuración óptima de *Stacking* para un dominio dado.

A fin de estimar la influencia de los parámetros de aprendizaje asociados al método propuesto, se han realizado experimentos en dieciocho dominios con el fin de determinar la mejor configuración de estos parámetros.

Con el propósito de validar el método propuesto, se han llevado a cabo experimentos utilizando cinco dominios para medir su rendimiento frente a variantes de

Stacking y otros métodos de generación de conjuntos.

6.1. Conclusiones

Las principales conclusiones que se extraen al analizar los resultados obtenidos en la evaluación del método propuesto se exponen a continuación.

GA-Stacking posee, además de los parámetros inherentes a los algoritmos genéticos, tres parámetros adicionales: el número máximo de posibles clasificadores base, el número de algoritmos de aprendizaje disponibles y la opción de incluir los parámetros de aprendizaje de estos algoritmos en el proceso de búsqueda. Combinando estos parámetros se han evaluado seis configuraciones de *GA-Stacking*. Analizando los resultados obtenidos por las configuraciones evaluadas se demuestra que el ampliar el espacio de búsqueda, no implica que se obtendrá una mejora significativa en las soluciones encontradas. Por ejemplo, si sólo se incrementa el número de posibles clasificadores base, los resultados muestran que, a igual número de generaciones, se pueden obtener resultados inferiores a los obtenidos por configuraciones de *GA-Stacking* con menos clasificadores. Sin embargo, si además de incrementar el número de clasificadores, se incluyen los parámetros de aprendizaje de los algoritmos que los generan, los resultados mejoran significativamente.

Con el propósito de comparar *GA-Stacking* con otros algoritmos de generación de conjuntos, incluyendo las variantes más recientes de *Stacking*, se utilizó la versión de *GA-Stacking* cuyo espacio de búsqueda es mayor.

Los resultados empíricos demuestran que las soluciones que encuentra *GA-Stacking* generan conjuntos de clasificadores que al ser comparados con los métodos de generación de conjuntos homogéneos, *Bagging* y *Boosting*, muestran mejores resultados. De igual forma, si se compara con el mejor de los clasificadores generados por los algoritmos disponibles seleccionado por validación cruzada o con la combinación estos a través de votos, los resultados de *GA-Stacking* siguen siendo mejores.

Por otro lado, al comparar los resultados de *GA-Stacking* con las variantes de *Stacking* más recientes, los resultados varían de acuerdo a la configuración propia de cada algoritmo. Pero, en cualquier caso, *GA-Stacking* obtiene mejores resultados en la mayoría de los dominios utilizados.

Las principales diferencias de *GA-Stacking* con respecto a los trabajos previos relacionados con *Stacking* es que en éste no se seleccionan “a priori” algunos de los parámetros de *Stacking*. Por ejemplo, no se determinan con anterioridad a la construcción del conjunto de clasificadores parámetros como: qué algoritmo debe ser utilizado para generar el clasificador del meta-nivel, los parámetros de aprendizaje de este algoritmo, el número de clasificadores base, cuáles de los algoritmos disponibles utilizar para generar los clasificadores base, ni los parámetros de aprendizaje

de estos algoritmos.

La principal ventaja de *GA-Stacking* es su flexibilidad y el no fijar "a priori" de los parámetros de aprendizaje de *Stacking*. Este sistema es muy extensible. *GA-Stacking* se puede beneficiar de nuevos algoritmos de aprendizaje puesto que estos pueden ser fácilmente incorporados en el grupo de algoritmos disponibles, al igual que sus parámetros de aprendizaje, con cambios leves en la codificación del cromosoma. Otra ventaja de *GA-Stacking* es que las soluciones que encuentra son dependientes del dominio en el cual se aplica, como se puede ver en el análisis de las soluciones encontradas. De esta manera, *GA-Stacking* se adapta a los *bias* y atributos del dominio, mientras de los otros enfoques establecen la misma configuración, independientemente del dominio en el que se apliquen.

6.2. Limitaciones

La principal limitación de *GA-Stacking* es el recurso computacional que requiere para encontrar la configuración óptima de *Stacking* si se compara con el resto de los algoritmos que utilizan una configuración fija. Sin embargo, es importante señalar que, una vez encontrada la configuración y construido el conjunto de clasificadores, la eficiencia de éste es idéntica a la de un conjunto de clasificadores construido por otro método de generación de conjuntos de clasificadores, dependiendo, evidentemente, del número de clasificadores que formen parte del mismo.

6.3. Líneas de Investigación Futuras

El trabajo desarrollado en esta parte de la tesis plantea líneas de investigación que pueden ser estudiadas y desarrolladas en un futuro. Entre estas líneas se proponen las siguientes:

- Al incrementar la cantidad de algoritmos disponibles e incluir los parámetros de aprendizaje de éstos, el espacio de búsqueda se incrementa de manera notable. Al llevar a cabo la comparación de las diferentes configuraciones de *GA-Stacking*, se optó por seleccionar la versión que llevaba a cabo la búsqueda en el espacio más amplio. Los resultados demuestran que el valor de la función de *fitness* mantiene un constante crecimiento incluso al llegar al límite de generaciones fijado. Es necesario llevar a cabo un estudio con la finalidad de determinar el número adecuado de generaciones necesarias para lograr la convergencia de las soluciones en las distintas versiones de *GA-Stacking*.

- Otro factor a tomar en cuenta en futuros trabajos, es la influencia de la codificación de las soluciones utilizadas. Se plantea la utilización de una codi-

ficación diploide en donde se codifiquen por separado los algoritmos y sus parámetros de aprendizaje, con una tabla de dominancias, con el fin de hacer más flexible el proceso de aprendizaje.

- Dos de las variantes de *Stacking* estudiadas, SMRLE y SCMLR, llevan a cabo una ampliación o reducción de los datos del meta-nivel respectivamente. Estas variaciones pueden ser incluidas en *GA-Stacking* con la finalidad de determinar el tipo de datos de meta-nivel más adecuado a un dominio o configuración de *Stacking* dada.

- Al evaluar la función de *fitness*, *GA-Stacking* sólo toma en cuenta la precisión de un conjunto de clasificadores generado a partir de *Stacking*. Sin embargo, si dos conjuntos poseen la misma precisión, ambos tienen el mismo *fitness*, pero siguiendo el principio de la *navaja de Occam*, debería primar la solución más simple. Por esta razón se plantea modificar la función de *fitness* con la finalidad de encontrar configuraciones simples y precisas.

- Además de *Stacking*, existen otras técnicas de generación de conjuntos de clasificadores heterogéneos. Dados los resultados obtenidos al utilizar AG's en el proceso de optimización de los parámetros de *Stacking*, se plantea la posible aplicación de éstos a otras técnicas de generación de conjuntos.

Parte III

Modelado de Agentes mediante Aprendizaje Automático

Capítulo 7

Introducción

Un equipo de fútbol vuelve a jugar con un antiguo rival, aquel rival al que no ha podido ganar en los últimos años, pero esta vez el nuevo entrenador emplea la tecnología como parte del entrenamiento de su equipo y hace que sus jugadores observen una y otra vez una serie de vídeos de partidos anteriores del equipo contrario. Los defensas observan detalladamente las jugadas ensayadas de los delanteros rivales y los delanteros hacen lo propio con relación a la posición y distribución de la defensa rival. Lo que el entrenador busca es que su equipo *aprenda* de lo sucedido en el pasado, de tal forma que pueda utilizar este conocimiento en busca de mejor rendimiento. Una vez llegado el gran día, el equipo está preparado y da comienzo el partido pero el equipo contrario está alineando delanteros no habituales y que no juegan como juegan los habituales, pero la defensa del equipo está preparada y observa cuidadosamente el comportamiento del rival y utiliza esta información para evitar que los delanteros profundicen por las bandas y puedan penetrar en su área.

Un comando de elite del ejercito está realizando un reconocimiento aéreo del área enemiga y el comandante en jefe da la orden de silencio absoluto, no puede haber ninguna comunicación entre los miembros del equipo para evitar ser detectados por el enemigo. La experiencia y los conocimientos de los miembros del equipo pasan a ser de vital importancia para llevar a cabo la misión. Cada uno de los miembros del equipo puede utilizar la información de su radar para detectar los movimientos de los demás miembros del equipo y es capaz de interpretar los mismos de tal manera que se lleve a cabo una estrategia colectiva y el área enemiga sea reconocida sin ninguna baja.

Los ejemplos que se han mencionado pueden corresponder a situaciones de la vida real en donde el conocimiento que se posea de los miembros del equipo contrario, en el caso de situaciones competitivas, o de los otros miembros del equipo del que forma parte, en caso de situación colaborativa, proporcionan información útil a la hora de tomar una decisión y de llevar a cabo una acción.

El incremento en la investigación en la última década en lo referente a los agentes y sistemas multiagentes ha propiciado que existan ambientes simulados en donde los agentes se enfrentan a situaciones de la vida real como las descritas con anterioridad. Recientemente, una de las áreas de investigación que más auge ha tenido en cuanto a lo que agentes se refiere, es la capacidad de un agente de obtener información del oponente o de un compañero e intentar saber cuál es el comportamiento que está llevando a cabo, es decir, *modelar a los otros agentes*.

Según Kitano et al. [81] el modelado de agentes se puede definir como el modelado y razonamiento acerca de las metas, planes, conocimientos, capacidades o emociones de otro agente. Con tal fin se han utilizado una variedad de técnicas para obtener el modelo de un agente, entre las que se pueden mencionar las que han utilizado técnicas recursivas [55, 58], autómatas finitos [17] y técnicas probabilísticas [126], entre otras.

Otra manera de modelar el comportamiento de un agente es considerarlo como una caja negra e intentar modelar su comportamiento en términos de la relación existente entre sus entradas y salidas. Este tipo de técnicas denominadas IOAM (del inglés *Input-Output Agent Modeling*), han sido utilizadas con éxito en el modelado de usuarios, campo relacionado con el modelado de agentes.

En esta tesis doctoral se utiliza el enfoque que plantean las técnicas IOAM. Es decir, que los agentes se pueden modelar a partir de sus entradas y salidas. En primera instancia, se plantea la utilización de técnicas de aprendizaje automático con la finalidad de construir modelos de los agentes que interactúan con el agente que construye el modelo. A diferencia de otros enfoques IOAM utilizados en el área de modelado de usuarios, se pretende aplicar el enfoque propuesto a dominios dinámicos y complejos con la finalidad de identificar el tipo de técnicas necesarias para construir el modelo del agente.

Por otro lado, en la mayoría de los entornos en donde intervienen agentes no se puede tener acceso directo a las entradas y salidas del agente a modelar. Esta situación plantea un problema al enfoque basado en las entradas y salidas del agente. Por esta razón, se propone un nuevo enfoque que utiliza técnicas de aprendizaje automático para superar esta dificultad. Además, se propone utilizar técnicas de aprendizaje automático en la fase de utilización del modelo.

A fin de introducir al lector en el tema tratado en esta segunda parte de la tesis doctoral, en el capítulo 8 se da una panorámica general del estado del arte en cuanto a modelado de agentes se refiere. La propuesta para llevar a cabo la tarea de modelar un agente utilizando técnicas de aprendizaje automático se describe en el capítulo 9. En el capítulo 10 muestran los resultados obtenidos al evaluar la propuesta presentada. Finalmente, en el capítulo 12 se presentan las conclusiones y trabajos futuros.

Capítulo 8

Estado del Arte

En este capítulo se da una perspectiva global de las investigaciones relacionadas con el segundo tema abordado en esta tesis doctoral, el modelado de agentes.

La revisión de la literatura comienza con la definición del concepto de *agente*. A continuación se presentan las investigaciones relacionadas con el área de modelado de agentes, desde el enfoque clásico, representado por la teoría de juegos hasta el reconocimiento de planes.

8.1. ¿Qué es un Agente?

A pesar del creciente número de investigaciones que se han realizado y se están realizando dentro del área de agentes en los últimos años, no existe una definición aceptada por todos los implicados en lo referente al término “agente”. Existe una gran cantidad de definiciones de lo que es un agente, y en lo que todas estas definiciones parecen estar de acuerdo es en el concepto de autonomía (i.e. que puede actuar independiente de otros). Las demás cualidades que debe tener un agente dependen del entorno en donde se defina el concepto. Una de las definiciones más aceptadas es la de Wooldridge [146] en donde establece como requisitos fundamentales que debe poseer el agente, su autonomía, sociabilidad, capacidad para reaccionar y para tomar la iniciativa. Además, agrega otras características menos convencionales a los agentes como movilidad, benevolencia, veracidad y racionalidad. En [70, 147], se puede encontrar un compendio de definiciones y teorías de agentes.

8.2. Modelado de Agentes

En dominios competitivos, el conocer datos acerca del conocimiento que posee un oponente u oponentes proporciona una clara ventaja, dado que se puede utilizar esta información para predecir, de alguna manera, lo que va a hacer el oponente y actuar en función de dicha predicción [137]. De igual forma, en dominios colaborativos, el poseer información acerca de un compañero adquirida mediante la observación del comportamiento del mismo, sin tener que recurrir a una comunicación explícita, conlleva un ahorro de recursos y permite utilizar los conocimientos adquiridos en el momento de tomar una decisión de cómo actuar [127].

La tarea de modelar un agente se puede definir como *el modelado y razonamiento acerca de las metas, planes, conocimientos, capacidades o emociones de otro agente* [81].

Recientemente, las investigaciones sobre modelado de agente se han visto incrementadas especialmente desde el punto de vista de los Sistemas Multiagentes [122, 78] y la Interacción Hombre-Máquina debido sobre todo a la utilidad de poseer un modelo de los agentes con que se interactúa (agente o humano) [9].

La utilización de un modelo del oponente no es algo nuevo. Por ejemplo, en Teoría de Juegos, el objetivo es poder jugar óptimamente contra un oponente. En un principio los enfoques se centraban en la mecánica del juego pero después se comprendió que la utilización del conocimiento acerca del oponente podía incrementar la eficacia del sistema. Es por eso que se puede considerar a la Teoría de Juegos como la base del modelado de agentes. Otros conceptos que se asocian con el modelado de agentes son: el reconocimiento de planes, reconocimiento de comportamiento, modelado de usuario, monitorización de agentes, seguimiento de agentes y coordinación basada en observación.

Las investigaciones dentro del área de modelado de agentes se han desarrollado en una gran variedad de dominios tales como modelado de oponentes en situaciones competitivas, detección de fallos, monitorización de aplicaciones distribuidas y modelado de usuario, entre otras. También son muchas las técnicas que se han utilizado para abordar este tema como son el aprendizaje automático, razonamiento bayesiano, diagnóstico, etc. En las siguientes secciones se hace una revisión de los distintos modos de abordar este problema.

8.2.1. Enfoque clásico - Teoría de Juegos

En términos generales, la *Teoría de Juegos* examina el comportamiento estratégico para la toma de decisiones por parte de los participantes en situaciones de conflicto.

La Teoría de Juegos, desde el punto de vista clásico, se basa en la construcción

de una matriz que permite entender el conflicto entre las partes involucradas y sus posibles soluciones. En el enfoque clásico, se asume que un jugador posee un conjunto de las posibles acciones que puede realizar y que en cada jugada conoce dicho conjunto, al igual que conoce el conjunto de acciones que puede realizar el oponente. De la misma manera se asume que el jugador conoce la *utilidad* propia y la del contrario, que pueden recibir al llevar a cabo una jugada. También se asume que los jugadores son racionales, es decir, que buscan el máximo beneficio propio (utilidad).

Un ejemplo de técnicas utilizadas en Teoría de Juegos para juegos con oponentes es el algoritmo de búsqueda *minimax*. Minimax asume que se conocen las posibles acciones que puede llevar a cabo el oponente al igual que se supone que dicho contrincante está actuando de manera óptima. En otras palabras, se asume que el oponente (AGENTE A) actúa para maximizar sus beneficios lo cual, probablemente, sea lo peor para el modelador (AGENTE B).

Sin embargo, minimax es un algoritmo diseñado para dominios con contrincantes (por turno), información completa y acciones discretas. Por el contrario, el objetivo de esta tesis doctoral es la creación de un marco que permita el modelado de agentes en entornos con información incompleta, ruidosa, con un espacio continuo de acciones y en donde no hay turnos. Por otra parte, según [74] en juegos complejos un agente puede actuar de manera sub-óptima puesto que es muy difícil o imposible encontrar una estrategia óptima.

Otra de las suposiciones que se encuentran detrás del algoritmo minimax es que no se posee información acerca del proceso de toma de decisiones del oponente. Por esta razón Carmel y Markovitch [16] proponen una generalización del mismo para que incorpore modelos de oponentes en la búsqueda. En dicho trabajo se asume que existe un conjunto fijo de modelos que abarcan a todos los posibles oponentes. Carmel y Markovitch definen el modelo del oponente como una estructura recursiva que está formada por la función de evaluación del oponente y su modelo de jugador. Al seleccionar una acción que llevar a cabo, el algoritmo $M*$ genera los sucesores del estado actual, luego aplica el modelo del oponente sobre cada sucesor para obtener la respuesta del oponente. Por último, evalúa cada una de estas respuestas aplicando el algoritmo de manera recursiva (con una profundidad definida).

Al igual que el minimax, este algoritmo está diseñado para juegos de turnos y no se pueden aplicar directamente a sistemas multiagente en donde se interactúe por turnos. Por otra parte, la utilización de este algoritmo en sistemas en donde el número de acciones posibles a realizar no sea discreto, hace que la comprobación recursiva de todos los posibles estados no sea factible. Por otra parte, puede resultar difícil la creación de una función de evaluación eficiente y precisa.

En un trabajo posterior [17], Carmel y Markovitch presentan un método donde el modelo del oponente es inferido basándose en su comportamiento previo (en-

trada/salidas). Este modelo es representado como un autómata finito determinista (AFD). Una vez que se posee el modelo, se utiliza éste (predicciones) para diseñar la estrategia que maximice la recompensa del agente modelador en un entorno de juego repetitivo de dos jugadores. Puesto que el AFD utiliza una tabla de observaciones para mantener un modelo consistente con el comportamiento del oponente, este enfoque se ve limitado a dominios discretos. De igual forma posee una alta sensibilidad al ruido. Carmel y Markovitch limitan las estrategias del oponente a estrategias que puedan ser modeladas con un AFD.

8.2.2. Modelos de Usuarios

Otra área de estudio relacionada con el modelado de agentes es el modelado de usuario o *user modeling*. En principio este campo era designado con el nombre de modelado de estudiantes, pero debido al el incremento del comercio electrónico y las aplicaciones en la *world-wide-web*, las investigaciones dentro del área de recuperación de la información se han visto incrementadas, lo que ha provocado el cambio de denominación [143].

Se define modelo de estudiante como una aproximación, posiblemente parcial, de una representación cualitativa del conocimiento del estudiante sobre un dominio, tema o característica del dominio particular, teniendo en cuenta total o parcialmente los aspectos específicos del comportamiento del estudiante [119].

Entre los distintos enfoques utilizados para encarar el problema del modelado de usuarios se encuentra el aprendizaje automático. Según Webb [143], dependiendo del propósito para el cual se desee adquirir el modelo del usuario, éste varía en su formación. Según Webb, los modelos de usuarios pueden buscar describir:

- los procesos cognitivos detrás de las acciones del usuario
- las diferencias entre las habilidades del usuario y las habilidades del experto
- los patrones de comportamiento del usuario o preferencias del usuario, o
- características del usuario.

La mayor parte de las aplicaciones del aprendizaje automático al modelado de usuarios se centran en los dos primeros puntos. Sin embargo en [142] Webb presenta un paradigma para el modelado basado-en-característica (*Feature-Based Modeling*- FBM) que se centra en la adquisición de patrones de comportamiento del usuario y no en intentar modelar el proceso cognitivo subyacente. El trabajo de Webb es un ejemplo del tipo de técnicas denominadas IOAM (*Input-Output Agent Modeling*). Este tipo de técnicas han sido utilizadas con éxito en el modelado de estudiantes. Por ejemplo el modelado basado en la relación (*Relational Based Modeling* - RBM) [86], el FBM y el C4.5-IOAM [21] han demostrado una alta

precisión en la predicción en el dominio de la resta elemental. Tanto el FBM como el RBM utilizan métodos de inducción diseñados para esta tarea, mientras que C4.5-IOAM utiliza C4.5 [110] como mecanismo de inducción.

En esta tesis se utilizan técnicas de aprendizaje automático, entre ellas C4.5, con representación atributo-valor, en donde los atributos son las características de las acciones y del contexto de las tareas. Es decir, el modelo adquirido posee relaciones del tipo $X \rightarrow a$, en donde X es un conjunto de características del entorno y a es una acción simple. El presente trabajo utiliza este enfoque a la hora de adquirir el modelo de otro agente, aunque se aplica a otro tipo de dominios (dinámicos, continuos, ruidosos, etc).

8.2.3. Reconocimiento de Planes

El reconocimiento de planes, como su nombre indica, mantiene una relación con la planificación tradicional en Inteligencia Artificial. Sin embargo, a diferencia de la planificación tradicional, en la cual la tarea principal es la generación de una serie de pasos a seguir para llegar a una meta y la ejecución de dicho plan consiste en la aplicación secuencial de las acciones planificadas, en el reconocimiento de planes la tarea principal es la inferencia del plan o planes que está siguiendo un agente a partir de la observación de sus acciones.

Existe una gran cantidad de investigaciones realizadas en el área de reconocimiento de planes en las últimas dos décadas. Por ejemplo, a principios de los años 80 Cohen et al. [24] distinguen entre dos clases de reconocimiento de planes, los denominados *"Key hole"* y los intencionados. Kautz y Allen [80] definen el reconocimiento de planes como el problema de identificar un conjunto mínimo de acciones de alto nivel suficientes para explicar el conjunto de acciones observadas. Esta investigación es considerada como la base de una gran parte de los trabajos de reconocimiento de planes [60].

Pollack [104] hace una formalización lógica del reconocimiento de planes en situaciones en donde los agentes pueden construir planes inválidos. Además, argumenta que un plan puede ser visto, no como una fórmula estática que determina una acción, sino como actitudes mentales complejas.

En [136] van Beek y Cohen estudian el reconocimiento de planes en sistemas pregunta-respuesta, de tal forma que este reconocimiento del plan, detrás de las preguntas del usuario, permita generar una respuesta apropiada. Plantean la necesidad de resolver o no la ambigüedad utilizando la comunicación con el usuario.

Charniak y Goldman [19] proponen un enfoque probabilístico general para el reconocimiento de planes que puede explícitamente razonar sobre la incertidumbre en sistemas multiagente. Este trabajo está relacionado con [68], en donde Huber et al. describen métodos que trasladan los posibles planes que puede ejecutar otro agente generados por un planificador a redes de creencias probabilísticas para

respaldar la tarea de reconocimiento de planes. En [67], Huber utiliza el enfoque anterior aplicándolo a la monitorización del trabajo de equipos en un dominio militar de reconocimiento, incentivado, sobre todo, por lo poco fiable y costoso de las comunicaciones en este dominio. En [69], Huber estudia el uso del reconocimiento probabilístico de planes aplicados a la coordinación por observación en el dominio Netrek [1]. Huber demuestra que los agentes que utilizan el reconocimiento de planes para coordinarse superan a los agentes que utilizan comunicación para coordinarse.

Devaney y Ram [29] realizan una combinación de reconocimiento de patrones, reconocimiento de planes y seguimiento de objetos para el reconocimiento de tácticas militares durante batallas de entrenamiento. De igual forma, Intille y Bobick [73] presentan un marco probabilístico para reconocer jugadas de fútbol americano. Las jugadas están descritas en forma de metas para los agentes y limitadas por restricciones temporales entre las acciones. Al igual que Charniak y Goldman [19] y Huber [68], utilizan redes de creencias para incorporar las observaciones. Ambas investigaciones utilizan datos generados a partir de agentes humanos.

Washington [139] propone una variante del reconocimiento de planes en la coordinación con otros agentes o procesos representados como Procesos de Decisión de Markov Parcialmente Observables (POMDP). Washington se enfoca sobre todo en hacer este proceso computacionalmente tratable. Cabe señalar que en este trabajo el agente que monitoriza no ejerce ninguna influencia sobre el agente monitorizado.

Recientemente Goldman et al. [60] presentaron un marco probabilístico para el reconocimiento de planes basado en la ejecución de los planes y no en la concepción de los planes como objeto formal.

Tambe [128] presenta un algoritmo para el "seguimiento de agentes" en entornos flexibles y reactivos (*RESC: REal-time Situated Commitments*). Mediante *RESC* un agente puede llevar a cabo el seguimiento de otro agente infiriendo una jerarquía de operadores (del agente modelado) aprovechando su propia arquitectura. De esta forma ejecuta el modelo del otro agente y compara las predicciones generadas por el modelo con las acciones del otro agente para verificar posibles fallos. Además, *RESC* utiliza una técnica de retroceso para recuperarse de los fallos que se puedan dar por la presencia de ambigüedades. Los agentes se desarrollan utilizando la arquitectura *SOAR* [88]. Esta técnica está basada en lo que se conoce como reconocimiento de planes reactivos y elaborado para funcionar en tiempo real y con aplicación práctica en dominios militares con información parcialmente observable. Algunos autores no consideran esta investigación dentro del área de reconocimiento de planes porque no se ciñe estrictamente a la definición dada por Kautz y Allen [80]. En cambio otros autores la consideran como reconocimiento

[1] Netrek es un juego de simulación de batalla multijugador basado en Start Trek

de planes reactivos.

En [127] Tambe amplía el trabajo realizado en [128] con el fin de reconocer planes ejecutados por equipos, como equipos y no como agentes individuales. *RESC*$_{team}$ está basado en *RESC*, pero incluye mejoras para afrontar la tarea de formación de sub-equipos, asignación de roles y desviación de sub-equipos. Los modelos de equipo que utiliza están basados en el *Marco de Intenciones Conjuntas* [23]. *RESC*$_{team}$ se enfoca explícitamente en la explotación del razonamiento del trabajo en equipo. Kaminka et al. [79], desarrollan un algoritmo de reconocimiento de planes reactivos llamado *RESL* (*REal-time Situated Least-commitment*). La representación que utiliza *RESL* es similar a la que utiliza *RESC*$_{team}$ con la diferencia de que cuando *RESC*$_{team}$ se utiliza en entornos no colaborativos utiliza una heurística del peor-coste. De esta forma sólo razona sobre las hipótesis que implican el mayor costo para el agente que realiza la monitorización. En cambio, *RESL* permite la representación de varias hipótesis y la utilización de distintos métodos de eliminar la ambigüedad.

Las investigaciones sobre reconocimiento de planes citadas hasta este punto asumen que el agente modelador posee cierto conocimiento acerca de los planes que puede ejecutar el agente modelado, i.e. librería de planes. La tarea del agente entonces es identificar cuál de estos planes es el que está siguiendo el otro agente.

Por otra parte, Kaminka et al. [78], proponen técnicas para convertir entradas dinámicas, complejas, continuas y multivariadas correspondientes a los estados del mundo, en series temporales de comportamientos atómicos reconocidos que luego son analizados para encontrar subsecuencias repetidas de eventos que caractericen el comportamiento de un equipo. Para llevar a cabo esto, se utilizan "reconocedores" que interpretan las entradas de los sensores en busca de "eventos". Una vez almacenados los eventos en un *trie* [82] utiliza técnicas estadísticas que permitan realizar un análisis estadístico de los eventos en busca de secuencias de comportamiento.

8.2.4. Otros Enfoques en Sistemas Multiagentes

Además de los métodos utilizados para el modelado de agentes que se han mencionado, basados en la teoría de juegos y el reconocimiento de planes, existen otro grupo de trabajos relacionados con el modelado de agentes propiamente dicho.

Entre estos métodos se puede señalar el Método de Modelado Recursivo (*Recursive Modeling Method - RMM*) [38, 39, 55, 56, 57, 58, 101] muy relacionado con la teoría de juegos. Mediante *RMM*, un agente puede representar y utilizar el conocimiento que posee sobre sus pagos esperados al realizar una acción y el pago que reciben los demás mediante las matrices de pago. De esta manera un agente puede modelar el estado interno de otro agente y sus estrategias a la hora de llevar a cabo una acción. Este método es recursivo porque un agente A puede modelar a

otro agente B y éste a su vez modelar al agente A y el agente A a su vez modelar lo que cree que el agente B sabe de él y así sucesivamente (modelos anidados). Sólo se deja de modelar cuando se alcanza el conocimiento cero, es decir cuando no se posee información del agente que está siendo modelado. Esta estrategia puede llegar a conducir a jerarquías (matrices de pago) muy profundas y costosas de analizar. Vidal y Durfee [137] y Durfee [37] proponen técnicas para limitar la profundidad de la jerarquía de matrices.

Un problema del enfoque del *RMM* es que asume que conoce el estado interno de los otros agentes, al igual que sus posibles acciones con el fin de construir la matriz de pago. Esta suposición es poco realista puesto que en entornos reales se tiene poca o ninguna información del estado interno de los otros agentes, problema que se busca abordar en esta tesis doctoral.

A pesar de que las matrices de pago utilizadas en *RMM* resumen la información contenida en los diagrama de influencia [101], Suryadi y Gmytrasiewicz [126] utilizan éstos para representar los modelos de los agentes indicando que los diagramas de influencia [65] brindan una mejor percepción del problema de aprendizaje. Mediante estos modelos un agente puede interactuar con otros agentes y predecir su comportamiento. Presentan un marco para crear modelos de agentes basados en sus capacidades, creencias y preferencias. Parten de modelos previos, representados como diagramas de influencia, y de un historial del comportamiento de un agente. Se pueden construir nuevos modelos ajustando los parámetros del diagrama de influencia que representa el modelo.

Hu y Wellman [66] llevan a cabo una serie de experimentos en donde crean modelos de otros agentes utilizando métodos de regresión. Realizan las predicciones de dos maneras. Mediante la primera sólo se toma en cuenta el comportamiento previo del otro agente y se modela éste utilizando series temporales sin intentar modelar el proceso de decisión del agente. En el segundo caso, se asume que el otro agente está intentando maximizar su recompensa, de tal forma que existe una relación funcional entre las acciones de los agentes y sus estados internos. Este método posee niveles de recursividad atendiendo al tipo de forma funcional que se asuma. Este trabajo considera la versión “on-line” del modelado de agentes y utiliza como dominio de prueba un sistema de subastas (doble). Hu y Wellman asumen que todos los estados del otro agente son observables y que cada agente conoce su función de pago.

Garrido et al. [53] llevan a cabo un estudio empírico cuyo fin es cuantificar los beneficios que puede obtener un agente al modelar a otro. Para esto utilizan como dominio de aplicación el Juego de Asignación de Reuniones (*Meeting Scheduling Game*) [52]. Utilizan distintos tipos de estrategias, desde una estrategia aleatoria en donde el agente no utiliza información acerca de los otros agentes, hasta estrategia de oráculo en donde el agente supone correctamente toda la información acerca del otro agente. Adicionalmente, crean estrategias que van decrementando el conocimiento que posee el agente acerca del otro agente e incrementan la capa-

cidad de modelado utilizando modelos probabilísticos. En [54] amplían el trabajo arriba mencionado e introducen un agente capaz de modelar a otros agentes utilizando modelos probabilísticos y capaz de actualizar dichos modelos de manera incremental e iterativa mediante un mecanismo bayesiano.

Quizás uno de los dominios de prueba más utilizados, en cuanto a agentes se refiere, es el simulador de fútbol de la *RoboCup* [81]. Dentro de las competiciones que se llevan a cabo en la *RoboCup*, existen categorías que incluyen agentes físicos y otras que se desarrollan en entornos simulados. Uno de estos entornos simulados está basado en un sistema cliente/servidor denominado *Soccer Server System* [100]. Basados en este servidor, existen actualmente tres competiciones, la liga 2D, la liga 3D y la competición de entrenadores (*coach*). Las ligas 2D y 3D son partidos de fútbol simulados en donde interactúan 22 agentes, 11 por equipo, simulando un partido de fútbol en dos y tres dimensiones respectivamente. Por otra parte, en la liga de *coach*, los 22 agentes que interactúan son estándar y la competición consiste en crear un agente *entrenador* que dé consejos al equipo que entrena con la finalidad de aprovechar la visión del partido que éste posee. Dado que en los últimos años las ligas de simulación se han convertido en un conocido *testbed* en el área de agentes, han sido desarrolladas una serie de investigaciones relacionadas con el modelado de agentes en estos dominios. Cabe destacar que la mayoría de los trabajos relacionados con el modelado de agentes dentro de la liga de simulación se basan en la utilización del agente *coach* o entrenador. Ejemplos de estos trabajos se detallan a continuación.

Stone et al., [124] proponen *IMBBOP* (*Ideal Model Based Behavior Outcome Prediction*). Esta técnica predice las acciones de otro agente (compañero u oponente) en relación con el comportamiento ideal del agente en una situación dada. Es decir, *IMBBOP* no asume que el agente esté llevando a cabo las acciones definidas como "óptimas" para el agente, sino que describe su comportamiento esperado como una desviación de este óptimo. Este comportamiento ideal es independiente del agente. De esta forma, éste puede ser calculado basándose sólo en un modelo de la dinámica del entorno (dominio). A excepción de los trabajos que se detallan posteriormente, Stone lleva a cabo un modelado de bajo-nivel (agente-agente). Cabe señalar que asumir que el contrario está actuando de manera óptima podría llevar a conclusiones erróneas.

En [34], Druecker et al. utilizan una red de neuronas artificiales para clasificar los posibles tipos de formación del equipo contrario con el fin de comunicar a los agentes una *contra-formación*. En esta tesis doctoral se busca realizar un modelado a bajo nivel (agente-agente) y no reconocer formaciones del oponente, aunque esta información pueda servir como información adicional a la hora de resolver ambigüedades.

En [112], Riley asume que se conocen *"a priori"* un número de "clases de adversarios" e intenta clasificar el comportamiento del equipo oponente basándose en la equiparación de los datos de los sensores con las clases predefinidas.

Recientemente, Riley y Veloso [113], presentaron *ATAC* (*Adaptive Team - Adversarial Coaching*) en donde el agente *coach* (entrenador) genera planes *on-line* en forma de "Redes Temporales Simples" [26] basadas en el reconocimiento de los planes del oponente y luego los comunica a sus compañeros de equipo para que lleven a cabo el plan de manera distribuida. El agente *coach* posee *"a priori"* un conjunto de modelos del oponente los cuales son representaciones probabilísticas de las posiciones de los oponentes. A la hora de seleccionar un modelo utilizan Naive Bayes [77].

Basándose en [112], Steffens [122] presenta un marco llamado *FBDOM* (*Feature Based Declarative Opponent Modeling*) para el modelado de oponentes en sistemas multiagente. Este sistema asume que un agente puede identificar ciertas características en el oponente que describen su comportamiento. Dicho comportamiento puede ser formalizado utilizando una extensión del lenguaje que utiliza el agente "entrenador" para comunicarse con el resto de los jugadores en el simulador. De esta forma, cuando se observa el comportamiento de un equipo, se busca emparejar este comportamiento con alguno de los modelos de oponentes que se poseen *"a priori"*.

8.3. Conclusiones

Se ha presentado una serie de investigaciones relacionadas con el modelado de agentes aplicadas a una amplia gama de dominios. Algunos de estos trabajos asumen la existencia de información completa o alguna suposición de conocimiento interno de los otros agentes. En esta tesis doctoral se busca la creación del modelo de otros agentes basándose sólo en la observación de su comportamiento. En cuanto a las investigaciones que han utilizado como dominio de prueba el simulador de fútbol de la *RoboCup*, uno de los dominios de prueba de esta tesis, una gran parte de éstas se centra en el modelado a alto-nivel (equipos). Esta tesis doctoral tiene por objetivo el modelado de agentes a bajo-nivel, es decir, de agente a agente. Por ejemplo, un delantero podría aprender a predecir si el portero va a salir en busca o no de la pelota, y de esta manera actuar en consecuencia.

Capítulo 9

Modelado de Agentes

Se supone que el comportamiento de un agente puede ser descrito en términos de sus entradas y salidas. Si se piensa en todas las posibles entradas, éstas pueden ser representadas como un grupo de parámetros de entrada. Por lo tanto, hay una clara analogía con la tarea de clasificación en la cual cada parámetro de entrada del agente puede ser representado como un atributo que puede tener tantos valores como el correspondiente parámetro. En relación con la salida, puesto que se han seleccionado un grupo de tareas que poseen la característica de generar decisiones en *un paso* (soluciones que no requieren de un conjunto de pasos, como en planificación), se puede pensar en éstas como salidas atómicas. En términos de una tarea de clasificación, esto permite definir una clase por cada posible salida. De esta manera, la tarea de modelado se convierte en una tarea de clasificación. Por ejemplo, en el simulador de fútbol (sistema cliente-servidor) de la *RoboCup*, uno de los dominios seleccionados para probar la viabilidad de esta propuesta, las entradas del agente son los datos que recibe a través de sus sensores de cuerpo, auditivo y de visión (enviados por el servidor) y la salida son las posibles acciones que puede llevar a cabo (enviándolas al servidor).

Una vez se ha determinado la tarea de clasificación, cualquier técnica de clasificación puede ser empleada para resolver esta tarea: aprendizaje basado en instancias [1], árboles de decisión [105], aprendizaje de reglas [94, 107], o redes de neuronas [114]. Sin embargo, además de obtener el modelo de un agente, se quiere que este modelo sea relativamente fácil de entender y depurar para lo cual se necesita utilizar algoritmos que utilicen una representación declarativa. Por esta razón, se propone utilizar técnicas de aprendizaje automático como árboles de decisión o generadores de reglas.

En la realización de esta tesis doctoral se han tomando dos vertientes. En la primera, a la cual se le ha denominado *Modelado de Agentes Basado en Trazas* (MABT), se presupone que el agente modelador tiene acceso directo a las entradas y salidas del agente a modelar, mas no así a su estructura interna. Es decir, se

poseen datos del comportamiento del agente a modelar generados en interacciones previas. La segunda de las vertientes desarrolladas en esta tesis, aborda la tarea de modelado en situaciones más complejas, en donde el agente modelador no tiene acceso directo a las entradas y salidas del agente a modelar. Por esta razón, el modelado debe realizarse en dos fases, la observación y recolección de datos y, el modelado en sí. El esquema propuesto para resolver esta tarea ha sido denominado *Modelado de Agentes Basado en la Observación* (MABO).

9.1. Modelado de Agentes Basado en Trazas (MABT)

Una suposición inherente a MABT es el acceso directo a las entradas y salidas del agente a modelar. Es decir, el agente modelador, AGENTE B, tiene acceso a los pares entrada/salida generados por el AGENTE A en situaciones pasadas.

En la Figura 9.1 se muestra el marco general del Modelado de Agentes Basado en Trazas. Como se puede apreciar, el proceso de modelado requiere una etapa previa en donde se registran las interacciones del AGENTE A con el entorno y con el propio AGENTE B. En esta etapa se forman los pares entrada/salida en donde la entradas son los datos recibidos por los sensores del AGENTE A y la salida es la acción llevada a cabo por éste.

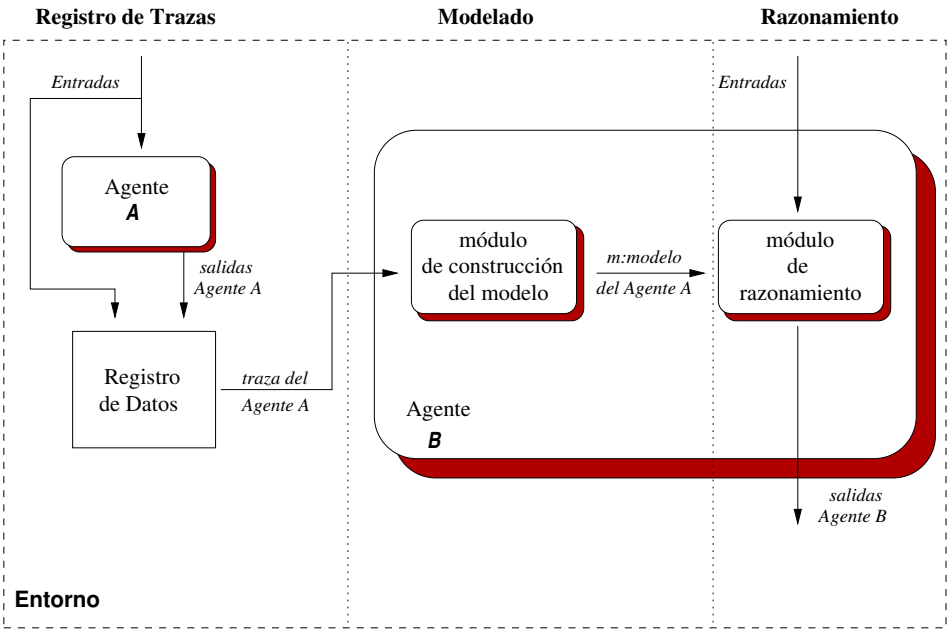


Figura 9.1: Marco general del Modelado de Agentes Basado en Trazas (*MABT*).

En cuanto al AGENTE B o agente modelador, MABT propone la incorporación de dos módulos a la arquitectura del agente con la finalidad de llevar a cabo la tarea

de modelado del AGENTE A. El primero de estos módulos, denominado *Módulo de Construcción del Modelo* (MCM), es el encargado, como su nombre indica, de llevar a cabo la construcción del modelo del AGENTE A. El otro módulo propuesto dentro de MABT se denomina *Módulo de Razonamiento*, (MRA), cuya función principal es utilizar el modelo del AGENTE A en el proceso de tomar la decisión sobre qué acción llevar a cabo.

El núcleo del MCM son una o varias técnicas de aprendizaje automático capaces de generar el modelo del AGENTE A, *m*, basándose en los datos adquiridos en la etapa del registro de trazas. Por otra parte, la arquitectura del MRA puede ser desde el modelo del AGENTE A, en el caso más simple, hasta procesos de razonamiento complejos que incorporen parte de la arquitectura original del AGENTE B con técnicas de aprendizaje automático con el fin de determinar la mejor acción en un momento dado.

La tarea de modelado llevada a cabo en este enfoque, entra dentro de las técnicas conocidas como IOAM (*Input/Ouput Agent Modelling*). Los enfoques IOAM asumen que se puede modelar a un agentes a partir de sus entradas y salidas. Un enfoque similar al utilizado en MABT es el modelado basado-en-características (*Feature Based Modelling* - FMB) [142]. FMB es aplicado al modelado de estudiantes. Los dominios de aplicación del MABT van más alla del modelado de estudiantes.

El objetivo de este esquema es conocer el límite superior que se puede alcanzar en el modelado de otros agentes, puesto que en la práctica en la gran mayoría de los dominios en donde intervienen agentes, no se tiene acceso directo a las entradas/salidas del agente a modelar.

9.2. Modelado de Agentes Basado en la Observación (MABO)

En MABT se dispone de antemano de un conjunto de datos que representan las entradas y salidas del agente a modelar. Este enfoque se puede aplicar en dominios estáticos en donde se disponga de los datos *"a priori"* y se desee obtener un modelo del AGENTE A. Sin embargo, en gran parte de los dominios en donde intervienen agentes no se puede contar *"a priori"* con este conjunto de datos. Por esta razón, se extendió el marco propuesto anteriormente a un nuevo enfoque que incluye un módulo para la obtención de estos datos. Por ejemplo, en uno de los dominios utilizados para la evaluación de esta propuesta, el simulador de fútbol de la *RoboCup*, un equipo o en este caso un jugador, no tiene acceso directo a las entradas del oponente (o compañero) en tiempo real, es decir, a lo que realmente está observando para tomar decisiones de que acciones ejecutar. Tampoco conoce qué acción ha realizado. Por esta razón el agente modelador tiene que obtener datos sobre el oponente desde su punto de vista e inferir la acción que ha llevado a cabo.

En otras palabras, el agente que lleva a cabo la tarea de modelado debe ser capaz de inferir las acciones que ha realizado el agente a modelar en instantes anteriores a partir de sus propias entradas.

Dada la necesidad de adquirir los datos para llevar a cabo la tarea de modelado MABO lleva a cabo en dos fases. La primera de estas fases es la creación de un módulo genérico capaz de etiquetar la última acción llevada a cabo por el AGENTE A basado en la observación del comportamiento de éste (*Módulo de Etiquetado de Acciones* - MEA). La necesidad de utilizar MEA surge al querer modelar el comportamiento de otros agentes en dominios dinámicos en donde no se tiene acceso directo a las entradas y salidas de los demás agentes que interactúan en el entorno (lo que exactamente está percibiendo el otro agente a través de sus sensores y la acción que lleva a cabo en un momento dado). La segunda fase dentro de MABO es la creación del modelo del otro agente basado en los datos generados por MEA. Esta tarea es llevada a cabo por el *Módulo de Creación del Modelo*(MCM). Dado que el poseer el modelo del agente con que se está interactuando proporciona una ventaja comparativa sólo si éste es utilizado, en MABO se propone la incorporación de un módulo adicional denominado *Módulo de Razonamiento* (MRA), cuya finalidad es la de utilizar el modelo generado en la etapa previa.

La Figura 9.2, se muestra el marco general del *Modelado de Agentes Basado en la Observación*.

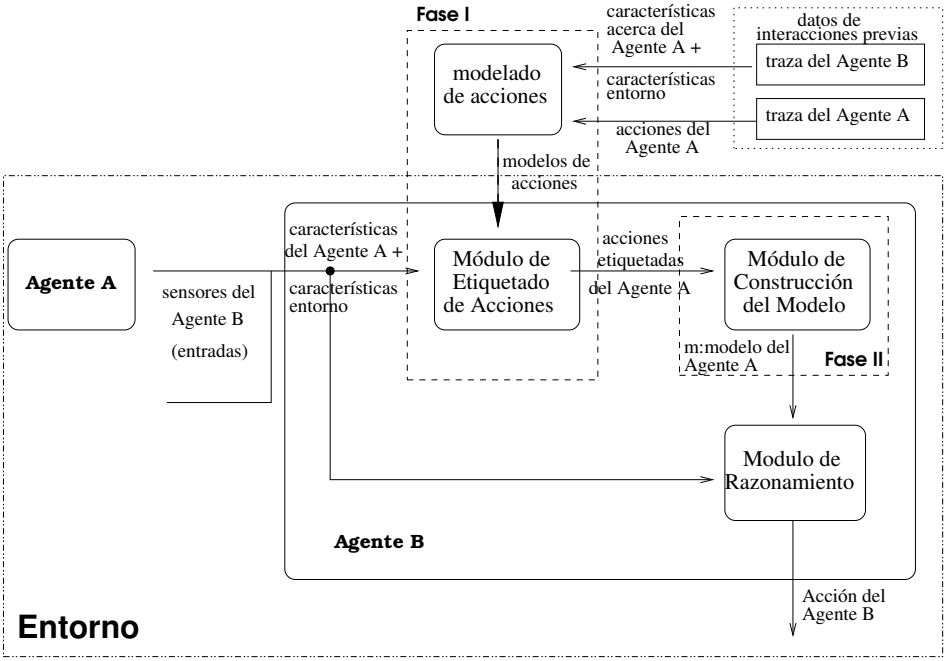


Figura 9.2: Marco general del Modelado de Agentes Basado en la Observación (*MABO*).

9.2.1. Módulo de Etiquetado de Acciones (MEA)

Con el propósito de predecir el comportamiento del agente a modelar (AGENTE A), es necesario obtener un número suficiente de instancias de la forma entrada/salida, de tal forma que éstas puedan ser utilizadas para aprender. Sin embargo, en la mayoría de los dominios en donde intervienen agentes, las entradas y salidas del AGENTE A no son accesibles directamente por parte del agente modelador (AGENTE B). O, mejor dicho, las acciones del AGENTE A deben ser inferidas por el AGENTE B mediante la observación del comportamiento de éste.

El propósito del MEA es clasificar las acciones llevadas a cabo por el AGENTE A basándose en las observaciones de éste realizadas por el AGENTE B. Esto puede ser visto como una tarea de clasificación. En este caso, se necesitan instancias de la forma *datos_del_agente_A, acción_del_agente_A* en donde los datos del AGENTE A son generados por el AGENTE B, tal y como éste lo percibe cuando realiza la acción.

Una descripción general del módulo de etiquetado de acciones se muestra en la Figura 9.3

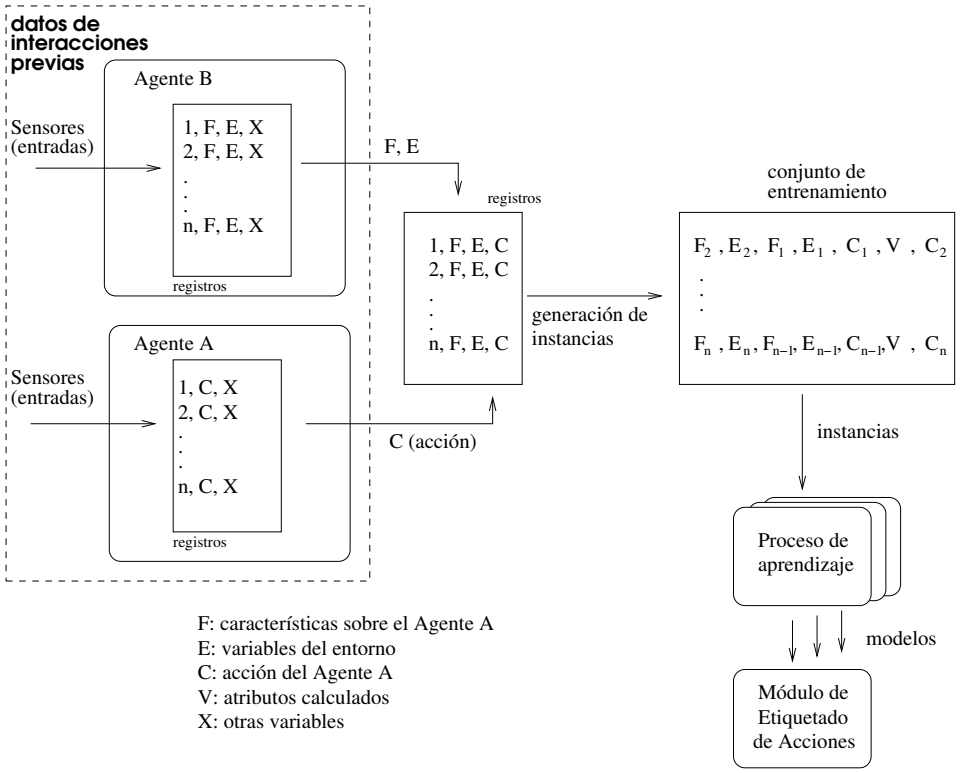


Figura 9.3: Creación del Módulo de Etiquetado de Acciones.

Dependiendo del dominio, existe una serie de acciones genéricas que se eje-

cutan de la misma manera independientemente del agente que la ejecute. Por esta razón, el MEA es independiente del AGENTE A, y puede ser utilizado para inferir las acciones de cualquier agente. Dada la generalidad del clasificador, éste sólo se construye una vez.

A continuación se detallan los pasos llevados a cabo en la construcción del MEA:

1. El AGENTE A y el AGENTE B interactúan un número determinado de veces. En cada instante, algunas variables relacionadas con el AGENTE A, variables relacionadas con el entorno obtenidas por el AGENTE B, y las acciones del AGENTE A son registradas para producir una traza del comportamiento del AGENTE A desde el punto de vista del AGENTE B.

2. Cada ejemplo I en la traza está compuesto por tres partes: un grupo de atributos relacionados con el AGENTE A, F, algunas variables relacionadas con el entorno, E, y la acción llevada a cabo por el AGENTE A, C, en un instante de tiempo dado, t. En otras palabras $I_t = F_t + E_t + C_t$. A partir de esta traza es fácil obtener una serie de ejemplos, D, basado en los cuales el AGENTE B puede inferir mediante la aplicación de técnicas de aprendizaje automático, la acción llevada a cabo por el AGENTE A, utilizando ejemplos de dos instantes de tiempos consecutivos.

3. Sea D el conjunto completo de los ejemplos disponibles de la traza del AGENTE A. Cada ejemplo $d_i \in D$ está compuesto por dos partes: un vector n-dimensional que representa los atributos, $a(d_i)$ y el valor de $c(d_i)$ que representan la clase a la que pertenece el ejemplo. En más detalle, $a(d_i) = F_t, E_t, F_{t-1}, E_{t-1}, C_{t-1}, V$ y $c(d_i) = C_t$. V representan una serie de atributos calculados basándose en la comparación de las diferencias de variables entre los instantes de tiempo.

4. Una vez que el clasificador o los clasificadores han sido construidos, éstos son utilizados con la finalidad de etiquetar la acción llevada a cabo por el AGENTE A. Este clasificador o grupo de clasificadores, constituyen el núcleo del MEA.

9.2.2. Módulo de Construcción del Modelo (MCM)

Una vez construido e incorporado MEA en la arquitectura del AGENTE B, el siguiente paso consiste en crear un modelo capaz de predecir el comportamiento del AGENTE A basado en las observaciones realizadas desde el punto de vista del AGENTE B. Para llevar a cabo esta tarea, se debe partir de ejemplos de la forma F_t, E_t, C_{mea_t} registradas durante las interacciones del AGENTE A con el AGENTE B, en donde C_{mea_t} es la acción etiquetada por MEA a partir de las observaciones en t y $t-1$.

En más detalle, los datos consisten en tuplas, $I's$, con características acerca del AGENTE A, algunas variables relacionadas con el entorno, y la acción que el AGENTE B infiere que ha llevado a cabo el AGENTE A. En vez de considerar utilizar sólo un instante de tiempo, se consideran varios instantes de tiempo en una misma instancia de aprendizaje. Además, las tuplas de aprendizaje son de la forma $I_t, I_{t-1}, ...I_{t-(w-1)}$, en donde w es el tamaño de la ventana de tiempo considerada. Al igual que en MEA se han utilizado atributos calculados, V.

Los pasos llevados a cabo para obtener el modelo del AGENTE A son los siguientes:

1. MEA se incorpora a la arquitectura del AGENTE B. De esta manera, éste puede etiquetar (inferir) las acciones del AGENTE A.

2. El AGENTE A y el AGENTE B interactúan en diferentes situaciones. En cada instante de tiempo, el AGENTE B obtiene información acerca del AGENTE A al igual que la acción llevada a cabo, la cual es etiquetada por el MEA. Toda esta información es registrada con la finalidad de producir una traza del comportamiento del AGENTE A.

3. Al igual que en la construcción del MEA, cada ejemplo, I, en la traza está compuesto por tres partes: un grupo de atributos relacionados con el AGENTE A, F, algunas variables relacionadas con el entorno, E, y la acción llevada a cabo por el AGENTE A, C (etiquetada por el MEA), en un instante de tiempo dado, t. En otras palabras $I_t = F_t + E_t + C_t$. En este caso se quiere predecir la acción que lleva a cabo el AGENTE A en un instante de tiempo dado y se necesita información acerca de éste de unos instantes de tiempo atrás. El número de instantes de tiempo utilizados para llevar a cabo la tarea de modelado es denominada w.

4. Sea D el conjunto completo de los ejemplos disponibles en un instante de tiempo dado. Cada ejemplo $d_i \in D$ esta compuesto por dos partes: un vector n-dimensional que representa los atributos, $a(d_i)$ y el valor de $c(d_i)$ que representan la clase a la que pertenece el ejemplo. En más detalle, $a(d_i) = F_t, E_t, F_{t-1}, E_{t-1}, C_{t-1}, ...F_{t-(w-1)}, E_{t-(w-1)}, V$ y $c(d_i) = C_t$. En donde V representa una serie de atributos calculados basándose en la comparación de las diferencias de variables entre los instantes de tiempo.

5. A partir de D se crea al clasificador capaz de predecir la acción que lleva a cabo el AGENTE A en un instante de tiempo dado.

9.2.3. Módulo de Razonamiento (MRA)

Predecir las acciones de un agente no es suficiente. Las predicciones se deben utilizar de manera tal que el agente modelador se pueda anticipar y reaccionar a las

acciones del oponente en situaciones competitivas, o colaborar con el compañero en situaciones colaborativas.

Al igual que en el MEA y el MCM, en este módulo se pueden aplicar técnicas de aprendizaje automático con la finalidad de aprovechar el modelo que se posee del agente o agentes con los cuales se interactúa. En el más simple de los casos, el núcleo de este módulo puede ser hecho a mano aprovechando la arquitectura del agente que utiliza MABO.

Capítulo 10

Evaluación: MABT

En el capítulo anterior se han descrito dos propuestas para llevar a cabo la tarea de modelar el comportamiento de un agente. En este capítulo se muestran los resultados obtenidos en la evaluación del primero de los enfoques propuestos, el *Modelado de Agentes Basado en Trazas* (MABT).

Con la finalidad de llevar a cabo la evaluación del MABT, se ha realizado una serie de experimentos que van desde aquéllos en donde se pretende demostrar la viabilidad de la propuesta (sección 10.1), hasta la evaluación del MABT en dominios complejos (sección 10.3), pasando por la aplicación del enfoque con vistas a utilizar el modelo generado (sección 10.2).

10.1. Modelado de Agentes en Dominios Estáticos

Con vistas a determinar la viabilidad del MABT, como primera aproximación, se ha simulado la interacción del agente modelador y el agente a modelar en situaciones denominadas estáticas. Es decir, se han utilizado dominios de clasificación a partir de los cuales se ha generado el comportamiento del agente a modelar o AGENTE A, para posteriormente generar el modelo de éste, partiendo de sus entradas y salidas.

10.1.1. Configuración Experimental

Para determinar si el conocimiento generado por el AGENTE B es capaz de modelar el comportamiento del AGENTE A, considerado como una caja negra, se ha llevado a cabo una serie de pasos que se detallan a continuación. A efectos de la experimentación, se considera que el AGENTE A está basado en un clasificador generado a partir de un algoritmo de aprendizaje. Por esta razón, antes de registrar el comportamiento del AGENTE A, el núcleo de éste, debe ser generado. En la

Figura 10.1 se muestra el proceso llevado a cabo para generar el clasificador que sirve de base al AGENTE A. Se asumen que se dispone de ejemplos para poder generar al núcleo del AGENTE A. Igualmente se muestra el proceso de generación del modelo.

El proceso de generación del núcleo del AGENTE A y la posterior generación del modelo de éste por parte del AGENTE B, se detallan a continuación:

1. Sea T el conjunto completo de las instancias disponibles. Cada ejemplo $t_i \in T$ consta de dos partes: un vector n-dimensional que representa los atributos $a(t_i)$ y un valor $c(t_i)$ que representa la clase a la cual pertenece. El conjunto T se divide aleatoriamente en tres partes distintas, llamadas T^1, T^2 y T^3.

2. T^1 es utilizado para generar el clasificador que será la base del AGENTE A. Las entradas que recibe el AGENTE A son los atributos de los ejemplos y la salida que produce éste es la clase a la que el clasificador generado determina que pertenecen estos ejemplos.

3. A continuación, los atributos de cada ejemplo, $a(t_i^2)$, en T^2 son utilizados como entrada al AGENTE A, el cual genera una predicción de clase de t_i^2. Esta clase se denomina $\hat{c}(t_i^2)$. Después, un nuevo conjunto de ejemplos $\hat{T}_2$ es creado a partir de cada par $a(t_i^2), \hat{c}(t_i^2)$.

4. El nuevo conjunto $\hat{T}^2$ es utilizado como entrada al MCM del AGENTE B, el cual genera un modelo del AGENTE A. Como la clase de los ejemplos en $\hat{T}^2$ son las salidas del AGENTE A, las reglas obtenidas (modelo) deben ser capaces de modelar el comportamiento del AGENTE A. Esto quiere decir que el conocimiento adquirido por el AGENTE B debería ser capaz de predecir la salida del AGENTE A, sin importar si las predicciones que el AGENTE A realiza son correctas o no.

Para determinar la capacidad de modelado del AGENTE B sobre el comportamiento del AGENTE A se lleva a cabo el siguiente proceso:

1. El conjunto de datos T^3 es utilizado como entrada para ambos, AGENTE A y AGENTE B, como se muestra en la Figura 10.2.

2. Las salidas producidas tanto por el AGENTE A como por el AGENTE B se comparan. Esta comparación se mide como el número de ejemplos en que la clase que predice el modelo en el AGENTE B difiere de la dada por el AGENTE A supuestos los mismos atributos de entrada. Por razones experimentales, estas diferencias se miden a partir de un conjunto de datos llamado $\widehat{T^3}$ cuyo vector de atributos corresponde a los atributos que posee T^3 y la clase a la que corresponden estos atributos es la predicción que realiza el AGENTE A. De esta forma, la precisión que logre el AGENTE B sobre este conjunto de datos brinda una estimación de la precisión del modelo.

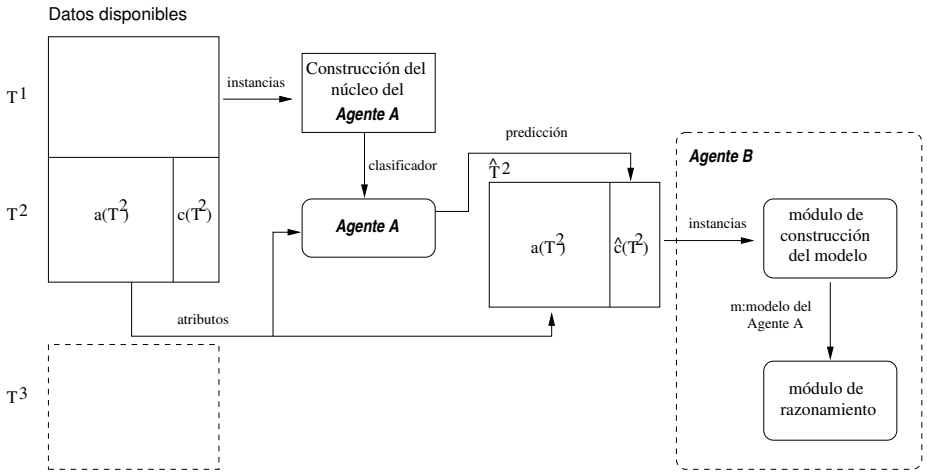


Figura 10.1: Registro de trazas y construcción del modelo del AGENTE A.

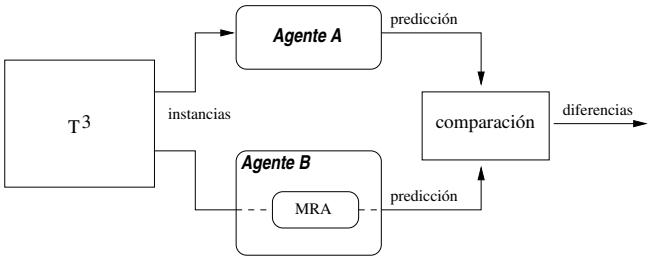


Figura 10.2: Validación del modelo obtenido por el AGENTE B.

Para la comprobación experimental de este enfoque, se ha utilizado como núcleo del AGENTE A una red de neuronas generada mediante el algoritmo de retropropagación y, como técnica de aprendizaje automático base del MCM del AGENTE B, se ha utilizado el algoritmo de generación de árboles de decisión C4.5 [107]. El *Stuttgart Neural Network Simulator* (SNNS) [149] ha sido utilizado como herramienta para la generación de la red de neuronas del AGENTE A.

En cuanto a los datos de prueba utilizados, se han utilizado tres dominios del conocido repositorio de datos del UCI [6]. La selección de estos dominios se ha llevado a cabo tomando en consideración la tarea de aprendizaje de cada uno de éstos, considerando que estos dominios son ejemplos típicos en donde un modelado entrada/salida a partir de la interacción de dos agentes resulta de gran importancia. A continuación se detalla la tarea de clasificación en cada dominio y el posible uso del proceso de modelado.

- Datos de registros de votos del Congreso de los Estados Unidos de América. Este es un ejemplo de un posible agente de negociación. En este caso, el modelo del otro agente generado por el AGENTE B, puede ayudar a decidir

cómo negociar con el otro agente observando y categorizando los rasgos políticos de un tercer agente.

- Datos Tic-Tac-Toe (Endgame). Este dominio es un ejemplo de dominios en donde poseer información sobre el comportamiento de los otros agentes es de mucha importancia. Si se posee un modelo del comportamiento del oponente en cualquier juego de conocimiento completo de suma-cero, se puede utilizar éste en la búsqueda de la mejor jugada dada una situación de juego. En estos casos, una búsqueda *alfa-beta* puede ser transformada en un tipo de técnica de búsqueda del mejor-primero, permitiendo una búsqueda más profunda en ramificaciones de interés. Cuando la búsqueda *alfa-beta* tiene que expandir el grupo de posibles movimientos del oponente a un número impar de niveles del árbol, el modelo aprendido del oponente puede ser utilizado para predecir la salida del otro agente. Esto permite podar el resto de las ramas, convirtiendo un árbol minimax en un árbol max (el movimiento seleccionado es aquél cuyas ramas llevan al nodo hoja con el máximo valor de la función heurística).

- Datos de Cáncer de Senos. El modelo obtenido en este dominio podría ser un ejemplo de cuán útil puede ser este enfoque cuando los datos utilizados para construir el AGENTE A no están disponibles y algún tipo de conocimiento sobre este agente es necesario, como en las técnicas de envoltura (*wrappers*) para el desarrollo de agentes [75]. Por ejemplo, en este caso, un hospital podría haber creado, en el pasado, un sistema de diagnóstico basado en agentes cuyo motor de inferencia son redes de neuronas, y puede no poseer los datos originales con el que fue creado. Registrando su comportamiento, y utilizando el esquema propuesto, se podría tener acceso a un conjunto de reglas que declarativamente pueden describir este conocimiento.

La características de los dominios utilizados se reflejan en la Tabla 10.1. Para obtener una estimación apropiada de la capacidad de modelado del AGENTE B, los resultados que se muestran son el promedio de una validación cruzada de diez carpetas. Es decir, se han generado diez grupos diferentes de T^1, T^2 y T^3. Por otra parte, las redes de neuronas, que son el núcleo del AGENTE A, se obtienen a partir del entrenamiento del algoritmo de aprendizaje hasta alcanzar la convergencia.

Tabla 10.1: Dominios utilizados para evaluación del MABT en situaciones estáticas.

Dominio	Atributos	Clases	Instancias
Registros de votos	16	2	435
Tic-Tac-Toe	9	2	958
Cáncer de Senos	30	2	569

10.1.2. Resultados

En la Tabla 10.2 se muestran los resultados obtenidos en el proceso de evaluación del modelo generado por el AGENTE B. La segunda columna muestra la tasa de aciertos obtenida por el AGENTE A (basado en redes de neuronas) sobre el conjunto de datos T^3. En la segunda columna se muestra la precisión del modelo generado por el MCM del AGENTE B sobre el mismo conjunto de prueba. Sin embargo, los resultados más interesantes son los que se muestran en la última columna, en donde se refleja la precisión del modelo creado por el MCM del AGENTE B sobre el conjunto de datos a partir de $\widehat{T}^3$. En otras palabras, esta columna refleja las diferencias entre la salida del AGENTE A y la predicción realizada por el AGENTE B sobre T^3.

Tabla 10.2: Tasa de aciertos (en %) del AGENTE A y del modelo de éste generado por el AGENTE B sobre el conjunto de datos T^3 y la tasa de aciertos del modelo sobre el conjunto de datos $\widehat{T}^3$.

Dominio	AGENTE A/T^3	AGENTE B/T^3	AGENTE B/ $\widehat{T}^3$
Votos	95.6	94.6	96.4
Tic-Tac-Toe	97.9	90.2	90.7
Cáncer	97.4	94.0	94.5

En los tres dominios utilizados, el modelo del AGENTE A generado por el AGENTE B basado en el algoritmo de aprendizaje C4.5, obtiene una tasa de precisión superior al 90 % sobre el conjunto de datos $\widehat{T}^3$, lo que indica que el modelo generado reproduce de forma aproximadamente correcta la misma salida que el AGENTE A.

10.2. Utilización del Modelo Generado

Una vez determinada la viabilidad de la construcción de modelos de agentes mediante la aplicación del MABT en el caso de dominios de clasificación, denominados dominios estáticos, se plantea la necesidad de aplicar MABT a dominios en donde la tarea de aprendizaje sea distinta. Por otra parte, el poseer un modelo del agente con el cual se está, o pretende, interactuar puede proporcionar una ventaja sólo si este modelo es utilizado en el proceso de razonamiento del agente que los posee. Por esta razón, además de utilizar MABT en dominios con tareas de aprendizaje distintas a las denominadas tareas de clasificación, en esta serie de experimentos se utiliza el modelo adquirido por el agente modelador mediante el MCM como reemplazo del motor de inferencia del AGENTE B (MRA). El objetivo es comprobar si el comportamiento original del AGENTE A y el comportamiento generado por el modelo son similares.

10.2.1. Configuración Experimental

El dominio seleccionado para evaluar el MABT en este caso es el *Simulador Distribuido de Agentes Autónomos - SimDai* [121]. SimDai permite la simulación de robots autónomos equipados con una serie de sensores y distintas arquitecturas de control.

La situación simulada es el movimiento de un robot en un entorno bidimensional en el cual se encuentran presentes obstáculos con diversidad de formas. El objetivo del robot es moverse, de manera eficiente, hasta un punto marcado como meta.

La Figura 10.3 muestra la descripción del robot. El mismo posee cinco sensores. Tres de ellos informan al robot sobre cuán cerca están los obstáculos (sensores de proximidad). Los otros dos miden cuán lejos se encuentra el robot de la posición final y cuál es el ángulo a ese punto de destino. El robot posee dos ruedas que se pueden mover a distintas velocidades v_1 y v_2[1]. De esta manera, el robot puede llevar a cabo giros. Sin embargo, por razones experimentales, la velocidad de la rueda 1, v_1, se considera constante. En otras palabras, las dos ruedas están en movimiento, pero la dirección del robot se controla mediante v_2.

Con vistas a aplicar MABT, el robot es considerado como el agente a modelar, es decir, el AGENTE A. El control de este agente está basado en el esquema de Braitenberg [10] en donde las relaciones entre los sensores y los actuadores son definidas por una red de neuronas obtenida mediante Coevolución Uniforme [5]. Como se ha mencionado, la velocidad de una de las ruedas del robot, v_1, se considera constante, y por esta razón, el objetivo de la red de neuronas es controlar la velocidad de la rueda dos, es decir, v_2.

En cuanto al AGENTE B o agente modelador, éste se considera como un robot idéntico al AGENTE A con la salvedad de que es controlado por el modelo adquirido del AGENTE A. Es decir, el MCM construye *off-line* el modelo del AGENTE A, m, basado en la traza del comportamiento de éste. Posteriormente, m pasa a ser el MRA del AGENTE B.

El MCM del AGENTE B utiliza como técnica de aprendizaje automático un algoritmo de generación de árboles de regresión y un algoritmo de generación de árboles de decisión, dependiendo de la tarea de aprendizaje, tal y como se describe más adelante.

En todos los experimentos realizados en este dominio para estimar la precisión en la tarea de generación del modelo del AGENTE A, se ha utilizado una validación cruzada de diez carpetas.

[1] La velocidad se encuentra en el rango [-1.0, 1.0].

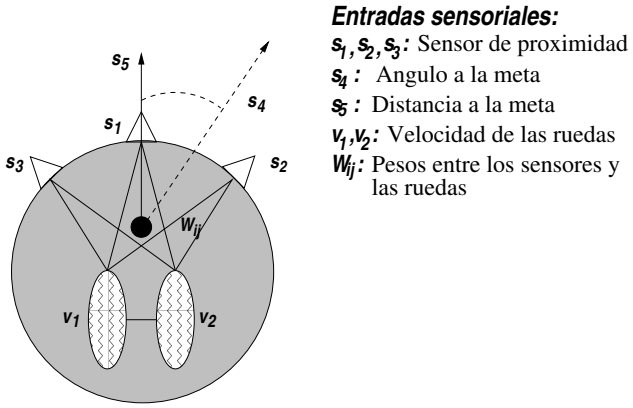


Figura 10.3: Descripción del robot utilizado en SimDai.

Modelo Basado en Arboles de Regresión

En la fase del registro de trazas, se han obtenido los datos correspondientes a seis simulaciones del AGENTE A. El número total de instancias es de 976 y el valor a predecir es la velocidad de la rueda dos, v_2, del robot. Para generar el árbol de regresión se ha aplicado el algoritmo M5 [111]. La versión de M5 utilizada es la implementada en WEKA (versión 3.1.7) [144].

Modelo Basado en Arboles de Decisión

Además de demostrar la viabilidad de aplicar el MABT a este tipo de dominios, otro objetivo de estos experimentos es obtener un modelo del AGENTE A más fácil de entender si se compara con una red de neuronas. A pesar de que los árboles de regresión son relativamente más fáciles de entender que una red de neuronas, los modelos lineales asociados a cada clase disminuyen en cierta medida su comprensibilidad. Con vistas a superar este problema, se ha aplicado el algoritmo de generación de árboles de decisión C4.5. Dado que C4.5 sólo puede trabajar con clases discretas, es necesario transformar los datos correspondientes a la velocidad de la rueda dos, v_2, en datos discretos. Esta discretización ha sido realizada manualmente tomando en consideración la distribución de los datos. Se han considerado un total de 11 clases que se muestran en la Tabla 10.3.

El número de instancias utilizadas en estos experimentos es igual que en la generación de árboles de decisión del punto anterior (976 instancias).

Una vez que el modelo del AGENTE A, m, ha sido generado, se utiliza éste como si fuese el MRA del AGENTE B. Cabe señalar que m es capaz de predecir valores discretos y la velocidad v_2 tiene que ser continua. Por esta razón, para pasar de clases discretas a continuas, se reemplaza la predicción de m por un valor correspondiente a la media de los datos incluidos en el intervalo discretizado uti-

Tabla 10.3: Intervalos de velocidad de la rueda dos (v_2) y su equivalencia en clases discretas.

intervalo	clase discreta	instancias
-1.0000	mínima	158
-0.7501 a -0.9999	baja	60
-0.5001 a -0.7500	medio_baja	101
-0.2501 a -0.5000	poco_baja	58
-0.0001 a -0.2500	cerca_0_negativo	94
0.0000	nula	6
0.0001 a 0.2500	cerca_0_positivo	102
0.2501 a 0.5000	poco_alta	81
0.5001 a 0.7500	media_alta	158
0.7501 a 0.9999	alta	145
1.0000	máxima	13

lizado para entrenar. Por ejemplo, para el intervalo comprendido entre $-0{,}5001$ y $-0{,}7500$ correspondiente a la etiqueta *poco_baja* el valor asignado es $-0,6206$ equivalente a la media del valor de la clase de las instancias incluidas en dicho intervalo.

Una vez configurado el AGENTE B, se utiliza el simulador para crear situaciones con el fin de comparar al AGENTE A con el AGENTE B. Concretamente, se realizaron 50 ejecuciones (simulaciones de búsqueda de blancos) tanto para el AGENTE A como para el AGENTE B, cuyo MRA es m. Cada ejecución comienza desde un punto distinto en un mundo bi-dimensional (Figura 10.4) y consiste en alcanzar la meta de manera eficiente.

Para comparar el comportamiento del AGENTE B con el comportamiento del agente modelado, AGENTE A, se han utilizado las variables *distancia recorrida* y *tiempo utilizado*. La distancia recorrida se refiere a la distancia que recorre el robot desde el punto de partida al punto en donde se encuentra el objetivo. De igual forma, el tiempo utilizado representa al tiempo que consume el robot en alcanzar el objetivo. Por razones experimentales el tiempo máximo está fijado a 2000 ciclos.

10.2.2. Resultados

En esta sección de muestran los resultados obtenidos en el proceso de generación del modelo del AGENTE A por parte del AGENTE B. Además, se muestran los resultados del proceso de utilización del modelo por parte del AGENTE B.

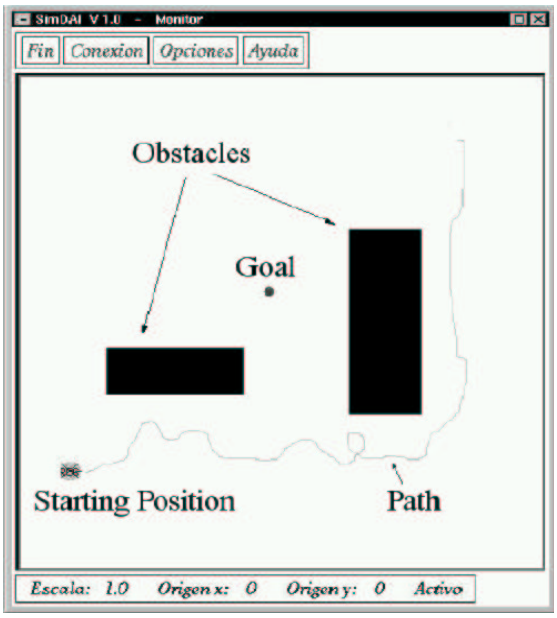


Figura 10.4: Mundo Bi-dimensional utilizado en SimDai.

Arboles de Regresión

En la Tabla 10.4 se muestran los resultados que obtiene M5 en el proceso de generación del modelo del AGENTE A. Como se puede apreciar, se obtiene un coeficiente de correlación cercano al 1, lo que indica que el modelo adquirido por el AGENTE B es muy similar al comportamiento del AGENTE A. Por otra parte, en la Tabla 10.5 se muestra el resumen del árbol de regresión generado por M5. Cada nodo hoja del árbol tiene asociado un modelo lineal (Tabla 10.6) que estima el valor de la clase (velocidad de la rueda 2 del robot).

Tabla 10.4: Resultados de la aplicación de M5.

Coeficiente de Correlación	0.9954
Media del Error Absoluto	0.0342
Raíz Media Error Cuadrático	0.0641

Arboles de Decisión

En la Tabla 10.7 se muestran los resultados obtenidos al utilizar C4.5 como técnica de aprendizaje automático dentro del MCM del AGENTE B. Como se puede apreciar, tanto C4.5 como C4.5-RULES, generan un modelo que, en alrededor del 84 % de los casos, genera la misma salida que el AGENTE A para los mismos

Tabla 10.5: Reglas del árbol de regresión generado por M5.

$Sensor\ 1$	$Sensor\ 2$	$Sensor\ 3$	$Sensor\ 4$	$Sensor\ 5$	Modelo
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$ -0.841	$\leq$0.29	LM1
$\leq$ 0.0333	-	$\leq$ 0.233	$>$ -0.841 and $\leq$ -0.743	$\leq$0.29	LM2
$\leq$ 0.0333	-	$\leq$ 0.233	$>$ -0.743	$\leq$0.29	LM3
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.29 and $\leq$0.761	LM4
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.761 and $\leq$0.975	LM5
$\leq$ 0.0333	-	$\leq$ 0.233	$\leq$-0.59	$>$0.975	LM6
$\leq$ 0.0333	-	$\leq$ 0.233	$>$-0.59 and $\leq$ -0.453	-	LM7
$\leq$ 0.0333	-	$\leq$ 0.233	$>$-0.453	-	LM8
$\leq$ 0.0333	-	$>$ 0.233	$\leq$-0.161	-	LM9
$>$ 0.0333 and $\leq$0.22	-	-	$\leq$-0.161	-	LM10
$>$0.22 and $\leq$0.587	-	$\leq$0.213	$\leq$-0.161	-	LM11
$>$0.22 and $\leq$0.587	-	$>$0.213	$\leq$-0.161	-	LM12
$>$0.587	-	-	$\leq$-0.161	-	LM13
$\leq$0.193	-	-	$>$-0.161 and $\leq$0.134	-	LM14
$\leq$0.193	-	-	$>$0.134 and $\leq$0.711	-	LM15
$\leq$0.193	-	-	$>$0.711	-	LM16
$>$0.193	-	-	$>$-0.161	-	LM16

Tabla 10.6: Modelos lineales generados por M5.

Modelo	Predicción	Factor	$Sensor\ 1$	$Sensor\ 2$	$Sensor\ 3$	$Sensor\ 4$	$Sensor\ 5$
LM1:	clase =	0.36	-0.0936	+0.0711	-0.895	-0.723	- 0.276
LM2:	clase =	0.358	-0.0936	+0.129	-1.25	-0.668	-0.35
LM3:	clase =	0.0201	-0.0936	+0.0288	-1.2	-1.15	-0.423
LM4:	clase =	0.201	-0.0936	+0.0751	-1.29	-0.969	-0.531
LM5:	clase =	0.886	-0.0936	+0.0365	-1.11	-0.427	-0.859
LM6:	clase =	0.933	-0.0936	+0.0365	-1.11	-0.427	-0.943
LM7:	clase =	0.0814	-0.0936	+0.0257	-0.312	-1	-0.134
LM8:	clase =	-0.0041	-0.0936	+0.0202	-1.22	-1.2	-0.453
LM9:	clase =	-0.00956	-0.0936	+0.0155	-1.14	-1.12	-0.166
LM10:	clase =	-0.073	-2.6	+0.0441	-0.964	-1.03	-0.264
LM11:	clase =	-0.149	-2.23	+0.0065	-0.407	-0.895	-0.0413
LM12:	clase =	-0.427	-1.33	+0.0065	-0.424	-0.488	-0.0413
LM13:	clase =	-0.719	-0.62	+0.0065	-0.226	-0.305	-0.0413
LM14:	clase =	-0.0313	-2.87	+0.0897	-1.14	-1.13	-0.33
LM15:	clase =	-0.0435	-2.25	+0.0879	-1.11	-1.11	-0.329
LM16:	clase =	-0.408	-0.757	+0.0364	-0.38	-0.652	
LM17:	clase =	-0.839	-0.126	+0.013	-0.129	-0.135	

valores de los atributos de entrada. Estos resultados dan una idea de la precisión del modelo, pero no permiten estimar su utilidad en una situación real. Por esta razón el modelo, m, se utiliza como si fuese el MRA del AGENTE B como se explicó en la configuración experimental.

Como se puede ver en la Figura 10.5, la distancia cubierta por el AGENTE A y el AGENTE B es muy similar. De igual manera, el tiempo consumido en alcanzar la meta, (Figura 10.6) es similar. El AGENTE A no alcanza la meta en una ocasión (tiempo $>$ 2000) mientras que el AGENTE B controlado por m, no alcanza la meta

Tabla 10.7: Tasa de aciertos (en %) de C4.5 y C4.5-RULES en el proceso de generación del modelo.

	Aciertos/C4.5	**Aciertos**/C4.5**Rules**
Promedio	84.44 %	84.02 %
Desviación	3.8624	4.2424

Tabla 10.8: Ejemplo de reglas generadas por C4.5-RULES.

Rule 13:		
sensor5	$>$	0.940451
	class =	null [79.4 %]
Rule 1:		
sensor1	$\leq$	0.08 AND
sensor3	$\leq$	0.426667 AND
sensor4	$\leq$	-0.863739 AND
sensor5	$\leq$	0.298303
	class =	very-high [89.9 %]
Rule 73:		
sensor1	$>$	0.346667 AND
sensor2	$\leq$	0.16 AND
sensor5	$\leq$	0.141961 AND
	class =	very-low [98.1 %]
Rule 80:		
sensor1	$>$	0.573333
	class =	super-low [98.0 %]
Rule 69:		
sensor1	$>$	0.28 AND
sensor4	$>$	-0.039041
	class =	super-low [97.1 %] ...

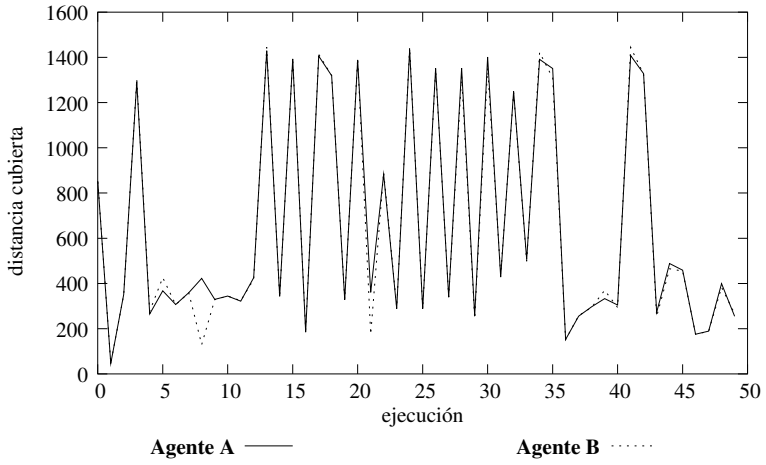


Figura 10.5: Distancia recorrida por el AGENTE A y el AGENTE B (controlado por el modelo generado utilizando C4.5) antes de alcanzar el objetivo.

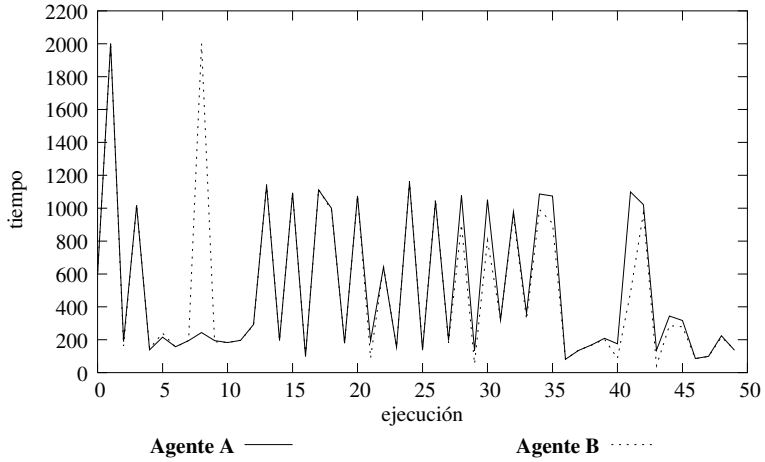


Figura 10.6: Tiempo consumido por el AGENTE A y el AGENTE B (controlado por el modelo generado utilizando C4.5) en alcanzar el objetivo.

en dos ocasiones. Estos resultados indican que a pesar de que la precisión en la construcción del modelo no sobrepasa el 85 %, el conocimiento reflejado en el modelo captura en gran medida el comportamiento del AGENTE A.

10.3. Modelado en Entornos Dinámicos

La *Robot World Cup Initiative* (*RoboCup*) [81] es una iniciativa internacional que busca promover la investigación en Inteligencia Artificial y Robótica proponiendo un problema estándar donde un amplio rango de tecnologías pueden ser integradas y estudiadas. La *RoboCup* ha seleccionado el fútbol como tópico central de la investigación, teniendo como objetivo innovaciones que puedan ser aplicadas en problemas sociales significativos y de la industria. La última meta del proyecto de la *RoboCup* es que para el año 2050 se haya desarrollado un equipo de robots humanoides totalmente autónomos que pueda ganar en un partido contra el campeón del mundo de fútbol de humanos. Entre las tecnologías que pueden ser integradas en un equipo de robots están el diseño de agentes autónomos, colaboración multiagente, adquisición de estrategias, razonamiento en tiempo real, robótica y fusión de sensores.

A pesar de que el objetivo final de la *RoboCup* contempla el desarrollo de robots , actualmente existen simuladores que permiten el desarrollo de investigaciones enfocadas, sobretodo, en la adquisición de estrategias y el comportamiento de los agentes y no en las características físicas de estos. La liga o categoría de fútbol simulada se basa en el *Soccer Server System* [100]. Este es un sistema que permite que dos equipos de 11 agentes, implementados en diversos lenguajes de programación, jueguen un partido de fútbol.

Para ello, se utiliza una arquitectura cliente-servidor. El servidor, proporciona un campo virtual y simula los movimientos de los jugadores y el balón. La comunicación entre el servidor y los clientes se realiza vía *sockets* UDP/IP. Así, los clientes pueden ser implementados en cualquier arquitectura que permita comunicaciones de este tipo.

Los clientes definen el comportamiento de los jugadores, controlando sus movimientos de forma que cada uno de ellos dirige a un jugador. Mediante la conexión UDP/IP, recibe información sensorial (visual, auditiva y sobre el estado de su cuerpo) desde el servidor, y también a través de dicha conexión envía sus órdenes, o lo que es lo mismo, las acciones que quiere ejecutar sobre el entorno.

La información visual que los agentes reciben del entorno es variada. Pueden recibir información sobre la posición del balón, de los jugadores, y sobre marcas situadas por el campo (ver Figura 10.7) que dan idea a los jugadores de dónde están situados los diversos elementos del entorno (como las porterías y el centro del campo). Toda esta información que reciben los agentes es subjetiva, es decir, relativa a su propia posición. Esto significa que un agente no recibe nunca un mensaje visual de que, por ejemplo, el balón está en la posición x e y del campo, sino que el balón está a una distancia d de él, y que lo está "viendo" con un ángulo a. Si se añade a esto que la información que se recibe del simulador, se recibe con un ruido proporcional a la distancia de los objetos, se comprende una de las principales dificultades del dominio consistente en obtener una representación completa de la realidad.

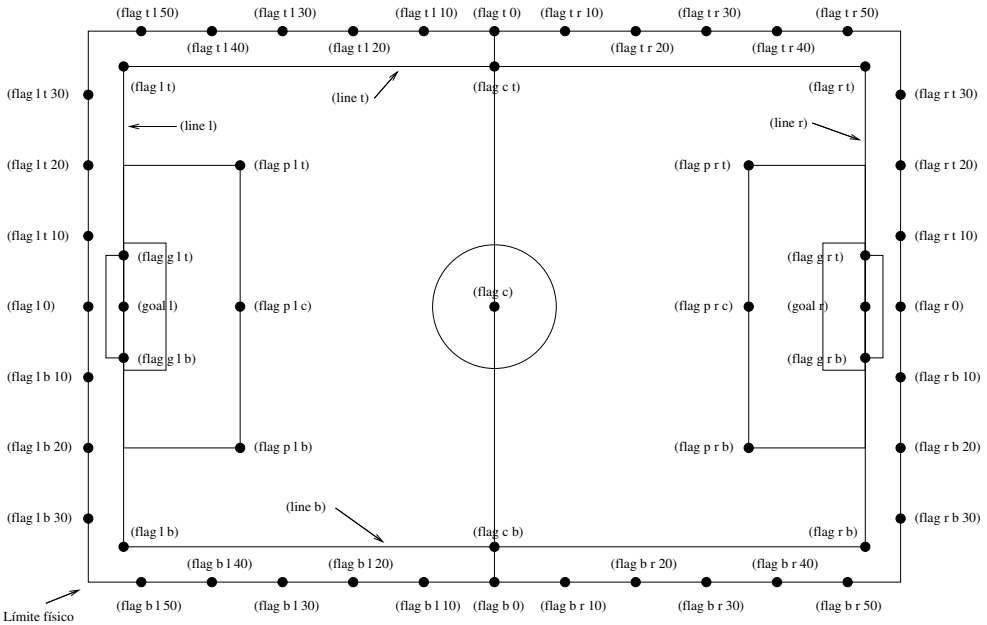


Figura 10.7: Marcas o banderas del campo de fútbol dentro del simulador de la *RoboCup*.

El simulador de fútbol ha sido seleccionado como dominio de prueba del MABT con vistas a determinar su aplicabilidad en entornos dinámicos.

Para determinar que el conocimiento generado por el agente, o JUGADOR B es capaz de modelar el comportamiento del agente o JUGADOR A, considerado como una caja negra, se han llevado a cabo dos etapas, la generación de trazas y el proceso de modelado.

El proceso de generación de trazas se detalla a continuación:

1. Selección de un jugador de un equipo situado entre los mejores de las últimas competiciones como JUGADOR A. Al seleccionar un jugador de un equipo situado entre los mejores, se asegura que el comportamiento de éste sea correcto en una situación dada. Es estos experimentos se ha utilizado un jugador del equipo TsinghuAeolus [148], equipo campeón de la competición del año 2001.

2. El JUGADOR A es utilizado en una situación en donde no tiene contrarios ni compañeros en el terreno de juego, puesto que se desea determinar si se puede obtener un modelo correcto del oponente en el caso más sencillo posible. Su objetivo es dirigir el balón hacia la portería contraria y disparar el balón con el fin de marcar gol.

Una vez que ha sido adquirida la traza del JUGADOR A, se procede a obtener el conocimiento que intenta modelar el comportamiento de éste. El detalle de la fase de construcción del modelo se detalla a continuación:

1. Por razones experimentales la experimentación relacionada con el JUGADOR B se limita a la generación del modelo del JUGADOR A mediante el módulo de construcción del modelo (MCM). Es decir, el JUGADOR B no llega a ser implementado en el simulador.

2. Se han diseñado dos tareas de aprendizaje: modelado simple y el modelado jerárquico. En el modelado simple, se utiliza C4.5 como técnica de aprendizaje automático para generar el modelo, mientras que en el modelado jerárquico se utilizan C4.5 y M5. Ambos algoritmos de aprendizaje están implementados en WEKA [144].

10.3.1. Modelado Simple

Como primera aproximación al problema, se ha aplicado el enfoque utilizado en las secciones anteriores con el fin de obtener el modelo del JUGADOR A. En este caso, sólo se ha utilizado un conjunto de entrenamiento con 291 instancias correspondientes a medio tiempo de un partido de fútbol simulado.

La información que se ha utilizado para generar el modelo del JUGADOR A corresponde a la información en bruto que recibe éste a través de sus sensores. Se han utilizado 120 atributos con información sobre la posición relativa del JUGADOR A con respecto a las banderas de localización y líneas del terreno de juego (distancia y dirección a cada punto y línea en un instante de tiempo dado). Además, cuatro atributos relativos a la posición del balón (distancia, dirección, cambio de radio, cambio de ángulo). Al igual que los atributos relacionados con el terreno de juego, los atributos relacionados con el balón son recibidos a partir del sensor de visión. Por otra parte, la información sobre el sensor del cuerpo está compuesta por 16 atributos (el tiempo de simulación, dos atributos relacionados con la fuerza, dos atributos relacionados con el tipo de visión, dos atributos relacionados con la velocidad, el ángulo de la cabeza, y ocho contadores de acciones realizadas). En resumen, se ha utilizado un total de 140 atributos.

Un jugador puede llevar a cabo entre 6 y 7 acciones distintas dependiendo del tipo de jugador (i.e. *catch, dash, kick, move, say, turn y turnneck*). Algunas de estas acciones se pueden llevar a cabo en el mismo instante de tiempo. Además, la mayoría de estas acciones tiene asociado uno o varios parámetros numéricos. En estos experimentos la clase de cada instancia es la combinación de las acciones que se pueden ejecutar en el mismo ciclo (e.g. $dash - turn\_neck$) con el correspondiente valor numérico si éste es el caso (e.g. $dash100 - turn\_neck103{,}13$). Estos valores numéricos son obtenidos discretizando los valores originales con una variante del algoritmo de Lloyd generalizado [44].

Los resultados de este experimento están etiquetados como ejecución 01 en la Tabla 10.9. El MCM del JUGADOR B obtiene más de un 45 % de precisión en la predicción de la acción que va a ejecutar el oponente asociada con el parámetro discretizado. Si se considera que existen 51 clases y que se utiliza como entrada 140 atributos, los resultados se pueden catalogar como regulares. Todos los resultados reflejados en la Tabla 10.9 han sido obtenidos mediante una validación cruzada de diez carpetas.

Con el propósito de mejorar los resultados anteriores, se incrementó el número de instancias aumentando la duración del partido. Adicionalmente se añadieron dos nuevos atributos: las coordenadas X e Y dadas por la arquitectura del JUGADOR A mediante un pre-procesado de los datos en bruto recibidos a través de los sensores. De esta manera, se añade información sobre la posición del JUGADOR A en lugar de que el modelo intente deducirla mediante los atributos de banderas del campo. Como consecuencia de la introducción de estos nuevos atributos, el número de total de atributos se reduce de manera drástica al eliminar la mayoría de los que están relacionados con las banderas de campo (sólo se conservan los atributos relacionados con el centro del campo y los relacionados con el centro de ambas porterías). En esta ejecución el número de clases se incrementa debido a la existencia de un mayor número de instancias, y el rango numérico también crece. Se aplicó nuevamente la variante del algoritmo de Lloyd generalizado para discretizar

las clases continuas. Los resultados de este experimento se pueden apreciar en la Tabla 10.9 etiquetados como ejecución 02. En este caso se logra un 55.72 % de precisión, a pesar del incremento en el número de clases.

En el análisis del modelo obtenido en este experimento, se observó que atributos con un gran número de valores desconocidos generaban un modelo muy difícil de entender. Por lo tanto, se sustituyeron los valores desconocidos por valores numéricos muy elevados para representar objetos que se encontraban fuera del campo de visión del JUGADOR A (por ejemplo, una bandera que se encuentra muy lejos para ser vista). Llevando a cabo este reemplazo, la precisión en la predicción permanecía casi sin variación (55.57 %) pero el modelo obtenido tenía más sentido. Los resultados de esta ejecución se muestran en la Tabla 10.9 etiquetados como ejecución 03.

Al analizar la matriz de confusión de la ejecución 03, se observó que muchas clases solo estaban instanciadas en pocos casos, con lo cual se hacía muy difícil obtener reglas para la clasificación de estas instancias. Por esta razón se decidió realizar pruebas limitando la tarea de aprendizaje a la predicción de la acción, sin incluir el parámetro correspondiente, lo que provoca una gran disminución en el número de clases. Por ejemplo, todas las instancias con clase del tipo $dash100 - turn\_neck103{,}13$ son compactadas a $dash - turn\_neck$, para con posterioridad, llevar a cabo el aprendizaje de los parámetros. El número de clases es de 7: *narrowhigh-turn-turnneck, wide-high, dash, dash-turnneck, kick, turn, y turnneck*. La clase *dash-turnneck* quiere decir que las acciones se llevan a cabo de manera concurrente. Como en la ejecución anterior, se llevó a cabo la experimentación reemplazando los valores desconocidos. La precisión en la predicción mejora hasta el 69.66 % (ejecución 04, con valores desconocidos) y 72.82 % (ejecución 05, sin valores desconocidos), como se muestra en la Tabla 10.9. Esta precisión en la predicción es razonable dado el nivel de ruido que existe en el simulador.

Tabla 10.9: Resultados obtenidos en el proceso de generación del modelo del AGENTE A. * sin valores desconocidos.

Ejecución	Instancias	Atributos	Clases	Precisión
01	291	140	51	45.36 %
02	2595	32	69	55.72 %
03*	2595	32	69	55.57 %
04	2595	32	7	69.66 %
05*	2595	32	7	72.82 %

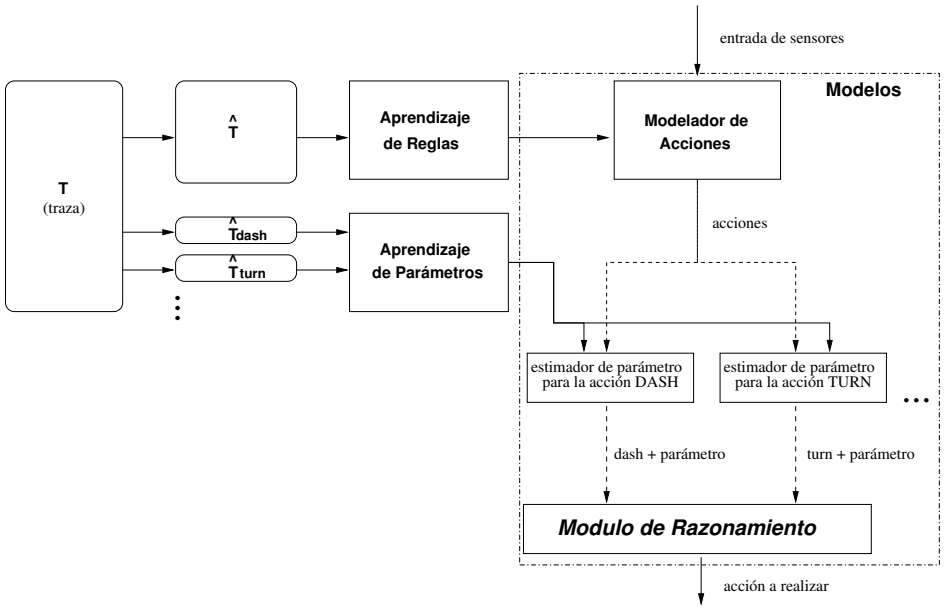


Figura 10.8: Arquitectura del aprendizaje jerárquico.

10.3.2. Modelado Jerárquico

Una vez analizados los resultados obtenidos, se decidió realizar un aprendizaje de las acciones y los parámetros por separado. A este tipo de aprendizaje secuencial se le ha llamado aprendizaje jerárquico. En la Figura 10.8 se muestra la arquitectura general del modelado jerárquico.

Conjuntamente, se decidió aprender sólo las acciones principales (*view, dash, kick y turn*) que son mutuamente excluyentes ya que son las más relevantes a la hora de utilizar el modelo aprendido. En estos experimentos se han utilizado 32 atributos (las banderas de campo principales e información del sensor de cuerpo), al igual que en las ejecuciones 02,03,04 y 05 reflejadas en la Tabla 10.9.

Primero, se utiliza el conjunto completo de instancias de entrenamiento para obtener el modelo que es capaz de predecir la acción del oponente utilizando para ello C4.5. Los resultados de esta experimentación están reflejados en la Tabla 10.10 etiquetados como ejecución 01. Se obtiene un 72.74 % de precisión, resultado similar las ejecuciones 04 y 05 mostradas en la Tabla 10.9. Después, para cada acción se genera un árbol de regresión utilizando el algoritmo M5, con el fin de modelar los parámetros (con valores continuos). Estos parámetros son: *Turn-Angle* (TA), *Dash-Power* (DP), *Kick-Power* (KP) y *Kick-Direction* (KD). Los resultados son mostrados en la Tabla 10.10, etiquetados como ejecuciones 02 a la 05.

Los resultados para *Dash-Power* (DP) y *Kick-Power* (KP) se pueden considerar como buenos (0.98 C.C. y 0.83 C.C., respectivamente). Pero en los parámetros re-

Tabla 10.10: Tasa de acierto (en % y coeficiente de correlación - C.C.) obtenidas por las distintas técnicas de aprendizaje automáticos utilizadas en la generación del modelo. Clase C indica clase continua.

Ejecución	Predicción	Algoritmo	Instancias	Clases	Precisión
01	acción principal	C4.5	2594	4	72.74 %
02	TA	M5	321	C	0.52 C.C.
03	DP	M5	1331	C	0.94 C.C.
04	KP	M5	929	C	0.83 C.C.
05	KD	M5	929	C	0.58 C.C.
06	KD	C4.5	929	5	62.10 %
07	TA	C4.5	321	5	62.30 %
08	KD	NB	929	5	45.64 %
09	TA	NB	321	5	48.91 %

lacionados con ángulos, *Turn-Angle* (TA) y *Kick-Direction* (KD), los coeficientes de correlación son peores (0.52 C.C. y 0.58 C.C., respectivamente). Por esta razón, se han discretizado, tomando en cuenta la distribución de los datos, los valores de estos parámetros y se ha aplicado sobre estos datos dos algoritmos (C4.5 y Naive Bayes - NB) que trabajan con clases discretas. Después de la discretización, resultan 5 clases: centro (50^o a -50^o), centro-izquierda (50^o a 100^o), centro-derecha (-50^o a -100^o), izquierda (100^o a 180^o), y derecha (-100^o a -180^o). Se trata de discretizar de la mejor manera estos ángulos en donde la predicción del comportamiento del oponente es más útil. Por ejemplo, se considera que es mas útil predecir la dirección del movimiento (o la dirección de un disparo) cuando el oponente se dirige hacia adelante que cuando va hacia atrás. Los resultados son denominados ejecuciones 06, 07, 08, y 09 en la Tabla 10.10. Estos muestran que utilizando C4.5 se obtienen mejores resultados (62.10 % KD y 62.30 % TA) que con Naive Bayes (45 % KD y 48.91 % TA). También se quieren comparar los resultados sobre el conjunto de datos discretizados con los resultados obtenidos sin la discretización. Esta tarea resulta difícil, dado que los resultados sobre datos discretizados obtenidos utilizando C4.5 son menos precisos que los valores dados por el modelo generado por M5, y por lo tanto la predicción sobre el comportamiento del oponente posee más incertidumbre. Pero asumiendo que las 5 clases discretas (centro, centro-derecha, etc) son todo lo que se necesita a la hora de utilizar el modelo, entonces M5 y C4.5 pueden ser comparados discretizando la salida de M5 en 5 clases y calculando el porcentaje de precisión. Para llevar a cabo esta comparación, se ha dividido aleatoriamente las instancias en conjunto de entrenamiento y conjunto de prueba (80 %/20 %). En relación al *Turn-Angle*, C4.5 obtiene 64 % mientras que M5 da un 54 % de precisión. Resultados similares se han obtenidos en la predicción de *Kick-Direction*: 61 % para C4.5 y 44 % para M5. De esta manera, y si se considera que una salida discreta es lo que hace falta a la hora de la utilización del modelo obtenido, C4.5 es la mejor opción.

Capítulo 11

Evaluación: MABO

Es evidente que en dominios como el simulador de fútbol de la *RoboCup*, en donde el agente modelador no tiene acceso directo a las entradas y las salidas del agente a modelar, MABT solo puede ser aplicado con propósitos experimentales. Dado que esta situación es común a la gran mayoría de los dominios en donde interactúan agentes, es necesaria la utilización de MABO con la finalidad de llevar a cabo la tarea de modelado en este tipo de dominios.

En este capítulo se muestran los resultados obtenidos en las diferentes etapas de la aplicación de MABO al dominio del simulador de fútbol.

11.1. Configuración Experimental

El MABO consta de tres módulos: módulo de etiquetado de acciones, módulo de construcción del modelo y el módulo de razonamiento. Dado que cada uno de estos módulos involucra una serie de experimentos, se ha creado una situación simulada común con el fin de determinar la utilidad de MABO en este dominio.

A diferencia de los experimentos realizados sobre el dominio del simulador de fútbol aplicando el MABT, en este caso, las entradas que son consideradas no son las entradas directas que recibe el agente a través de los sensores. En este caso, tanto en la creación del MEA como en el MCM y el MRA, se aprovecha la arquitectura del agente modelador el cual realiza un procesado de las entradas de los sensores con la finalidad de utilizar información de más alto nivel.

El AGENTE A utilizado en estos experimentos es un portero del equipo ORCA [99] y el AGENTE B o agente modelador es un delantero basado en el código del equipo CMUnited-99 [125]. El objetivo del AGENTE B es conducir la pelota hacia la portería que defiende el AGENTE A con el fin de marcar un gol. Por el contrario, el AGENTE A debe evitar que el AGENTE B marque goles.

11.2. Módulo de Etiquetado de Acciones

Los datos utilizados para la creación del MEA son una combinación de la percepción acerca del AGENTE A que posee el AGENTE B y la acción real llevada a cabo por el AGENTE A basada en registros de interacciones previas (partidos anteriores). Para generar los datos necesarios para la construcción del MEA, se simuló una situación en donde interviene un atacante (AGENTE B y un portero AGENTE A. Mientras ambos agentes están interactuando, se lleva a cabo un registro del comportamiento de éstos. A partir de la traza generada por el AGENTE B, se genera un conjunto de entrenamiento con un total de 68 atributos (ver Apéndice C). De estos atributos, 44 son datos acerca del AGENTE A y del entorno en dos instancias de tiempo consecutivas, t y $t-1$. El resto de los atributos, 24, son atributos calculados mediante la comparación de los atributos relacionados con el AGENTE A y el entorno. Por otra parte, la clase asociada a cada instancia es tomada de la traza del comportamiento del AGENTE A. Los valores de la clase puede ser *turn*, *kick*, *dash* y *none*. Además existe otra clase denominada *desconocida* que es asignada a una instancia para la cual no se posee la clase real por motivos del ruido existente en el dominio.

Una vez que se han generado los datos, se procede a generar los clasificadores capaces de deducir las acciones realizadas por el oponente.

Cabe mencionar que para la creación MEA se ha seguido el enfoque jerárquico descrito en la sección 10.3. Para generar el clasificador que etiqueta la clase principal se ha utilizado PART mientras que los parámetros numéricos son etiquetados por un clasificador generado por M5. Los resultados obtenidos en la generación de estos clasificadores se detallan en la Tabla 11.1.

Tabla 11.1: Resultados de la creación de los clasificadores que forman el núcleo del MEA.

Tarea	Instancias	Atributos	Clases	Precisión
Principal	5095	69	5	70.81 %
Turn	913	69	continua	0.007 C.C.
Dash	3711	69	continua	0.21 C.C.

C.C.: coeficiente de correlación

Existen tres filas en la Tabla 11.1. La primera de éstas refleja la predicción de la acción del AGENTE A, mientras que las otras dos filas muestran la predicción de los parámetros numéricos de dos acciones, *turn* y *dash*. Dado que el AGENTE A es un portero, por razones experimentales, sólo se consideran relevantes los parámetros de estas acciones y no se consideran los parámetros numéricos asociados a la acción *kick*. Las columnas representan el número de instancias utilizadas en la tarea de aprendizaje, el número de atributos utilizados y el número de clases (continua para clases numéricas). En la última columna se muestra la precisión ob-

tenida en la predicción. Para los parámetros numéricos, se muestra el coeficiente de correlación. Estos resultados han sido obtenidos utilizando una validación cruzada estratificada de 10 carpetas.

El clasificador generado para etiquetar las acciones del AGENTE A y que forma parte del MEA, obtiene un 70 % de precisión, lo cual es un resultado aceptable si se considera que el simulador añade ruido a la ya incierta tarea del etiquetado de la acción. Por otro lado, los resultados obtenidos en la predicción de los parámetros numéricos asociados a las acciones son pobres. Quizás las técnicas utilizadas para construir los modelos numéricos no son las más apropiadas, o quizás se pueden obtener mejores resultados discretizando los valores continuos de la clase. Generalmente no es necesario predecir los valores numéricos con gran precisión; una estimación aproximada es suficiente si se busca tomar ventaja de la predicción. Por ejemplo, puede ser suficiente predecir si el portero va a girar hacia la derecha o a la izquierda en vez de predecir el ángulo exacto en que va a girar. En estos experimentos se utilizará solo la predicción de la acción principal con la finalidad de llevar a cabo una acción en una situación dada.

11.3. Módulo de Construcción del Modelo

Una vez que han sido generados los clasificadores que son el núcleo del MEA, se incorpora éste a la arquitectura del agente con el fin de generar información sobre el oponente. La situación simulada es la misma que la utilizada en la construcción del MEA. De igual forma, se han utilizado los mismos atributos que al construir el MEA con la única salvedad de que esta vez se incluyen los atributos correspondientes a la clasificación de la acción llevada a cabo por el oponente. Un total de tres atributos por cada uno de los dos instantes de tiempo tomados en consideración son añadidos. En cada instante de tiempo se incluyen la acción realizada por el oponente y sus dos parámetros numéricos asociados. Por otra parte, se suprime el atributo *OpponentNumber* por considerarlo de poca utilidad en la tarea de aprendizaje. En tanto que el número de clase se ve incrementado en uno ya que se añade la clase *no_hay* que se asigna a las instancias a las cuales el MEA no puede clasificar.

Al igual que en el proceso de creación del MEA se ha utilizado PART y M5 para construir los modelos. Los resultados obtenidos a la hora de crear estos modelos se encuentran reflejados en la Tabla 11.2. Como se puede apreciar, el clasificador que genera el MCM con la finalidad de predecir la acción que llevará a cabo el AGENTE A, obtiene una precisión por encima del 80 % lo cual es un resultado aceptable en esta tarea. Por otra parte, los clasificadores generados por el MCM para llevar a cabo la predicción de los parámetros numéricos asociados a las acciones *turn* y *dash* obtienen resultados mejorables.

Tabla 11.2: Resultados de la creación de los clasificadores que forman parte del modelo del AGENTE A llevado a cabo por el MCM.

Tarea	Instancias	Atributos	Clases	Precisión
Principal	5352	73	6	81.13 %
Turn	836	73	C	0.67 C.C.
Dash	4261	73	C	0.41 C.C.

C.C.: coeficiente de correlación

11.4. Módulo de Razonamiento

11.4.1. Utilización del Modelo

Una vez que el JUGADOR B ha adquirido el modelo, m, del JUGADOR A y ha sido incorporado a su arquitectura (probablemente dentro de un módulo de razonamiento), éste puede ser utilizado para predecir las acciones del oponente en una situación dada. La tarea seleccionada para probar el modelo adquirido es *cuando disparar* [124]. Cuando el jugador atacante se aproxima a la portería, éste tiene que decidir si dispara a la portería o si sigue avanzando con el balón. En este caso, el AGENTE B (el atacante) tomará esa decisión basado en el modelo del comportamiento del portero (AGENTE A).

Cuando decide disparar, el AGENTE B primero selecciona un punto dentro de la portería como el *blanco del lanzamiento*. En este caso, un punto a cada lado la portería. El agente entonces considera su propia posición y la posición del portero para seleccionar que punto será el blanco del lanzamiento. Una vez que el agente esta cerca de la portería, utiliza el modelo del portero construido por el MCM con el propósito de predecir la reacción del portero y decide si disparar o no en un instante de tiempo dado. Por ejemplo, si se predice que el portero permanecerá quieto, el atacante avanza con el balón hacia la portería.

Con el propósito de estimar la efectividad del MABO en un partido de fútbol simulado, se han realizado 100 simulaciones en donde solo dos jugadores están presentes en el terreno de juego. Para cada simulación, el atacante (AGENTE B) y el balón son colocados en 30 posiciones diferentes del campo seleccionadas aleatoriamente. Esto hace un total de 3000 oportunidades de ataque. El portero se coloca cerca de la portería. La tarea del atacante es marcar gol mientras que la del portero es evitarlo.

Para probar la utilidad del modelo se compara a un atacante que utiliza el modelo del portero con un atacante que no utiliza dicho modelo. En todas las situaciones, el atacante conduce el balón hacia la portería hasta que decide cuándo disparar. El delantero que no utiliza el modelo decide cuándo disparar basándose sólo en la dis-

tancia hasta la portería, mientras que el atacante que utiliza el modelo, considera la distancia a la portería y la predicción de la acción del portero.

En la Figura 11.1 se muestra la situación simulada. La distancia a la cual el atacante toma de decisión de disparar es de 25 metros. Por otra parte el *blanco de lanzamiento* dependerá de la posición del portero. Si el ángulo que hace el portero con el *blanco de lanzamiento* más cercano al delantero es inferior a 20^o, el disparo se dirige a ese punto en concreto. En caso contrario se elegirá como *blanco de lanzamiento* el lado opuesto de la portería. En el caso del atacante que utiliza el modelo, además de la distancia, utiliza la predicción de la acción que llevará a cabo el portero. Es decir, si el atacante se encuentra dentro del área que comprende los 25 metros desde el centro de la portería, y el modelo predice que el portero avanzará hacia su posición (realizará un *dash* suponiendo que la dirección es hacia el balón), el atacante llevará a cabo el disparo, en caso contrario seguirá avanzando con el balón. Adicionalmente, se ha fijado un área de 15 metros desde el centro de la portería en donde el atacante disparará siempre que esté en ella, ya que se encuentra muy cerca del portero el cual puede evitar que siga avanzando.

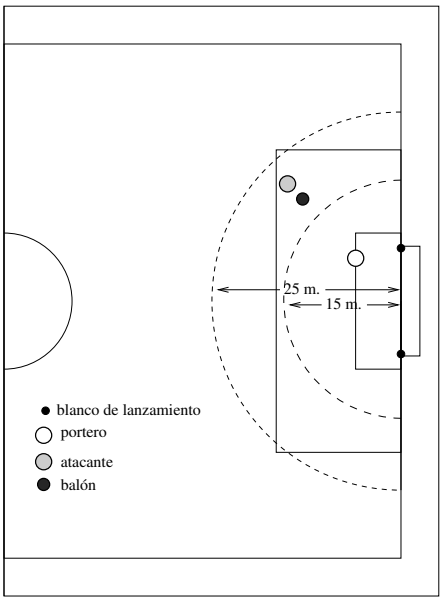


Figura 11.1: Situación simulada para estimar la utilidad del modelo del portero adquirido por el atacante.

Los resultados obtenidos en la realización de estos experimentos se muestran en la Tabla 11.3.

Como muestran los resultados, la media de goles utilizando el modelo es superior a la media de goles sin utilizar el modelo. Estos resultados pueden ser resumidos como que, de cada 30 disparos, un gol extra es marcado si se utiliza el modelo. Además, los tiros dirigidos hacia fuera de la portería se reducen si se uti-

Tabla 11.3: Resultados de comparativos de la utilización del modelo.

Delantero	Media de Goles	Media de tiros fuera
sin modelo	4.65	11.18
con modelo	5.88	10.47

liza el modelo. Se ha realizado un prueba *t*-test para determinar si estas diferencias son significativas con un $\alpha = 0{,}05$. Los resultados demuestran que sí existe diferencia significativa entre utilizar el modelo o no. A pesar de que el AGENTE B utiliza de una manera muy simple el modelo del AGENTE A, se obtiene una mejora significativa.

11.4.2. Utilización Automática del Modelo

En el apartado anterior se utilizó el modelo del AGENTE A para ayudar al AGENTE B a decidir que acción ejecutar. Pero la estrategia era fija y programada a mano. En este apartado se quiere generar de manera automática dicha estrategia mediante la utilización de técnicas de aprendizaje automático. Así pues, con vistas a utilizar de forma automática el modelo del AGENTE A adquirido por el AGENTE B, se llevó a cabo un nuevo experimento en donde la decisión de disparar a portería o seguir avanzando se realiza de manera automática mediante un clasificador (C_{Au}).

Al igual que en la sección anterior, el modelo del AGENTE A y C_{Au} son sólo una parte del MRA ya que éste se basa, en gran medida, en la arquitectura del agente modelador.

Para construir C_{Au} es necesario generar una gran cantidad de datos. El proceso llevado a cabo con vistas a generar los datos necesarios para la construcción de C_{Au} se detalla a continuación:

- La situación simulada para generar los datos es la misma que se utilizó en la sección 11.4.1. Un atacante (agente modelador) y un portero (agente a modelar) interactúan un número elevado de veces.
- Una vez que el AGENTE B se encuentra en el área de 25 metros, decide aleatoriamente si disparar o no.
- Si al disparar a portería, el AGENTE B marca gol, el ejemplo es considerado como positivo. En caso contrario se considera negativo.
- Cada ejemplo está formado por 18 atributos (ver Tabla 11.4) que corresponden a los valores de algunas variables del entorno además de las predicciones

Tabla 11.4: Atributos utilizados en la creación del C_{Au}.

Nombre	Descripción
BallKickable	¿se puede disparar el balón?
OpponentX	coordenada X del oponente
OpponentY	coordenada Y del oponente
OpponentDistance	distancia al oponente
BallX	coordenada X del balón
BallY	coordenada Y del balón
MyX	coordenada X del agente modelador
MyY	coordenada Y del agente modelador
Distance_RM_Their_Goal	distancia hasta la portería
Distance_RM_LF_Flag	distancia al poste izquierdo de la portería
Distance_RM_RF_Flag	distancia al poste derecho de la portería
AngleFromBody_RM_Their_GR_Flag	ángulo a la portería
LastAction	última acción realizada
LastAngle	ángulo de la última acción
LastPower	fuerza de la última acción
OpponentActionPrediction	predicción de la acción del oponente
OAPrediction	predicción de el ángulo de la acción del oponente
OPPrediction	predicción de la fuerza de la acción del oponente
class	positivo si se marca gol, negativo en caso contrario

llevadas a cabo por el modelo del AGENTE A en el instante de tiempo en que realiza el disparo. Además de estos atributos, cada instancia es etiquetada positivo o negativo según sea el caso.

- Una vez generado los datos, se utiliza un algoritmo de aprendizaje generar el C_{Au}. En este caso se ha utilizado el algoritmo de generación de reglas PART [47].

- Por último, se incorpora el C_{Au} generado a la arquitectura del AGENTE B para decidir cuándo disparar.

Puesto que se quiere determinar la influencia de las predicciones realizadas por el modelo del AGENTE A incorporado en la arquitectura del AGENTE B, se han realizado experimentos en donde no se incluyen éstas. De esta manera, el clasificador es generado a partir de 15 atributos y la clase.

La distribución original de los datos generados en el proceso de simulación posee 7414 casos negativos y 2082 casos positivos. Al aplicar PART, se lograba un porcentaje de aciertos del 77.85 %. Pero al analizar la matriz de confusión, esta revela que la practica totalidad de los ejemplos eran clasificados como negativos. Por esta razón, se decidió incrementar el número de instancias positivas cuadriplicando las existentes para hacer un total de 8328 instancias positivas. El nuevo conjunto de datos posee 15742 instancias y el número de atributos es 19 cuando se utilizan las predicciones del modelo y de 16 cuando no se utilizan. PART obtiene un 63.49 % de aciertos sobre el conjunto de datos que no incorpora las predicciones del modelo y un 68.06 % sobre el conjunto de datos que si las incorpora. Estos

resultados y el análisis del conjunto de reglas generados, indican que la utilización de las predicciones influye positivamente a la hora de generar el C_{Au}.

Los resultados obtenidos en la generación del C_{Au} muestran que la utilización del modelo del AGENTE A incrementa el porcentaje de aciertos en el proceso de decisión de si disparar o no (positivo y negativo respectivamente). Pero como se quiere determinar si este clasificador es útil con esa precisión, se incorpora éste a la arquitectura del AGENTE B. Una vez que el C_{Au} es parte de la arquitectura del AGENTE B, se utiliza éste en el simulador con las mismas condiciones en las cuales se evaluaron el delantero sin modelo y con modelo cuyos resultados se reflejan en la sección anterior. En la Tabla 11.5 se muestran los resultados obtenidos por el delantero sin modelo, el delantero con modelo y el delantero con modelo y el C_{Au}, el cual determina la acción a llevar a cabo.

Tabla 11.5: Resultados comparativos de la utilización automática del modelo.

Delantero	Media de Goles	Media de tiros fuera
sin modelo	4.65	11.18
con modelo	5.88	10.47
con modelo + C_{Au}	5.97	9.61

Como se puede apreciar, el AGENTE B que utiliza automáticamente el modelo mediante el C_{Au} obtiene un media superior de goles. Sin embargo al aplicar un *t*-test con una significación del 95 %, no existe diferencia significativa con el AGENTE B que utiliza el modelo mediante una programación realizada a mano. Por otra parte, la utilización del modelo con C_{Au} por parte del AGENTE B es significativamente mejor que no utilizar modelo. En cuanto a la media de tiros fuera, el agente que utiliza el modelo y el C_{Au} reduce de manera significativa éstos si se compara con cualquiera de los otros dos agentes. Estos resultados indican que la utilización del C_{Au} por parte del AGENTE B, obtiene resultados, al menos, comparables a la utilización del modelo mediante la programación del comportamiento realizado por un experto.

Capítulo 12

Conclusiones y Trabajos Futuros

En esta parte de la tesis se han presentado dos enfoques para la obtención del modelo de un agente basado en la observación de su comportamiento. En entornos en donde intervienen agentes, el conocimiento que se posea sobre éstos proporciona una clara ventaja al agente que es capaz de adquirir dicho conocimiento. Existen diversas formas de obtener el modelo de un agente. Algunas de ellas equiparan las observaciones realizadas sobre el comportamiento del agente con modelos previamente construidos. Otras, asumen que el agente se comporta de manera óptima y parten de esta suposición para construir un modelo. Una manera de modelar el comportamiento de un agente es considerar a éste como una caja negra y realizar el modelado intentando inferir la relación existente entre las sus entradas y salidas. Este tipo de modelado, conocido como modelado de agentes entrada/salida (IOAM por sus siglas en inglés), ha sido aplicado con éxito en el área de modelado de usuarios.

En esta tesis se presentan dos esquemas para el modelado de agentes los cuáles entran dentro de las técnicas IOAM. Estos esquemas aplican técnicas de aprendizaje automático con la finalidad de construir el modelo del agente. En primer lugar se propone un esquema denominado *Modelado de Agentes Basado en Trazas* (MABT) en donde se asume que el agente modelador tiene acceso a los datos de entrada y salida del agente a modelar. MABT está compuesto esencialmente por dos módulos, el módulo de construcción del modelo (MCM) y el módulo de razonamiento (MRA).

Además, se extiende el esquema anterior con la finalidad de poder utilizarlo en dominios en donde no se tiene acceso directo a las entradas y salidas del agente a modelar. Esto da origen al esquema denominado *Modelado de Agentes Basado en la Observación* (MABO). Este nuevo esquema incorpora un módulo encargado de inferir las acciones llevadas a cabo por el agente a modelar basándose en información de interacciones previas (módulo de etiquetado de acciones - MEA).

Con la finalidad de evaluar el MABT se han llevado a cabo experimentos tanto

en dominios estáticos como dinámicos. Por otra parte se quiere determinar el límite superior al que se puede llegar utilizando las entradas y salidas del agente a modelar como base del proceso de modelado.

Para validar MABO, en el cuál asume que no se tiene acceso directo a las entradas y salidas del agente a modelar, se han realizado experimentos en el conocido dominio del simulador de fútbol de la *RoboCup*.

12.1. Conclusiones

Las principales conclusiones que se extraen de la evaluación de los resultados del MABT se exponen a continuación.

En la evaluación del MCM de MABT se ha observado que la utilización de técnicas de aprendizaje automático con el propósito de generar el modelo de un agente han demostrado una alta precisión. Además, en uno de los dominios de prueba, al utilizar el modelo generado como núcleo del MRA, el agente modelador se comporta de manera muy similar al agente modelado.

Sin embargo, en un dominio en donde las entradas son complejas y las acciones realizadas por el agente a modelar involucran una combinación de parámetros, se hace necesaria la utilización de un enfoque de aprendizaje automático jerarquizado como núcleo del MCM. Los resultados obtenidos en el proceso de construcción del modelo en este dominio, reflejan que la complejidad de las entradas tiende a limitar la calidad del modelo adquirido.

En cuanto al proceso de evaluación del MABO se pueden extraer las siguientes conclusiones.

Al aplicar el aprendizaje jerárquico con la finalidad de construir el núcleo del MEA se obtienen resultados diversos. Es decir, de acuerdo a la tarea de aprendizaje que estén llevando a cabo las técnicas de aprendizaje automático utilizadas, se obtienen resultados muy distintos. A pesar de esto, se puede considerar que los resultados son aceptables para el uso que se pretende dar al modelo.

En cuanto a los resultados obtenidos en la creación del modelo (MCM) se puede considerar que el aprendizaje jerárquico obtiene resultado aceptables. Pero quizás los resultados más interesantes son los obtenidos en la evaluación del MRA. A pesar de que el agente que incorpora MABO dentro de su arquitectura utiliza el modelo de una forma simple, éste obtiene mejores resultados que un agente que no utiliza MABO.

12.2. Limitaciones

Una de las principales limitaciones de MABT es que éste asume que puede tener acceso directo a las entradas/salidas del agente a modelar. Sin embargo, MABT se puede utilizar en dominios en donde esto sea cierto. Por otra parte, dependiendo de la complejidad de las entradas y/o salidas en un dominio dado, MABT puede ver limitado su rendimiento.

En cuanto a las limitaciones de MABO, quizás la más importante puede ser el tiempo necesario para construir el modelo del agente si éste se utiliza en un dominio en donde el tiempo es un factor determinante. Sin embargo, dependiendo del dominio se pueden establecer estrategias para limitar el tiempo de construcción y actualización de los modelos generados. Otra limitación de MABO, en dominios como el simulador de fútbol de la *RoboCup* es la cantidad de datos necesarios para construir el modelo.

12.3. Líneas de Trabajo Futuro

La realización de este trabajo plantea líneas de investigación que pueden ser estudiadas, desarrolladas y evaluadas en un futuro. A continuación se detallan algunas de estas líneas.

- En las evaluaciones de MABT realizadas hasta ahora, el uso de técnicas de aprendizaje automático se limita a la creación del modelo del agente. Sin embargo existen situaciones donde no basta con reemplazar el núcleo del MRA por el modelo adquirido. Para este tipo de situaciones se propone la utilización de técnicas de aprendizaje automático con la finalidad automatizar la utilización del modelo.
- La calidad del modelo generado depende en gran medida de la calidad de los datos etiquetados por el MEA. Por esta razón se propone la utilización de otras técnicas de aprendizaje y/o estadísticas con la finalidad de incrementar la calidad de los datos etiquetados por el MEA.
- Se propone la utilización del modelo adquirido en comportamientos más complejos. Puesto que el MRA utiliza de una forma simple el modelo adquirido por el agente, se propone que se utilice éste en comportamientos en donde se vean involucrados más de dos agentes. Por ejemplo, en el simulador de fútbol de la *RoboCup* se podrían crear situaciones en donde el agente modelador tenga que utiliza el modelo en una situación que involucre oponentes y/o compañeros.
- Utilización de otras técnicas de aprendizaje automático en el MRA que puedan aprovechar aún más el modelo adquirido. Un ejemplo de estas técnicas

sería el aprendizaje por refuerzo.

- El proceso de modelado realizado en el simulador de fútbol de la *RoboCup* es un modelado agente-agente, considerado de bajo nivel. Por esta razón se propone modelar series de acciones y no una acción en especifico para intentar eliminar las ambigüedades presentes en este tipo de dominios.

- Evaluación de MABT y MABO en otros dominios. Tomando en consideración las restricciones de MABT, éste puede ser utilizado en otros dominios en donde intervienen agentes. Por otra parte, MABO podría ser utilizado en entornos multiagentes como el dominio de subastas electrónicas o el de la competición de agentes *entrenadores* dentro del propio simulador de fútbol de la *RoboCup*.

Parte IV

Conclusiones Generales

Capítulo 13

Conclusiones Generales

13.1. Sumario

Como se ha mencionado en la introducción de esta memoria, existen distintas maneras de resolver un problema mediante la aplicación de técnicas que implican la utilización de la Inteligencia Artificial.

Existen entornos en donde técnicas o resolvedores de problemas deben colaborar con la finalidad de cumplimentar la tarea asignada. Dentro de estos entornos se encuentran los conocidos conjuntos de clasificadores, los cuales combinan las decisiones de un grupo de clasificadores (resolvedores de problemas) con la finalidad de llevar a cabo la tarea de asignada. Uno de los objetivos de esta tesis ha consistido en el desarrollo y experimentación de un método capaz de encontrar, basándose en un algoritmo de generación de conjuntos, un grupo óptimo de resolvedores de problemas para un problema específico. Este método, denominado *GA-Stacking*, utiliza algoritmos genéticos con el propósito de encontrar la configuración óptima de los parámetros del algoritmo de generación de conjuntos heterogéneos conocido como *Stacking*.

Para conseguir este objetivo ha sido necesario el diseño y desarrollo de las codificaciones de las soluciones, la función de *fitness* necesaria para la utilización de los AG's al igual que la evaluación del método propuesto. Finalmente, la experimentación ha permitido determinar la configuración adecuada del método propuesto con la finalidad de resolver una tarea asignada.

En otros entornos en donde los resolvedores de problemas no sólo cooperan con otros resolvedores de problemas, sino que pueden llegar a competir con éstos, cualquier información que se posea sobre los demás individuos que interactúan en el entorno resulta de mucha utilidad. En este tipo de entornos, los resolvedores de problemas poseen cierto grado de autonomía. El objetivo de este trabajo, relacionado con este tipo de resolvedores de problemas, ha sido desarrollar un esquema ge-

neral que permita adquirir el modelo de un agente basándose en el comportamiento de éste. Cabe señalar que en realidad se han desarrollado dos esquemas cuyo fin es el modelado del comportamiento de otros agentes, el *Modelado de Agentes Basado en Trazas* (MABT) y el *Modelado Basado en la Observación* (MABO). Ambos han sido evaluados experimentalmente.

Con el propósito de conseguir este objetivo, ha sido necesaria la conceptualización y desarrollo de los módulos que se incorporan a la arquitectura del agente modelador. Además, fue necesario determinar las relaciones existente entre estos módulos. Por otra parte, ha sido necesario el desarrollo de un esquema experimental que permitiese evaluar los métodos propuestos. Este marco experimental ha permitido validar el correcto funcionamiento de los métodos propuestos.

13.2. Publicaciones

En esta sección se enumeran las publicaciones a las que ha dado origen la elaboración de esta tesis doctoral.

Título:	*Empirical Evaluation of Optimized Stacking Configurations*
Autores:	Agapito Ledezma, Ricardo Aler, Araceli Sanchis, y Daniel Borrajo
Congreso:	The 16th IEEE International Conference on Tools with Artificial Intelligence
Publicación:	por publicar
Lugar de publicación:	
Año:	2004

Título:	*Predicting Opponent Actions by Observation*
Autores:	Agapito Ledezma, Ricardo Aler, Araceli Sanchis y Daniel Borrajo
Congreso:	RoboCup 2004 Symposium
Publicación:	por publicar
Lugar de publicación:	
Año:	2004

Título:	*From Continuous Behaviour to Discrete Knowledge*
Autores:	Agapito Ledezma, Fernando Fernández, Ricardo Aler
Congreso:	7th International Work-Conference on Artificial and Natural Neural Networks
Publicación:	Proceedings of IWANN 2003
Lugar de publicación:	España
Año:	2003

Título:	*Predicting Opponent actions in the RoboSoccer*
Autores:	Agapito Ledezma, Ricardo Aler, Araceli Sanchis y Daniel Borrajo
Congreso:	2002 IEEE International Conference on Systems, Man and Cybernetics
Publicación:	Proceedings of the SMC 02
Lugar de publicación:	Túnez
Año:	2002

Título:	*Heuristic Search-Based Stacking of Classifiers*
Autores:	Agapito Ledezma, Ricardo Aler y Daniel Borrajo
Libro:	Heuristic and Optimization for Knowledge Discovery
Editorial:	Idea Group Publishing
Páginas:	54-67
Lugar de publicación:	Reino Unido
Año:	2001

Título:	*Automatic Symbolic Modelling of Co-evolutionarily Learned Robot Skills*
Autores:	Agapito Ledezma, Antonio Berlanga y Ricardo Aler
Congreso:	6th International Work-Conference on Artificial and Natural Neural Networks
Publicación:	Proceedings of IWANN 2001
Páginas:	799-806
Lugar de publicación:	España
Año:	2001

Título:	*Extracting Knowledge from Reactive Robot Behavior*
Autores:	Agapito Ledezma, Antonio Berlanga y Ricardo Aler
Congreso:	AGENTS-01 Workshop on ”Learning Agents”
Publicación:	Proceeding of the AGENTS-01 Workshop on ”Learning Agents”
Páginas:	7-12
Lugar de publicación:	Canadá
Año:	2001

Título:	*Learning Models of Other Agents*
Autores:	Agapito Ledezma, Ricardo Aler, Daniel Borrajo e Inés Galván
Congreso:	AGENTS-00/ECML-00 - Workshop on ”Learning Agents”
Publicación:	Proceeding of the AGENTS-00/ECML-00 Workshop on ”Learning Agents”
Páginas:	1-5
Lugar de publicación:	España
Año:	2000

Bibliografía

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, jan 1991.

[2] K. M. Ali and M. J. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24(3):173–202, 1996.

[3] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1(72):81–138, 1995.

[4] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, 36(1):105–139, 1999.

[5] A. Berlanga, A. Sanchis, P. Isasi, and J. M. Molina. A general coevolution method to generalize autonomous robot navigation behavior. In *Proceedings of the Congress on Evolutionary Computation*, pages 769–776, La Jolla, San Diego (CA) USA, July 2000. IEEE Press.

[6] C. Blake and C. Merz. UCI repository of machine learning databases. databases http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[7] A. Blum and R. Rivest. Training a 3-node neural network is NP-complete (extended abstract). In *In Proceedings of th 1988 Workshop on Computational Learning Theory*, pages 9–18, San Francisco, CA, 1988. Morgan Kaufmann.

[8] D. Borrajo and M. Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 11(1-5):371–405, February 1997.

[9] J. G. Boticario and E. Gaudioso. A multiagent architecture for a web-based adaptive educational system. In S. Rogers and W. Iba, editors, *Adaptive User Interfaces, Papers from the 2000 AAAI Spring Symposium*, pages 24–27, Standford, CA, March 2000. AAAI Press.

[10] V. Braitenberg. *Vehicles: experiments on synthetic psychology*. MIT Press, Massachusets, 1984.

[11] P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Proceedings of the 7th European Conference on Machine Learning (ECML-94)*, pages 83–102, Cagliari, Italy, 1994. Springer-Verlag.

[12] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[13] L. Breiman. Stacked regressions. *Machine Learning*, 1(24):49–64, 1996.

[14] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[15] J. Carbonell and Y. Gil. Learning by experimentation: The operator refinement method. In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach (Volume III)*, pages 191–213. Kaufmann, San Mateo, CA, 1990.

[16] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *Proceedings of the Thirteenth National Conference on Artifial Intelligence (AAAI)*, Portland, Oregon, 1996. AAAI Press.

[17] D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proccedings of Thirteenth National Conference on Artificial Intelligence (AAAI96)*, pages 65–67, Portland, Oregon, 1996.

[18] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In M. Kaufmann, editor, *Proceedings of Twelfth International Conference on Machine Learning*, pages 90–98, 1995.

[19] E. Charniak and R. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[20] K. Cherkauer. Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 15–21, 1996.

[21] B. C. Chiu and G. I. Webb. Using C4.5 as an induction engine for agent modelling: An experiment of optimisation. In *Proceedings of the User Modelling Conference UM'97*, 1997.

[22] J. G. Cleary and L. E. Trigg. K*: an instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning*, pages 108–114, 1995.

[23] P. Cohen and H. Levesque. Teamwork. *Nous*, 35, 1991.

[24] P. Cohen, C. Perrault, and J. Allen. *Strategies for Natural Language Processing*, chapter Beyond Question Answering, pages 245–274. Lawrence Erlbaum Associates, 1981.

[25] W. W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[26] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[27] G. Demiroz and H. A. Guvenir. Classification by voting feature intervals. In *Proceedings of the 9th European Conference on Machine Learning*, pages 85–92, 1997.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 1(38):1–38, 1977.

[29] M. Devaney and A. Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proceedings of the Fithteenth National Conference on Artificial Intelligence (AAA1-98)*, pages 942–947, Madison, WI, 1998.

[30] T. G. Dietterich. Machine-learning research:four current directions. *AI Magazine*, 18(4):97–136, 1997.

[31] T. G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *Multiple Classifiers Systems: first international workshop; proceedings /MCS 2000*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15, Cagliari, Italy, June 2000. Springer.

[32] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decisions trees: Bagging, boosting and randomization. *Machine Learning*, 40(2):139–157, August 2000.

[33] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[34] C. Druecker, C. Duddeck, S. Huebner, H. Neumann, E. Schmidt, U. Visser, and H.-G. Weland. Virtualweder: Using the online-coach to change team formations. Technical report, TZI-Center for Computing Technologies, University of Bremen, 2000.

[35] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Addison-Wesley, 1973.

[36] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu. *Evolutionary Computation*. CRC Press, 2000.

[37] E. H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In V. Lesser and L. Gasser, editors, *Proceedings of the First International*

Conference on Multi-Agent Systems (ICMAS-95), pages 406–413, Menlo Park, CA, 1995. AAAI Press.

[38] E. H. Durfee, P. J. Gmytrasiewicz, and J. S. Rosenschein. The utility of embedded communications: Toward the emergence of protocols. In *Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop*, pages 85–93, 1994.

[39] E. H. Durfee, J. Lee, and P. J. Gmytrasiewicz. Overeager reciprocal rationality and mixed strategy equilibria. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 225–230, Washington, DC, USA, 1993. The AAAI Press/The MIT Press.

[40] S. Dzeroski and B. Zenko. Stacking with multi-response model trees. In J. K. Fabio Roli, editor, *Proceedings of Multiple Classifier Systems, Third International Workshop, MCS 2002*, Lecture Notes in Computer Science, Cagliari, Italy, 2002. Springer.

[41] S. Dzeroski and B. Zenko. Is combining classifiers better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[42] D. Fan, S. Stolfo, and P. Chan. Using conflicts among multiple base classifiers to measure the performance of stacking. In *Proceedings of the ICML-99 Workshop on Recent Advances in Meta-Learning and Future Work*, pages 10–17, 1999.

[43] M. Faupel. http://www.micropraxis.com/gajit/index.html, 1998.

[44] F. Fernández and D. Borrajo. VQQL. Applying vector quantization to reinforcement learning. In *RoboCup-99: Robot Soccer World Cup III*, number 1856 in Lecture Notes in Artificial Intelligence, pages 292–303. Springer Verlag, 2000.

[45] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[46] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

[47] E. Frank and I. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann, 1998.

[48] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Springer-Verlag, editor, *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, 1995.

[49] Y. Freund and R. Schapire. Experiment with a new boosting algorithm. In M. Kaufmann, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[50] J. Gama and P. Brazdil. Cascade generalization. *Machine Learning*, 41(3), 2000.

[51] R. García-Martínez and D. Borrajo. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems*, 29(1):47–78, September 2000.

[52] L. Garrido and R. Brena. The meeting scheduling game: a multiagent testbed. Technical report, Center for Artificial Intelligence ITESM-Campus Monterrey, Monterrey, México, 1998.

[53] L. Garrido, R. Brena, and K. Sycara. Towards modeling other agents: A simulation-based study. *Multi-Agent Systems and Agent-Based Simulation, LNAI Series*, 1534, 1998.

[54] L. Garrido, K. Sycara, and R. Brena. Quantifying the utility of building agents models: An experimental study. In *Proceedings of the Learning Agents Workshop at the Fourth International Conference on Autonomous Agents (Agents 2000)*, 2000.

[55] P. J. Gmytrasiewicz. *A Decision-Theoretic Model of Coordination and Communication in Autonomous Systems (Reasoning Systems)*. PhD thesis, University of Michigan, 1992.

[56] P. J. Gmytrasiewicz. An approach to user modeling in decision support systems. In *Proceedings of the Fifth International Conference on User Modeling*, pages 121–128, 1996.

[57] P. J. Gmytrasiewicz and E. H. Durfee. A rigorous, operational formalization of recursive modeling. In V. Lesser and L. Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 125–132, Menlo Park, CA, 1995. AAAI Press.

[58] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. A decision-theoretic approach to coordinating multi-agent interactions. In R. R. John Mylopoulos, editor, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 62–68, Sydney, Australia, 1991. Morgan Kaufmann.

[59] D. E. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[60] R. Goldman, C. Geib, and C. Miller. A new model of plan recognition. In K. B. Laskey and H. Prade, editors, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, 1999.

[61] L. Hansen and P. Salamon. Neural network emsembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[62] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 2nd edition, 1999.

[63] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[64] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 2ª edition, 1992.

[65] R. Howard and J. Matheson. *Readings on the Principles and Applications of Decision Analysis*, volume 2, chapter Influence diagrams, pages 719–762. Strategic Decisions Group, Menlo Park, 1984.

[66] J. Hu and M. P. Wellman. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the second International Conference on Autonomous Agents (Agents-98)*, pages 239–246, 1998.

[67] M. Huber and E. H. Durfee. On acting together: Without communication. In *Working Notes of the AAAI Spring Symposium on Representing Mental States and Mechanisms*, pages 60–71, Stanford, CA, 1995. American Association for Artificial Intelligence.

[68] M. Huber, E. H. Durfee, and M. P. Wellman. The automated mapping of plans for plan recognition. In R. L. de Mantaras and D. Poole, editors, *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 344–351, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers.

[69] M. Huber and T. Hadley. Multiple roles, multiple teams, dynamic environment: Autonomous netrek agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 332–339, Marina del Rey, CA, 1997. ACM Press.

[70] M. N. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, 1998.

[71] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Infomation Processing Letters*, 5(1):15–17, 1976.

[72] W. Iba and P. Langley. Induction of one-level decision trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, 1992.

[73] S. S. Intille and A. F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 518–525. AAAI Press, 1999.

[74] P. Jansen. *Computers, Chess and Cognition*, chapter Problematic Positions and Speculative Play, pages 169–182. Springer, New York, 1990.

[75] N. Jennings, L. Varga, R. Aarnts, J. Fuchs, and P. Skarek. Transforming standalone expert systems into a community of cooperating agents. *Engineering Applications of Artificial Intelligence*, 6(4):317–331, 1993.

[76] G. John and P. Langley. Estimating continuous distribution in bayesian classifiers. In M. Kaufmann, editor, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[77] G. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[78] G. Kaminka, M. Fidanboylu, A. Chang, and M. Veloso. Learning the sequential behavior of teams from observations. In *Proeedings of the 2002 RoboCup Symposium*, 2002.

[79] G. A. Kaminka, M. Tambe, and C. M. Hopper. The role of agent-modeling in agent robustness. In *Working Notes of AI Meets the Real-World: Lessons Learned (AIMTRW-98)*, 1998.

[80] A. Kautz and J. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI)*, pages 32–37, Menlo Park, CA, 1986. AAAI Press.

[81] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The robocup synthetic agent challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 24–49, San Francisco, CA, 1997.

[82] D. Knuth. *Sorting and Searching. Volume 3 of The Art of Computer Programming*. Addison-Wesley, 1973.

[83] R. Kohavi. The power of decision tables. In *Proceedings of the Eighth European Conference on Machine Learning*, 1995.

[84] J. F. Kolen and J. B. Pollack. Back propagation is sensitive to initial conditions. In *Advances in Neural Information Processing Systems*, pages 860–867, 1991.

[85] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[86] M. Kuzmycz. A dynamic vocabulary for student modelling. In *Proceedings of the Fourth International Conference on User Modeling*, pages 185–190, 1994.

[87] S. W. Kwok and C. Carter. Multiple decision trees. In *Uncertainty in Artificial Intelligence 4*, pages 327–335. North-Holland, Amsterdam, 1990.

[88] J. E. Laird, A. Newell, and P. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[89] P. Lanzi, W. Stolzmann, and S. Wilson, editors. *Learning Classifier Systems From Foundations to Applications*, volume 1813 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[90] M. LeBlanc and R. Tibshirani. Combining estimates in regression and classification. In *Technical Report 9318*. Deparment of Statistic, Univesity of Toronto, 1993.

[91] S. P. Lloyd. Least squares quantization in PCM. In *IEEE Transactions on Information Theory*, number 28 in IT, pages 127–135, March 1982.

[92] B. Martin. Instance-based learning : Nearest neighbor with generalization. Master's thesis, University of Waikato, 1995.

[93] C. J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1-2):33–58, 1999.

[94] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 1983.

[95] S. Minton, J. Carbonell, C. Knoblock, O. Etzioni, and Y. Gil. Explanation-Based Learning: A problem-solving perspective. *Artificial Intelligence*, 40, 1989.

[96] T. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

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

[98] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[99] A. G. Nie, A. Honemann, A. Pegam, C. Rogowski, L. Hennig, M. Diedrich, P. Hugelmeyer, S. Buttinger, and T. Steffens. The osnabrueck robocup agents project. Technical report, Institute of Cognitive Science, Osnabrueck, 2001.

[100] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multi-agent systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

[101] S. Noh and P. J. Gmytrasiewicz. Agent modeling in antiair defense. In *Proceedings of the Sixth International Conference on User Modeling*, pages 389–400, 1997.

[102] B. Parmanto, P. Munro, and H. Doyle. Improving committe diagnosis with resampling techniques. In D. S. Touretzky, M. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 882–888. MIT Press, 1996.

[103] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning*, Stanford, CA, 2000.

[104] M. E. Pollack. *Intention in Communication*, chapter Plans as Complex Mental Attitudes. MIT Press, 1990.

[105] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[106] J. Quinlan. Learning with continuous classes. In *Proceedings of the fifth Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

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

[108] J. Quinlan. Bagging, boosting, and C4.5. In A. Press and the MIT Press, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[109] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

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

[111] J. R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236–243, Amherst, MA, June 1993. Morgan Kaufmann.

[112] P. Riley and M. Veloso. *Distributed Autonomous Robotic Systems*, volume 4, chapter On Behavior Classification in Adversarial Environments, pages 371–380. Springer-Verlag, 2000.

[113] P. Riley and M. Veloso. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling (AIPS-2002)*, 2002.

[114] D. Rummelhart, J. McClelland, and the PDP Research Group. *Parallel Distributed Processing Foundations*. The MIT Press, Cambridge, MA, 1986.

[115] C. Schaffer. Cross-validation, stacking and bi-level stacking: Methods for classification learning. In P. Cheeseman and W. Oldford, editors, *Selecting models from data: Artificial Intelligence and Statistics IV*, pages 51–59. Springer-Verlag, 1994.

[116] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[117] A. K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In A. G. H. Claude Sammut, editor, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, Sidney, Australia, July 2002. Morgan Kaufmann.

[118] A. K. Seewald and J. Fürnkranz. An evaluation of grading classifiers. In F. Hoffmann, D. J. Hand, N. M. Adams, D. H. Fisher, and G. Guimarães, editors, *Advances in Intelligent Data Analysis, 4th International Conference, IDA 2001, Proceedings*, Lecture Notes in Computer Science, pages 115–124, 2001.

[119] R. Sison and M. Shimura. Student modelling and machine learning. *International Journal of Artificial Intelligence in Education*, 9:128–158, 1998.

[120] D. B. Skalak. *Prototype Selection for Composite Nearest Neighbor Classifiers*. PhD thesis, University of Massachusetts Amherst, 1997.

[121] L. Sommaruga, I. Merino, V. Matellán, and J. Molina. A distributed simulator for intelligent autonomous robots. In *In Proccedings of Fourth International Symposium on Intelligent Robotic Systems*, pages 393–399, 1996.

[122] T. Steffens. Feature-based declarative opponent-modelling in multi-agent systems. Master's thesis, Institute of Cognitive Science Osnabrück, 2002.

[123] P. Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1999.

[124] P. Stone, P. Riley, and M. Veloso. Defining and using ideal teammate and opponent agent models. In *Proceedings of the Twelfth Innovative Applications of Artificial Intelligence Conference (IAAI-2000)*, 2000.

[125] P. Stone, M. Veloso, and P. Riley. The CMUnited-98 champion simulator team. *Lecture Notes in Computer Science*, 1604:61–76, 1999.

[126] D. Suryadi and P. J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *Proceedings of the Seventh International Conference on User Modeling*, pages 223–232, Banf, CA, 1999.

[127] M. Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference , AAAI 96, IAAI 96*, pages 80–87, Portland, Oregon, 1996. The MIT Press.

[128] M. Tambe and P. Rosenbloom. RESC: An approach for real-time, dynamic agent tracking. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, (IJCAI 95)*, pages 103–111, Montréal, Québec, Canada, 1995. Morgan Kaufmann.

[129] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, Amherst, MA, June 1993. Morgan Kaufman.

[130] K. Ting and I. Witten. Stacked generalization: when does it work? In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[131] K. M. Ting. Decision combination based on the characterisation of predictive accuracy. *Intelligent Data Analysis*, 1(1-4):181–205, 1997.

[132] K. M. Ting and I. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[133] L. Todorovski and S. Dzeroski. Combining multiple models with meta decision trees. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 54–64, 2000.

[134] G. G. Towell, J. W. Shavlik, and M. O. Noordenier. Refinement of approximate domain theories by knowledge based neural network. In *Proceedings of the 8th National Conference on AI (AAAI-90)*, volume 2, pages 861–866, 1990.

[135] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–403, 1996.

[136] P. van Beek and R. Cohen. Resolving plan ambiguity for cooperative response generation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJAIT-91)*, pages 938–944, Sidney, Australia, 1991.

[137] J. M. Vidal and E. H. Durfee. Recursive agent modeling using limited rationality. In V. Lesser and L. Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 376–383, Menlo Park, CA, 1995. AAAI Press.

[138] X. Wang. Learning planning operators by observation and practice. In *Artificial Intelligence Planning Systems*, pages 335–340, 1994.

[139] R. Washington. Markov tracking for agent coordination. In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98)*, pages 70–77. ACM Press, 1998.

[140] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[141] C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[142] G. Webb and M. Kuzmycz. Feature based modelling: A methodology for producing coherent, consistent, dynamically changing models of agents's competencies. *User Modeling and User Assisted Interaction*, 5(2):117–150, 1996.

[143] G. Webb, M. Pazzani, and D. Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(19-20), 2001.

[144] I. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000.

[145] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[146] M. J. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: A survey. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents · ECAI-94 Workshop on Agent Theories, Architectures, and Languages. Proceedings*, number 890 in Lecture Notes in Computer Science, pages 1–39. Springer, 1995.

[147] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 2(10), 1995.

[148] C. Yunpeng, C. Jiang, Y. Jinyi, and L. Shi. Global planning from local perspective: An implementation of observation-based plan coordination in robocup simulation games. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer-Verlag, 2002.

[149] A. Zell, N. Mache, R. Huebner, M. Schmalzl, T. Sommer, and T. Korb. Snns: Stuttgart neural network simulator. Technical report, University of Stuttgart, Stuttgart, 1992.

[150] Z. H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than al. *Artificial Intelligence*, 137(1-2), 2002.

Apéndice A

Algoritmos de Generación de Conjuntos de Clasificadores

En este apéndice se detallan los algoritmos de construcción de conjuntos homogéneos más utilizados. En la Figura A.1 se muestra el algoritmo de generación de conjuntos de clasificadores homogéneos denominado *Bootstrap Aggregating* o *Bagging*. Por otro lado, en la Figura A.2 se detalla el algoritmo AdaBoostM1 conocido algunas veces como *Boosting*.

Algoritmo Bagging

Entradas

- Conjunto de entrenamiento, S
- Algoritmo de aprendizaje base, B
- Número de muestras *bootstrap*, T

Procedimiento

Para $i = 1$ hasta T {
  $S' =$ muestra *bootstrap* de S(S' es una muestra con reemplazo de S)
  $C = B(S')$ (crea un nuevo clasificador a partir de S')
}

$$C^*(x) = \arg\max_{y \in Y} \sum_{i C_i(x)=y} 1(etiqueta\ \text{"}\mathbf{y}\text{"}\ mayoritaria)$$

Salida

Clasificador C^*

Figura A.1: Algoritmo de generación de conjuntos homogéneos *Bootstrap Aggregating* (*Bagging*).

Algoritmo AdaBoostM1. La fórmula $[\![E]\!]$ es 1 cuando es E es cierto y 0 de otra forma

Entradas

- Conjunto de entrenamiento S, de instancias etiquetadas: $S = \{(x_i, y_i), i = 1, 2, ..., m\}$
- Clases $y_i \in Y = \{1, ..., K\}$
- Algoritmo de aprendizaje base (clasificador débil) B
- Número de iteraciones T

Procedimiento

Inicializar para todo $i : w_1(i) = 1/m$	Asigna el mismo peso a todas las instancias
Para $t = 1$ a $T\{$	
para $i : p_t(i) = w_t(i)/(\sum_i w_t(i))$	Normaliza el peso de las instancias
$C_t = B(p_t)$	Aplica el algoritmo base con los pesos normalizados
$\varepsilon_t = \sum_i p_t(i)[\![C_t(x_i) \neq y_i]\!]$	Calcula el error de C_t
si $\varepsilon_t > 1/2$ entonces	
$T = t - 1$	
termina bucle	
$\beta_t = \varepsilon_t/(1 - \varepsilon_t)$	
Para todo $i : w_{t+1}(i) = w_t(i)\beta_t^{1-[\![C_t(x_i)\neq y_i]\!]}$	cálculo de nuevos pesos
$\}$	

Salida

$$Clasificador\ C^* = \arg\max_{y \in Y} \sum_{t=1}^{T} \left(\frac{1}{\beta_t}\right) [\![C_t(x_i) = y_i]\!]$$

Figura A.2: Algoritmo de generación de conjuntos homogéneos *AdaBoostM1* (*Boosting*).

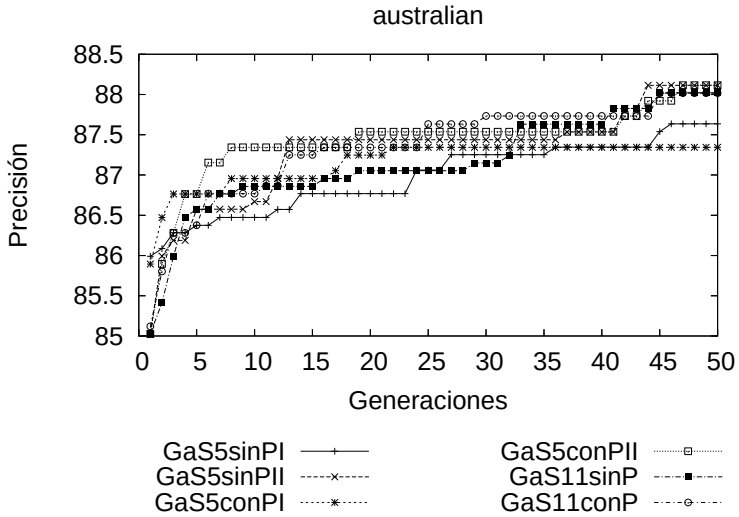
Apéndice B

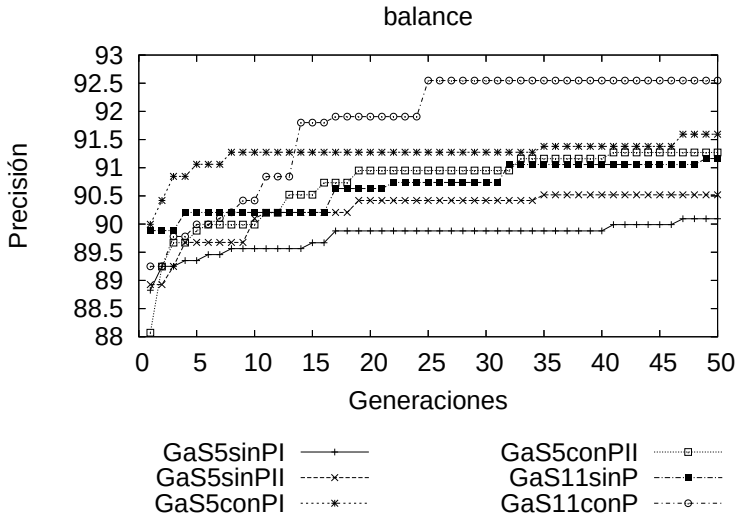
Configuraciones de *GA-Stacking*

En este apéndice se muestran los resultados obtenidos en el proceso de evaluación de las diferentes configuraciones de *GA-Stacking*. En la Tabla B.1 se puede apreciar la evolución del fitness en los dominios utilizados de las diferentes configuraciones de *GA-Stacking*. Las gráficas correspondientes a los dominios de *echo* y *sonar* no se muestran pues en la primera generación se alcanzaba el *fitness* máximo.

Por otro lado, en las Tablas B.2, B.3, B.4, B.5, B.6, B.7 se muestran los resultados de la comparación de los individuos encontrados por las diferentes configuraciones de *GA-Stacking*.

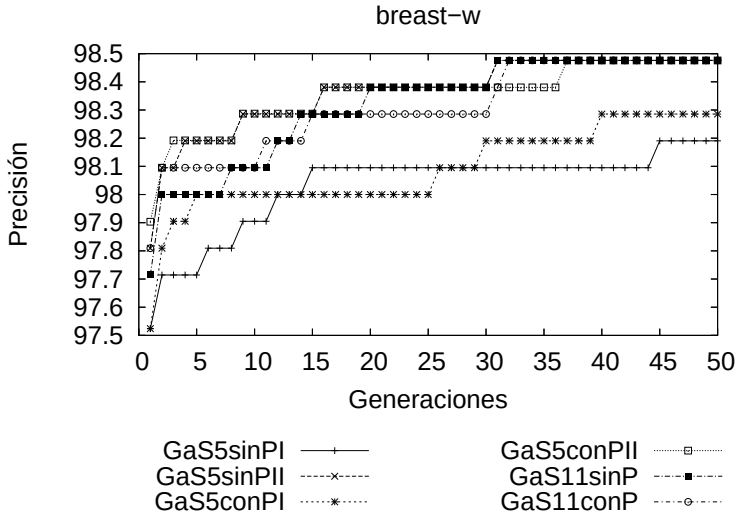
Tabla B.1: Evolución del *fitness* en los distintos dominios con cada una de las configuraciones de *GA-Stacking*. El valor reflejado es el promedio de las tres ejecuciones del algoritmo sobre el conjunto de datos.

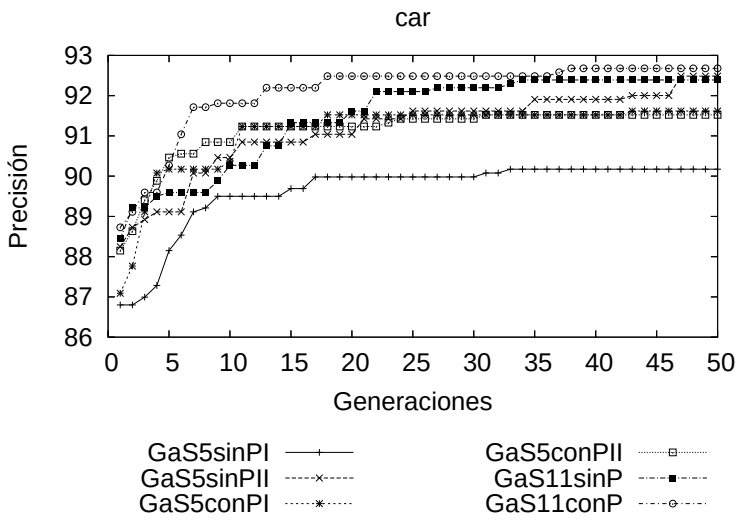




(Continúa en la siguiente página)

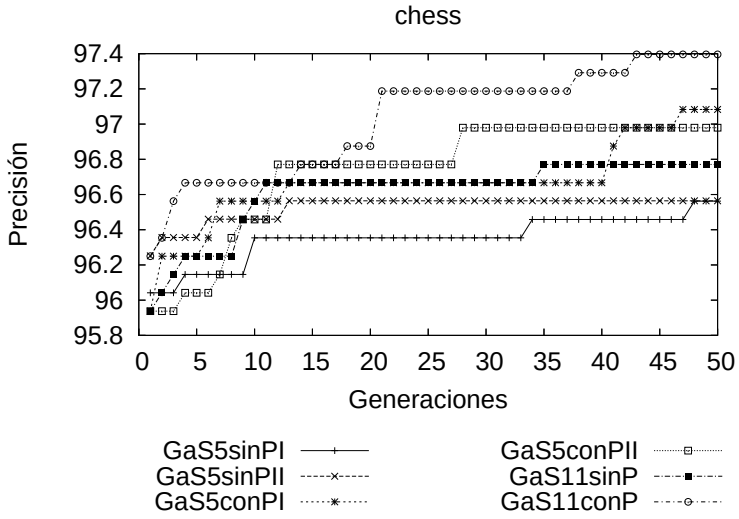
Tabla B.1. (*Continuación*)

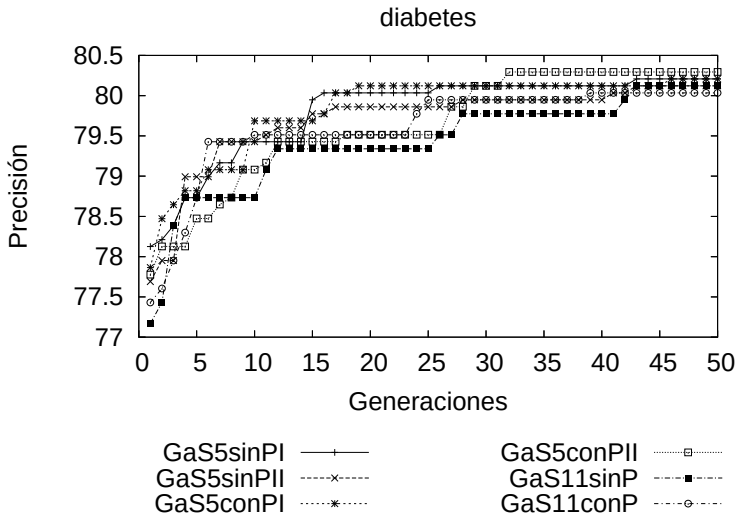




(Continúa en la siguiente página)

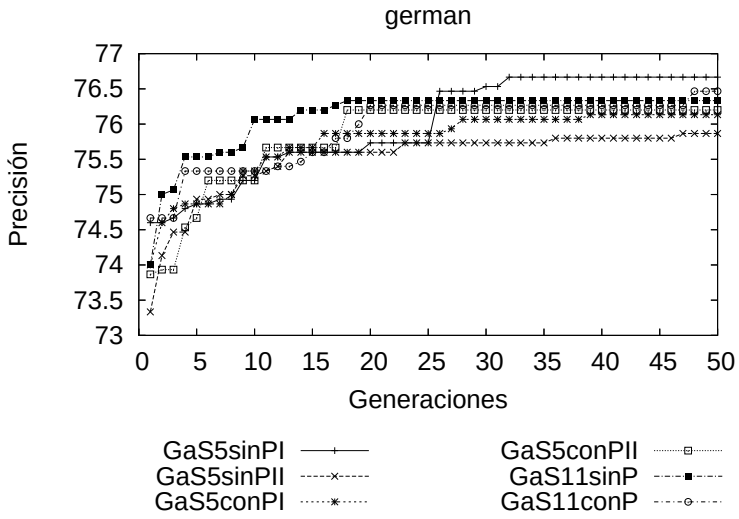
Tabla B.1. (*Continuación*)

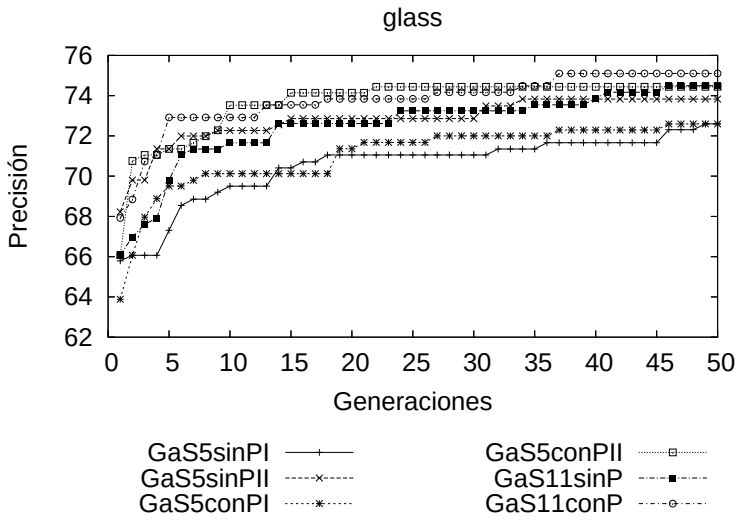




(Continúa en la siguiente página)

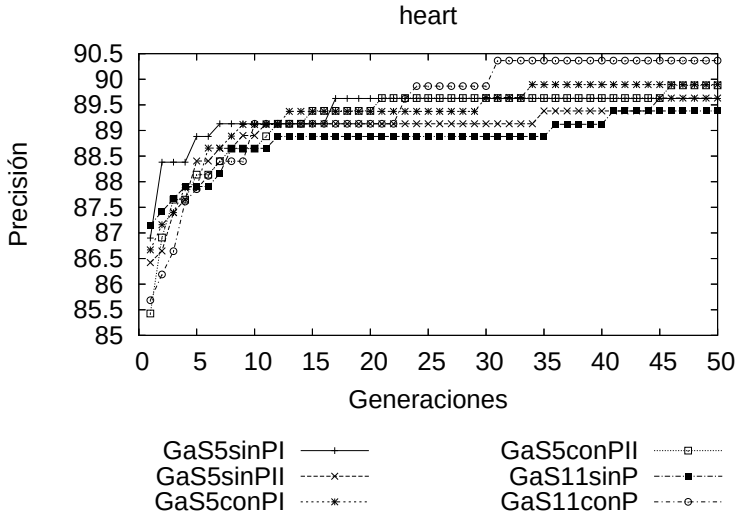
Tabla B.1. (*Continuación*)

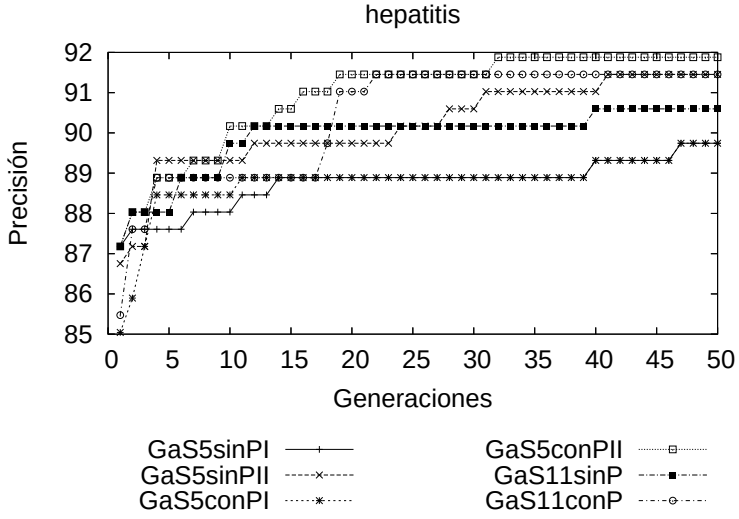




(Continúa en la siguiente página)

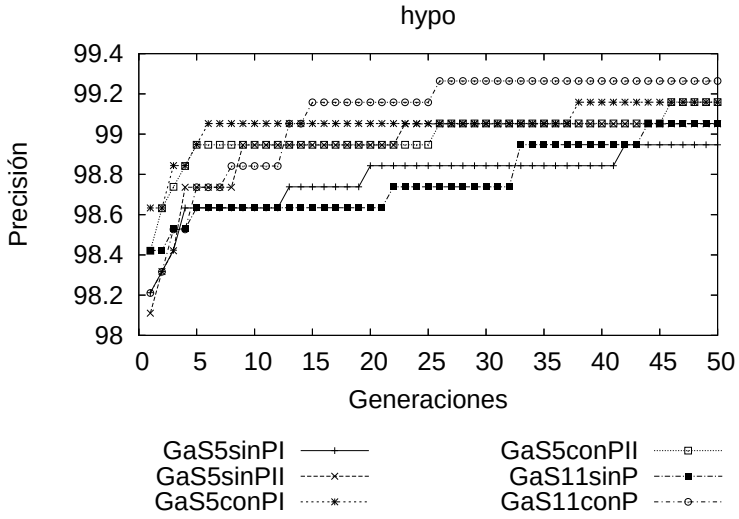
Tabla B.1. (*Continuación*)

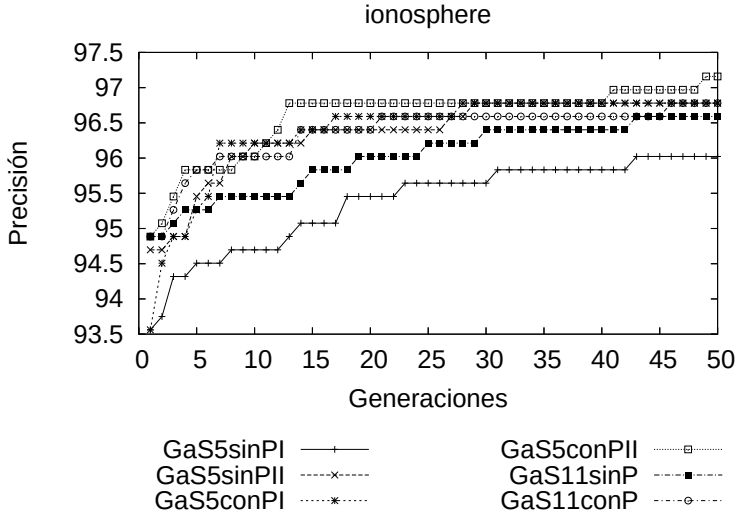




(Continúa en la siguiente página)

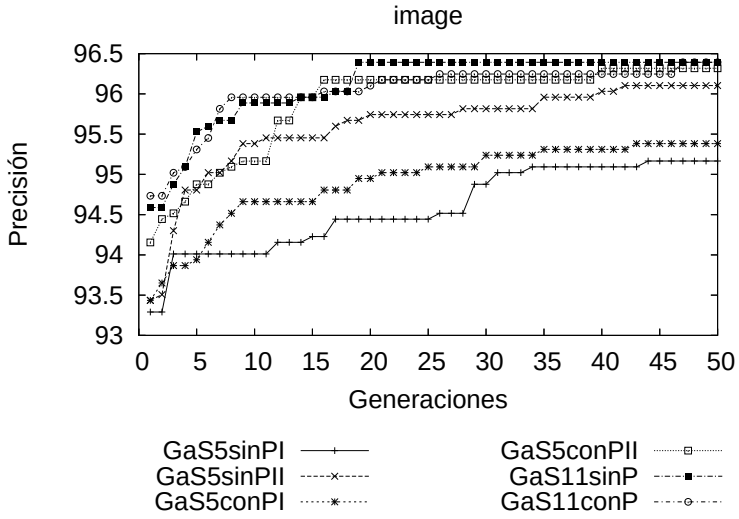
Tabla B.1. (*Continuación*)

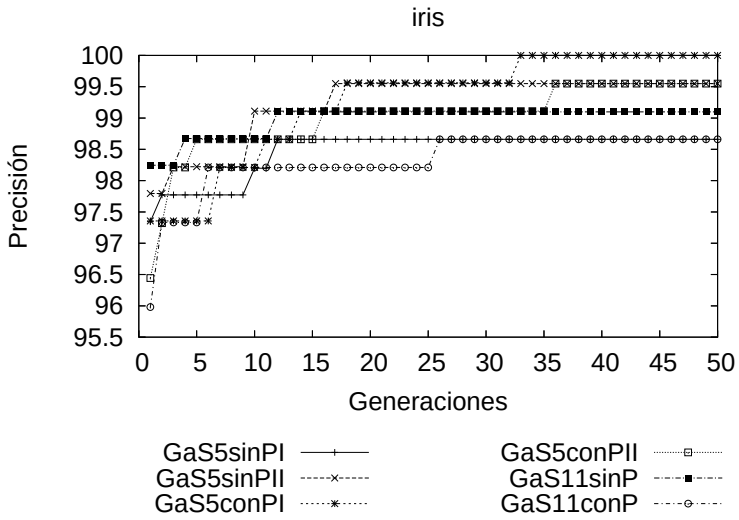




(Continúa en la siguiente página)

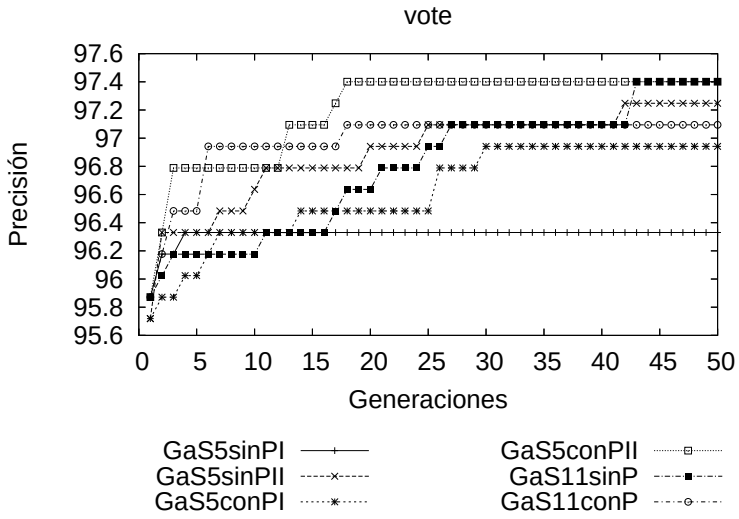
Tabla B.1. (*Continuación*)





(Continúa en la siguiente página)

Tabla B.1. (*Continuación*)



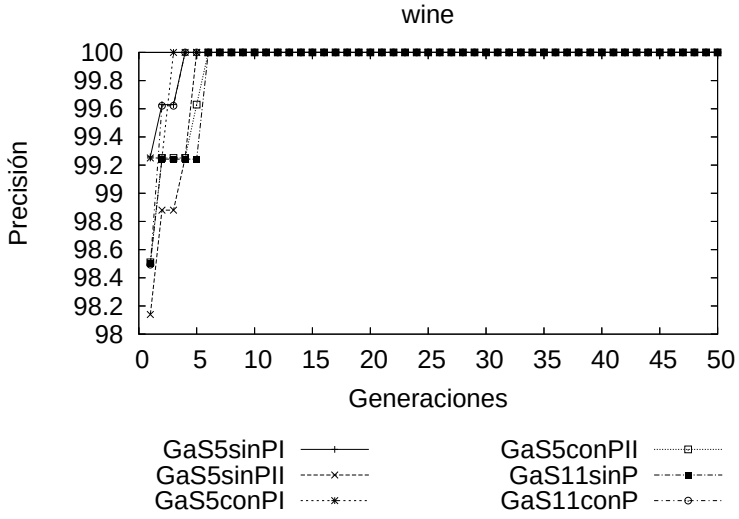


Tabla B.2: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS5SPI con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS5SPI	GAS5SPII	GAS5CPI	GAS5CPII	GAS11SP	GAS11CP
australian	86.30	87.10	88.10	87.20	88.90	88.80
balance	90.80	94.30 +	94.40 +	92.50 +	94.20 +	94.30 +
breast	95.20	95.10	97.40	95.90	96.70	96.80
car	96.20	95.50	97.30 +	97.80 +	95.80	97.70 +
chess	99.70	99.70	99.30	99.40	99.40	99.70
diabetes	73.10	73.30	75.40 +	73.30	74.90	75.40
echo	86.70	84.70	87.50	90.00	84.70	90.00
german	74.00	75.40	71.00	75.00	71.00	73.60
glass	62.50	63.50	67.70	76.00 +	66.70	74.50 +
heart	82.50	83.70	82.30	82.80	75.40 −	82.50
hepatitis	80.90	80.90	79.40	78.70	80.10	79.40
hypho	99.30	99.20	99.60	98.80 −	98.50 −	99.30
images	97.40	97.90	97.60	98.60	97.20	98.10
ionosphere	89.20	90.30	91.30	89.10	90.60	90.30
iris	93.70	92.20	93.70	92.20	94.20	97.20
sonar	97.80	94.70	91.60 −	93.60 −	97.80	95.70
vote	95.70	93.70 −	95.40	95.70	91.50 −	94.00 −
wine	94.40	96.70	98.90	97.80	94.10	92.20

Tabla B.3: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS5SPII con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS5SPII	GAS5SPI	GAS5CPI	GAS5CPII	GAS11SP	GAS11CP
australian	87.10	86.30	88.10	87.20	88.90	88.80
balance	94.30	90.80 −	94.40	92.50	94.20	94.30
breast	95.10	95.20	97.40 +	95.90	96.70	96.80
car	95.50	96.20	97.30 +	97.80 +	95.80	97.70 +
chess	99.70	99.70	99.30	99.40	99.40	99.70
diabetes	73.30	73.10	75.40	73.30	74.90	75.40
echo	84.70	86.70	87.50	90.00	84.70	90.00
german	75.40	74.00	71.00	75.00	71.00 −	73.60
glass	63.50	62.50	67.70	76.00	66.70	74.50 +
heart	83.70	82.50	82.30	82.80	75.40 −	82.50
hepatitis	80.90	80.90	79.40	78.70	80.10	79.40
hypho	99.20	99.30	99.60	98.80	98.50	99.30
images	97.90	97.40	97.60	98.60	97.20	98.10
ionosphere	90.30	89.20	91.30	89.10	90.60	90.30
iris	92.20	93.70	93.70	92.20	94.20	97.20
sonar	94.70	97.80	91.60	93.60	97.80	95.70
vote	93.70	95.70 +	95.40	95.70 +	91.50	94.00
wine	96.70	94.40	98.90	97.80	94.10	92.20

Tabla B.4: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS5CPI con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS5CPI	GAS5SPI	GAS5SPII	GAS5CPII	GAS11SP	GAS11CP
australian	88.10	86.30	87.10	87.20	88.90	88.80
balance	94.40	90.80 $-$	94.30	92.50	94.20	94.30
breast	97.40	95.20	95.10 $-$	95.90	96.70	96.80
car	97.30	96.20 $-$	95.50 $-$	97.80	95.80 $-$	97.70
chess	99.30	99.70	99.70	99.40	99.40	99.70
diabetes	75.40	73.10 $-$	73.30	73.30	74.90	75.40
echo	87.50	86.70	84.70	90.00	84.70	90.00
german	71.00	74.00	75.40	75.00	71.00	73.60
glass	67.70	62.50	63.50	76.00	66.70	74.50
heart	82.30	82.50	83.70	82.80	75.40 $-$	82.50
hepatitis	79.40	80.90	80.90	78.70	80.10	79.40
hypho	99.60	99.30	99.20	98.80 $-$	98.50	99.30
images	97.60	97.40	97.90	98.60	97.20	98.10
ionosphere	91.30	89.20	90.30	89.10 $-$	90.60	90.30
iris	93.70	93.70	92.20	92.20	94.20	97.20
sonar	91.60	97.80 $+$	94.70	93.60	97.80 $+$	95.70 $+$
vote	95.40	95.70	93.70	95.70	91.50 $-$	94.00
wine	98.90	94.40	96.70	97.80	94.10 $-$	92.20

Tabla B.5: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS5CPII con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS5CPII	GAS5SPI	GAS5SPII	GAS5CPI	GAS11SP	GAS11CP
australian	87.20	86.30	87.10	88.10	88.90	88.80
balance	92.50	90.80 $-$	94.30	94.40	94.20	94.30
breast	95.90	95.20	95.10	97.40	96.70	96.80
car	97.80	96.20 $-$	95.50 $-$	97.30	95.80 $-$	97.70
chess	99.40	99.70	99.70	99.30	99.40	99.70
diabetes	73.30	73.10	73.30	75.40	74.90	75.40
echo	90.00	86.70	84.70	87.50	84.70	90.00
german	75.00	74.00	75.40	71.00	71.00 $-$	73.60
glass	76.00	62.50 $-$	63.50	67.70	66.70	74.50
heart	82.80	82.50	83.70	82.30	75.40 $-$	82.50
hepatitis	78.70	80.90	80.90	79.40	80.10	79.40
hypho	98.80	99.30 $+$	99.20	99.60 $+$	98.50	99.30 $+$
images	98.60	97.40	97.90	97.60	97.20	98.10
ionosphere	89.10	89.20	90.30	91.30 $+$	90.60	90.30
iris	92.20	93.70	92.20	93.70	94.20	97.20
sonar	93.60	97.80 $+$	94.70	91.60	97.80 $+$	95.70
vote	95.70	95.70	93.70 $-$	95.40	91.50 $-$	94.00 $-$
wine	97.80	94.40	96.70	98.90	94.10	92.20 $-$

Tabla B.6: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS11SP con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS11SP	GAS5SPI	GAS5SPII	GAS5CPI	GAS5CPII	GAS11CP
australian	88.90	86.30	87.10	88.10	87.20	88.80
balance	94.20	90.80 $-$	94.30	94.40	92.50	94.30
breast	96.70	95.20	95.10	97.40	95.90	96.80
car	95.80	96.20	95.50	97.30 $+$	97.80 $+$	97.70 $+$
chess	99.40	99.70	99.70	99.30	99.40	99.70
diabetes	74.90	73.10	73.30	75.40	73.30	75.40
echo	84.70	86.70	84.70	87.50	90.00	90.00
german	71.00	74.00	75.40 $+$	71.00	75.00 $+$	73.60
glass	66.70	62.50	63.50	67.70	76.00	74.50
heart	75.40	82.50 $+$	83.70 $+$	82.30 $+$	82.80 $+$	82.50 $+$
hepatitis	80.10	80.90	80.90	79.40	78.70	79.40
hypho	98.50	99.30 $+$	99.20	99.60	98.80	99.30 $+$
images	97.20	97.40	97.90	97.60	98.60	98.10
ionosphere	90.60	89.20	90.30	91.30	89.10	90.30
iris	94.20	93.70	92.20	93.70	92.20	97.20
sonar	97.80	97.80	94.70	91.60 $-$	93.60 $-$	95.70
vote	91.50	95.70 $+$	93.70	95.40 $+$	95.70 $+$	94.00
wine	94.10	94.40	96.70	98.90 $+$	97.80	92.20

Tabla B.7: Comparación de los resultados obtenidos por los individuos seleccionados mediante GAS11CP con el resto de los individuos encontrados con las demás configuraciones de *GA-Stacking* y su significación estadística ($+/-$ es mejor/peor significativamente).

Dominio	GAS11CP	GAS5SPI	GAS5SPII	GAS5CPI	GAS5CPII	GAS11SP
australian	88.80	86.30	87.10	88.10	87.20	88.90
balance	94.30	90.80 $-$	94.30	94.40	92.50	94.20
breast	96.80	95.20	95.10	97.40	95.90	96.70
car	97.70	96.20 $-$	95.50 $-$	97.30	97.80	95.80 $-$
chess	99.70	99.70	99.70	99.30	99.40	99.40
diabetes	75.40	73.10	73.30	75.40	73.30	74.90
echo	90.00	86.70	84.70	87.50	90.00	84.70
german	73.60	74.00	75.40	71.00	75.00	71.00
glass	74.50	62.50 $-$	63.50 $-$	67.70	76.00	66.70
heart	82.50	82.50	83.70	82.30	82.80	75.40 $-$
hepatitis	79.40	80.90	80.90	79.40	78.70	80.10
hypho	99.30	99.30	99.20	99.60	98.80 $-$	98.50 $-$
images	98.10	97.40	97.90	97.60	98.60	97.20
ionosphere	90.30	89.20	90.30	91.30	89.10	90.60
iris	97.20	93.70	92.20	93.70	92.20	94.20
sonar	95.70	97.80	94.70	91.60 $-$	93.60	97.80
vote	94.00	95.70 $+$	93.70	95.40	95.70 $+$	91.50
wine	92.20	94.40	96.70	98.90	97.80 $+$	94.10

Apéndice C

Detalles de la Evaluación del MABO

En este apéndice se muestran en detalle las características de los datos utilizados con el propósito de evaluar el proceso de generación del módulo de etiquetado de acciones. En la Tabla C.1 se muestran los atributos del conjunto de datos utilizados y la descripción de los mismos.

Tabla C.1: Atributos utilizados en el proceso de construcción de los clasificadores que son el núcleo del MEA dentro de MABO.

Nombre	**Descripción**
	ATRIBUTOS DEL INSTANTE t
SeeOpponent	¿puedo ver al oponente?
OpponenteNumber	número del oponente
BallKickableForOpponent	¿puede el oponente disparar el balón?
CanFaceOpponentWithNeck	¿puedo encarar al oponente girando el cuello?
CanSeeOpponentWithNeck	¿puedo ver al oponente girando el cuello?
BallMoving	¿el balón se esta moviendo?
BallKickable	¿puedo disparar el balón?
OpponentPositionValid	grado de certeza sobre la posición del oponente
OpponentDistance	distancia al oponente
OpponentSpeed	velocidad del oponente
OpponentAngleFromBody	ángulo del oponente desde mi cuerpo
OpponentAngleFromNeck	ángulo del oponente desde mi cuello
BallPositionValid	grado de certeza sobre la posición del balón
BallSpeed	velocidad del balón
BallDistance	distancia del balón
BallAngleFromBody	ángulo del balón desde mi cuerpo
BallAngleFromNeck	ángulo del balón desde mi cuello
MyBodyAng	ángulo de mi cuerpo
MySpeed	mi velocidad
MyAction	mi acción
MyActionAngle	el ángulo asociado a mi acción
MyActionPower	el poder asociado a mi acción
	ATRIBUTOS DEL INSTANTE $t-1$
Iguales a los anteriores	se utilizan los mismos que en el instante t
	ATRIBUTOS CALCULADOS
DIF-BKFO	diferencia en dos instantes de tiempo del atributo BallKickableForOpponent
DIF-CFOWN	diferencia en dos instantes de tiempo del atributo CanFaceOpponentWithNeck
DIF-CSOWN	diferencia en dos instantes de tiempo del atributo CanSeeOpponentWithNeck
DIF-BM	diferencia en dos instantes de tiempo del atributo BallMoving
DIF-BK	diferencia en dos instantes de tiempo del atributo BallKickable
DIF-OX	diferencia en dos instantes de tiempo de la coordenada X del oponente
DIF-OY	diferencia en dos instantes de tiempo de la coordenada Y del oponente
DESP-O	desplazamiento del oponente de un instante de tiempo a otro
DIF-OD	diferencia en dos instantes de tiempo del atributo OpponentDistance
DIF-OS	diferencia en dos instantes de tiempo del atributo OpponentSpeed
DIF-OAFB	diferencia en dos instantes de tiempo del atributo OpponentAngleFromBody
DIF-OAFN	diferencia en dos instantes de tiempo del atributo OpponentAngleFromNeck
DIF-BX	diferencia en dos instantes de tiempo de la coordenada X del balón
DIF-BY	diferencia en dos instantes de tiempo de la coordenada Y del balón
DESP-Ball	desplazamiento del balón de un instante a otro
DIF-BS	diferencia en dos instantes de tiempo del atributo BallSpeed
DIF-BD	diferencia en dos instantes de tiempo del atributo BallDistance
DIF-BAFB	diferencia en dos instantes de tiempo del atributo BallAngleFromBody
DIF-BAFN	diferencia en dos instantes de tiempo del atributo BallAngleFromNeck
DIF-MyX	diferencia en dos instantes de tiempo de mi coordenada X
DIF-MyY	diferencia en dos instantes de tiempo de mi coordenada Y
DESP-My	mi desplazamiento de un instante a otro
DIF-MyBA	diferencia en dos instantes de tiempo del atributo MyBodyAng
DIF-MyS	diferencia en dos instantes de tiempo del atributo MySpeed
CLASS	acción llevado a cabo por el agente a modelar en el instante $t-1$