Tesis Doctoral

Metodología para la definición y uso de sistemas multiclasificadores basados en la especialización de múltiples clasificadores autónomos.

Universidad de Castilla-La Mancha

Abril 2013

Autor

Ester del Castillo Herrera
Licenciada en Informática

Director

Dr. Luis Jiménez Linares

Autor:
ESTER DEL CASTILLO HERRERA

Director:
DR. LUIS JIMÉNEZ LINARES

LICENCIA:
©Ester del Castillo Herrera. Se permite la copia y distribución de la totalidad o parte de este documento sin ánimo de lucro. Toda copia total o parcial deberá citar expresamente el nombre del autor, de la Universidad de Castilla-La Mancha y deberá incluir esta misma licencia, añadiendo, si es copia literal, la mención *"copia literal"*.

Se autoriza la modificación y traducción de la obra sin ánimo de lucro siempre que se haga constar en la obra resultante de la modificación el nombre de la obra originaria, el autor de la obra originaria y el nombre de la Universidad de Castilla-La Mancha. La obra resultante también deberá ser libremente reproducida, distribuida, comunicada al público y transformada en términos similares a los expresados en esta licencia.

Este documento fue maquetado con LaTeX. Imágenes generadas con Blender/Yafray, compuestas con Gimp y OpenOffice.

Resumen

Un clasificador es un sistema que ha aprendido las características de un modelo, usando un conjunto de datos, para predecir las características de nuevos datos. En los últimos años han aparecido numerosos trabajos en los que se obtienen diferentes clasificadores base que representan al modelo, de forma que la clasificación se produce como combinación de las predicciones de los clasificadores base. En la literatura encontramos diferentes nombres para estos sistemas: agregación de clasificadores, fusión de clasificadores, etc. Llamaremos sistema multiclasificador a un clasificador que construye un conjunto de clasificadores base y combina sus predicciones de alguna forma para obtener la clasificación de un dato.

Un caso particular de estos sistemas es cuando los clasificadores base se generan para ser especialistas en algún subdominio del problema, de forma que el conjunto de especialistas modelan, en su conjunto, el problema original. En este caso, la clasificación de un dato requiere de la selección del especialista o especialistas que mejor modelan las características del dato, para proceder a la combinación de sus predicciones. Decimos que estos sistemas tienen un mecanismo de selección dinámica de expertos.

Este trabajo presenta un sistema multiclasificador que genera especialistas en subdominios y utiliza selección dinámica de expertos. La generación de expertos se ha planteado desde la perspectiva de sistemas de enjambres, donde los clasificadores base son agentes autónomos que exploran el espacio de datos y se especializan en subzonas del dominio. Se han desarrollado tres modelos con estas características, consiguiendo, en el modelo final, obtener un conjunto de agentes clasificadores que cubren el dominio total del problema. Además, se ha definido un método de reducción del conjunto de expertos que consigue que el conjunto reducido mantenga la propiedad de definir el dominio general.

Respecto a la clasificación del sistema, se han definido las predicciones de los expertos de forma difusa, lo que permite realizar un análisis completo de sus respuestas.

Agradecimientos

Mi agradecimiento a quienes, de un modo u otro, han hecho posible la realización de esta tesis, especialmente a mi director Dr. D Luis Jiménez Linares por su paciencia y aliento durante este tiempo. Mi agradecimiento también a mis compañeros del Departamento de Tecnologías y Sistemas de Información, a la Escuela Superior de Informática de Ciudad Real, al servicio de Supercomputación y, en general, a la Universidad de Castilla-La Mancha por el apoyo y los medios que me han brindado para poder realizar este trabajo.

Finalmente, y no por ello menos importante, quiero agradecer a mi marido su lucha, empeño y apoyo en todos los momentos difíciles por los que ha pasado esta tesis. A mis hijas Ana y Marina, por prescindir de su madre en sus ratos de ocio. A mis padres y hermanas que me han apoyado y vivido conmigo esta experiencia.

Índice general

1. Objetivos **1**

2. Sistemas Multiclasificadores **7**
2.1. Introducción a los sistemas multiclasificadores 7
2.2. Generación de clasificadores básicos 11
2.3. Métodos para la combinación de clasificadores 17
2.4. Algunos métodos relevantes en los sistemas multiclasificadores 20
2.4.1. Votación 20
2.4.2. Plantillas de decisión 21
2.4.3. Mezcla de expertos 22
2.4.4. Stacked generalization 24
2.4.5. Meta Aprendizaje 25
2.5. Generación o identificación de especialistas 26
2.5.1. Propuesta: generación y combinación de clasificadores especialistas evolutivos 28

3. Modelos multiclasificadores basados en Sistemas de Enjambres **31**
3.1. Introducción 31
3.2. Algoritmo de agrupamiento de Lumer y Faieta. Variantes 35
3.3. Reducción del conjunto de entrenamiento 44
3.4. Modelo de multiclasificador basado en el algoritmo LF (MBLF) . 47
3.4.1. Estructura de un agente clasificador 48
3.4.2. Creación de un agente 50
3.4.3. Entrenamiento de un agente 50
3.4.4. Sistema multiclasificador 56
3.4.4.1. Generación del conjunto de agentes 57
3.4.4.2. Selección y combinación de agentes 60
3.4.5. Experimentos 62
3.5. Modelo de multiclasificador con memoria local (McML) 64
3.5.1. Estructura del agente 64
3.5.2. Definición de la zona de especialización del agente 65
3.5.3. Criterios para la evaluación de la calidad del agente . . . 69
3.5.4. Creación de un agente 71
3.5.5. Entrenamiento de un agente 72
3.5.6. Uso de un agente 76
3.5.7. Sistema multiclasificador 80
3.5.7.1. Generación del conjunto de agentes 80
3.5.7.2. Multiclasificador asociado 84
3.5.8. Experimentos 87

4. Sistema multiclasificador autoorganizativo difuso (MAD) 91
4.1. Estructura del agente 92
4.2. Criterios para la evaluación de la calidad del agente 93
4.3. Creación del agente 95
4.4. Entrenamiento de un agente 95
4.5. Uso de un agente 97
4.5.1. Clasificación de un dato por un agente 97
4.6. Generación del conjunto de agentes 100
4.6.1. Creación e inicialización del conjunto de agentes 101
4.6.2. Entrenamiento básico del conjunto de agentes 105
4.6.3. Generación de nuevos agentes para zonas no cubiertas . 112
4.6.4. Reducción del conjunto agentes 115
4.7. Multiclasificador asociado 124
4.7.1. Selección de los mejores agentes para un dato 124
4.7.2. Combinación de los mejores agentes 129
4.8. Evaluación del sistema multiclasificador 131

5. Experimentos 135
5.1. Datos con clases bien definidas 136
5.2. Datos con clases que no están bien definidas 141
5.3. Bases de datos del repositorio UCI 145

6. Conclusiones 149
6.1. Compendio de aportaciones 149
6.2. Trabajos futuros 153

Apéndice 155

A. Sistemas multiclasificadores generados 157
A.1. Conjunto de agentes para D2C5 157
A.2. Conjunto de agentes para D2C4 1c2z 161
A.3. Conjunto de agentes para D2C5 z2clases 165
A.4. Conjunto de agentes para Cone Torus 168
A.5. Conjunto de agentes para Iris 173
A.6. Conjunto de agentes para Pendigits 174
A.7. Conjunto de agentes para Wine 178

Índice de figuras

2.1. Taxonomía propuesta para los sistemas multiclasificadores . . . 10
2.2. Boostrapping-Bagging. 13
2.3. Boosting. 14
2.4. Stacked generalization 23

3.1. Grafico Kibler 45
3.2. Zona de especialización y subzonas de especialización 68

4.1. Estructura de un agente 93
4.2. Efecto del parámetro k en el valor de la calidad del agente V_c . . 95
4.3. Selección de dos agentes para un valor de corte de 0,8 128
4.4. Selección de tres agentes para un valor de corte de 0,8 129
4.5. Combinación de tres agentes para clasificar un dato x 132

5.1. Datos para el conjunto D2C5 136
5.2. Agentes creados para D2C5 138
5.3. Agentes obtenidos en la primera iteración del EB para D2C5 . . 138
5.4. Datos para el conjunto D2C4 1c2z 140
5.5. Agentes después de aplicar la reducción para D2C4 1c2z 140
5.6. Datos para el conjunto D2C5 1z2clases 141
5.7. Datos para el conjunto Cone Torus 143
5.8. Datos en el ciclo de detección de zonas no cubiertas para Cone Torus 145

Índice de tablas

3.1. Resultado de aplicar K-vecinos a las bases de datos Iris y Pendigits 62
3.2. Resultado de aplicar el modelo MBLF con 30 agentes a Iris . . . 63
3.3. Resultado de aplicar el modelo MBLF con 1150 agentes a Pendigits . . . 63
3.4. Resultado de aplicar el modelo McML para Pendigits con 187 agentes para 1, 4 y 7 agentes seleccionados . . . 89
3.5. Resultado de aplicar el modelo McML para Pendigits con 187 agentes y 10, 13 y 16 agentes seleccionados . . . 89

5.1. Distribución normal para los datos D2C5 . . . 136
5.2. Resultados para los datos D2C5 . . . 137
5.3. Clases de equivalencia en la reducción para los datos D2C5 . . 139
5.4. Distribución normal para los datos D2C4 1c2z . . . 139
5.5. Resultados para los datos D2C4 1c2z . . . 140
5.6. Distribución normal para los datos D2C5 1z2clases . . . 141
5.7. Resultados para los datos D2C5 1z2clases . . . 142
5.8. Resultados para los datos Cone Torus . . . 143
5.9. Composición de los agentes en las tres fases para los datos Cone Torus . . . 144
5.10. Clases de los agentes en cada zona detectada para los datos Cone Torus . . . 144
5.11. Características de Iris, Pendigits y Wine . . . 146
5.12. Resultados para Iris . . . 146
5.13. Resultados para Wine . . . 146
5.14. Resultados usando los algoritmos Bl-1, K* y el modelo MAD . . 147

Lista de algoritmos

1. Bagging 11
2. AdaBoost 15
3. Stacked Generalization 24
4. Algoritmo LF 38
5. AntClass 42
6. Generación de los agentes en el modelo basado en el algoritmo LF 52
7. Actualizar agente: modelo MBLF 55
8. Inicialización de la memoria de un agente: modelo McML 71
9. Entrenamiento de un agente: modelo McML 72
10. Enfriamiento simulado 73
11. Entrenamiento de un agente con enfriamiento 75
12. Actualizar agente: modelo McML 76
13. Generación del conjunto de agentes usando enfriamiento 81
14. Inicialización de un agente: modelo MAD 95
15. Entrenamiento de un agente con un dato: modelo MAD 96
16. Generación de agentes: modelo MAD 102
17. Obtención del número de agentes y tamaño de la memoria: modelo MAD 103
18. Creación e inicialización del conjunto de agentes: modelo MAD . 105
19. Asignación de un dato a los agentes con comportamiento colaborativo 106
20. Asignación de un dato a los agentes con comportamiento competitivo, aplicando el criterio de pertenencia 107
21. Asignación de un dato a los agentes con comportamiento competitivo, aplicando el criterio de calidad 108
22. Asignación de un dato a los agentes con un comportamiento autónomo 109
23. Iteración en el entrenamiento básico del conjunto de agentes . . 111
24. Entrenamiento básico del conjunto de agentes 112
25. Generación de agentes para zonas no cubiertas 115
26. Similitud de dos agentes respecto a su zona 117
27. Similitud de dos agentes respecto a la clasificación 120
28. Obtención de las clases de equivalencia 122

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

1

Objetivos

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

1 Objetivos

El reconocimiento de patrones [95] es el conjunto de técnicas y modelos que permiten identificar la pertenencia de un determinado objeto, desconocido o parcialmente observado, a una clase o colección bien definida. Esto permite inducir todo el conocimiento general sobre la clase y aplicarlo al objeto estudiado.

Para realizar esta acción de reconocimiento, es necesario definir y establecer las clases a estudiar. Partiendo de un conjunto observado de objetos, y sin ninguna información previa que caracterice a cada clase, el primer proceso que se ha de realizar es la agrupación de los objetos en conjuntos, con la propiedad de que se maximicen las similitudes entre sus miembros. Este proceso se denomina aprendizaje automático no supervisado o clustering y proporciona las herramientas necesarias para obtener una colección de clases a partir de un conjunto de objetos observados. Fijadas las clases de interés, es necesario explicitarlas y definir el criterio de pertenencia de un determinado objeto a una determinada clase. Este proceso concluye con la especificación de la función característica o de pertenencia de cada una de las clases, con la que podremos concluir si un objeto desconocido pertenece o no pertenece a una determinada clase. Es la inducción y definición de esta función y su uso lo que constituye el núcleo del aprendizaje de clasificadores.

Si la inducción de la función característica de cada clase se realiza mediante el análisis de parejas de datos (objeto,clase) estaremos ante técnicas denominadas de aprendizaje supervisado. El aprendizaje supervisado es una técnica de aprendizaje automático que, partiendo de un conjunto de ejemplos $S = \{(x_1, y_1) \dots (x_m, y_m)\}$, genera una hipótesis sobre una función desconocida $f(x) = y$ donde a cada objeto x le asigna una clase y. El valor x_i es un vector con n atributos en la forma $x_i = \{x_{i,1} \dots x_{i,n}\}$ y el valor y_i es la clase asociada al dato x_i, donde $y_i \in C$, siendo C un conjunto finito y discreto que denota al conjunto de clases posibles. Esta hipótesis generada es un clasi-

ficador, el cual, dado un valor nuevo x, establece el valor correspondiente y.

Un sistema multiclasificador es formalmente un clasificador que obtiene la clase de un objeto mediante la combinación de las decisiones individuales de un conjunto de clasificadores base, al aplicar cada clasificador al objeto. En los últimos años se han desarrollado distintos estudios teóricos y prácticos [41, 55, 59, 73], que demuestran que un sistema multiclasificador mejora a un sistema con un único clasificador si se consigue que los clasificadores base sean lo suficientemente exactos en sus predicciones y, al mismo tiempo, estos clasificadores sean diversos, en el sentido de generar errores no correlacionados. Merz [73] define esta propiedad de la siguiente forma: *Las mejores condiciones para combinar ocurren cuando los modelos aprendidos son lo suficientemente exactos, pero lo suficientemente independientes en los errores que cometen.*

En la mayoría de los sistemas multiclasificadores que aparecen en la literatura, cada clasificador se entrena en el dominio total, con lo que cualquier dato nuevo puede ser clasificado por todos los clasificadores que componen el sistema. En general, se habla de fusión de clasificadores cuando el sistema multiclasificador utiliza las predicciones de todos los clasificadores base para obtener la clasificación final. En contrapartida, se habla de selección dinámica de clasificadores cuando el sistema aprende y determina las zonas específicas del dominio en el que es mejor cada clasificador, de forma que, dado un nuevo dato a clasificar, el sistema elije solamente aquellos clasificadores que previsiblemente predicen mejor en la zona del dato nuevo. La principal ventaja de una selección dinámica es que el coste de clasificación decrece siempre que el procedimiento de selección no sea demasiado complejo.

Aunque la detección de las zonas donde cada clasificador es experto tiene un coste computacional, esto se realiza una sola vez cuando los clasificadores se construyen. El problema de estos métodos es que si los clasificadores se entrenan en el dominio total, es posible que no haya grandes diferencias, respecto al ratio de error, en diferentes zonas del dominio con lo que, en la práctica, no hay clasificadores realmente expertos en zonas específicas.

Siguiendo estos principios, el objetivo general de esta tesis es **diseñar y definir un sistema multiclasificador formado por un conjunto de agentes clasificadores autónomos que se especialicen en zonas específicas del dominio estudiado**. Con la especialización de cada agente se pretende que el mecanismo de combinación de clasificadores no incluya las decisiones de todos ellos, sino que se aplique un mecanismo de selección dinámica del mejor o mejores agentes para clasificar un nuevo dato.

A diferencia de los sistemas multiclasificadores enmarcados en la selección dinámica, el sistema que se propone generará explícitamente expertos en subdominios, con lo que no necesita un aprendizaje de las

zonas de cada clasificador base. El sistema multiclasificador no necesitará que todos los clasificadores sepan predecir la clase de un dato. En nuestro caso, debemos garantizar que al menos existe un clasificador que lo clasifica correctamente.

La especialización de clasificadores en zonas específicas del dominio de datos permite afrontar problemas con grandes cantidades de datos, segmentándolos y paralelizando actividades, acciones estas que mejoran los métodos tradicionales ineficientes con este tipo de problemas. De igual forma, para la generación de estos clasificadores especialistas se propone utilizar el marco de la inteligencia de enjambres [9, 65, 77], que simula el comportamiento de sociedades de insectos, los cuales actúan de forma autónoma y sus acciones producen unos resultados emergentes y colectivos.

El sistema multiclasificador propuesto, por tanto, se basa en mantener el concepto de selección dinámica pero forzando a cada clasificador a ser experto en zonas específicas del dominio, es decir, haciendo relevantes los clasificadores solo en esas zonas y asegurando que el conjunto de clasificadores generados cubre el total del dominio. Las ventajas que este planteamiento pone de manifiesto son:

- Los clasificadores necesitan menos datos para ser entrenados.
- No se necesita un procedimiento externo de identificación de expertos y son generados explícitamente.
- La selección del mejor o mejores expertos para cada dato es sencilla se realiza mediante la asignación del dato a una zona y elección de los expertos de esa zona.
- Posibilidad de generación distribuida y paralela de los expertos.

Respecto al método por el cual se generan los expertos, una novedad importante es que cada clasificador deberá seleccionar los datos que servirán para su entrenamiento, moviéndose en un entorno cambiante. En este sentido, la construcción de los expertos es un proceso dinámico y emergente, que puede ser autoajustado por cada agente o reajustado por un proceso externo.

Subobjetivos parciales

Fijado el marco y el objetivo general de esta tesis, para alcanzarlo se establece el siguiente conjunto de subobjetivos parciales que se desarrollarán a lo largo del trabajo y que configuran la estructura de esta memoria.

Los subojetivos son:

Estudiar y analizar el estado del conocimiento de los sistemas multiclasificadores. Enmarcar el trabajo desarrollado dentro del entorno de los sistemas multiclasificadores; introducir los conceptos generales de los sistemas multiclasificadores; las distintas alternativas y métodos que se desarrollan en la actualidad, incidiendo en las dificultades observadas y criterios que orientarán la solución propuesta.

Definir y crear los agentes clasificadores básicos. Las características que se proponen para el agente clasificador se basan en la idea de autonomía, es decir, que el propio agente sea capaz de guiar su entrenamiento para lograr ser especialista en un subdominio del problema. Los datos de entrenamiento estarán disponibles para todos los agentes pero será el agente quien va a decidir si un dato está en el subdominio en el que se está especializando y si su uso como dato de entrenamiento mejora su calidad. El agente se adaptará al entorno, modificando en su entrenamiento la zona específica donde se está especializando. En general, el agente deberá tomar decisiones que le lleven a especializarse en zonas homogéneas que contengan el menor número de clases posibles.

Definir el sistema multiclasificador con múltiples agentes especialistas. Este sistema debe generar un conjunto de agentes clasificadores que se especialicen en zonas específicas, de forma que se garantice que el total de los agentes cubre el dominio original. La opinión del agente especialista solo es tenida en cuenta si el sistema decide que debe intervenir en la predicción de la clase de un dato. La gestión dinámica de los agentes en la fase de entrenamiento deberá posibilitar al sistema multiclasificador la capacidad de crear, destruir o modificar los agentes obtenidos.

Probar y validar el sistema multiclasificador. Una vez definido el sistema multiclasificador, se ha de implementar y estudiar su comportamiento con diferentes casos de pruebas, confrontando los resultados obtenidos con los esperados; así como comparándolos con los resultados de otros clasificadores.

El desarrollo de esta memoria se orienta a la culminación de cada uno de los distintos subobjetivos propuestos, por lo que en el Capítulo 2 presentaremos el estado del conocimiento de los sistemas multiclasificador, fijando una taxonomía para enmarcar los elementos y conceptos relevantes que permiten contextualizar nuestra propuesta. En el Capítulo 3 presentamos varias propuestas de sistemas multiclasificadores, fundamentadas en los algoritmos de agrupamiento de Lumer [65], como una primera aproximación al modelo de sistema multiclasificador buscado. Tras una pequeña introducción a las técnicas de agrupamiento, se proponen dos modelos, el primero de ellos utiliza a las hormigas de Lumer como clasificadores, y el segundo modelo amplía el modelo anterior dotando a las hormigas de más información. Estos cambios culminan en el Capítulo 4, donde se presenta el modelo del sistema multiclasificador propuesto, eliminando el concepto de arena de los modelos

anteriores e incorporando las capacidades de detección de zonas específicas no cubiertas y la simplificación y modificación del conjunto de agentes básicos. En el Capítulo 5 se realizan las pruebas y comprobaciones necesarias para validar el modelo con varios conjuntos de datos, tanto datos artificiales y generados por nosotros, como datos de bases de datos clásicas en el ámbito de los sistemas clasificadores. La memoria terminará con el Capítulo 6 donde se expondrán las conclusiones del trabajo y las líneas de trabajo que han quedado abiertas y puedan mejorar el sistema propuesto.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

2

Sistemas Multiclasificadores

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

2

Sistemas Multiclasificadores

En este capítulo se introducen los conceptos básicos que fijan el problema de la clasificación automática y las mejoras que aportan los sistemas multiclasificadores, planteando una taxonomía referida a los elementos y acciones que configuran los sistemas multiclasificadores. Se muestran las técnicas básicas y más frecuentes para la obtención de los clasificadores básicos, así como los mecanismos de combinación de estos en un sistema multiclasificador. El capítulo finaliza resaltando como la diversidad de los clasificadores favorece a los sistemas multiclasificadores, por lo que deberá ser una característica básica de los mismos.

2.1. Introducción a los sistemas multiclasificadores

El aprendizaje supervisado [95] consiste en obtener un clasificador C, Ecuación 2.1, a partir de un conjunto de ejemplos L. El conjunto de ejemplos está formado por tuplas (x_i, c_i) donde $x_i \in X$ y $c_i \in Cl$, siendo X el dominio del problema y Cl el conjunto de clases posibles.

$$C(x) = c \tag{2.1}$$

En este ámbito, en los últimos años han aparecido numerosos trabajos sobre la inducción de un conjunto clasificadores H, a partir de un conjunto de datos L, y su combinación, como se indica en la Ecuación 2.3, para la obtención de un único clasificador C [50, 51, 52, 63, 105, 102, 111].

$$H = \{C_1, ..., C_n\} \tag{2.2}$$

$$C(x) = g(C_1(x), ..., C_n(x)) \tag{2.3}$$

Siendo g un método de combinación que se aplica sobre los resultados obtenidos por los distintos clasificadores C_i.

La idea original de estos estudios es la siguiente: ya que existen técnicas bien conocidas sobre como construir un clasificador, *¿Podríamos mejorar la calidad de la clasificación construyendo varios clasificadores y combinando sus resultados?*, para que esta pregunta tenga una respuesta afirmativa es necesario determinar ciertas propiedades generales y específicas que deben cumplir los sistemas con múltiples clasificadores para mejorar la clasificación respecto al uso tradicional de un único clasificador.

Los objetivos que se pretenden cubrir cuando se desarrollan modelos con múltiples clasificadores son muy variados. Uno de los objetivos generales de todo método de aprendizaje es la minimización del error cometido en la clasificación de un dato, siendo muy importantes los requerimientos computacionales de los mismos, tanto en la generación del modelo como en su uso. A grandes rasgos, es razonable pensar que se pueda mejorar el error en la clasificación si se construyen varios clasificadores para un conjunto de datos de entrada, ya que podríamos evaluar las clasificaciones de cada uno de ellos y obtener una única clasificación en base a esta evaluación. Respecto a los costes computacionales, en un principio puede parecer que aumentan, al tener que desarrollar y usar varios clasificadores, pero no tienen que ser así si se tienen en cuenta características particulares de varios de los métodos de aprendizaje, como la generación y uso distribuido de los clasificadores o el manejo de datos en localizaciones y tiempos diferentes.

Por estas razones, algunas de las características específicas buscadas al desarrollar un sistema con múltiples clasificadores son:

1. **Manejar gran cantidad de datos de entrada**. Hay muchos algoritmos de aprendizaje que no soportan computacionalmente el tratamiento de grandes cantidades de datos. Desde esta perspectiva, se ve la generación de múltiples clasificadores como una oportunidad de mejora. Este objetivo es explotado en campos como la *Minería de Datos*, donde cada clasificador puede ser entrenado en una partición del conjunto total de un dominio de entrada extenso [12, 43, 75, 94].

2. **Definir y utilizar algoritmos de aprendizaje distribuidos**. Es posible que los datos de entrada no estén disponibles desde los mismos dispositivos físicos, incluso que no estén disponibles al mismo tiempo. Una arquitectura de multiclasificación permitiría activar un clasificador para cada uno de los dispositivos. Por ejemplo, datos de distintos hospitales, datos geográficos, etc. El aprendizaje distribuido invita al uso de agentes clasificadores que se comunican y combinan para realizar una tarea.

3. **Especialización de los clasificadores en zonas parciales de datos**. Partiendo de la premisa de que se van a generar varios clasificadores, se actuará para que cada uno de ellos se especialice o sea especialista en distintas zonas parciales de los datos de entrada. Esto puede implicar que un clasificador no tenga que manejar todos los datos de entrada, e incluso, y más importante aún, que solo se activen unos pocos clasificadores cuando se tenga que clasificar un dato nuevo. Esta especialización se puede plantear desde la propia creación de los clasificadores, o bien aplicando un método que descubra donde cada clasificador clasifica mejor. Si se generan explícitamente los especialistas solamente se debe definir un mecanismo de selección de los mejores especialistas para el dato. Si no se generan especialistas, se puede aplicar un algoritmo de detección de zonas donde cada clasificador obtiene mejores resultados y, aplicar la selección de los clasificadores con esta información.

Desde el punto de vista de la *combinación de clasificadores*, definición de la función g, hay enfoques metodológicos bien diferenciados. El primero, llamado por algunos autores "*fusión de clasificadores*" [50, 51, 52, 63, 61, 62, 105, 84, 91], está basado en la idea de que todos los clasificadores van a ser utilizados para construir la solución final. Desde esta perspectiva, todos los clasificadores se entrenan para la misma tarea de predicción, pero solo se logra mejorar la clasificación si hay diversidad entre los clasificadores aprendidos. Esta característica, que resume este enfoque, se refleja perfectamente en la siguiente frase de Merz [73]: *"Las mejores condiciones para combinar ocurren cuando los modelos aprendidos son lo suficientemente exactos, pero lo suficientemente independientes en los errores que cometen".*

Esta propiedad implica que los clasificadores deben cometer errores independientes para poder garantizar un incremento en la predicción del sistema con respecto a la predicción de un solo clasificador. Muchos trabajos coinciden en la importancia de que los errores de los clasificadores no estén correlacionados. En general, se podría afirmar que si el propio método de generación de los clasificadores garantiza la propiedad de que los errores que comete cada uno no están correlacionados, el método de combinación puede ser sencillo, mientras que si la forma de generar estos clasificadores no garantiza esta no correlación, el método de combinación necesita de un análisis de las relaciones entre cada uno de los clasificadores para lograr mejorar el ratio de error de un único clasificador.

El segundo enfoque, llamado por algunos autores "*Selección dinámica de clasificadores*" [2, 25, 35, 37, 38, 60, 61, 84, 30], se basa en la idea de determinar dinámicamente que clasificador es mejor para clasificar un nuevo dato de entrada, de forma que el clasificador o clasificadores que van a contribuir en la clasificación de ese dato es un subconjunto del conjunto total de clasificadores. Un ejemplo clásico de construcción de un sistema con selección dinámica de clasificadores es obtener particiones del conjunto

de entrada y observar, para cada partición, los ratios de error de cada clasificador. Así que, cuando se usa el sistema multiclasificador, el dato a clasificar se asigna a una de las particiones y se utiliza el mejor o mejores clasificadores para esa partición.

Para poder establecer un marco conceptual amplio, y a su vez concreto, del estado del conocimiento en sistemas multiclasificadores, especificaremos una taxonomía (Figura 2.1) respecto a las dos etapas en la construcción del sistema multiclasificador: por un lado, la generación del conjunto de clasificadores base (C_i) y, por otro lado, los posibles métodos de combinación de esos clasificadores base.

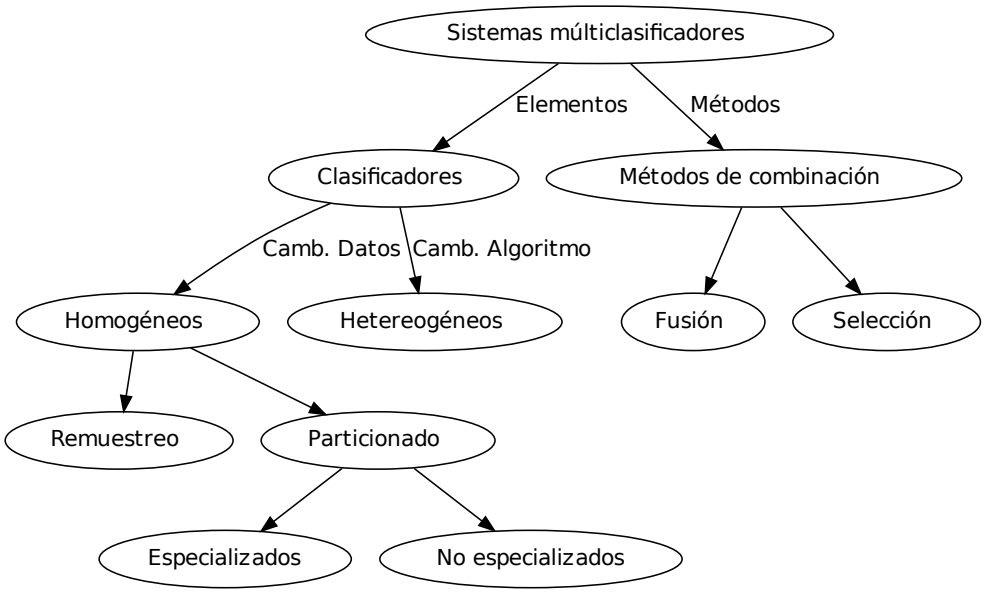

Figura 2.1: Taxonomía propuesta para los sistemas multiclasificadores

Todo sistema multiclasificador se compone de un conjunto de clasificadores base y un método de combinación de las predicciones de los clasificadores. La unión de estos dos componentes determinará las características del sistema. La taxonomía expuesta de un sistema multiclasificador irá encaminada a definir, por un lado, de que formas se pueden generar el conjunto de clasificadores base (Sección 2.2 y, por otro lado, qué métodos se pueden definir para combinar sus respuestas (Sección 2.3). En la Sección 2.4 veremos algunos de los métodos más relevantes definidos como sistemas multiclasificadores, para terminar, en la Sección 2.5, centrándonos en los métodos que generan o identifican especialistas. Este último marco será el de la propuesta que se plantea en esta memoria.

2.2. Generación de clasificadores básicos

Varios métodos han sido diseñados para obtener diversidad en los errores cometidos en el conjunto de clasificadores aprendidos H. La mayoría de ellos varían en los algoritmos de aprendizaje o bien en los datos de entrenamiento. Los modelos que son generados variando los algoritmos de aprendizaje son referenciados como *heterogéneos*, mientras que los que son generados variando el conjunto de datos de entrada son referenciados como *homogéneos*.

En los algoritmos de aprendizaje heterogéneos, la propia diversidad de los algoritmos de aprendizaje conduce a la diversidad en los clasificadores obtenidos, por tanto, no hay una taxonomía explícita para este tipo de método ya que pueden existir tantas como combinaciones de métodos de aprendizaje se utilicen.

Por contra, la variación del conjunto de entrenamiento utilizando un solo algoritmo de aprendizaje es la forma más utilizada en los métodos propuestos en la literatura. Es este el motivo por el que en la sección de modelos homogéneos se desarrollarán las técnicas más usadas, así como los métodos propuestos más relevantes.

Modelos heterogéneos

Ya que los clasificadores van a ser construidos con distintos algoritmos de aprendizaje, estos modelos en si mismos podrían garantizar la generación de clasificadores diversos. Para fomentar esta diversidad, los métodos pueden variar la representación de los datos y/o el método de búsqueda en esa representación.

Algoritmo 1 Bagging

Entrada: El conjunto de entrenamiento Datos=(x,c), algoritmo de inducción **I** y un entero **T** (número de muestras bootstrap).
Salida: El clasificador C*

1: **para** i =1 to T **hacer**
2: $Datos_i = (x,c)_i = Boostrap(Datos)$
3: $C_i = I(Datos_i)$
4: **fin para**
5: $C^*(x) = \underset{c \in Cl}{argmax} \sum_{i:C_i(x)=c} 1$

Un ejemplo de está técnica la podemos ver en [72], donde Merz hace un estudio de varias estrategias de combinación aplicadas al conjunto de modelos aprendidos. Estos modelos son un árbol de decisión, una lista

de decisión, una red neuronal, un clasificador bayesiano, un algoritmo de aprendizaje basado en instancias y un clasificador producido con la técnica del vecino más próximo. Está técnica la podemos también contemplar en el trabajo de Klein [53], donde se estudia el uso de clasificadores heterogéneos para eliminar la ambigüedad semántica de las palabras; así como en [39], donde varios clasificadores distintos son utilizados para la detección de peatones. En [68], la identificación de tablas para la construcción de bases de datos relacionales, o la detección de intrusos en los sistemas [110].

Modelos homogéneos

En estos modelos se utiliza el mismo algoritmo de aprendizaje para todos los clasificadores base que se generan, por tanto estos modelos se basan fundamentalmente en variar los datos de entrada, las condiciones iniciales o los parámetros del algoritmo de aprendizaje. Comenzaremos describiendo dos grandes grupos técnicas de modificación de los datos de entrada: las técnicas de remuestreo y las técnicas de particionado.

Técnicas de remuestreo

El método más común de generar modelos homogéneos es tomar repetidas muestras del conjunto inicial de datos de entrenamiento. Bradley Efrom, en 1979 [28], con el objetivo de mejorar las estimaciones de los estadísticos de una muestra de tamaño n, propuso un método consistente en la extracción de n elementos con reemplazamiento para configurar una nueva muestra. De esta forma se podrán extraer m muestras que tendrán, en esencia, la misma distribución que la original, por lo que la estimación de sus estadísticos como media de los estadísticos de las m muestras será más robusta y fiel que la correspondiente a la muestra inicial. Bradley denominó a esta técnica **boostrapping.**

Un ejemplo del uso del boostrapping para la generación de clasificadores es lo que propone Breiman [10], en lo que el llama "bootstrap aggregating" o "*Bagging*" (Algoritmo 1). En este método se obtienen n conjuntos de ejemplos formados con reemplazamiento del conjunto inicial, de forma que cada uno de estos n conjuntos tiene el mismo número de elementos que el original. La generación del conjunto H con n clasificadores se produce aplicando el algoritmo de aprendizaje a cada uno de los nuevos conjuntos.

Una representación gráfica de este método se puede observar en la Figura 2.2, donde se han distinguido las fases de la generación del conjunto de clasificadores y su uso en la clasificación. Como muestra la figura, una técnica de remuestreo, boostrapping en este caso, genera cuatro nuevas muestras de datos (x, c) con igual tamaño. Aplicando a cada uno de estas muestras una técnica de inducción de clasificadores, se construyen los cuatro clasificadores. Finalmente, la fase de utilización del clasificador hace uso de los clasificadores inducidos para que, con un criterio del más votado

o de media de los votos, proporcione el resultado final.

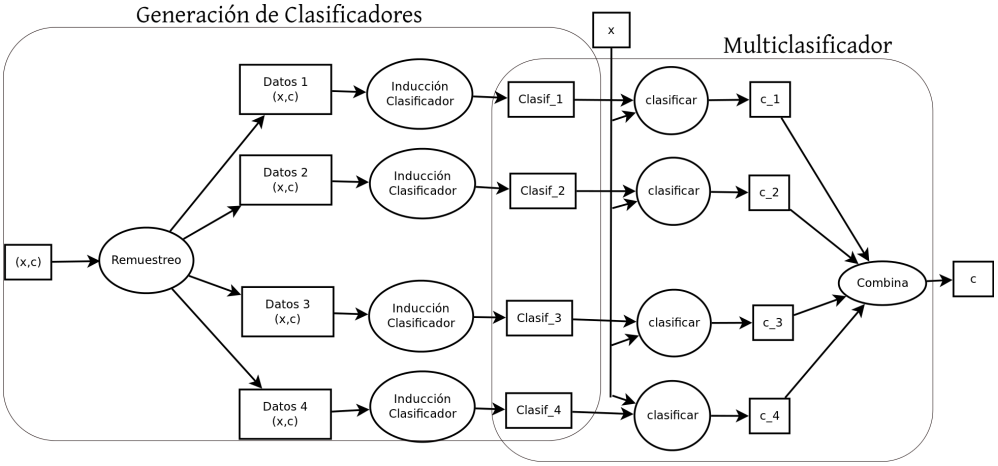

Figura 2.2: Boostrapping-Bagging.

Otro enfoque distinto al remuestreo simple, como el que hace boostrapping, es considerar la existencia de una distribución de probabilidad W asociada al conjunto de datos de entrada (x, c) usado para generar cada clasificador base. La distribución W refleja el éxito de la clasificación usando un modelo débil de aprendizaje. Las modificaciones sucesivas de la distribución W van reflejando distintos clasificadores realizados. El proceso así descrito se muestra en la Figura 2.3, donde un proceso iterativo va creando una colección de clasificadores que posteriormente son combinados.

Boosting es un método desarrollado por Schapire [87], basado en la utilización de un algoritmo débil de aprendizaje para los clasificadores base, obteniendo un clasificador sólido al combinar sus resultados. El algoritmo más extendido basado en Boosting es AdaBoost (Algoritmo 2), el cual se basa en el entrenamiento secuencial de N clasificadores, donde el conjunto de entrenamiento del clasificador i contiene, con mayor frecuencia, a datos que no han sido bien clasificados por el clasificador i-1. Cada dato de un conjunto de entrenamiento tiene un peso relacionado con su aparición en el conjunto. En el primer clasificador entrenado, todos los datos tienen el mismo peso $(1/m)$, donde m es el número de ejemplos. Una vez generado el clasificador, se evalúa su eficiencia y a los ejemplos que son mal clasificados se les aumenta el peso, de forma que este cambio de peso en los datos provoca que el siguiente clasificador sea entrenado en un nuevo conjunto de datos formado por la elección con reemplazamiento de datos del conjunto anterior de acuerdo con la distribución de pesos de estos.

Según Margineantu y Dietterich [69], Boosting es el método más efectivo para generar modelos diversos. Dos estudios independientes dirigidos por Quinlan [81] y Freund y Schapire [33] indican que Boosting es superior a Bagging. Schapire y Freund en [88] afirman que la efectividad de Boosting

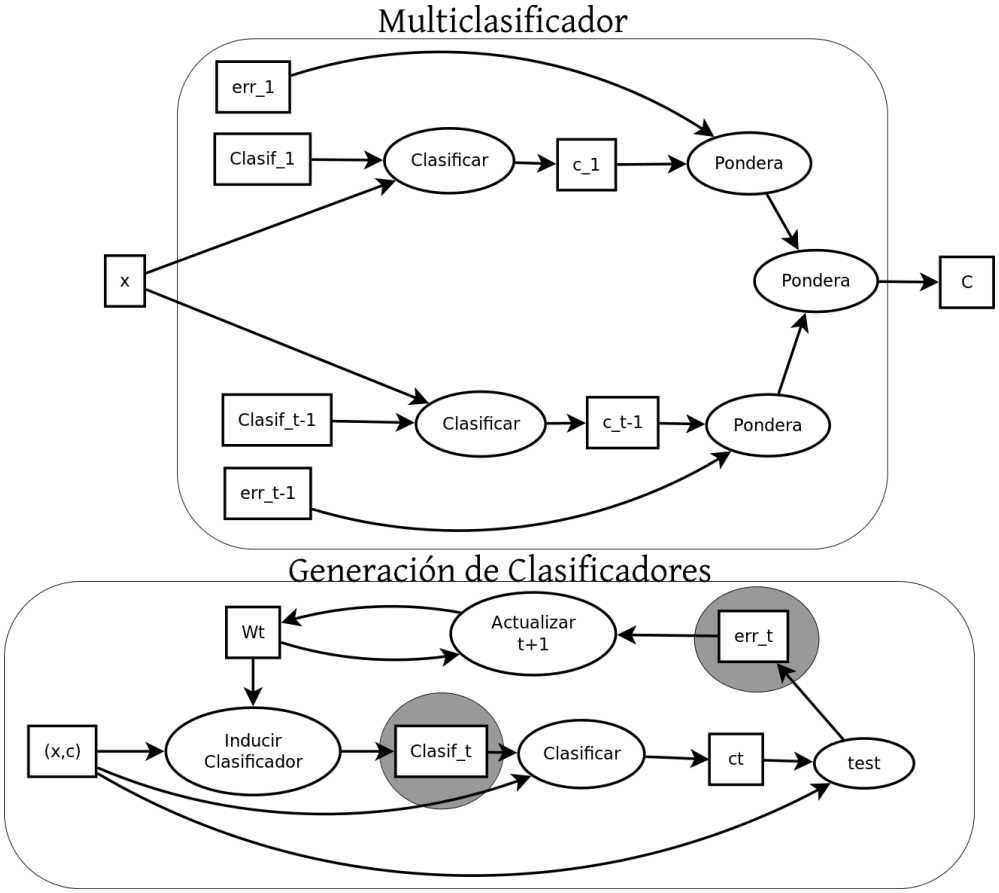

Figura 2.3: Boosting.

radica en su habilidad para clarificar las fronteras entre clases incluso cuando el error de entrenamiento es el cero por ciento. La clarificación de las fronteras entre clases es medida en términos de "margen" entre en los ejemplos de entrenamiento, respecto al esquema de combinación generado, donde el margen de un ejemplo es la diferencia entre el número de votos correctos y el número máximo de votos obtenido por cualquier clase incorrecta.

Otro enfoque distinto es el presentado por Krogh y Vedelsby [55], que diseñan un método conocido como "Aprendizaje Activo" basado en preguntas mediante lo que llaman comités. En esta metodología se entrena simultáneamente a un conjunto de redes neuronales, de forma que patrones que tienden a variar considerablemente son más probables de ser incluidos en el siguiente ciclo del entrenamiento.

Algoritmo 2 AdaBoost

Entrada: Conjunto de entrenamiento de tamaño m, algoritmo de inducción I, entero T (número de pruebas)

Salida: El clasificador C*

1: $S' = S$ con pesos para las instancias inicializados a 1
2: **para** $i = 1$ to T **hacer**
3: $C_i = I(S')$
4: $\varepsilon_i = \frac{1}{m} \sum_{x_j \in S': C_i(x_j) \neq c_j} peso(x)$ (error ponderado en el conjunto de entrenamiento)
5: **si** $\varepsilon_i > 1/2$ **entonces**
6: obtener una muestra bootstrap de S con peso 1 y asignarla a S' para cada instancia e ir a 4
7: **fin si**
8: $\beta_i = \varepsilon_i / (1 - \varepsilon_i)$
9: **para todo** $x_j \in S'$ **hacer**
10: **si** $C_i(x_j) = c_j$ **entonces**
11: $peso(x_j) = peso(x_j).\beta_j$
12: **fin si**
13: **fin para**
14: Normalizar los pesos de las instancias de forma que el peso total de S' sea m.
15: **fin para**
16: C* $(x) = argmax_{c \in Cl} \sum_{i: C_i(x) = c} log \frac{1}{\beta_i}$

Técnicas de particionado

Las técnicas de particionado se basan en generar modelos que son entrenados en diferentes subconjuntos del conjunto inicial de datos y son particularmente apropiadas cuando se dispone de cantidades masivas de datos y se pretende que los clasificadores se formen de forma paralela y/o distribuida. Dependiendo del objetivo que se pretenda cubrir, podemos hablar de particionado orientado a la especialización de los clasificadores o no orientado a la especialización.

No orientada a la especialización

Si se parte de un conjunto de datos entrenamiento lo suficientemente extenso, la obtención de un subconjunto del original con muestreo sin reemplazamiento, puede representar al dominio total y, en este sentido, puede ser utilizado para entrenar a un clasificador base. La partición del conjunto de datos no afecta al dominio del clasificador que es entrenado en una partición.

Sollich y Krogh [92] usan un método donde los modelos se generan usando un conjunto de datos parcialmente superpuesto. Cuando se está construyendo cada modelo, una porción diferente del conjunto de datos es retenida. Algunos métodos usan particiones de datos completamente

diferentes como Meir [71]; Krogh y Vedelsby [55]; Chan y Stolfo [16] y Wolpert [101].

Orientada a la especialización

En este caso, la partición del conjunto de entrenamiento va dirigida a obtener grupos de datos semejantes y los clasificadores serán entrenados en estos grupos. Un proceso de agrupamiento previo de los datos origina la partición del conjunto inicial de datos en los diversos conjuntos de entrenamiento.

Una metodología de partición de datos ha sido definida por Baxt [5], donde dos redes neuronales diferentes se entrenan en datos desde una clase diferente. La idea es producir redes especializadas en diferentes tipos de ejemplos, diseñando un área específica del experto.

Tresp y Taniguchi [96], dividen los ejemplos del conjunto de entrenamiento inicial en k clusters, y cada cluster es dividido aleatoriamente por la mitad produciendo dos conjuntos de entrenamiento. Perceptrones multicapa son entonces entrenados en cada uno de los nuevos conjuntos de entrenamiento. La idea de este método es que cada algoritmo de aprendizaje sea expuesto a los distintos tipos de ejemplos.

Giacinto [35] genera particiones del conjunto inicial de entrenamiento de forma que cada partición es asignada a un clasificador que estará especializado en esa partición. La manera de obtener cada partición es la siguiente: suponiendo que los datos de entrenamiento identifican a M clases, separamos los datos de forma que cada grupo contenga solo datos de la misma clase. Para cada grupo (clase) aplicamos un algoritmo de clustering obteniendo un número M_i de clusters. Supongamos que M_k es el número mayor de clusters obtenidos para una clase, entonces para el resto de clases (con menor número de clusters) se seleccionan aleatoriamente tantos cluster de su clase como la diferencia entre $M_k - M_i$ de forma que todas las clases tengan el mismo número de clusters. A partir de estos datos obtendremos M_i particiones escogiendo un cluster de cada clase. Con esta partición se asegura que cada clasificador va a ser entrenado en todas las clases pero cada clasificador será especialista en la zona del cluster que le fue asignado para la clase. Este método es particularmente interesante porque permite definir un modelo de selección del clasificador más apropiado para un patrón de entrada.

2.3. Métodos para la combinación de clasificadores

Kuncheva [61] establece la agrupación de métodos de combinación diferenciando entre los que usan la respuesta de todos los clasificadores base para obtener la predicción final y los que seleccionan un subconjunto de clasificadores base para cada dato nuevo. En general podemos decir que la combinación por **fusión** se utiliza cuando todos los clasificadores base han sido entrenados en todo el espacio de características y por tanto cualquier clasificador es susceptible de etiquetar a un dato nuevo. En contraposición, la noción de **selección** de clasificadores va ligada a la idea de que cada clasificador responde correctamente cuando los datos pertenecen a una región del espacio y por tanto predice correctamente los datos pertenecientes a esa región.

Sea el problema:

$$C(x) = g(C_1(x), ..., C_n(x)) \tag{2.4}$$

Para los métodos de *Fusión*, g será una función que establezca los criterios de combinación partiendo de los resultados de todos los clasificadores $C_i(x)$.

Para los métodos de *Selección*, g será una función que calcule que clasificador C_i es el más apropiados para el dato x.

En general, podemos decir que la mayoría de los métodos descritos en la literatura utilizan esquemas de fusión de clasificadores. Algunos de los métodos más importantes se estudian en la Sección 2.4 , por lo que en esta sección nos centraremos en revisar la llamada "selección de clasificadores".

Selección de clasificadores

Según Ko [54], existen tres esquemas de selección: selección estática de clasificadores, selección dinámica de clasificadores y selección dinámica de subconjuntos de clasificadores.

Selección estática de clasificadores

En la selección estática de clasificadores el objetivo es evaluar los clasificadores base que han sido generados, para obtener el subconjunto de clasificadores que previsiblemente mejor van a clasificar a cualquier dato. Se dice que es una selección estática porque esta evaluación se hace antes de usar el sistema multiclasificador para predecir la clase de un dato y, por tanto, el subconjunto de clasificadores es el mismo para todos los datos que se quieran clasificar.

Una de las formas más sencillas y a la vez efectivas para obtener el mejor subconjunto de clasificadores es escoger aquellos clasificadores cuyos ratios de error sobre el conjunto de test sea menor.

Selección dinámica de clasificadores (DCS)

Se habla de selección dinámica de clasificadores cuando se selecciona un único clasificador entre el conjunto de clasificadores base y esta selección depende del dato que se quiera clasificar, es decir, el método de selección del clasificador analiza cual es, previsiblemente, el mejor clasificador para ese dato.

Giacinto y Roli [36] definen lo que llaman un método de selección 'a priori', que determina el mejor clasificador teniendo en cuenta solamente la localización del dato, antes de tener conocimiento de las clases que los clasificadores pueden predecir para el dato. La efectividad de un clasificador puede ser ponderada por las distancias entre los ejemplos de entrenamiento de la región local y el dato a clasificar. De igual forma, definen el método llamado 'a posteriori' en el que la efectividad de un clasificador se calcula teniendo en cuenta la predicción de los clasificadores.

Una forma de elegir al mejor clasificador es analizar la eficiencia local de cada clasificador en ejemplos que sean 'parecidos' al dato que se quiere etiquetar. Woods [102] define la eficacia de un clasificador en lo que llama efectividad de clases local (local class accuracy, LCA). El método consiste en calcular los k-vecinos más próximos al dato y obtener la clase que predice el clasificador para cada uno de ellos. Si el clasificador predice una clase c_i para el dato x, la efectividad del clasificador para x se calculará como la proporción de los k-vecinos cuya clase predicha es también c_i. El análisis de la efectividad local es, por tanto, dinámico, en el sentido de que se realiza un análisis de eficiencia diferente para cada dato a clasificar.

El inconveniente de un análisis de la eficiencia dinámico es que, para cada dato, hay que encontrar a sus k-vecinos más próximos y, seguidamente calcular la efectividad de cada clasificador en esos vecinos. Para decrementar la complejidad computacional, Kuncheva [61] propone precalcular lo que llama 'regiones de competencia' y asignar a cada clasificador las regiones de competencia donde cada clasificador es más efectivo. De esta forma, el proceso de clasificación de un dato se reduce a asignar dicho dato a una de las regiones, y elegir el clasificador mejor para esa región.

Siguiendo con la idea de preestimar las regiones de competencia, Kuncheva [60, 61] propone un método de selección de clasificadores llamado "clustering y selección". El método consiste, por un lado, en entrenar a L clasificadores sobre el conjunto inicial de entrenamiento. A partir de los datos de entrenamiento, se obtienen k particiones que serán usadas para averiguar que clasificador predice mejor en cada región, de forma que cuando se quiere etiquetar un dato nuevo, se asigna el dato a la región más cercana y se elige al clasificador que mejores resultados obtuvo en esa región. A continuación

se define este método en dos partes, la primera fase de entrenamiento y la segunda fase de uso.

Fase de entrenamiento:

1. Diseñar los clasificadores individuales $D_1, ...D_L$ usando el conjunto de datos etiquetados Z. Elegir el número de regiones K.
2. Sin tener en cuenta las etiquetas de clase, formar K clusters a partir de Z, $C_1, ..., C_k$, usando p.e. la técnica k-means. Encontrar los centroides de cada cluster, $v_1, ..., v_k$ obtenidos como la media aritmética de todos los puntos del cluster.
3. Para cada cluster C_j (el cual define a la región R_j), estimar la efectividad en la clasificación de $D_1, ...D_L$. Tomar el clasificador más efectivo para R_j y llamarlo $D_{i(j)}$.
4. Devolver $v_1, ..., v_k$ y $D_{i(1)}, ...D_{i(k)}$.

Fase de uso:

1. Dada una entrada $x \epsilon \Re^n$, encontrar el centro del cluster más cercano (v_j) a x.
2. Usar el clasificador $D_{i(j)}$ para clasificar x.

El procedimiento de definición de las regiones $R_1, ..., R_k$ puede realizarse antes o después de la generación de los clasificadores, ya que son procesos independientes. Una modificación al algoritmo anterior es usar cada región R_j para construir un clasificador experto $D_{i(j)}$ de forma que la fase de uso es la misma que si cada clasificador se entrena en todo el conjunto de inicial de entrenamiento.

Lui y Yuan [105] también intentan encontrar las regiones donde cada clasificador es más eficaz. Para cada clasificador generado, obtienen un conjunto con los ejemplos bien clasificados y otro con los mal clasificados. A partir de estos dos conjuntos, obtiene diferentes clusters que son usados para medir la eficacia del clasificador. En la fase de uso, se asigna el dato a clasificar a la región más cercana y se elige el mejor clasificador para esa región.

Selección dinámica de un subconjunto de clasificadores (DES)

Ko [54] propone la selección dinámica de un conjunto de clasificadores en lo que llama "dynamic ensemble selection", en vez de la selección de un único clasificador para un patrón nuevo. En este trabajo propone cuatro esquemas de selección de un subconjunto de clasificadores y como combinar sus predicciones.

El primer esquema, llamado Knora-Eliminate, elige k-vecinos del patrón x y obtiene un conjunto de clasificadores que clasifican correctamente a los k vecinos. Cada unos de los clasificadores obtenidos es usado entonces para clasificar al patrón x , de forma que la clase mayoritaria de todos los clasificadores es la clase final. Si no hay ningún clasificador que clasifique correctamente a todos los vecinos se decrementa el valor de k hasta que al menos un clasificador clasifica bien a los k-vecinos.

El segundo esquema, llamado Knora-eliminate-w, funciona igual que el primero pero cada voto de un clasificador es ponderado por la distancia del patrón x a sus vecinos.

El tercer esquema, llamado Knora-Unión, no es tan restrictivo respecto a que un clasificador tenga que clasificar bien a los k-vecinos. En este esquema se obtiene el conjunto de clasificadores que predice bien a cada vecino j, de forma que la unión de todos los conjuntos conforma el conjunto de clasificadores que son utilizados en la clasificación final, aportando un voto por cada clasificador.

El cuarto esquema, llamado Knora-Union-w, vuelve a ser una modificación del anterior donde el voto de cada clasificador es ponderado por la distancia del patrón x al vecino j. Ko aplica estos cuatro esquemas a bases de datos del repositorio de la UCI y muestra mejoras respecto a métodos de selección estática.

Do Santos [30] también trata el concepto de selección dinámica de un subconjunto de clasificadores en una estrategia llamada “sobreproduce y selecciona”. El objetivo de la estrategia sobreproduce y selecciona es generar un conjunto amplio de clasificadores y usar un método de selección que elige al mejor subconjunto de clasificadores. Do Santos propone dos niveles en la fase de selección: un primer nivel de optimización, que genera un subconjunto de clasificadores óptimo, y un segundo nivel en el que se elige dinámicamente a los mejores clasificadores para predecir la clase de un dato nuevo.

2.4. Algunos métodos relevantes en los sistemas multiclasificadores

2.4.1. Votación

La votación es el método más claro de asignación de pesos constantes a la predicción de un clasificador base y se suele aplicar cuando hablamos de fusión de clasificadores.

El objetivo en este esquema es combinar múltiples clasificadores tomando

una combinación lineal sus salidas. El esquema más simple derivado de esta metodología es el llamado *Votación Simple*, en el que la predicción final es aquella que haya obtenido la mayoría de los votos. Una variación de la votación simple es la *Votación Ponderada*, en la que a cada clasificador se le asocia un peso, que normalmente es determinado en relación a la efectividad del clasificador en un conjunto de validación. Cada predicción se pondera por el peso asignado al clasificador, se suman todos los pesos para cada predicción y la predicción final es aquella cuya suma ponderada ha sido mayor.

Littlestone [64] propone varios algoritmos de mayoría ponderada (Weighted majority algorithms) para combinar diferentes clasificadores. En su trabajo, los clasificadores son algoritmos de predicción diferentes, no necesariamente aprendidos, y los datos de entrenamiento son solo usados para calcular los pesos. El algoritmo básico asocia un peso inicial a cada clasificador; cada ejemplo en el conjunto de entrenamiento es procesado por el clasificador y la predicción final para cada ejemplo es generada mediante "weighted voting". Si la predicción final es mala, los pesos de los clasificadores cuyas predicciones fueron incorrectas son multiplicados por una constante β que decrementa el peso del clasificador, donde $0 \leq \beta < 1$. Una variante de este algoritmo básico es WML, en el que no se permite que los pesos sean decrementados más allá de un límite predefinido.

Tanto Boosting como Bagging fueron descritos en la generación de clasificadores como técnicas basadas en el remuestreo. Ambos métodos utilizan voting como técnica de combinación. En el caso de Bagging, la clase final es la que obtiene una mayoría de votos, mientras que en Boosting se utiliza un esquema de "voto ponderado" donde el peso de cada clasificador depende de su rendimiento en el conjunto de entrenamiento usado para construirlo. Otra posibilidad, dentro de este esquema, es que cada clasificador ofrezca información sobre el grado con el que cada una de sus clases ha sido seleccionada, estableciendo un ranking de clases y/o estimando un peso con estos datos, por ejemplo un peso proporcional a la diferencia de las dos salidas con grado más alto.

2.4.2. Plantillas de decisión

Kuncheva [62] ha diseñado un esquema de combinación, llamado "decision templates", donde se genera un perfil de decisión para un dato a clasificar x. Este perfil de decisión para el dato se compara con cada una de las plantillas de decisión creadas para cada clase, a partir del conjunto de entrenamiento, de forma que, usando una medida de similaridad, se obtiene como clase de x, la clase cuya plantilla es más similar al perfil de decisión de x.

El perfil de decisión para un dato x (PD(x)) se obtiene construyendo una matriz con tantas filas como clasificadores tengamos y tantas columnas

como clases. Un elemento $d_{i,j}$ representa la decisión del clasificador i respecto al grado de pertenencia del dato x a la clase j, por lo que la matriz representa las decisiones de cada clasificador para cada clase. Algunos métodos calculan la predicción para cada clase i usando solo la columna i de la matriz y eligiendo así la clase "mejor puntuada".

Para cada clase i se obtiene una plantilla de decisión que resume en una matriz la media de los perfiles de decisión de los ejemplos del conjunto de entrenamiento con clase i .

La plantilla de decisión para la clase i se construye solamente con los datos de entrenamiento que tienen clase i, obteniendo para cada dato de entrenamiento su perfil de decisión. El elemento k, j de esta matriz es la media de las predicciones del clasificador k para la clase j para todos los datos de entrenamiento con clase i.

La clasificación de un nuevo dato se produce construyendo su matriz (perfil de decisión) y comparándola, mediante alguna medida de similaridad, con todas las plantillas de decisión (una por clase), de forma que la plantilla más similar es la que se asocia a la clase final.

2.4.3. Mezcla de expertos

El método llamado mezcla de expertos (Mixture of Experts) es un método claramente enfocado a la asignación de pesos no constante en la predicción de un clasificador base, pero sí se necesita de la evaluación de todos los expertos.

En el esquema de votación, los pesos de los clasificadores son constantes sobre el espacio de entrada. En la arquitectura de mezcla de expertos, los pesos de obtienen dinámicamente en función del dato a clasificar. Cada “experto” (clasificador aprendido), es guiado en su construcción para que se especialice en una región local del espacio de entrada, de forma que los expertos son forzados a competir y no solaparse. El objetivo es que, dado un dato para clasificar, podamos tener una función (o nuevo experto) que identifique que subconjunto de expertos puede dar una mejor respuesta. Esta información dependerá de la mayor o menor inclusión del nuevo dato en cada una de las regiones de los expertos. La metodología llamada mezcla de expertos estaría incluida dentro de lo que hemos llamado selección dinámica de clasificadores.

El método de mezcla de expertos es visto como un método de combinación fuertemente acoplado con el proceso de generación de los modelos. Dietterich [26] da un ejemplo para la tarea de reconocer rápidamente las distinciones entre las palabras *bee, tree, gate* y *mate*. Puede necesitarse un experto para distinguir entre *bee y tree*, y otro para las palabras *gate y mate*. Zhao (et. al) [112] ha aplicado la metodología de mezcla de expertos al reconocimiento

rápido continuo. Melia [70] ha aplicado una variante de esta metodología para aprender el movimiento de un brazo de robot.

El método prevalente de mezcla de expertos fue propuesto por Jacobs [44]. En este método, se entrena un conjunto de redes neuronales y cada red (experto) ofrece una distribución de probabilidad de clase para cada ejemplo, de forma que una red, llamada *gating network* selecciona a los expertos que tienen la distribución más apropiada para el ejemplo. Durante el entrenamiento, los cambios de los pesos se localizan solo en los expertos seleccionados y en la red de selección. A los expertos que clasifican bien el ejemplo se les da más responsabilidad para ese ejemplo (aumenta su peso) y a los que no lo clasifican bien se les decrementa su peso. Esto es debido a que los expertos elegidos están especializados en gran medida en la zona del ejemplo. No hay cambios en los pesos de los expertos que están especializados en casos claramente diferentes. Durante el entrenamiento, los pesos de los expertos se van desacoplando entre ellos conforme se especializan en pequeñas zonas del espacio de entrada.

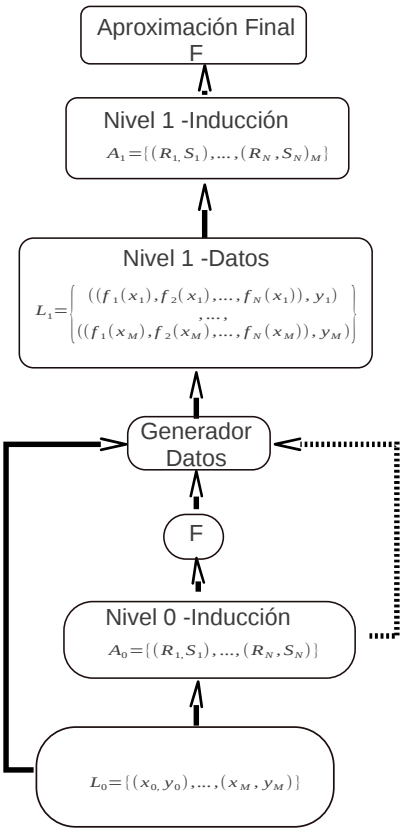

Figura 2.4: Stacked generalization

En la versión jerárquica de esta arquitectura, los expertos están en una

estructura de árbol, donde los nodos internos son redes de selección y las hojas son los expertos. Las redes de selección hacen divisiones suaves, permitiendo que un ejemplo se encuentre simultáneamente en múltiples regiones.

2.4.4. Stacked generalization

Stacked generalization es un método genérico de fusión de clasificadores, donde las salidas de todos los clasificadores son las entradas de un nuevo algoritmo de aprendizaje.

Esta es una técnica propuesta por Wolpert [100], la cual es una extensión de la votación en el sentido de que la combinación de las salidas de los clasificadores aprendidos no necesita ser lineal. La premisa básica es que el problema de la combinación de los clasificadores puede ser visto como otro problema de inducción donde el espacio de entrada está compuesto por las salidas de los modelos aprendidos.

En este marco, Wolpert define dos niveles de inducción. En el primer nivel de inducción, los datos L_0 (llamados datos de nivel 0) son la entrada a una colección de algoritmos A y son usados para crear una colección de modelos aprendidos F. A partir de aquí, un generador de datos genera los datos del nivel 1 (L_1) que son utilizados en el segundo nivel de inducción. Se puede observar este esquema en la Figura 2.4.

Algoritmo 3 Stacked Generalization

Entrada: Conjunto de entrenamiento L_0, número de particiones V
Salida: El conjunto L_1
1: Dividir los datos L_0 en V particiones
2: $L_1 \leftarrow \emptyset$
3: **para todo** partición v **hacer**
4: Entrenar cada algoritmo en todas las particiones excepto en la partición v para obtener $\{f_n^{-v}\}$
5: Aplicar un test a cada modelo aprendido en $\{f_n^{-v}\}$ utilizando la partición v
6: Emparejar las predicciones en cada ejemplo en la partición v con la correspondiente salida para formar L_1^v
7: $L_1 \leftarrow L_1 \cup L_1^v$
8: **fin para**
9: Devolver L_1

El generador de datos puede actuar básicamente de dos formas:

- La más simple, toma como entrada del nivel 1 las salidas del nivel 0 de los N modelos para cada ejemplo.

- En la segunda forma, las salidas de los N modelos son aproximadas usando la metodología descrita en el Algoritmo 3.

El término Stacking está asociado en la literatura a dos diferentes nociones. En el sentido más general se utiliza como marco para combinar las salidas de múltiples clasificadores de nivel 0 a través de uno o más niveles de algoritmos de combinación. El otro sentido en el que se utiliza el termino Stacking está referido al método para generar el nivel 1 de datos.

2.4.5. Meta Aprendizaje

Meta-Aprendizaje (Meta-Learning) es una técnica general definida por Chan [15] como un método que combina el resultado de diferentes algoritmos de aprendizaje entrenados en un conjunto de entrenamiento. Además de obtener un conjunto de clasificadores base, se va a generar un meta-clasificador usando una estrategia de meta-aprendizaje, donde el meta-clasificador es entrenado con información de los clasificadores base generados. Dependiendo de la estrategia de meta-aprendizaje que se elija, se obtendrán diferentes sistemas multiclasificadores. En este sentido, la definición de stacked generalization hecha por Wolpert [100] es un caso concreto de estrategia de meta-aprendizaje.

Chan [15] define dos tipos de información que se puede combinar para generar el meta-clasificador. Por un lado, propone que se combinen los clasificadores base, es decir, que se combinen las descripciones conceptuales que subyacen en los clasificadores base. Dado que estos clasificadores base han sido generados con distintos algoritmos de aprendizaje, las representaciones de los conceptos en ellos serán diferentes (por ejemplo, árboles de decisión, reglas, redes, etc). En este sentido, propone generar un meta-clasificador que tenga una representación uniforme de todos ellos.

La segunda estrategia para generar el meta-clasificador es integrar las predicciones de los clasificadores base para el conjunto de entrenamiento. Chan define dos estrategias de meta-aprendizaje para generar el meta-clasificador: la primera estrategia llamada "combiner" y la segunda "arbiter".

Combiner. La estrategia llamada "combiner" consiste en encontrar la relación entre las predicciones generadas por los clasificadores base y las predicciones correctas. El conjunto de entrenamiento para generar el meta-clasificador (combiner) está compuesto por las predicciones de los clasificadores base sobre el conjunto de entrenamiento original. Básicamente, un dato de entrenamiento para generar el "combiner" está compuesto de las predicciones de los clasificadores base para un dato de entrenamiento x, añadiendo la clase correcta del dato x . Incluso, Chan propone que cada clasificador no ofrezca una sola clase sino una lista de posibles clases para el dato.

Cuando se quiere clasificar un nuevo dato se clasifica el dato en cada uno de los clasificadores base, de forma que las salidas de los clasificadores base son la entrada del combiner, y este decide la clase final.

Arbiter. Es un nuevo clasificador entrenado con los ejemplos del conjunto de test que hacen que los k clasificadores no predigan consistentemente. El arbiter es entrenado con el mismo algoritmo de aprendizaje que los clasificadores. Una instancia pasará a ser una instancia de entrenamiento para el arbiter si ninguna de las clases obtenidas tiene k/2 votos (siendo k el número de clasificadores).

Cuando se quiere clasificar un nuevo dato, se pasa la nueva instancia a los k clasificadores y al arbiter (k +1 salidas) y estas salidas pasan a una regla de arbitraje que decide la clase final (la del arbiter en caso de empate).

En estudios posteriores, Chan [17] entrena a cada clasificador base en un subconjunto del conjunto de entrenamiento original. Los clasificadores base se construyen en paralelo y se va generando una estructura de árbol donde las hojas son los clasificadores base y los nodos son los meta-clasificadores de alto nivel. Esta idea es refinada [79, 80], podando el árbol para eliminar clasificadores redundantes y aplicando estas ideas a sistemas distribuidos.

Stolfo [93] define una herramienta, llamada JAM (Java Agents for Meta-Learning), que implementa una serie de algoritmos de aprendizaje junto con un conjunto de agentes de meta-aprendizaje para combinar los resultados de los clasificadores base y, posteriormente, Prodromidis [78] usa JAM para discutir los sistemas de meta-aprendizaje aplicados a sistemas de data mining distribuido.

2.5. Generación o identificación de especialistas

En la introducción de este trabajo se habló de la necesidad de obtener clasificadores diversos para que su combinación fuera óptima. Desde este punto de vista se hace necesario hacer las siguientes reflexiones:

- **Identificación de especialistas**. Suponiendo que no generemos explícitamente clasificadores especialistas en zonas de datos, el algoritmo de combinación debería actuar de alguna de las siguientes formas:

  1. Detectar si varios de los clasificadores construidos predicen de forma semejante. En este caso habría clasificadores muy correlacionados por lo que sería aconsejable mantener solo uno de ellos. En definitiva estaríamos eliminando información redundante y además, provocaríamos que para un nuevo ejemplo se activasen

menos clasificadores. Merz en su tesis utiliza la técnica de "*Singular Value Descomposition*". El objetivo es encontrar la correlación entre los elementos de una matriz A, de dimensión MxN, donde M es el número de modelos aprendidos y N es el número de ejemplos. El elemento A[i,j] es la predicción del clasificador i sobre el dato j, y obtener una nueva estructura con la información comprimida. Esta técnica detecta que elementos son combinaciones lineales de otros y los transforma.

2. Intentar descubrir en que zonas de datos los clasificadores son especialistas. Koppel y Engelson [29] intentan integrar varios clasificadores encontrando las zonas donde ellos son expertos. Para ello construyen un árbol de decisión para cada clasificador con un conjunto de ejemplos. El objetivo es aprender en que tipo de ejemplos es experto un clasificador de forma que, para un ejemplo nuevo, cada árbol de cada clasificador decidirá si es o no es el apropiado para el dato.

- ***Generación de especialistas***. La otra posibilidad es generar explícitamente clasificadores que sean especialistas en zonas de datos. El objetivo es entrenar a cada clasificador solamente en aquellos datos que están en la misma zona. Para ello podemos hacer una agrupación previa que establezca las zonas y, por tanto, el número de clasificadores que vamos a activar. Las ventajas de este método general son las siguientes:

  - El método de selección del mejor/es clasificador/es es tan sencillo como asignar a que zona pertenece el dato que queremos clasificar.
  - Cada clasificador no es entrenado en el total de los datos. Por lo que es un método muy apropiado cuando tenemos gran cantidad de datos.
  - Se necesita muy poca o nula comunicación entre cada clasificador. Por lo que se puede pensar en una arquitectura paralela y/o distribuida para su implementación.

  Podemos pensar en obtener la agrupación de los datos de dos formas:

  - *Estática*. La agrupación es un proceso anterior a la generación de los clasificadores, es decir, una vez que obtenemos las agrupaciones, entrenamos a cada clasificador en una agrupación.
  - *Dinámica*. La obtención de las zonas se realiza al mismo tiempo que la generación de los clasificadores. El objetivo aquí es diseñar un método donde los datos se vayan agrupando en zonas y que, en un momento dado de este proceso, se detecten zonas lo suficientemente definidas como para activar un clasificador para esa zona. Esta manera de agrupar los datos es especialmente apropiada cuando no se disponen de todos los datos al inicio del proceso.

2.5.1. Propuesta: generación y combinación de clasificadores especialistas evolutivos

Una vez expuesta una visión general de la situación actual en el ámbito de la multiclasificación y considerando las últimas observaciones expuestas, podemos centrar los objetivos generales definidos en el Capítulo 1.

Este trabajo propone un nuevo método enmarcado dentro del enfoque de "Selección dinámica de Clasificadores". Nuestro objetivo va a ser la determinación dinámica del mejor clasificador pero desde una perspectiva diferente: "*Forzar a que los clasificadores que generamos sean especialistas en regiones de los datos de entrada".* Si cada clasificador predice correctamente datos pertenecientes a su región de entrenamiento, no es necesario un procedimiento en el que se aprenda en que zona obtiene un mejor ratio de error cada clasificador. Consecuentemente, la predicción de la clase de un dato nuevo se realiza obteniendo el o los clasificadores que se han especializado en la zona del dato y usando sus predicciones.

Haciendo un símil con la definición mencionada anteriormente de Merz [73], referida a la fusión de clasificadores; nuestra propuesta, dentro de la selección dinámica, se podría resumir de la siguiente forma:

"Las mejores condiciones para combinar podrían ocurrir cuando los modelos aprendidos se construyen de forma que son especialistas en zonas de datos y para cada nuevo dato podamos determinar dinámicamente que subconjunto de especialistas es el más apropiado, además, esta función de selección podría ser el resultado del propio proceso de formación de los clasificadores".. Un cambio fundamental en nuestra propuesta es que cada clasificador base inducido no tiene porque predecir correctamente todas las zonas del dominio de entrada. Para garantizar que el sistema multiclasificador pueda predecir correctamente la clase de cualquier dato, el conjunto de los especialistas generados debe cumplir que cubran el total del dominio de los datos, es decir, que al menos se va a generar un clasificador.

En la generación de los clasificadores base se pretende que cada clasificador actúe como un agente que interactúa con su entorno, compuesto de datos de entrenamiento y otros agentes, de forma que cada agente selecciona los datos del entorno que van a servir para su entrenamiento.

De igual forma, un agente devolverá al entorno aquellos datos que, aun habiendo sido seleccionados con anterioridad, ofrecen peor valoración al agente. El resultado será un conjunto de agentes que son especialistas en diferentes zonas del dominio.

El uso del sistema multiclasificador se basará en la elección, para cada dato a clasificar, de los agentes cuyas zonas de especialización incluyen al dato a clasificar y la combinación de las respuestas de este subconjunto de agentes.

Las ventajas de esta metodología son especialmente importantes cuando pensamos en ciertas características respecto a la adquisición y tratamiento de los datos, por ejemplo:

- La mayoría de los métodos de generación de clasificadores necesitan tener todos los datos de entrenamiento disponibles antes de empezar el entrenamiento. Si los datos de entrenamiento se obtienen en tiempo real no tendríamos que esperar a tener todos los datos para comenzar a construir los clasificadores, habríamos conseguido un método capaz de interactuar con el entorno.
- Los datos pueden ser producidos por distintas fuentes, que a su vez pueden estar distribuidas o tener un origen diverso. Por ejemplo, sensores distintos que identifican distintos componentes de un mismo dato, o sensores distintos que aun recogiendo características similares estas corresponden a datos de distinta naturaleza.
- Es un sistema adaptativo, en el sentido de que nuevos datos de entrenamiento pueden ser incluidos en el entorno y estos servir para que los agentes sigan entrenándose, y por tanto adaptándose a las nuevas condiciones. Esta característica hace que sea un sistema abierto, en el que se puede seguir mejorando el clasificador siempre que haya datos nuevos de entrenamiento. En este sentido, es una metodología idónea para el aprendizaje online.

Las propiedades que se pretende que cumpla el sistema multiclasificador son las siguientes:

- El conjunto de agentes debe representar a todas las zonas del dominio inicial para poder asegurar que, para cualquier dato que queramos clasificar, va a existir al menos un agente que lo pueda clasificar correctamente. También sería deseable que el sistema multiclasificador detecte agentes similares y los sustituya por un solo agente.
- El sistema multiclasificador debería detectar cuando un agente ya está formado y por tanto parar o ralentizar su entrenamiento.
- Detección de zonas del dominio en las que ningún clasificador se esté especializando o bien detectar pequeñas zonas que pueden representar a datos con ruido o excepciones. En este caso el sistema multiclasificador debería tratar de forma independiente estas zonas activando un clasificador para ellos.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

3

Modelos Multiclasificadores Basados en Sistemas De Enjambres

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

3

Modelos multiclasificadores basados en Sistemas de Enjambres

En este capítulo se proponen dos sistemas multiclasificadores especialistas basados en sistemas de enjambre. En particular, los modelos se han construido en base al comportamiento agrupador de cierto tipo de hormigas reales. Se ha utilizado un algoritmo de agrupamiento emergente basado en estas hormigas como modelo de acción e interacción con el entorno, en el que las hormigas se corresponden con los agentes clasificadores que se generan y se utilizan en el sistema multiclasificador.

3.1. Introducción

Tal y como se ha descrito en los Capítulos 1 y 2, los modelos que vamos a proponer se enmarcan dentro de sistemas multiclasificadores especialistas con selección dinámica, esto es, sistemas multiclasificadores que generan o parten de clasificadores expertos en subdominios y cuyo modo de uso se basa en la selección del mejor subconjunto de clasificadores cada vez que se quiere predecir la clase de un nuevo dato.

Estos modelos requieren, por tanto, la definición de una metodología para generar el conjunto de agentes especialistas y, por otro lado, la definición de un mecanismo de selección y combinación de los mejores expertos para predecir la clase del dato.

En el capítulo anterior se ha expuesto una taxonomía general de los sistemas multiclasificadores, así como ejemplos y algoritmos concretos muy extendidos y referentes a este ámbito. Esto permite centrar nuestra aportación de forma precisa en un marco tan general. Se ha dejado fuera del capítulo anterior el concepto de inteligencia de enjambre (swarm

intelligence [9]), ya que es la base de los dos primeros modelos propuestos y será expuesto detenidamente en este capítulo. En términos generales, los sistemas basados en inteligencia de enjambre simulan el comportamiento de determinados insectos en la naturaleza, como abejas, hormigas etc, que logran un objetivo global como resultado de su interacción con el entorno y de un comportamiento simple y autónomo de sus miembros.

En los modelos estudiados de sistemas multiclasificadores, los algoritmos de aprendizaje que suelen ser utilizados para obtener los clasificadores son los algoritmos clásicos de aprendizaje definidos para obtener un único clasificador. Estos algoritmos han sido diseñados para poder clasificar cualquier dato, por lo que su uso dentro de un sistema multiclasificador se realiza reajustando el entrenamiento para el conjunto de clasificadores.

Los modelos propuestos van a construir el conjunto de clasificadores desde una perspectiva diferente: cada clasificador va a ser un agente autónomo que se mueve en un entorno en el que hay datos de entrenamiento y otros agentes. El entrenamiento del conjunto de agentes emerge como la evolución de un sistema donde cada agente compite en algún sentido con el resto de agentes y donde cada agente selecciona los datos de entrenamiento del entorno y devuelve los datos que no le sirven para su entrenamiento.

Desde este punto de vista, la definición de un modelo concreto responderá, entre otras, a las siguientes cuestiones:

- ¿Cuál es la estructura de un agente especialista y cómo representamos su zona de especialización?
- ¿Cuántos agentes especialistas necesitamos para cubrir todo el dominio?
- ¿Cuál es el entorno del conjunto de agentes y cómo estos interactuan con él?.
- ¿Cuál es el criterio de selección de un dato del entorno como dato de entrenamiento?.
- ¿Cómo se entrena al agente con el dato seleccionado?.
- ¿Cómo seleccionar al mejor subconjunto de especialistas para clasificar un dato nuevo?.
- ¿Cómo combinar las soluciones del subconjunto de especialistas?

Las distintas respuestas a estas preguntas determinarán distintos modelos posibles.

De forma general, en los dos modelos que vamos a definir en este capítulo, pretendemos que los agentes sean lo más simples posibles, tanto en su

formación como en las acciones que realizan en el entorno y, a su vez, que sean independientes del resto de agentes.

Esquemáticamente, un agente se mueve en el entorno y, si encuentra un dato, debe decidir si lo utiliza como dato de entrenamiento o lo devuelve al entorno. En caso de decidir que es un dato válido para su construcción, el entrenamiento del agente con el dato se produce de forma incremental, es decir, el agente no dispone de todos los datos de entrenamiento al mismo tiempo, con lo que el un nuevo dato debe producir una transformación del agente en cierto sentido. Por tanto, la formación de cada agente como especialista implica que cada agente va recibiendo los datos de entrenamiento conforme este interactúa con el entorno.

Como marco general para la construcción de los agentes en el entorno descrito, se ha utilizado el modelo de agrupación de hormigas definido por Lummer y Faieta [66] (algoritmo LF). En este modelo, el conjunto de agentes, que son elementos agrupadores del algoritmo LF, se mueven en un espacio en el que se encuentran los datos. El objetivo de un agente es transportar datos de un lugar a otro dentro de su entorno, en un espacio bidimensional en el que se encuentran los datos, de forma que las acciones del conjunto de agentes provocan que los datos que estaban aleatoriamente esparcidos en el espacio, queden agrupados en el mismo espacio, formando grupos de elementos parecidos. Las acciones de un agente se limitan a coger un dato si este no está bien agrupado y el agente no transporta ningún dato y, depositar el dato en la posición actual si el agente lleva un dato y este es parecido a los datos de esa posición.

La aplicación de este modelo a nuestro problema intenta aprovechar el carácter exploratorio del algoritmo LF basado en múltiples agentes transportadores, para convertir a cada agente no en un mero transportador, sino en un clasificador especializado. En la primera aproximación del modelo, los datos de entrenamiento serán aquellos que no estén bien agrupados y que pertenezcan a la zona del dominio de los datos del agente clasificador especialista. Por tanto, el objetivo del algoritmo LF de obtener agrupaciones se convierte, en nuestro caso, en un medio para que los agentes se muevan en el entorno y seleccionen los datos de entrenamiento, y los agentes, que en el algoritmo LF son meros transportadores, son los agentes clasificadores que queremos obtener.

En este caso, un agente solo elegirá un dato que encuentre si se cumplen las condiciones del algoritmo LF y, además, si el dato encontrado es relevante para fijar su zona de especialización. El tratamiento de zonas donde haya datos de distintas clases hace que se generen agentes cuya estructura contiene información de ellas, y análogamente, zonas con una sola clase provocarán que existan agentes cuya estructura contiene solo a esa clase.

Esta aproximación que proponemos obtiene como resultado un modelo de combinación de clasificadores en el que podemos encontrar bastantes analogías con problemas relacionados con la reducción del conjunto de

entrenamiento. Nuestro objetivo es, por tanto, ubicar este método dentro de un marco general de algoritmos de reducción de datos y evaluar los resultados desde esta perspectiva. Para ello, en la Sección 3.3 se hace un breve repaso de las técnicas existentes para reducción de datos.

En la segunda aproximación que proponemos, vamos a conservar el marco definido por Lumer para definir el entorno y los componentes del sistema, pero eliminando las acciones referidas a obtener agrupaciones de los datos. Además, vamos a redefinir la estructura del agente para permitir una mejor representación de este como clasificador. Se van a desarrollar características para un agente que permitan valorar su calidad, tanto en su formación como cuando se usa como clasificador base en el sistema multiclasificador. En general, un dato del entorno será escogido por un agente para su entrenamiento si el dato mejora la calidad de agente. La zona de especialización de un agente va a estar representada por una memoria que contiene a los m mejores datos usados por el agente para su entrenamiento, es decir, aquellos datos que hacen que su calidad sea máxima. Dado que el objetivo del agente es especializarse en una zona del domino, la calidad se va a medir en términos de la varianza de los datos que componen la zona del agente y de su entropía. De igual forma, un agente devolverá al entorno aquel dato que hace que su calidad sea peor. La actuación de un agente es autónoma en el sentido de que el objetivo del agente es maximizar su función de valoración y tiene una interacción mínima con el entorno ya que coge datos del entorno y devuelve datos al entorno. En un nivel superior al conjunto de agentes está el sistema multiclasificador, que debe encargarse de inicializar el sistema, evaluar su evolución y aplicar los cambios o reajustes necesarios para lograr que el conjunto de agentes generados cumpla las propiedades deseadas.

En la Sección 3.2 se describen, por un lado, los fundamentos del algoritmo LF así como otras modificaciones del algoritmo encontradas en la literatura. En la Sección 3.4 vamos a definir una primera aproximación de nuestro modelo, en la que definiremos la estructura y la funcionalidad de un agente, para posteriormente definir el sistema multiclasificador en términos de generación del conjunto de agente y definición de selección y combinación de agentes. Haremos, en la Sección 3.5, una estructuración análoga al definir la segunda aproximación de nuestro modelo, discutiendo las mejoras en los cambios aportados respecto al primer modelo. Terminaremos, en ambos casos, con la explicación de los experimentos realizados y las conclusiones que podemos sacar de estas dos aproximaciones.

3.2. Algoritmo de agrupamiento de Lumer y Faieta. Variantes

El objetivo de esta sección es establecer los antecedentes de los aspectos que afectan a la definición y comprensión de los modelos que se proponen en este capítulo.

Como se ha descrito, los dos modelos que se definen en este capítulo utilizan como marco de trabajo el modelo descrito por Lumer [66] para la agrupación de datos mediante inteligencia de enjambre. En esta sección se define este modelo y se estudian varias modificaciones encontradas en la literatura.

En el mundo real se ha observado que ciertos tipos de hormigas agrupan cadáveres para formar un cementerio u ordenan sus larvas formando varios montones. Este comportamiento no se comprende totalmente aún, pero un modelo simple, en el cual los agentes se mueven aleatoriamente en el espacio y cogen y depositan objetos en base a información local, puede explicar algunos de los rasgos característicos del agrupamiento y la ordenación de las hormigas reales. El mecanismo básico que subyace en este tipo de agrupaciones es la atracción entre cadáveres. Esta atracción induce a una hormiga que transporta un cadáver a dirigirse, con mayor probabilidad, hacia un grupo grande de cadáveres antes que a un grupo pequeño de estos, de forma que distintos grupos de cadáveres crecen atrayendo a las hormigas para que depositen más cadáveres en ellos. Esta retroalimentación positiva es la que conduce a la formación de agrupaciones cada vez mayores.

El modelo también puede ser aplicado al análisis de datos y particionado de grafos: objetos con diferentes atributos o nodos de un grafo pueden ser considerados como objetos que pueden ser ordenados. Objetos que han sido depositados cerca unos de otros por el algoritmo de ordenación tienen atributos similares, y nodos depositados cerca unos de otros por el algoritmo de ordenación estarán fuertemente conectados en el grafo.

La agrupación distribuida, y posteriormente la ordenación mediante un enjambre de robots, ha servido como referencia para la robótica basada en enjambres. En todos los casos, el robot presenta un comportamiento extremadamente simple: actúa basándose únicamente en información local y comunicación indirecta.

Deneoubourg [23] propone dos modelos muy semejantes para describir el fenómeno de agrupación y ordenación observado en las hormigas. Ambos modelos se basan en el mismo principio siendo, de hecho, la agrupación un caso especial del ordenamiento.

Deneoubourg supone que los elementos son de un solo tipo, de forma que elementos aislados deberían ser cogidos por alguna hormiga y depositados en alguna posición donde hubiese más elementos. La probabilidad de coger un

elemento (Pc) por una hormiga que no lleva en el tiempo t ningún elemento es definida de la siguiente forma:

$$Pc = \left(\frac{k_1}{k_1 + f} \right)^2 \tag{3.1}$$

donde f es la fracción de elementos que son vecinos del agente y k_1 es una constante umbral. Cuando el agente tiene pocos vecinos a su alrededor ($f \ll k_1$), la probabilidad de coger el dato (Pc) se acerca a 1, es decir, la probabilidad de coger un elemento es alta cuando no hay muchos elementos a su alrededor. Análogamente, la probabilidad de coger el elemento (Pc) es pequeña cuando $f \gg k_1$, es decir, cuando hay muchos elementos alrededor del agente.

De igual forma, la probabilidad de dejar un elemento (Pd) que lleva un agente en la posición actual es:

$$Pd = \left(\frac{f}{k_2 + f} \right)^2 \tag{3.2}$$

Donde k_2 es otra constante umbral de forma que, si $f \ll k_2$ entonces Pd tiende a 0 y si $f \gg k_2$, Pd tiende a 1.

Respecto a la ordenación de elementos, el funcionamiento básico del algoritmo sería el mismo, con la diferencia de que ahora los elementos pueden ser de distintos tipos y la función f estará definida en términos de fracción de elementos entre los vecinos del agente que son iguales al que lleva o al que pretende coger el agente.

Algoritmo de Lumer y Faieta (Algoritmo LF)

Lumer y Faieta [66] han generalizado el algoritmo de Deneubourg [23] para aplicarlo al análisis de datos mediante exploración. Este algoritmo necesita definir una medida de distancia o disimilitud d entre objetos en el espacio de atributos. La definición de esta distancia es necesaria en cualquier problema de análisis de datos, donde se manejan objetos que pueden ser descritos por un número finito n de atributos. Un objeto puede ser visto como un punto en $\mathcal{R}^n$, siendo $d(O_i, O_j)$ una métrica como, por ejemplo, la distancia Euclídea.

El algoritmo descrito por Lumer y Faieta [66], en adelante algoritmo LF, proyecta el espacio de atributos en un espacio de dimensión menor, normalmente de dimensión z=2, y pretende construir agrupaciones con la siguiente propiedad: *Las distancias entre elementos de la misma agrupación deberían ser pequeñas respecto a las distancias entre elementos de diferentes*

agrupaciones. Con esta propiedad, Lumer y Faieta pretenden obtener tantas agrupaciones como tipos o clases de datos tengamos.

El algoritmo LF funciona de la siguiente forma: asumimos que $z = 2$, de forma que los agentes que se mueven en este espacio discreto pueden ver los objetos que hay alrededor de la posición r en la que están; exactamente el área que pueden ver es s^2 ($s \times s$ posiciones alrededor de r). Sea $d(O_i, O_j)$ la distancia entre dos objetos O_i y O_j en el espacio de atributos. Si se asume que un agente está en la posición r en el tiempo t, y encontramos un objeto O_i en esa posición, se puede definir la densidad local $f(O_i)$ respecto al objeto O_i en la posición r como:

$$f(O_i) = \begin{cases} \frac{1}{s^2} \sum_{O_j \in Vecinos(sxs)(r)} \left[1 - \frac{d(O_i,O_j)}{\alpha}\right] & \text{si } f > 0 \\ 0 & \text{en otro caso} \end{cases} \tag{3.3}$$

donde $f(O_i)$ es la similitud media del objeto O_i con los objetos O_j que son vecinos de O_i en la posición r y α es un factor que define la escala de disparidad. El objetivo de usar α es determinar cuando dos objetos deberían o no deberían estar próximos uno de otro. Si α es demasiado grande no habrá suficiente discriminación entre los objetos muy diferentes por lo que las agrupaciones podrán contener objetos que no deberían estar juntos, y si su valor es demasiado pequeño, la distancia entre objetos en el espacio de atributos se amplifica, con lo que objetos muy similares no estarán juntos en el mismo grupo debido a una gran discriminación entre ellos.

Examinando el valor de $f(O_i)$ se puede observar que cuando este es muy bajo, cercano a 0, la probabilidad de coger ese elemento por el agente debería ser grande al ser poco similar con sus vecinos, o simplemente por no tener vecinos, similares o no, que formen parte de un grupo. En este sentido, se define la probabilidad de coger un objeto en función de f de la siguiente forma:

$$Pc(O_i) = \left(\frac{k_1}{k_1 + f(O_i)}\right)^2 \tag{3.4}$$

De igual forma, se define la probabilidad de dejar el objeto O_i que lleva un agente como:

$$Pd(O_i) = \begin{cases} 2f(O_i) & \text{si} f(O_i) < k_2 \\ 1 & \text{si } f(O_i) \geq k_2 \end{cases} \tag{3.5}$$

Siendo k_1 y k_2 dos constantes umbrales con el mismo significado que el descrito en el algoritmo de Deneoubourg.

El funcionamiento del algoritmo LF se ilustra en el Algoritmo 4. En este algoritmo se muestra como cada agente se mueve aleatoriamente y, dependiendo de si lleva o no un elemento, comprueba si debe dejarlo en la posición actual o debe coger el elemento de la posición actual, en base a los criterios de semejanza y nivel de disparidad descritos anteriormente.

La inicialización del algoritmo consiste en depositar los objetos y los agentes aleatoriamente en la arena, de forma que las agrupaciones obtenidas tras ejecutar el algoritmo son también aleatorias.

Algoritmo 4 Algoritmo LF

Entrada: Conjunto de agentes, conjunto de datos de entrenamiento

{Inicialización}
para todo dato o_i **hacer**
  Colocar o_i aleatoriamente en la arena
fin para
para todo agente **hacer**
  Situar el agente aleatoriamente en la arena
fin para
{Bucle Principal}
para t=1 to t_{max} **hacer**
  para todo agente **hacer**
    si (agente sin carga) y (lugar ocupado por el dato o_i) **entonces**
      Calcular $P_c(o_i)$
      Generar un número aleatorio R uniforme entre 0 y 1
      si $R \leq P_c(o_i)$ **entonces**
        Coger el dato o_i
      fin si
    si no si (agente cargado con el dato o_i) y (lugar vacío) **entonces**
      Calcular $P_d(o_i)$
      Generar un número aleatorio R uniforme entre 0 y 1
      si $R \leq P_d(o_i)$ **entonces**
        Soltar el dato o_i
      fin si
    fin si
    Mover aleatoriamente al agente a un lugar vecino y libre
  fin para
fin para

Lumer y Faieta [66] ilustran el funcionamiento de este algoritmo usando un ejemplo simple, donde el espacio de atributos es $\mathcal{R}^2$ y los valores de los dos atributos para cada objeto corresponden a las coordenadas (x, y) en $\mathcal{R}^2$. Se generan cuatro agrupaciones de 200 puntos cada una, con x e y según la distribución Normal $\mathcal{N}(\mu, \sigma)$ de media μ y varianza σ^2:

$$\begin{array}{ll} x \propto \mathcal{N}(0{,}2, 0{,}1) & y \propto \mathcal{N}(0{,}2, 0{,}1) \\ x \propto \mathcal{N}(0{,}8, 0{,}1) & y \propto \mathcal{N}(0{,}2, 0{,}1) \\ x \propto \mathcal{N}(0{,}8, 0{,}1) & y \propto \mathcal{N}(0{,}8, 0{,}1) \\ x \propto \mathcal{N}(0{,}2, 0{,}1) & y \propto \mathcal{N}(0{,}8, 0{,}1) \end{array}$$

El resultado de este experimento fue que los objetos pertenecientes a la misma distribución fueron agrupados juntos y los que no pertenecían a la misma distribución se encontraron en diferentes agrupaciones. El problema de este resultado, según Lumer y Faieta, es que el número de agrupaciones obtenidas es mayor que el número de clases de los datos iniciales, con lo que varias agrupaciones de datos estarían representando a datos con la misma clase. Para corregir esta tendencia, Lumer y Faieta, han añadido tres características:

1. **Agentes con diferentes velocidades de movimiento**. El objetivo es que los agentes tengan distinta percepción de la similaridad entre un objeto y sus vecinos y, además, cada agente pueda en un solo movimiento recorrer más de una casilla. Si el número máximo de casillas recorridas en un movimiento es v_{max}, y cada agente tiene una velocidad asignada uniformemente entre $[1, v_{max}]$ entonces la densidad local $f(O_i)$ será:

$$f(O_i) = \begin{cases} \frac{1}{s^2} \sum_{O_j \in Vecinos(sxs)}(r) \left[1 - \frac{d(O_i, O_j)}{\alpha\left(1 + \frac{v-1}{v_{max}}\right)}\right] & \text{si f>0} \\ 0 & \text{en otro caso} \end{cases} \tag{3.6}$$

   Con esta definición, los agentes que tienen movimientos rápidos no son tan selectivos como los que tienen movimientos lentos, con lo que esta diversidad de agente permite formar agrupaciones sobre varias escalas simultáneamente.

2. **Una memoria pequeña**. Los agentes pueden recordar los últimos m elementos que han dejado en la arena junto con sus posiciones. Cada vez que el agente coge un elemento, lo compara con los m elementos que tiene memorizados y se desplaza a la posición del que sea más semejante en vez de elegir una nueva posición aleatoria. Este comportamiento conduce a una reducción del número de agrupaciones al dirigir al agente a un grupo anteriormente utilizado.

3. **Cambio de comportamiento**. Lumer y Faieta han añadido la posibilidad de que los agentes destruyan agrupaciones si ellos no han realizado ninguna acción en n pasos. Esto permite al sistema escapar de configuraciones locales no óptimas.

Evaluación del algoritmo de LF

El algoritmo LF no es paramétrico respecto al número de agrupaciones que se obtienen ni a la localización final de cada agrupación. En este sentido, la versión parametrizada de este algoritmo ha sido desarrollada por Franks y Deneubourg [24]. El algoritmo que proponen utilizar platillas para forzar a que los elementos sean depositados en un número de regiones predefinido, de forma que el número de agrupaciones y sus localizaciones en el espacio de representación se determinan a priori.

Si comparamos el algoritmo de LF con las técnicas de agrupación, este algoritmo se basa en la mezcla de dos metodologías: algoritmos jerárquicos y de particionado. En los algoritmos jerárquicos, todos los elementos se consideran agrupaciones, y los pares de agrupaciones que más se asemejan van siendo mezclados iterativamente, formando agrupaciones cada vez más grandes y ofreciendo soluciones al problema del agrupamiento en niveles de granuralidad diferentes. En contraste, los algoritmos de particionado necesitan generar una partición inicial de los datos e iterativamente las agrupaciones iniciales se van mejorando moviendo elementos a la agrupación con el centroide más cercano.

El algoritmo LF se asemeja a los algoritmos jerárquicos en que se incluye un elemento en una agrupación después de comparar los atributos del elemento con los elementos de la agrupación mientras que en los algoritmos de particionado se compara el elemento con la media de los atributos de los elementos de la agrupación, pero el algoritmo no permite agrupaciones en distintos niveles. Respecto a los algoritmos de particionado, el algoritmo LF se asemeja a estos en que los elementos se mueven de una agrupación a otra, refinando la partición en cada paso del algoritmo.

Algunos de los problemas y posibles modificaciones del algoritmo LF son enumerados a continuación:

1. Convergencia y parada del algoritmo.

2. Conflicto espacial entre distintas agrupaciones en la misma zona. Dado que la localización de las agrupaciones que se forman es aleatoria, puede ocurrir que agrupaciones que representan a clases diferentes se lleguen a unir. Esto implica que una hormiga situada en la frontera de ambas agrupaciones tenderá a coger un objeto al compararlo con sus vecinos, ya que los vecinos pertenecen a distintas agrupaciones, por lo que el crecimiento natural de una agrupación queda limitado.

3. No evaluación del tamaño de la agrupación y su homogeneidad respecto a la definición de la probabilidad de coger y dejar. En base a asegurar una convergencia rápida del algoritmo a la solución final, el algoritmo debería permitir que agrupaciones grandes y bien formadas sean menos propensas a que sus elementos sean cogidos por una hormiga. Esta

propiedad no se cumple debido a que la hormiga solo tiene información de sus vecinos, por lo que la actuación de esta es exactamente igual, tanto si la agrupación es grande como si es pequeña.

4. Asignación de todos los elementos a alguna agrupación. El algoritmo puede terminar sin haber asignado todos los elementos a alguna agrupación, tanto si hay hormigas con elementos cuando termina el algoritmo como si hay datos en la arena sin asignar a una agrupación.

Extensiones y aplicaciones realizadas sobre algoritmo LF

Para mejorar el rendimiento dela lagoritmo LF se han propuesto una serie de ampliaciones y modificaciones que comentamos a continuación.

Una de las características del algoritmo LF es que el número de agrupaciones que se obtienen y su localización no se puede determinar como parámetro de entrada. En algunos problemas reales se sabe a priori cual es el número de categorías en las que debemos agrupar los datos, por lo que la parametrización del algoritmo LF resulta imprescindible para poder aplicarlo a dichos problemas.

Una solución a la parametrización [24] es combinar el mecanismo de auto-organización que subyace en el algoritmo LF con la utilización de plantillas. El objetivo es que la probabilidad de coger o dejar un objeto no dependa únicamente de los vecinos del objeto sino también de la posición actual. Para ello se define la probabilidad de estar en una zona definida por una platilla $P_t(x, y)$, donde (x, y) es la coordenada en el plano del agente. Si se asume que el espacio está definido en $[0, 1]x[0, 1]$ y que queremos obtener cuatro agrupaciones, se puede definir $P_t(x, y)$ como:

$$P_t(x,y) = a\left[e^{-\frac{x^2+y^2}{\sigma^2}} + e^{-\frac{(x-1)^2+y^2}{\sigma^2}} + e^{-\frac{(x-1)^2+(y-1)^2}{\sigma^2}} + e^{-\frac{x^2+(y-1)^2}{\sigma^2}} - b\right] \tag{3.7}$$

donde σ^2 representa la pendiente de la plantilla y a y b son parámetros que ajustan a σ^2 tal que $0 \leq P_t(x, y) \leq 1$.

Si introducimos esta probabilidad en la definición de la probabilidad de coger P_c y en la probabilidad de dejar P_d del algoritmo LF, podríamos obtener las siguientes ecuaciones:

$$Pc(O_i) = w_1\left(\frac{k_1}{k_1 + f(O_i)}\right)^2 + w_2(1 - P_t(r_i)) \tag{3.8}$$

donde $r_i = (x_i, y_i)$ es la posición del objeto O_i, w_1 y w_2 son constantes que determinan el grado de influencia de cada uno de los términos. En

esta nueva definición, la probabilidad de coger un elemento dependerá tanto de sus vecinos como de su situación en el espacio, de forma que, si un elemento está dentro de una zona de una plantilla, este será cogido con menor probabilidad. De forma análoga se puede redefinir la probabilidad de dejar un elemento como:

$$Pd(O_i) = \begin{cases} 2P_t(r_i)f(O_i) & \text{si} f(O_i) < k_2 \\ P_t(r_i) & \text{si } f(O_i) \geq k_2 \end{cases} \quad (3.9)$$

Algoritmo de Monmarché: k-medias y agrupación mediante hormigas

N. Monmarché [76] propone un nuevo algoritmo, llamado AntClass, para obtener agrupaciones de datos basado en la hibridación del algoritmo LF y el algoritmo de agrupación k-medias.

El objetivo principal es definir un algoritmo de agrupación jerárquico en el que en un primer nivel de agrupación se obtiene una primera partición de los datos, aplicando la agrupación mediante hormigas y refinándola con el algoritmo k-media. El segundo nivel de este algoritmo consiste en volver a aplicar al agrupación mediante hormigas y el algoritmo k-media, siendo los datos de entrada a este nivel las agrupaciones obtenidas por la primera aplicación del algoritmo de hormigas y el refinamiento del algoritmo k-media (Algoritmo 5).

Algoritmo 5 AntClass

1: Aplicar la agrupación basada en hormigas sobre los datos iniciales, obteniendo como resultado k agrupaciones H_i, $i \in [1..k]$
2: Aplicar el algoritmo k-medias, tomando como entrada $H_1, ..., H_k$, $i \in [1..k]$, Obteniendo k' agrupaciones.
3: Aplicar nuevamente el algoritmo de agrupación de hormigas sobre $H_1, ..., H_{k'}$.
4: Aplicar nuevamente el algoritmo k-medias, tomando como entrada $H_1, ..., H_{k'}$, obteniendo k'' agrupaciones.

AntClass aporta importantes mejoras respecto al algoritmo LF debido a que construye explícitamente las agrupaciones. El algoritmo crea montones de objetos localizados cada uno en una posición distinta en el espacio de dos dimensiones, de esta forma, la hormiga es capaz de crear un montón, modificarlo e incluso eliminarlo. Con la definición de estos montones se consiguen los siguientes objetivos:

1. Fácil localización de una agrupación. El algoritmo LF ofrece únicamente una solución visual en el espacio 2D sobre los agrupamientos resultantes de aplicar el algoritmo. AntClass resuelve este problema ya que en un paso de algoritmo la hormiga tiene información de todas las

posibles agrupaciones que hay a su alrededor, aunque esta solución supone un cambio importante en la noción de "hormiga que ve solo su entorno".

2. Posibilidad de manejar información global de la agrupación. En este sentido AntClass define varias medidas sobre la agrupación. Si H es un montón, se define:

   - $D_{max}(H)$ es la distancia máxima entre dos objetos de H, donde:

   $$D_{max}(H) = max_{O_i,O_j \epsilon H} D(O_i, O_j) \tag{3.10}$$

   - O_{center} es el centro de masas de todos los objetos de H:

   $$O_{center} = \frac{1}{n_H} \sum_{O_i \epsilon H} O_i \tag{3.11}$$

   - O_{dissim} es el objeto que menos se asemeja a los demás objetos de H, el cual maximiza $D(O_{dissim}, O_{center})$.
   - D_{mean} es la distancia media entre los objetos de H y el centro de masas O_{center}:

   $$D_{mean} = \frac{1}{n_H} \sum_{O_i \epsilon H} D(O_i, O_{center}) \tag{3.12}$$

   Estas medidas globales sobre la agrupación permiten tener una heurística más exacta para coger, dejar o eliminar objetos del montón.

3. Evita la colisión entre agrupaciones. Aunque la localización de las agrupaciones en AntClass sigue siendo aleatoria, las fronteras entre cada una de ellas están perfectamente delimitadas.

Respecto a la evaluación general de AntClass se destacan a continuación las siguientes propiedades:

- Generación de la partición inicial con hormigas. Muchos algoritmos de agrupación (incluido el algoritmo K-media) requieren una partición inicial de los datos. La generación de esta partición inicial por el algoritmo de agrupación de hormigas, puede suponer una buena solución a este problema, sobre todo porque la parada del algoritmo no requiere condiciones estrictas de terminación.

- Todos los datos son asignados a alguna agrupación cuando termina el algoritmo.

3.3. Reducción del conjunto de entrenamiento

Los algoritmos de aprendizaje basados en casos (Instance-based learning algorithms) parten de un conjunto de entrenamiento formado con ejemplos llamados *instancias*, a partir del cual generan un modelo que será utilizado para clasificar (generalizar) nuevos ejemplos. La generalización del nuevo ejemplo se basa en calcular la distancia entre este y un conjunto de prototipos almacenados, de forma que la etiqueta asignada al nuevo ejemplo depende de las etiquetas de prototipos parecidos. El concepto de prototipo almacenado, en estos algoritmos, se entiende como una representación de las instancias del conjunto de entrenamiento. Por tanto, un prototipo puede ser tanto una instancia del conjunto original de instancias como un representante de ese conjunto (el cual no tiene por que ser una instancia real).

Tanto si se almacenan instancias reales como si se almacenan representaciones de estas, uno de los problemas a los que se enfrentan estos algoritmos es el de decidir que instancias o prototipos almacenar para su uso durante la generalización. Almacenar demasiados prototipos puede ocasionar grandes requerimientos de memoria, una velocidad de ejecución lenta y pueden causar sobresensibilidad al ruido.

Uno de los algoritmos basados en prototipos más simples es el algoritmo del vecino más próximo [20, 21, 42]. En este y otros algoritmos basados en instancias [1], los prototipos son instancias originales del conjunto inicial de entrenamiento y la clasificación de una nueva instancia se produce obteniendo la etiqueta de la instancia más cercana al nuevo dato (1-nn) o bien eligiendo la etiqueta mayoritaria del conjunto de las k instancias más cercanas (k-nn). La ventaja de este método es que el entrenamiento consiste únicamente en almacenar el conjunto de instancias pero tiene el inconveniente de que necesita gran cantidad de memoria para almacenarlas y que el tiempo de ejecución para calcular las distancias con todos las instancias almacenadas es grande.

El reto para mejorar este tipo de algoritmos es conseguir reducir el conjunto de prototipos almacenados sin que la tasa de error al generalizar aumente e, incluso, intentar que disminuya. En general, los algoritmos para reducir el conjunto de prototipos se caracterizan por los criterios que más influyen en su construcción. Estos criterios son: reducción de la memoria de almacenamiento, incremento de la velocidad de clasificación, decremento de la tasa de error y mayor tolerancia al ruido. Algunos artículos sobre técnicas de reducción son descritos por Wilson [99].

Una variación del clasificador del vecino más próximo es el *clasificador de mínima distancia*, el cual se basa en almacenar un único prototipo por cada clase. Cada clase está representada por un único prototipo, que es la media de todos los ejemplos de esa clase de forma que la clasificación de una nueva instancia asignará la clase del prototipo más cercano a esa

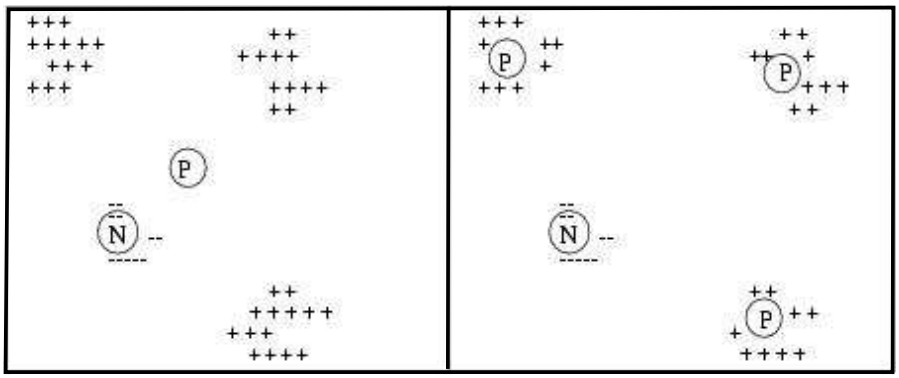

Figura 3.1: Grafico Kibler

instancia. Como consecuencia de la obtención de prototipos, el número de prototipos almacenados coincide con el número de clases que intervienen en el problema. Este método tiene el inconveniente de que solo funciona bien en dominios donde los ejemplos de una misma clase están agrupados en una única zona del espacio.

Kibler y Datta [22] proponen una modificación de este algoritmo que consiste, básicamente, en aprender más de un prototipo para cada clase. La motivación que dan los autores queda reflejada en la Figura 3.1. En esta figura se puede observar como un prototipo no representa espacialmente a los ejemplos de una clase, cuando estos forman a su vez subgrupos de elementos parecidos. En la misma figura se puede observar que la creación de un prototipo para cada subgrupo de elementos de una clase, produce un conjunto de prototipos que serán capaces de clasificar mejor en dominios donde los ejemplos están separados en grupos disjuntos y distantes. El algoritmo que proponen Kibler y Datta consiste en realizar agrupaciones utilizando el algoritmo de k-medias. También obtiene para cada clase el número de agrupaciones necesarias, siendo cada agrupación un prototipo.

Desde la perspectiva de sistemas multiclasificadores hay algunos trabajos que hacen referencia a la reducción del conjunto de entrenamiento usando/generando varios clasificadores [3, 56]. En estos trabajos se selecciona, en vez de un único subconjunto de ejemplos, varios subconjuntos de ejemplos complementarios en los errores que cometen, de forma que cada nuevo subconjunto representa un clasificador del vecino más próximo y la clasificación final es la clase más votada. La forma de obtener los subconjuntos es aplicar n veces una reducción del conjunto original, obteniendo n conjuntos que son utilizados por n clasificadores.

En los siguientes apartados se describen algunas técnicas de reducción del conjunto de prototipos, agrupándolas según obtengan un subconjunto del conjunto inicial de prototipos o un nuevo conjunto con una representa-

ción de sus elementos distinta a la inicial.

Reducción obteniendo un subconjunto del inicial

Referente a la regla del vecino más próximo, encontramos varios métodos que intentan seleccionar un subconjunto de los prototipos iniciales.

La regla del vecino más próximo condensada (CNN) fue definida por Hart [42]. Este algoritmo encuentra un subconjunto S del conjunto de entrenamiento T, tal que cada miembro de T está más cerca de un miembro de S con su misma clase que de un miembro de S con diferente clase. Cumpliéndose esta propiedad, el subconjunto S puede usarse para clasificar todas las instancias de T correctamente, pero no garantiza un conjunto mínimo. Este algoritmo es especialmente sensible a instancias con ruido, las cuales serán almacenadas para asegurar la propiedad anterior.

Wilson [98] describe el algoritmo del vecino más próximo editado (ENN). En este algoritmo se pretenden eliminar las instancias con ruido e instancias cercanas al borde. El algoritmo parte del conjunto S con los mismos elementos que el conjunto T original y cada elemento en S es eliminado si su clase no concuerda con la clase mayoritaria de sus k vecinos más cercanos. De forma más general, otros métodos de reducción aplicados a algoritmos basados en instancias son: la regla del vecino más próximo reducido presentado por Gates [34] y Kibler [48]); Model Class Selection (MCS) definido por Brodley [11]; el algoritmo TIBL descrito por Zhang [109] y Random Mutation Hill Climbing (Skalak, [90]).

Reducción generando nuevas representaciones

La otra forma de reducir el conjunto de datos almacenados es cambiando la representación de estos. El objetivo es partir de un conjunto original de instancias (T) y obtener un conjunto de datos (S) con menos elementos que T y donde cada nuevo dato tiene una representación distinta de las instancias originales. La idea general en estos métodos es agrupar la información de varias instancias en un único representante del nuevo conjunto.

Algunos de estos métodos son los siguientes:

Prototipos. Chang [18] y posteriormente Bezdek [6] presentan algoritmos que parten del conjunto inicial (T) donde cada instancia se considera un prototipo, y mezclan sucesivamente (por media simple o ponderada) los pares de prototipos más cercanos en una clase. Este procedimiento se repite hasta que la efectividad en la clasificación comienza a empeorar.

RISE. Domingos [27] presenta el sistema RISE 2.0, el cual trata cada

instancia de T como una regla en el nuevo conjunto R. Para cada regla r en R, se busca el ejemplo más próximo n en T tal que no esté cubierto por la regla R. La regla r se modifica para que cubra al ejemplo n.

EACH. Salzberg [86] presenta la teoría "Nested generalized Exemplar"(NGE), en la cual se usan hiper-rectángulos para ubicar a una o más instancias, reduciendo así los requerimiento de almacenamiento.

Wettscherck y Dietterich [97] presentan un algoritmo híbrido entre el vecino más próximo y el hiper-rectángulo más próximo. En este algoritmo se almacenan hiper-rectángulos y todas las instancias del conjunto original. La ganancia se produce al clasificar un nuevo dato, que en la mayoría de los casos se produce usando un hiper-rectángulo, y en los casos en los que la entrada no es cubierta por ningún hiper-rectángulo, se clasifica mediante los k vecinos más cercanos.

3.4. Modelo de multiclasificador basado en el algoritmo LF (MBLF)

En esta sección se desarrolla un modelo que pretende ser una primera aproximación al modelo final propuesto en el capítulo 4 y que está basado en el comportamiento de las hormigas del algoritmo LF. Llamaremos a este modelo: "Modelo basado en el algoritmo LF o modelo MBLF".

Nuestro objetivo es diseñar un sistema multiclasificador basado en la generación de múltiples agentes clasificadores independientes y simples, cuyo único mecanismo de comunicación es la modificación del entorno en el que se mueven, y cuyas acciones se basan en la elección y procesado de los datos que van a conformar su conjunto de entrenamiento. Para conseguir este objetivo, el método que se propone [14] va a aprovechar el carácter exploratorio del algoritmo LF, basado en múltiples agentes transportadores. El método va a convertir a cada agente en un clasificador especializado, cuyos datos de entrenamiento serán aquellos que no estén bien agrupados y que pertenezcan a la zona del dominio de los datos del agente clasificador especialista. Por tanto, el objetivo del algoritmo LF de obtener agrupaciones se convierte, en nuestro algoritmo, en un medio para seleccionar los datos de entrenamiento; y las hormigas, que en el algoritmo LF eran meras transportadoras, son en nuestro caso los agentes clasificadores que queremos obtener.

Dado que el conjunto de agentes va a ser el conjunto de clasificadores del sistema multiclasificador, vamos a definir nuestro modelo atendiendo a tres aspectos: como entendemos un agente como clasificador, como podemos definir el sistema multiclasificador en base al conjunto de agentes y, por último, como se usa el sistema multiclasificador.

Respecto a la definición del agente, se estudiará cual va a ser su estructura, es decir, que elementos lo constituyen y que características tienen estos elementos como componentes de un clasificador. A continuación se describirá cual es el comportamiento del agente cuando se mueve en el entorno; como afecta a su construcción un nuevo dato de entrenamiento; y que lo caracteriza como un clasificador. Una vez descrita la estructura y funcionalidad de un agente se estudian las características del sistema multiagente, así como su uso para la clasificación de un dato nuevo. Se finalizará la descripción de esta primera aproximación exponiendo una serie de experimentos sobre el modelo y las conclusiones sobre dichos experimentos.

3.4.1. Estructura de un agente clasificador

Para definir la estructura de un agente se debe tener en cuenta su formación iterativa y su posterior uso como parte del sistema multiclasificador.

Sea $H = \{A_1, ..., A_{n_A}\}$ el conjunto de agentes. La estructura de cada agente, A_i en un instante dado viene definida por la tupla $A_i = ((x_i, y_i), nd, n_{c_i}, P_A, c_A, Cl_A, d_t)$ donde :

- $(x_i, y_i) \in N^2$ es la localización del agente en el espacio bidimensional del agente.
- $nd \in N$ es el número de datos de entrenamiento utilizados por el agente.
- $n_{c_i} \in N$ es el número de datos que han sido cogidos con clase i.
- $P_A \in R^m$ es el valor prototípico del agente, donde m es el número de atributos que definen el prototipo.
- $c_A \in Cl$ es la clase asociada al prototipo del agente.
- $Cl_A \subseteq Cl$ es el subconjunto de clases que han intervenido en la formación del agente.
- d_t es el dato que transporta el agente.

Caracterización de la zona de especialización del agente

Un agente en el sistema multiclasificador va a representar a un clasificador especializado en una subzona del dominio total. Desde esta perspectiva, su estructura debe reflejar la información de la zona en la que se está especializando así como las clases que intervienen en esa zona de especialización.

Sea $E = \{e = (\vec{x}, c_i) | \ \vec{x_i} = (x_1, \ldots, x_m) \in R^m \text{ y } c_i \in Cl\}$ el conjunto que contiene a todos los datos de entrenamiento del sistema multiagente, donde

$e = (\vec{x}, c_i)$ es un dato de entrenamiento formado por un vector $\vec{x}$ con m atributos ($\vec{x} = (x_1, \ldots, x_m)$) y una clase c_i asociada a $\vec{x}$, donde $c_i \in Cl$; siendo Cl el conjunto de clases que intervienen en E. Sea E_A el conjunto de datos que han sido seleccionados y utilizados por el agente en su entrenamiento. Este conjunto representa a la zona en la que se ha especializado el agente.

Gran parte de las decisiones que el agente toma en su entrenamiento y uso están determinadas por su zona de especialización. Dado que una de las propiedades que queremos para el agente es que sea simple, no podemos considerar que contenga, como representación de su zona, al conjunto E_A. Por esta razón, es importante encontrar una estructura que resuma esta información y que represente adecuadamente a la zona.

Una posible estructura simple que represente a la zona de especialización es un valor prototípico, entendido como un valor con la misma estructura que un dato y que representa al conjunto de datos. Una opción para la elección del prototipo es asignar el valor del conjunto de datos que mejor represente al conjunto. En este caso, el prototipo es un dato real obtenido del conjunto pero tiene el inconveniente de que debemos usar un método que calcule el mejor dato en cada instante. La alternativa a elegir un dato real es obtener un dato ficticio que represente al centro de la zona de especialización del conjunto E_A.

Llamaremos prototipo del agente P_A al valor que representa a la media de los datos del conjunto E_A, obtenidos de la siguiente forma:

$$P_A = \{\bar{x}_1, \ldots, \bar{x}_m\} \tag{3.13}$$

donde $\bar{x_i}$ representa a la media del atributo i de todos los datos cogidos por el agente. Utilizaremos la media por ser el valor que hace mínimo el promedio de las distancias entre los datos y ella,

$$\bar{x_i} = \frac{1}{n} \sum_{j=1}^{n} x_{ij}$$

Caracterización de las clases que intervienen en un agente

La función de un agente dentro del sistema multiagente es predecir la clase de un dato si el agente es seleccionado para su clasificación, es decir, si el dato a clasificar está dentro de su zona de especialización. Si la zona de especialización está definida por los datos de entrenamiento que el agente ha utilizado en su formación, las posibles clases que puede predecir serán aquellas que han intervenido en su zona de especialización. Llamaremos Cl_A al conjunto de clases que intervienen en el conjunto de entrenamiento del agente, donde $Cl_A = \{c_1, \ldots, c_j\}$, siendo $Cl_A \subseteq Cl$.

Siguiendo la filosofía de construir un agente simple, vamos a utilizar una única clase que va a representar a la mejor clase dentro del agente. Esta clase, llamada c_A, va a ser la clase mayoritaria dentro del conjunto de datos E_A usado por el agentes. Para poder obtener la clase mayoritaria, se va a mantener información del número de datos en E_A que han sido utilizados para cada una de las clases (n_{c_i}), así como en número de datos de entrenamiento usados (nd).

Dado que un agente se va a corresponder con la noción de hormiga del algoritmo LF, cada agente mantendrá información de su posición actual dentro del espacio de dimensión 2x2 (arena). Esta posición se denota como (x_i, y_i). El agente también necesita poder contener a un dato que no estaba bien agrupado, para transportarlo a otra posición en la arena, por tanto, el valor d_t será este dato.

3.4.2. Creación de un agente

La información de inicialización de un agente marca el comportamiento que va a tener el agente en su entrenamiento y va a influir notoriamente en la subzona en la que va a ser especialista. Si el sistema multiclasificador hiciera un clustering inicial del conjunto E de entrenamiento se podría pensar en inicializar a cada agente con información de cada cluster, con lo que estaríamos dirigiéndolo a formarse en zonas preestablecidas.

La filosofía que se pretende seguir en la definición de nuestro modelo es la de construir agentes cuyo comportamiento sea lo más simple y autónomo posible, y que sea la propia evolución del sistema la que defina al agente final, en vez de predeterminar su evolución. Por este motivo, el agente va a ser lanzado sin información previa de ninguna zona. Esto significa que el primer dato que encuentre y que no esté bien agrupado va a ser el primer dato que defina su zona y, por tanto, va a ser el valor inicial del prototipo P_A.

Cuando se detalle posteriormente el comportamiento del sistema multiclasificador, se expondrán las técnicas y criterios utilizados para obtener el número de agentes del sistema y su creación.

3.4.3. Entrenamiento de un agente

Un agente se mueve dentro de la arena manteniendo el objetivo inicial del algoritmo LF de transportar datos. Por tanto, en un momento dado del entrenamiento, las acciones que realiza dependerán de su posición en la arena; de si transporta o no transporta un dato; y de si en la posición actual hay un dato o la posición está vacía.

Suponiendo que el agente está en la posición (x_i, y_i), podemos resumir su

operatividad en las cuatro siguientes situaciones:

1. Si el agente está sin carga y la posición actual está vacía:
   - Moverse a una nueva posición (x_j, y_j) aleatoria.
2. Si el agente está sin carga y la posición actual está ocupada por un dato e el agente debe:
   - Si e está mal colocado en la arena y e pertenece a la zona del agente:
     - Coger el dato e para transportarlo.
     - Actualizar el agente con el nuevo dato.
   - Moverse a una nueva posición (x_j, y_j) aleatoria.
3. Si el agente está cargado con e y la posición actual está vacía:
   - Si los vecinos del agente son parecidos a e:
     - Dejar el dato del agente en la posición actual.
   - Moverse a una nueva posición (x_j, y_j) aleatoria.
4. Si el agente está cargado con e y la posición actual está ocupada:
   - Moverse a una nueva posición (x_j, y_j) aleatoria.

La situación 1 y 4 provocan que el agente no pueda realizar ninguna acción respecto a coger o dejar datos, por lo que realiza un movimiento a una nueva posición en la arena. El punto 3 caracteriza a la situación en la que el agente transporta un dato, en cuyo caso debe soltarlo en la posición actual si está rodeado de datos parecidos, o bien, seguir transportándolo a una nueva posición.

El punto 2 es el que va a definir el entrenamiento del agente con el dato encontrado en la arena. En el algoritmo LF, si no va cargado y se encuentra con un dato mal colocado lo coge para transportarlo, moviéndose a una nueva posición aleatoria. El algoritmo que definimos va a añadir una restricción a esa acción: que el dato pertenezca a la zona del agente, de forma que el agente solamente coge el dato si le sirve en su entrenamiento y está mal colocado en la arena. Esta restricción produce, respecto al algoritmo LF, que el dato solo sea transportado por agentes afines a él. Por tanto, este es el contexto en el que el agente interactúa con el entorno y selecciona sus datos de entrenamiento.

A continuación se detallan cada una de las acciones que se han descrito anteriormente y que el agente debe llevar acabo, así como los elementos y

conceptos que intervienen en ellas.

El Algoritmo 6 muestra la generación del conjunto de agentes, que será explicado en la sección de generación de agentes dentro del apartado de sistema multiclasificador (Sección 3.4.4).

Algoritmo 6 Generación de los agentes en el modelo basado en el algoritmo LF

Entrada: Conjunto de agentes, conjunto de datos de entrenamiento
Salida: Conjunto de agentes entrenados

```
{Inicialización}
para todo dato e hacer
   Colocar e aleatoriamente en la arena
fin para
para todo agente hacer
   Situar el agente aleatoriamente en la arena
fin para
{Bucle Principal}
para t=1 to t_max hacer
   para todo agente hacer
      si (agente sin carga) y (lugar ocupado por el dato e) entonces
         Calcular P_c(e)
         si Random(0,1) ≤ P_c(e) and Random(0,1) ≤ Fp_A(e) entonces
            Coger el dato e
            Actualizar agente con e
         fin si
      si no si (agente cargado con el dato e) y (lugar vacío) entonces
         Calcular P_d(e)
         si Random(0,1) ≤ P_d(e) entonces
            Soltar el dato e
         fin si
      fin si
      Mover aleatoriamente al agente a un lugar vecino y libre
   fin para
fin para
```

Determinación de si un dato está bien agrupado en la arena

Para que un agente pueda coger un dato que se encuentra en la arena o dejar el dato que transporta en la posición actual, tiene que cumplirse la condición de que el dato esté mal agrupado, en caso de que queramos cogerlo, o de que esté bien agrupado en caso de querer dejarlo. En cualquier caso, el agente debe comparar el dato con los vecinos que están alrededor de la posición actual.

En este sentido, se va a mantener la definición de la función de similitud

f del algoritmo LF definida en la Ecuación 3.3, que obtiene la similitud entre el dato actual y los $s \times s$ vecinos de la posición actual.

$$f(e_i) = \begin{cases} \frac{1}{s^2} \sum_{e_j \in Vecinos(s \times s)(r)} \left[1 - \frac{d(e_i,e_j)}{\alpha}\right], & \text{si } f > 0 \\ 0, & \text{en otro caso} \end{cases}$$

La probabilidad de coger un dato, P_c, está definida, al igual que en el algoritmo LF (Ecuación 3.4), en orden inverso a la función de similitud: a mayor similitud entre los datos la probabilidad de coger un dato disminuye y viceversa,

$$P_c(e_i) = \left(\frac{k_1}{k_1 + f(e_i)}\right)^2$$

De igual forma, la probabilidad de dejar un dato, P_d, es proporcional a la similitud del dato con sus vecinos. Usaremos la definición del algoritmo LF (Ecuación 3.5),

$$P_d(e_i) = \begin{cases} 2f(e_i), & \text{si} f(e_i) < k_2 \\ 1, & \text{si } f(e_i) \geq k_2 \end{cases}$$

Determinación de si un dato pertenece a la zona del agente

El agente necesita saber si un dato pertenece a su zona de especialización en dos situaciones. En su entrenamiento, cuando encuentra un dato y este está mal agrupado, debe decidir si el dato va a servirle de entrenamiento. Cuando el sistema multiclasificador quiere clasificar un dato, va a elegir a un agente solamente si el dato pertenece a la zona del agente.

La caracterización de la zona de especialización de agente la hemos definido con un valor prototípico P_A, calculado como la media de los datos de entrenamiento vistos por el agente hasta el instante actual (Ecuación 3.13).

La ventaja de utilizar este prototipo es que tenemos una representación simple de la zona de especialización del agente, cuya estructura es la misma estructura de un dato $\vec{x}$. El prototipo, además, representa al centro de la zona de especialización por lo que es un buen representante de su zona.

Si se normaliza la distancia entre dos datos de forma que la distancia se encuentre en el intervalo [0,1], la misma función de distancia puede servir para evaluar el grado de pertenencia de un dato a la zona del agente. Si calculamos la distancia entre un dato y el prototipo, una distancia cercana a cero indica un alto grado de pertenencia del dato a la zona, mientras que si

la distancia se acerca a uno el dato tiene un grado de pertenencia muy bajo a la zona.

Definimos, por tanto una distribución de probabilidad Fp_A, que determina la pertenencia de un dato a la zona de agente como:

$$Fp_A(x) = 1 - d(x, P_A) \tag{3.14}$$

Dado que el agente necesita decidir si el dato pertenece o no a la zona, vamos a considerar un proceso de tipo montecarlo donde Fp_A será la probabilidad establecida. Generamos un número aleatorio uniforme en [0,1] y la pertenencia será cierta si el valor de $Fp_A(e)$ es mayor que ese número. El uso de un número aleatorio va a permitir que un dato con un grado de pertenencia alto sea muy probable que sea seleccionado pero no lo será en todos los casos, con lo que dotamos de cierto indeterminismo al sistema.

Actualización del agente con un dato de la arena

El algoritmo propuesto para la generación de los agentes (algoritmo 6) sigue la estructura del algoritmo LF, añadiendo el entrenamiento con el dato e, si este dato está mal colocado en la arena y el agente no va cargado con ningún dato. Estas son las condiciones para poder considerar a un dato como candidato para su entrenamiento.

Una vez que el dato cumple las condiciones para ser cogido, el algoritmo comprueba que el dato pertenece a la zona del agente y, en caso afirmativo, lo usa como dato de entrenamiento. La inclusión de estas acciones se ve reflejada con la inserción de los puntos 13,14 y 15 descritos a continuación:

- **Si** $Random(0,1) \leq P_c(e)$ and $Random(0,1) \leq Fp_A(e)$ **entonces**
  - Coger el dato e
  - Actualizar agente con e
- **endif**

Donde $Fp_A(e)$ es la función de pertenencia del dato e al agente (definida en la Ecuación 3.14). Al igual que se hace en el algoritmo LF, se va a generar un número aleatorio entre 0 y 1 ($Random(0,1)$) para evitar óptimos locales. Cuando se calcule la probabilidad de coger P_c o la probabilidad de dejar P_d, la acción de coger o dejar dependerá también del número aleatorio generado.

Si se el dato está mal colocado y pertenece a la zona del agente servirá para su entrenamiento, por lo que el agente se actualiza con el dato e.

La actualización va a consistir en incluir la información del dato e en el prototipo P_A del agente. Dado que el prototipo representa a la media de los datos que el agente ha seleccionado hasta el momento, el nuevo prototipo debe ser recalculado de forma iterativa para ser la media de todos los datos, incluyendo a e.

Si $e = (\vec{x}, c_i)$, donde $\vec{x} = (x_1, \ldots, x_m) \in R^m$, $c_i \in Cl$ es la clase asociada al dato e y $P_A = \{\bar{x}_1, \ldots, \bar{x}_m\}$, definimos $\Delta\bar{x}_j$ como la diferencia entre el atributo j del prototipo y el atributo j del dato de la siguiente forma:

$$\Delta\bar{x}_j = \bar{x}_j - x_j \tag{3.15}$$

Como el atributo j del prototipo es la media del atributo j de todos los datos nd cogidos hasta el momento, el nuevo valor del prototipo se calcula obteniendo las nuevas medias de sus atributos de la siguiente forma:

$$\bar{x}_j = \bar{x}_j + \frac{1}{nd}\Delta\bar{x}_j \quad \forall j = 1 \ldots m \tag{3.16}$$

De igual forma, la clase c_i asociada al dato $\vec{x}$ modifica el conjunto de clases que representa el agente e influye en la clase final que este obtendrá cuando se utilice para clasificar un dato. El agente va a manejar información de las clases que han intervenido hasta el momento en su entrenamiento, manteniendo el número de datos escogidos por cada clase posible (nd_{c_i}), de forma que, la clase que más haya intervenido (clase $c*$) se va a definir como la clase asociada a su prototipo. La descripción completa del proceso de actualizar un agente con el dato e se recoge en el algoritmo 7.

Algoritmo 7 Actualizar agente: modelo MBLF

Entrada: $e = (\vec{x}, c_i)$ dato con el que actualizar al agente A, $\vec{x}$ es el vector de atributos del dato y c_i la clase asociada a $\vec{x}$.

1: **si** $c_i \in Cl_A$ (el agente ha escogido anteriormente un dato con clase c_i) **entonces**
2: $\quad n_{c_i} = n_{c_i} + 1$
3: **si no**
4: $\quad n_{c_i} = 1$
5: **fin si**
6: $c_A = c^*$ `tal que` $n_{c^*} = \text{máx}_{c_i \in Cl_A}\{n_{c_i}\}$
7: $n_{c_A} = n_{c^*}$
8: $nd = nd + 1$
9: $\bar{x}_i = \bar{x}_i + \frac{1}{nd}\Delta\bar{x}_i \quad \forall j = 1 \ldots m$

Para finalizar la definición del agente, se establecen dos funciones que le permiten interactuar con el exterior. Por un lado, cuando el sistema multiclasificador va a clasificar un dato debe decidir si un agente interviene

en la clasificación. Dado un dato, la función de probabilidad Fp_A, definida en la Ecuación 3.14, sirve para decidir si el dato pertenece a la zona del agente y, por tanto, sirve también para decidir si este va a intervenir en la clasificación del dato.

Por otro lado, el agente debe ser capaz de predecir la clase de un dato nuevo. Si $\vec{x}$ es el dato a clasificar, podemos definir una función clasificador que, dado un dato como entrada, obtiene la clase asociada,

$$C_A(\vec{x}) = c_A \tag{3.17}$$

donde c_A es la clase mayoritaria del agente.

3.4.4. Sistema multiclasificador

Se ha definido a un agente como un clasificador entrenado en un subdomino, que es capaz de predecir la clase de un dato cuando es seleccionado como experto. Esto significa que su uso y su efectividad va ligada a una función de selección del agente, de forma que este debe predecir correctamente solo cuando es seleccionado por el sistema multiclasificador.

El sistema multiclasificador requiere la especificación de tres elementos: la generación del conjunto del clasificadores base; una función que seleccione el subconjunto de clasificadores base que van a intervenir en la clasificación y, por último, una función que combine los resultados de los clasificadores seleccionados para predecir la clase final.

Si llamamos H al conjunto de los n agentes generados ($H = \{A_1, ..., A_n\}$); g es la función que selecciona a los k mejores expertos para un dato; y $H^* = \{A_1, ..., A_k\}$ es el subconjunto de agentes seleccionados para clasificar, definimos la función g como:

$$g(\vec{x}, k) = H^* \tag{3.18}$$

La clasificación de un dato por el sistema multiclasificador se realizará seleccionando los k mejores agentes para ese dato, y combinando las predicciones de los k agentes para ofrecer una clase final. Sea $Comb$ la función que combina los resultados de las clasificaciones de los k agentes. El uso del sistema multiclasificador para clasificar un dato $\vec{x}$ se puede definir de la forma siguiente:

$$Smc(\vec{x}, k) = Comb(C_{A_1}(\vec{x}), ..., C_{A_k}(\vec{x})) \tag{3.19}$$

donde C_{A_i} es el clasificador asociado al agente i que va a clasificar al dato $\vec{x}$ y k es el número de agentes que se quiere que intervengan en la clasificación. La función $Comb$ obtiene una clase final c_f como resultado de combinar las clasificaciones de k agentes,

$$c_f = Comb(C_{A_1}(\vec{x}), ..., C_{A_k}(\vec{x})) \tag{3.20}$$

3.4.4.1. Generación del conjunto de agentes

El sistema multiclasificador debe cumplir el objetivo de todo clasificador, que es predecir correctamente la clase de cualquier dato del dominio. Dado que nuestro sistema está compuesto por múltiples clasificadores especialistas, debemos diseñar el conjunto de agentes de forma que se cumpla que hay al menos un agente especialista en la zona de cualquier dato del dominio, es decir, debemos asegurar que el conjunto de agentes cubre el dominio total de los datos.

Si el conjunto de clasificadores generados cumple esta propiedad y la función de selección elige correctamente al agente o agentes adecuados para un dato, podemos garantizar que el sistema multiclasificador va a poder clasificar cualquier dato.

La cuestión es, por tanto, determinar cuantos agentes necesitamos para cubrir el total del dominio y como determinamos la zona de especialización de cada uno de ellos para garantizar que el sistema cumple nuestras restricciones.

Número de agentes necesarios y zona de especialización

El número de agentes mínimo necesario para garantizar la cobertura de todo el dominio depende, en gran parte, de como se inicialicen los agentes. En general, si tenemos información previa de las subzonas en las que podemos dividir el dominio, podemos generar agentes inicializados con información de cada zona.

Nuestra propuesta parte de la idea de que las clases de los datos que representan al dominio no tienen porque representar a una única subzona del dominio, es decir, es posible que datos con la misma clase pertenezcan a zonas distintas. De igual forma, una misma zona del dominio puede contener a datos que, aun siendo parecidos, representen a diferentes clases. En este escenario, debemos intentar garantizar que todas las clases y que todas las zonas estén cubiertas, cumpliéndose en general que un agente no tiene porque cubrir a todas las zonas de una clase y que un agente puede representar a una zona con diferentes clases.

Para acotar el número de agentes adecuado, se puede definir como *cota*

inferior el número de clases que intervienen en el problema, ya que al menos se necesita un agente por cada clase para poder predecir cualquiera de ellas. La *cota superior* es el número de ejemplos de entrenamiento. En este caso, cada agente representaría a un ejemplo y el algoritmo no tendría más datos para entrenar a los agentes, por lo que, si ejecutásemos nuestro algoritmo con estos agentes, estaríamos representando al algoritmo del vecino más próximo, donde la selección del mejor agente se convierte en la selección del dato (agente) más cercano.

El cálculo del número de agentes necesarios, sin utilizar información previa de zonas y clases, debe asegurar un número suficiente de agentes, de forma que podamos garantizar que todas las subzonas y clases van a estar cubiertas por al menos un agente. Se puede pensar en una inicialización de s agentes para cada clase, con lo que aseguramos que todas las clases están representadas. Es decir, si tenemos z clases posibles en el dominio, el número de agentes del sistema será $z * s$. Por tanto, el número de agentes lo podemos calcular como:

$$nA = z * s \tag{3.21}$$

La elección de s determina el número de agentes, ya que el número de clases z es constante. Otra forma de obtener nA es determinando que porcentaje de agentes queremos respecto al número de datos de entrenamiento. Si queremos un porcentaje de agentes p, el número de agentes se calcula de la forma:

$$nA = ne * \frac{p}{100} \tag{3.22}$$

donde ne es el número de datos de entrenamiento.

Tomando un número suficientemente grande de agentes y dado que los datos de entrenamiento tienen asignada una clase, la inicialización de la zona de especialización de cada agente se puede hacer de forma aleatoria, escogiendo un dato de entrenamiento como prototipo inicial del agente.

Desde el punto de vista de la reducción del conjunto de entrenamiento (Sección 3.3), un número elevado de agentes (por ejemplo un 25 % del total de datos de entrenamiento) puede servir para realizar una compresión de datos, donde cada valor prototípico representa a un subconjunto de los datos iniciales que son muy parecidos entre si. El conjunto de agentes obtenido puede ser entonces utilizado como conjunto de entrenamiento para cualquier algoritmo de aprendizaje, donde cada valor prototípico de cada agente es un dato de entrenamiento.

Conjuntos de entrenamiento para los agentes

La ejecución del algoritmo 6 mantiene el funcionamiento del algoritmo

LF, donde los agentes se mueven en el entorno transportando datos, con el objetivo de formar agrupaciones en la arena. El agente solamente va a coger un dato para transportarlo si este no está bien agrupado y, una vez cogido, solo le va a servir de entrenamiento si pertenece a su zona. Esta interacción del agente con el entorno va a determinar cuales son las características de los datos de entrenamiento disponibles para el conjunto de agentes.

El cambio del entorno provoca que datos bien agrupados tiendan a no ser cogidos nuevamente para transportarlos y, por tanto, tiendan a no volver a ser datos de entrenamiento para otros agentes. Esto significa que los datos de entrenamiento que se van a encontrar con mayor probabilidad los agentes, son aquellos datos que no están bien agrupados en la arena. Si pensamos que estos datos pueden corresponder a datos de zonas mixtas o datos con ruido, estamos garantizando que estos datos van a ser usados con mayor frecuencia en el entrenamiento de los agentes. Por tanto *los datos mal agrupados tienen mayor peso en el conjunto de entrenamiento de los agentes*.

Otra consecuencia de la evolución del sistema es que un dato de entrenamiento puede servir de entrenamiento a más de un agente. Esta característica es importante, en el sentido de que estamos suponiendo que vamos a poder tener varios agentes especialistas en subdominios parecidos, por lo que tiene sentido que un dato de entrenamiento no se utilice exclusivamente por un solo agente.

Parada del algoritmo

El algoritmo LF está definido sobre un número determinado de iteraciones, donde en cada iteración cada agente realiza una acción. La forma básica de implementar el algoritmo es seguir este modelo, en el que el entrenamiento termina al finalizar un número determinado de iteraciones.

Un criterio posible para decidir si el entrenamiento debe terminar es evaluar el sistema sobre un conjunto de test, en una iteración concreta. En realidad, el sistema multiclasificador podría ser usado para clasificar un dato desde la primera iteración ya que la zona de especialización está definida por el prototipo, y aunque este estuviese formado con pocos datos de entrenamiento, su funcionalidad está determinada desde la inicialización del agente.

Un criterio puede ser evaluar el sistema en una iteración i y seguir entrenando a los agentes en intervalos de j iteraciones, de forma que cada j iteraciones se vuelva a evaluar al conjunto de agentes sobre el conjunto de test. Mientras que la serie de evaluaciones sea creciente, podemos afirmar que seguir con el entrenamiento del sistema va a incrementar su efectividad. En el momento en que la serie se mantenga o decrezca estaríamos en el punto en el que hay sobreentrenamiento, es decir, el conjunto de agentes puede no mejorar si se sigue entrenando o incluso empeorar. En este caso, tendríamos la condición para finalizar el entrenamiento de los agentes.

Cabe señalar que, al igual que en una determinada iteración pasamos un test a nuestro sistema, podríamos aplicar al sistema una evaluación de los agentes, con el objetivo de reorientar el entrenamiento. Por ejemplo, podríamos detectar duplicidades de agentes, en el sentido de tener agentes muy correlacionados, que podrían ser unificados formando un único agente. También se podrían detectar subzonas donde no hubiese especialistas, generando nuevos agentes especialistas en esas subzonas.

3.4.4.2. Selección y combinación de agentes

Una vez que hemos generado el conjunto H con n agentes, necesitamos definir un mecanismo de selección de los k mejores agentes (función g de la Ecuación 3.18) y una función de combinación de las predicciones de esos k agentes (función $Comb$ de la Ecuación 3.19).

Selección de los k mejores agentes para un dato

Para definir la selección de los k mejores agentes, debemos calcular cual es el valor adecuado para k, es decir, cuantos expertos van a intervenir en la clasificación del dato. Al igual que ocurría con la elección del número de agentes del sistema, el número de agentes expertos para el dato dependerá de las estimaciones que hagamos sobre el número de subzonas que se han generado y de cuantos agentes consideremos que puede haber para cada subzona.

Si el número de agentes lo hemos calculado multiplicando el número de clases z por el número estimado de agentes por cada clase s, una cota superior de k es el valor s ya que es previsible que tengamos, en general, s agentes por clase, aunque no siempre los s agentes estén en la misma zona,

$$k = s$$

Si el cálculo del número de agentes ha sido usando un porcentaje del número de datos de entrenamiento, podemos calcular s dividiendo el número de agentes entre el número de clases.

$$k = \frac{nA}{z}$$

Por otro lado, debemos determinar qué función vamos a aplicar para saber qué agentes son expertos en el dominio del dato a clasificar. En general, debemos elegir a aquellos agentes cuya probabilidad de que el dato pertenezca a su zona sea mayor. Dicha probabilidad ($Fp_A(x)$) fue definida en el entrenamiento de un agente (Sección 3.4.3) para saber si debíamos escoger el dato para el entrenamiento. La misma función va a servir para decidir si

escoger un agente A_i para clasificar un dato $\vec{x}$, donde la probabilidad de pertenencia de un dato al agente es máxima (1) ($Fp_A(x) = 1$) si su distancia del prototipo en mínima (0).

Por tanto, los k mejores agentes serán aquellos cuya función probabilidad para el dato a clasificar sea mayor.

Combinación de los k mejores clasificadores

En la definición del agente como clasificador, hemos simplificado la obtención de la clase final en la predicción como la clase que mayoritariamente ha intervenido en su entrenamiento, por tanto, cada agente ofrece su clase mayoritaria como clase final ($C_{A_i}(\vec{x}) = c_{A_i}$). La forma más sencilla para que el sistema multiclasificador combine las clases de los k clasificadores es aplicar votación simple sobre el conjunto de las k predicciones. Si llamamos V_{c_i} al número de veces que la clase c_i ha sido la predicción entre los k clasificadores, la función $Comb$, que combina las predicciones, va a elegir a la clase con más votos entre las z clases posibles, es decir a la clase cuyo valor V_{c_i} es mayor. Podemos definir la función de combinación de la siguiente forma:

$$Comb(C_{A_1}(\vec{x}), ..., C_{A_k}(\vec{x})) = max_{i=1}^{z}\{V_{c_i}\} \tag{3.23}$$

donde

$$V_{c_i} = \sum_{j=1}^{k} voto_{ij} \tag{3.24}$$

siendo $voto_{ij}$ igual a uno cuando la clase obtenida por el agente j es la clase c_i, y cero cuando es otra clase,

$$voto_{ij} = \begin{cases} 1 & \texttt{si } C_{A_j}(x) = c_i \\ 0 & \texttt{si } C_{A_j}(x) \neq c_i \end{cases} \tag{3.25}$$

En el esquema de selección y de combinación que se ha definido, los votos de los k agentes tienen el mismo peso. Esto tiene la ventaja de ser la forma más sencilla de combinar los resultados pero tiene el inconveniente de que, si k es mayor que el número de agentes que realmente son especialistas para el dato, los votos de los agentes menos especialistas pueden distorsionar el resultado final. En general, se puede afirmar que la condición para elegir k se puede relajar si la aportación de cada agente es directamente proporcional al grado de pertenencia del dato al agente.

3.4.5. Experimentos

El método propuesto se ha implementado utilizando el entorno de simulación SWARM, que ha sido desarrollado por The Swarm Development Group [32]. En esta versión del algoritmo, todos los agentes usan el mismo algoritmo de aprendizaje; utilizan la media de los datos vistos hasta el momento como valor prototípico y, como mecanismo de combinación, escogen la clase mayoritaria de los k agentes más cercanos.

Dada la similitud del algoritmo de los k-vecinos más cercanos con la función de selección de los k mejores agentes, se han escogido dos conjuntos de entrenamiento de UCI Machine Learning Repository (Iris y Pendigits) [8] para comparar ambos algoritmos.

La base de datos Iris consta de 150 ejemplos, donde cada uno contiene 4 atributos reales y 3 clases posibles. De esta base de datos se han escogido aleatoriamente 44 ejemplos, que forman el conjunto de test, y los 106 restantes ejemplos formarán el conjunto de entrenamiento.

La segunda base de datos es Pen-Based Recognition of Handwritten Digits. El número posible de clases es 10 (una por cada dígito) y el número de atributos es 16. En este caso, los autores ofrecen un conjunto de entrenamiento con 7.494 ejemplos y un conjunto de test con 3498 ejemplos.

En la Tabla 3.1 se muestran los resultados de aplicar el algoritmo de los k-vecinos más cercanos [4, 45, 46, 89, 95] a las bases de datos Iris y Pendigits, donde cada columna representa el número de vecinos utilizado para obtener la clase final, y cada casilla el porcentaje de aciertos sobre el conjunto de test para cada base de datos. La métrica que se ha usado para seleccionar a los k vecinos más cercanos es la distancia Euclídea; la misma que se ha utilizado en la implementación de este experimento.

	k=1	k=2	k=3
Iris	95.45 %	97.72 %	90.90 %
Pendigits	97.74 %	97.39 %	97.79 %

Tabla 3.1: Resultado de aplicar K-vecinos a las bases de datos Iris y Pendigits

Las Tablas 3.2 y 3.3 muestran los resultados obtenidos al aplicar nuestro algoritmo a la base de datos Iris y Pendigits. Las filas de las tablas reflejan el porcentaje de acierto al aplicar el conjunto de test cada IT iteraciones, entendiendo por iteración un movimiento del conjunto de agentes. Las columnas de estas tablas reflejan el número de clasificadores seleccionados más cercanos al dato a clasificar.

Se han realizado distintos experimentos variando el número de agentes. Los mejores resultados se han obtenido cuando el número de agentes es alto. Esto es debido a que la formación de cada clasificador es muy simple (media de los ejemplos vistos hasta el momento), con lo que en la práctica, estamos

	k=1	k=2	k=3
IT=2000	88.63 %	95.45 %	95.45 %
IT=4000	93.18 %	88.63 %	95.45 %
IT=6000	84.09 %	79.54 %	86.36 %
IT=8000	95.45 %	95.45 %	93.18 %
IT=10000	90.90 %	95.45 %	95.45 %
IT=12000	88.63 %	97.72 %	95.45 %
IT=14000	97.72 %	93.18 %	95.45 %
IT=16000	100.00 %	97.72 %	97.72 %
IT=18000	100.00 %	97.72 %	97.72 %

Tabla 3.2: Resultado de aplicar el modelo MBLF con 30 agentes a Iris

	k=1	k=2	k=3
IT=10000	96.19 %	95.39 %	96.19 %
IT=20000	96.65 %	95.36 %	96.45 %
IT=30000	96.65 %	95.51 %	96.34 %
IT=40000	96.28 %	95.56 %	95.94 %
IT=50000	96.54 %	95.74 %	96.02 %
IT=60000	96.68 %	95.68 %	96.05 %
IT=70000	96.71 %	95.59 %	96.11 %
IT=80000	96.79 %	95.59 %	96.22 %
IT=90000	96.99 %	96.02 %	96.36 %
IT=100000	96.96 %	96.08 %	96.36 %

Tabla 3.3: Resultado de aplicar el modelo MBLF con 1150 agentes a Pendigits

realizando una reducción del conjunto inicial de entrenamiento.

En el experimento con la base de datos Iris (Tabla 3.2), se consigue un 100 % de acierto a partir de la la iteración 16.000, eligiendo el clasificador más cercano a cada dato, y un 97,72 % con los dos y tres clasificadores más cercanos (43 aciertos de 44 datos). Estos resultados ofrecen una pequeña mejora respecto al algoritmo de los k vecinos más cercanos. Respecto al experimento con la base de datos Pendigits, (Tabla 3.3), los resultados son levemente inferiores a los ofrecidos por el algoritmo de los k vecinos más cercanos.

Comparando los resultados de ambos algoritmos, podemos observar que los resultados son muy similares. Sin embargo, el tiempo de clasificación se va a reducir notablemente usando nuestro modelo, ya que la comparación del dato a clasificar se produce con los n agentes y no con todos los datos de entrenamiento, como ocurre en el algoritmo k-vecinos.

Podemos concluir que los agentes obtenidos tienen una representación muy limitada respecto al concepto de clasificador, debido a que no hay suficiente información en el agente de la zona que representa y las clases

que la constituyen. Se ha demostrado con los experimentos que el conjunto de agentes tiene un comportamiento aceptable cuando el número de agentes es grande. El sistema obtenido, al usar un número elevado de agentes, es equivalente a aplicar un algoritmo de reducción de datos, donde el conjunto de agentes representa al conjunto de datos reducido.

3.5. Modelo de multiclasificador con memoria local (McML)

Las limitaciones encontradas en el modelo MBLF nos han conducido a su modificación, con el objetivo de obtener un nuevo modelo más robusto. La simplicidad buscada en la estructura y funcionamiento del agente del modelo MBLF deriva en que el sistema multiclasificador definido no explote toda la información disponible. Por un lado, el mantenimiento de un único prototipo y una única clase (la mayoritaria) en el agente hace que este no utilice información de todas las clases que han intervenido en el entrenamiento, con lo que agentes que representan a zonas heterogéneas respecto a sus clases no pueden explotar esta información.

Por otro lado, que el peso del voto de un agente sea el mismo para todos los agentes que intervienen en la clasificación hace que se distorsione el resultado de la combinación, pues agentes menos expertos deberían influir menos en la decisión final.

Nuestro objetivo es mejorar el modelo MBLF con una nueva aproximación, dotando al agente de más información respecto a su zona de especialización y definiendo un criterio de calidad del agente que sirva para ser aplicado en su entrenamiento [13]. También se van a modificar los criterios para la elección de los datos de entrenamiento, así como la definición de la selección de los mejores expertos y la combinación de sus predicciones, para conseguir explotar todas las características de los agentes. Llamaremos al nuevo modelo que vamos a definir en esta sección modelo basado en el algoritmo LF con memoria local (modelo McML en adelante).

En las siguientes secciones estudiaremos, como hicimos en el modelo MBLF, cada uno de los componentes del nuevo modelo, discutiendo las mejoras aportadas respecto al modelo inicial.

3.5.1. Estructura del agente

La formación, evolución y utilización de un agente como miembro del sistema multiclasificador vendrá determinada por la estructura que lo defina. Esta estructura está, a su vez, ligada al objetivo del agente de lograr especializarse en un subdominio y de poder ofrecer la información necesaria

para clasificar un dato. En este sentido, el agente debería tener las siguientes características:

- Una representación simple y compacta de la zona de datos en la que es especialista ya que el agente debe indicar claramente la adecuación de su clasificación a la zona de especialización.
- Un método eficiente y efectivo para clasificar un dato.

Si $H = \{A_1, ..., A_{n_A}\}$ es el conjunto de agentes, la estructura de cada agente A_i en un instante dado viene definida por la tupla $A_i = ((x_i, y_i), M, P_A, P_c)$ donde :

- $(x_i, y_i) \in N^2$ es la localización del agente en la arena.
- $M = \{e_1, \ldots, e_n\}$ donde $e_i = (\vec{x_i}, c)$ es la memoria del agente, compuesta por n datos de entrenamiento.
- $P_A \in R^m$ es el valor prototípico del agente, donde m es el número de atributos que definen el prototipo.
- $P_c = \{P_{c_1}, \ldots, P_{c_z}\}$ es el conjunto con los prototipos de las z clases posibles en la memoria M, donde $P_{ci} = \{\bar{x}_k | (\vec{x}_k, ci) \in M\}$ es el prototipo de los datos en M que tienen clase c_i.

Los componentes de este nuevo agente determinan su zona de especialización y permiten mantener información sobre las clases que intervienen en el agente. El agente en este modelo no necesita contabilizar el número de datos que han sido vistos de cada clase ya que la información de las clases que intervienen en cada momento está implícita en su memoria, por lo que no es necesario guardar esta información. Tampoco se necesita guardar un valor para la clase asociada al agente ya que esta clase podrá ser calculada como la mayoritaria dentro de la memoria, de forma análoga a como se hacía en el modelo MBLF o bien utilizando un examen de las clases que componen la memoria.

3.5.2. Definición de la zona de especialización del agente

El objetivo de que un agente sea un clasificador especialista en una subzona del domino general del problema hace que la representación de la zona del agente y su manipulación sean elementos fundamentales a la hora de establecer las características del mismo. A continuación se definen una serie de conceptos que nos van a permitir centrar las ideas que vamos a tratar en el desarrollo del agente y los elementos que constituyen su estructura.

Dominio del problema (X): sea $\vec{x} = (x_1, \ldots, x_m)$ un dato representado por un vector de m atributos. El dominio de los datos X queda establecido por cada subdominio X_i de los m atributos que componen el dato ($X_1 \times \ldots \times X_m$), donde $x_i \in X_i$.

.

Dato de entrenamiento (e) un dato de entrenamiento $e = (\vec{x}, c)$ es un par formado por un dato $\vec{x} \in X$ y una clase $c \in Cl = \{c_1, \ldots, c_z\}$ donde z es el número de clases posibles para los datos.

Zona de especialización del agente (Z_A): es la zona del dominio Z_A donde el agente A es especialista será un subconjunto del dominio X ($Z_A \subseteq X$), donde $Z_A = Z_{A,1} \times \ldots \times Z_{A,m} \subseteq X$ y $Z_{A,j}$ representa a un subconjunto del dominio del atributo j ($Z_{A,j} \subseteq X_j$).

Distancia entre dos datos ($d(x, y)$): sea $x, y \in R^m$, definimos la distancia entre x e y como una métrica que se aplica a los m atributos de x e y.

Memoria de un agente (M). La memoria de un agente va a representar a la zona de especialización del agente. Esta memoria está formada por los n datos de entrenamiento que mejor definen a su zona de especialización. La definición de "mejor conjunto de datos para el agente" va ligada a la definición de una función de valoración de la calidad del agente que evalúe la calidad de la zona que representa. Esta función de valoración, definida en la Sección 3.5.3, va a permitir al agente tener un criterio de decisión sobre si un dato de entrenamiento va a mejorar o no su zona, por tanto, sirve como criterio de optimización en su entrenamiento.

Si la memoria M del agente representa a la zona del agente Z_A, se cumple que todo elemento de la memoria pertenece a la zona del agente:

$$\forall e = (\vec{x}, c) \in M \text{ se cumple que } \vec{x} \in Z_A$$

Prototipo del agente (P_A). El entrenamiento y uso de un agente requiere del manejo continuo de la información sobre su zona de especialización. Vamos a definir un valor que represente la zona de especialización del agente. Dado un conjunto de datos M, definimos el punto prototípico P como el valor que mejor representa al conjunto de los n datos de M. Una característica deseable que vamos a considerar al establecer el prototipo P es que sea el punto que minimice la media de las distancias entre él y cualquier elemento del conjunto M.

Un valor apropiado para representar al punto prototípico es la media de todos los datos en M, ya que se cumple que: "la suma de los cuadrados de

las desviaciones de un conjunto M de números respecto de un cierto número a es mínima si y solo si se cumple que a es la media de todos los números en M", cumpliéndose también que la suma de las desviaciones típicas de un conjunto de números respecto a su media es cero.

Llamaremos P_A al valor medio, en adelante prototipo, de los n datos del conjunto M para el agente, calculado de la siguiente forma:

$$P_A = \{\bar{x}_1, \ldots, \bar{x}_m\} \tag{3.26}$$

donde $\bar{x}_j$ representa a la media de los n datos de M en el atributo j

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

Las características del prototipo P_A son:

- Define el centro de la zona de especialización del agente y la resume en un único valor, cumpliéndose que el prototipo está dentro de la zona del agente ($P_A \in Z_A$).
- El prototipo P_A se define con la misma estructura que un dato, es decir, un vector con m componentes (atributos), donde cada atributo del prototipo corresponde a la media de ese atributo para todos los datos de M.

Prototipos de Clase (P_c). El prototipo P_A permite resumir la información de la zona del agente, sin embargo, en este prototipo no tenemos información sobre las clases que intervienen en el agente. Si tenemos en cuenta que la memoria del agente puede contener datos de diferentes clases, podemos formar, de forma natural, tantas subagrupaciones como clases intervengan en la memoria. Una agrupación de los datos de M por cada clase permite establecer subzonas del agente, donde cada subzona está compuesta solamente por datos de la misma clase. Si resumimos los datos en M con la misma clase en un prototipo, obtenemos lo que llamamos un prototipo de clase.

Sea $M = \{(\vec{x_1}, c_1), \ldots, (\vec{x_n}, c_n)\}$ un conjunto de n datos de entrenamiento para un agente y sea P_A el prototipo asociado a M. Si z es el número de clases posibles, podemos formar z subconjuntos de M de forma que todos los datos que tienen la misma clase estén en el mismo subconjunto. Esto origina una partición de la memoria del agente, $M = \{M_1, \ldots, M_z\}$, tal que $M_i = \{(\vec{x_1}, c_i), \ldots, (\vec{x_k}, c_i)\}$.

Cada subconjunto M_i representa nuevamente a una subzona de Z_A, con

la característica de que contiene a una única clase c_i. Si $M_i = \varnothing$ la clase c_i no está representada en el agente.

Representando cada agrupación con su valor prototipo P_{c_i}, calculado como la media para cada atributo de los datos de clase i en M, podemos definir el conjunto de prototipos de clase P_c para el agente:

$$P_c = \{P_{c_1}, \ldots, P_{c_z}\}$$

donde

$$P_{c_i} = \begin{cases} \bar{\vec{x}}_k \ \ |(\vec{x}_k, c_i) \in M & \text{si } M_i \neq \varnothing \\ 0 & \text{si } M_i = \varnothing \end{cases}$$

El prototipo general P_A y los prototipos de clase P_{c_i} permiten representar en varios niveles la zona de especialización del agente (Figura 3.2). En un primer nivel, el prototipo general del agente resume las características generales de una zona sin tener en cuenta las clases que se han encontrado en ella, es decir, el prototipo representa a la agrupación donde se está especializando el agente. En un segundo nivel, podemos formar diferentes agrupaciones, una por cada una de las clases que intervienen en la definición de la zona del agente. Estas subagrupaciones, dentro de la agrupación general, están representadas por el conjunto P_c, donde cada agrupación está definida por un prototipo P_{c_i}.

En la Figura 3.2 se muestra la representación de la los datos de la memoria de un agente con tres clases. El prototipo del agente está representado por el punto central mientras que los prototipos de las clases representan a los centros de las zonas de cada clase.

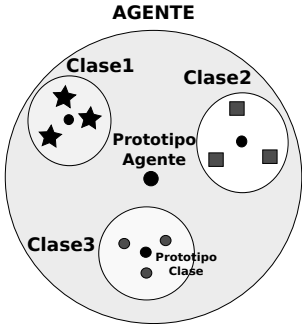

Figura 3.2: Zona de especialización y subzonas de especialización

3.5.3. Criterios para la evaluación de la calidad del agente

La zona de especialización de un agente determina el contexto en el que el agente va a ser usado por el sistema multiclasificador. La formación de la zona de especialización es la base del entrenamiento del agente, por lo que se necesitan medidas de la calidad de esta zona para guiar el entrenamiento. La evaluación del agente también es necesaria cuando el sistema multiclasificador decide si un agente interviene en la clasificación y de que forma lo hace.

A continuación se estudia la calidad de un agente respecto a su estructura, respeto a la especialización de sus clases y, por último, se define una medida de la calidad global como combinación de las anteriores.

Calidad del agente respecto a su estructura

Uno de los parámetros que miden la calidad de un agente es la homogeneidad de los datos que definen su zona de especialización Z_A, almacenados en la memoria M, de forma que la calidad del agente será mayor si su zona es homogénea. Una buena medida para evaluar esta homogeneidad de los datos es la varianza, que deberá ser calculada para cada uno de los m atributos de los datos de la memoria. Si llamamos $\vec{V}$ al vector de varianzas y V_j a la varianza para el atributo j tenemos:

$$\vec{V} = \{V_1, \ldots, V_m\}$$

donde

$$V_j = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

Donde $\bar{x}_j$ es la de media del atributo j para los n datos. Esta media es una constante en la fórmula y representa al atributo j del prototipo del agente P_A. Con el objetivo de obtener un cálculo iterativo de estos valores, podemos reescribir nuestra ecuación sacando el cuadrado de las medias ($\bar{x}_j^2$) fuera del sumatorio de la siguiente forma:

$$V_j = \left(\frac{1}{n}\sum_{i=1}^{n} x_{ij}^2\right) - \bar{x}_j^2$$

Si llamamos $\bar{x_j^2}$ a la media de los cuadrados de los datos para el atributo j:

$$\bar{x_j^2} = \frac{1}{n}\sum_{i=1}^{n} x_{ij}^2$$

la ecuación que define a la varianza en cada atributo queda de la siguiente forma:

$$V_j = \bar{x_j^2} - \bar{x}_j^2 \tag{3.27}$$

Por tanto, la varianza del agente queda definida por el vector de varianzas $\vec{V}$. Como nuestro objetivo es usar la varianza como medida de calidad de los datos, necesitamos una magnitud escalar de ella. Por este motivo, vamos a usar la media de los atributos de la varianza ($\bar{V}$) definida de la siguiente forma:

$$\bar{V} = \frac{1}{m}\sum_{j=1}^{m} V_j \tag{3.28}$$

Calidad del agente respecto a la especialización de sus clases

Otra medida de la calidad del agente es el desorden de las clases que intervienen en la memoria M, es decir, la cantidad de clases distintas que tienen los datos de su zona de especialización. Un agente cuyo desorden sea pequeño podrá discriminar mejor la clase correcta que un agente cuya distribución de clases sea heterogénea. Una métrica para medir el desorden de los datos respecto a sus clases es la entropía, que se define en siguiente ecuación.

$$E = -\sum_{i=1}^{z} \frac{nc_i}{n} log_z \frac{nc_i}{n} \tag{3.29}$$

donde z representa al número de clases que intervienen en los datos de entrenamiento y nc_i es el número de datos con clase i.

Calidad combinando ambos criterios

Para usar simultáneamente estas dos medidas de la calidad de un agente se define una función de valoración del agente, V_c, que llamamos valoración de compacidad, y que combina linealmente, de una manera ponderada, la varianza y la entropía. Para determinar el peso de cada uno de estos valores

en la valoración total, se va usar un parámetro $\beta \in [0, 1]$, que distribuye el peso de cada componente como indica la ecuación:

$$V_c = E\beta + \bar{V}(1 - \beta) \tag{3.30}$$

3.5.4. Creación de un agente

En este modelo se va a mantener la idea de que la zona de especialización de cada agente sea inicializada aleatoriamente. Como la zona viene definida por la memoria de n datos de entrenamiento, debemos decidir que mecanismo usamos para obtener esos n primeros datos. Una posibilidad es escoger el primer dato que se encuentre y, a partir de ese momento, seguir el entrenamiento con la fórmula general. En este caso, aplicaríamos el criterio de calidad del agente y, si el dato nuevo mejorase la calidad, lo añadiríamos a la memoria.

En este caso, tenemos la ventaja de que el agente, desde el comienzo del entrenamiento, cumple las condiciones de calidad pero el inconveniente de que las características de la zona van a estar muy determinadas por el primer dato en la memoria. Hay que tener en cuenta que si empezamos con un dato en la memoria, difícilmente un nuevo dato va a mejorar la calidad de la zona respecto a la varianza y entropía, con lo que tendríamos que relajar las condiciones iniciales de valoración.

En este sentido, hemos optado por lo que sería el caso extremo de relajación de estas condiciones iniciales de valoración, en el que la valoración de la mejora con el nuevo dato va a ser máxima si la memoria no está llena. Por lo que, en la práctica, la obtención de los n datos de la memoria se va a realizar llenando la memoria con los n primeros datos de entrenamiento que encuentre el agente. La inicialización de la memoria de un agente se describe en el Algoritmo 8.

Algoritmo 8 Inicialización de la memoria de un agente: modelo McML

Entrada: agente con la memoria vacía
Salida: agente con la memoria llena

```
1: Situar al agente aleatoriamente en la arena
2: mientras la memoria del agente no este llena hacer
3:    si hay un dato en la posición del agente entonces
4:       Meter el dato en la memoria
5:       Mover el agente a una nueva posición aleatoria
6:    fin si
7: fin mientras
```

Esta inicialización nos proporciona una zona inicial del agente sobre el

dominio general del problema pero con una valoración de la calidad de la zona baja, con lo que, cuando empiece el entrenamiento, los nuevos datos tienen una probabilidad alta de mejorar su calidad. Esta situación provoca que el agente cambie frecuentemente al comienzo del entrenamiento y se estabilice conforme se avanza en las iteraciones del algoritmo.

Como en el modelo MBLF, la filosofía que queremos implementar es la de agentes que se construyen sin información previa, para potenciar su formación autónoma. Claramente, una inicialización con información de preparticiones haría converger a los agentes a mejores soluciones pero estaríamos eliminando el componente de evolución y autoorganización que queremos imprimir en nuestros modelos.

Algoritmo 9 Entrenamiento de un agente: modelo McML

Entrada: agente
Salida: agente entrenado después de t_max iteraciones
1: **para** t=1 to t_{max} **hacer**
2: **si** Si lugar ocupado por dato e **entonces**
3: Obtener el dato k de la memoria que hace que la valoración total sea menor
4: Obtener la valoración sin k y con e ($V_c(k, e)$)
5: **si** $V_c(k, e) < V_c$ (Si la valoración sin k y con e es menor que la valoración inicial) **entonces**
6: Dejar el dato k en la arena
7: Coger el dato e y actualizar el agente
8: **fin si**
9: **fin si**
10: Mover aleatoriamente al agente a una nueva posición
11: **fin para**

3.5.5. Entrenamiento de un agente

La formación del agente en este modelo se puede resumir en la idea de un proceso de optimización, donde la función a optimizar es la calidad de la zona de especialización del agente V_c, es decir, la calidad de su memoria respecto a la varianza y la entropía. El entrenamiento va a consistir en, iterativamente, ir mejorando la calidad de la zona con la evaluación de la mejora que aporta un dato nuevo. En una iteración, un agente que encuentra un dato debe decidir si el dato mejora su calidad, en cuyo caso almacena el dato en su memoria y deja el dato que menor calidad aporta en la arena, con el fin de mantener los n datos de la memoria.

En contraposición con el modelo MBLF, no se va a mantener el objetivo de agrupar datos del algoritmo LF. Esto significa que el agente no va a transportar un dato para colocarlo en otro montón sino que el dato que encuentre lo va a guardar (o no) en su memoria para configurar su zona. El

agente no va a tener que calcular si el dato en la arena está bien agrupado por lo que, en este sentido, mejoramos la eficiencia del agente. Otra modificación respecto al modelo MBLF es que no se calcula la pertenencia del dato a la zona, como requisito para cogerlo. El criterio de la mejora de la calidad de la zona va a sustituir al criterio de pertenencia. En realidad, si un agente mejora la varianza de la memoria es porque pertenece a la zona del agente, con lo que una ponderación alta de la varianza en la valoración de la calidad asegura el mantenimiento de este criterio de pertenencia. Cabe notar que la calidad del agente mejora con un nuevo dato si la valoración de compacidad V_c disminuye, ya que se busca minimizar la varianza y la entropía. La descripción de este procedimiento de entrenamiento y optimización se detalla en el Algoritmo 9.

Hasta este momento, el procedimiento seguido para la actualización del agente y su comportamiento ha sido el de un enfoque voraz, en referencia a la optimización o mejora iterativa de sus características. Este tipo de esquemas tienen algunos problemas asociados, destacando la posibilidad de estancamiento en mínimos o máximos locales, o la dificultad de avanzar sin cambios en las características evaluadas.

Para disminuir estos problemas, Kirkpatrick, Gelatt y Vecchi [49] proponen un método probabilístico de optimización global denominado "Simulated annealing" (enfriamiento simulado), que, fundamentándose en el algoritmo de Metrópolis [74], establece los mecanismos necesarios para poder escapar de los mínimos y máximos locales. El algoritmo 10 muestra el comportamiento general de esta estrategia.

Algoritmo 10 Enfriamiento simulado

Entrada: Elemento Inicial S_0, tiempo para el algoritmo T_0, función de vecindad $V(s)$, función de evaluación $Val(S)$, función para decrementar T $Dec(T)$ y función $Iter(T)$ del número de iteraciones para T.
Salida: sol

1: $Sol = S_{act} = S_0, T = T_0$
2: **mientras** $T > 0$ **hacer**
3: $S_{cand} = V(S_{act})$
4: $\Delta V = Val(S_{cand}) - Val(S_{act})$
5: **para** $i = 1$ **hasta** $Iter(T)$ **hacer**
6: **si** ($\Delta V < 0$) o $Rand(0,1) < \exp^{\frac{-\Delta V}{T}}$ **entonces**
7: $S_{act} = S_{cand}$
8: **fin si**
9: **fin para**
10: $T = Dec(T)$
11: **si** $S_{act} > sol$ **entonces**
12: $sol = S_{act}$
13: **fin si**
14: **fin mientras**
15: **devolver** sol

En definitiva, el algoritmo que se propone introduce un cambio fundamental en el criterio de selección, dentro de un proceso general de mejora iterativa. Este cambio consiste en permitir la selección de opciones que no mejoran nuestra situación, atendiendo a dos criterios básicos: cuanto se empeora o deteriora la situación (ΔV) y cuanto tiempo llevamos en el proceso de mejora (T). En un ámbito donde el objetivo es minimizar el valor de un criterio V, se considera un empeoramiento del mismo cuando el incremento ΔV, al considerar otro punto, es mayor que 0, donde ΔV es la diferencia entre la valoración con el dato nuevo y la valoración actual.

Estos dos criterios se combinan en una distribución de probabilidad de ser seleccionado, como se indica en la Ecuación 3.31, en la que valores grandes para T y pequeños valores de ΔV obtienen probabilidades altas de selección.

$$e^{\frac{-\Delta V}{T}} \tag{3.31}$$

En estos términos, considerando T relacionado con el "tiempo que queda para finalizar", podemos decir que nuestro método de selección es poco riguroso cuando comienza el entrenamiento (T alto) y es estricto (T cercano a 0) cuando va a finalizar.

Vamos a aplicar este criterio en el entrenamiento del agente. Si V_c es la valoración de la calidad del agente con la memoria actual, calculamos una nueva valoración de la calidad como resultado de añadir el dato e a la memoria y eliminar el dato k que hace que la calidad de la memoria sea peor. Si llamamos a esta nueva valoración $V_c(k, e)$, podemos definir el incremento en la valoración de calidad al añadir e y soltar el dato k, respecto a la valoración inicial de la siguiente forma:

$$\Delta V_c = V_c(k, e) - V_c$$

V_c es el valor asociado al criterio de mejora, por lo que la probabilidad de que el dato sea seleccionado se resume en la fórmula:

$$e^{\frac{-\Delta V_c}{T}} \tag{3.32}$$

En cada iteración, el valor de T debe ir disminuyendo. Para lograr esta disminución se va a multiplicar T por un parámetro $\alpha \in [0, 1)$. Un valor de α cercano a 1 hará que T decrezca despacio mientras que un valor cercano a 0 produce que T decrezca rápidamente en pocas iteraciones.

La generación de un valor aleatorio u, uniformemente distribuido, va a permitir decidir si finalmente el dato e es seleccionado para el agente. El entrenamiento de un agente usando enfriamiento se detalla en el algoritmo 11.

Actualización del agente con un dato de la arena

El agente va a modificarse con el dato e de la arena si este mejora su calidad o se cumplen las condiciones referentes al enfriamiento. Como la memoria del agente contiene a un número n constante de datos, el dato k, que es el que produce la peor calidad del agente, va ser devuelto al entorno, depositándolo en la posición que tiene el dato e en la arena.

Algoritmo 11 Entrenamiento de un agente con enfriamiento

Entrada: agente
Salida: agente entrenado

para t=1 to t_{max} **hacer**
  si lugar ocupado por dato e **entonces**
    Obtener el dato k de la memoria que hace que la valoración total sea menor
    Obtener la valoración sin k y con e ($V_c(k, e)$)
    Obtener ΔV_c como $\Delta V_c = V_c(k, e) - V_c$
    Generar un número aleatorio u uniforme ($u \in [0, 1]$)
    si $\Delta V_c < 0$ or $u < e^{\frac{-\Delta V_c}{T}}$ **entonces**
      Soltar el dato k en la arena
      Coger el dato e
      Actualizar el agente con e
    fin si
  fin si
  $T = \alpha * T$
  Mover aleatoriamente al agente a una nueva posición
fin para

El nuevo dato e produce un cambio en la estructura y en los valores de los elementos del agente, que deben ser actualizados teniendo en cuenta la eliminación del dato k. Al igual que en el modelo MBLF, debemos volver a calcular el prototipo general del agente P_A. Este prototipo se va a construir con los datos de la nueva memoria (sin k y con e). Además, debemos actualizar los prototipos de clase con e. Si la clase c_i asociada a e no está en la memoria, el agente debe crear un nuevo prototipo que represente a dicha clase. Como solamente tendríamos un dato con clase c_i, el prototipo P_{c_i} será el vector de atributos $\vec{x}$ del dato e.

En el caso de que la clase c_i ya aparezca en la memoria, el prototipo de clase P_{c_i} debe ser actualizado añadiendo el vector $\vec{x}$ a su composición. Con estas actualizaciones estamos reconfigurando el prototipo de la zona general del agente P_A, y los prototipos de las subzonas (P_{c_i}).

Al añadir e y soltar x_k en la arena, también varía la varianza y entropía de los datos de la memoria, por lo que estos dos valores tienen que volver a

ser calculados. El método descrito para actualizar el agente se resume en el Algoritmo 12.

Algoritmo 12 Actualizar agente: modelo McML

Entrada: $e = (\vec{x}, c_i)$ dato con el que actualizar al agente A, x_k: elemento de la memoria a soltar en la arena

1: Incluir el dato e en la memoria
2: **si** $c_i \in Cl_A$ (el agente tiene al menos un dato en la memoria con clase c_i) **entonces**
3: Actualizar prototipo de clase c_i (P_{c_i}) con e.
4: **si no**
5: Añadir un nuevo prototipo para la clase c_i (P_{c_i}) con e, donde $P_{c_i} = \vec{x}$
6: **fin si**
7: Actualizar prototipo del agente (P_A) con e y sin x_k
8: Recalcular la varianza y entropía con e y sin x_k

3.5.6. Uso de un agente

El sistema multiclasificador va a utilizar a un agente como especialista para decidir la clase de un dato nuevo. El agente debe ser capaz de decidir si un dato pertenece a su zona de especialización y en que grado se produce esta pertenencia. Por otro lado, el agente debe predecir la clase de un dato cuando el sistema multiclasificador considere que el agente es especialista en el dato. A continuación veremos como se define la pertenencia de un dato al agente y como predice el agente la clase de un dato cuando el sistema multiclasificador solicita su clasificación.

Pertenencia de un dato al agente usando la distancia máxima al prototipo

Dado que la zona de un agente está determinada por los datos de su memoria, podemos definir la pertenencia de un dato al agente buscando una cota superior de la zona del dominio que representan los datos de la memoria para cada uno de sus atributos.

Distancia máxima en un atributo (ε_j). Dado el prototipo $P_A = \{\bar{x}_1, \ldots, \bar{x}_m\}$, se define ε_j como la distancia máxima en el atributo j entre todos los datos $\vec{x}_i$ en M y la media del atributo j como sigue:

$$\varepsilon_j = max_{i \in [1,n]} \{d(\bar{x}_j, x_{ij})\}$$

donde x_{ij} representa al atributo j del dato $\vec{x}_i$ y $d(\bar{x}_j, x_{ij})$ es una medida de

distancia. Esta distancia máxima al prototipo define una cota superior para la zona en el atributo j ($Z_{A,j}$).

Distancia máxima (ε). Si calculamos ε_j para todos los atributos ($j = 1, \ldots, m$) obtenemos un vector $\vec{\varepsilon}$ de distancias máximas al centro:

$$\vec{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_m)$$

Si consideramos las distancias máximas al prototipo para todos los atributos como una representación de la zona, podemos definir la pertenencia de un dato a la zona de la siguiente forma:

Pertenencia de un dato x a la zona del agente. Un valor x_j del dominio general X_j pertenece al dominio $Z_{A,j}$ si su distancia al prototipo es menor que el máximo de las distancias en el atributo j. Se puede extender esta definición para saber si un dato $\vec{x}$ del dominio general X pertenece a la zona Z_A. Una posible extensión será la que exija que todos los atributos de $\vec{x}$ estén dentro sus zonas respectivas de forma que se cumpla:

$$\forall \vec{x} \in X, \vec{x} \in Z_A \text{ sii } d(\bar{x}_j, x_j) \leq \varepsilon_j, \forall j = 1, \ldots, m$$

Esta formulación expresa la pertenencia de un dato $\vec{x}$ a la zona del agente Z_A en términos discretos, de forma que un dato pertenece a la zona del agente solamente si todos sus atributos pertenecen a las zonas de los atributos del agente ($Z_{A,j}$). En este sentido, se puede definir una función de pertenencia en la que, dado un dato $\vec{x}$ y la zona de un agente Z_A, nos diga si el dato pertenece o no pertenece a la zona del agente

$$f_{Z_A}(\vec{x}) = \begin{cases} 1, & \text{si } d(\bar{x}_i, x_i) \leq \varepsilon_j, \forall i = 1, \ldots, m \\ 0, & \text{en otro caso} \end{cases}$$

Si resumimos el vector de distancias $\vec{\varepsilon}$ en un único valor ε que represente a la media de las distancias máximas:

$$\varepsilon = \frac{1}{m} \sum_{i=1}^{m} \varepsilon_j$$

podemos redefinir la función de pertenencia de un dato $\vec{x}$ a la zona Z_A. Si llamamos $\bar{d}_{\vec{x}}$ a la media de las distancias entre los atributos del nuevo dato y los atributos del prototipo:

$$\bar{d}_{\vec{x}} = \frac{1}{m}\sum_{j=1}^{m} d(\bar{x}_j, x_j)$$

Diremos que $\vec{x}$ pertenece a la zona del agente si la distancia media de $\vec{x}$ al prototipo es menor que la distancia media máxima de los datos de la memoria al prototipo ($\bar{d}_{\vec{x}} < \varepsilon$). Podemos definir la función de pertenencia de $\vec{x}$ al agente ($f_{Z_A}(\vec{x})$) como:

$$f_{Z_A}(\vec{x}) = \begin{cases} 1, & \text{sii } \bar{d}_{\vec{x}} \leq \varepsilon \\ 0, & \text{en otro caso} \end{cases}$$

La diferencia entre la media de las distancias máximas al prototipo (ε) y la media de las distancias entre el nuevo dato y el prototipo ($\bar{d}_{\vec{x}}$) ofrece un grado de pertenencia del dato a la zona del agente, de forma que si $\bar{d}_{\vec{x}} \leq \varepsilon$ entonces $\varepsilon - \bar{d}_{\vec{x}}$ cumple que es un valor positivo. Un valor cercano a 0 significa que el dato $\vec{x}$ está muy en el limite de la zona del agente, por lo que el grado de pertenencia a la zona es bajo. Por el contrario, un valor cercano a ε indica una pertenencia máxima del dato a la zona. En este caso, el dato $\vec{x}$ está muy cerca del prototipo, por lo que $\bar{d}_{\vec{x}} \to 0$.

Con esta definición, el dominio de la función de pertenencia es $[0, \varepsilon]$ y se define como:

$$f_{Z_A}(\vec{x}) = \varepsilon - \bar{d}_{\vec{x}} \tag{3.33}$$

Si trasladamos el dominio de la función al intervalo $[0, 1]$ obtenemos una función de pertenencia de un conjunto difuso, según Zadeh [106]. Esta función queda definida de la siguiente forma:

$$f_{Z_A}(\vec{x}) = \frac{\varepsilon - \bar{d}_{\vec{x}}}{\varepsilon} \tag{3.34}$$

Clasificación de un dato

Respecto a la clasificación de un nuevo dato, se ha dotado al agente de información de todas las clases que lo representan y se ha construido un prototipo para cada subzona del agente. Esta información puede ser explotada en la clasificación de un dato de diversas formas, que detallamos a continuación. En general, si llamamos c_A a la clase que predice el agente, definimos la clasificación de un dato $\vec{x}$ por un agente A como:

$$C_A(\vec{x}) = c_A$$

Si llamamos N_{c_k} al numero de datos con clase c_k en la memoria; c_{M_i} a la clase del dato i-ésimo; y nc al número de clases que intervienen en la memoria, vamos a definir tres métodos para que un agente obtenga la clase c_A para un dato:

1. La clase c_A que asigna el agente es la clase mayoritaria entre los elementos de la memoria,

$$c_A = c_k / max_{k \in [1..nc]} \{N_{c_k}\} \tag{3.35}$$

donde el número de datos con clase k se calcula como:

$$N_{c_k} = \sum_{i=1}^{|M|} acierto_{i,j}$$

siendo el valor $acierto_{i,j}$ igual a uno si la clase del dato i de la memoria es la clase k.

$$acierto_{i,j} = \begin{cases} 1 & \text{si } c_{M_i} = c_k \\ 0 & \text{si } c_{M_i} \neq c_k \end{cases}$$

Los datos de la memoria representan a la zona del dominio del agente. En general, excepto en zonas con ruido, la mayoría de los datos de la memoria corresponderán a una misma clase, ya que el entrenamiento del agente se ha basado en elegir datos cercanos entre si y el componente de la entropía en la formación del agente lo induce a tener pocas clases.

2. La clase c_A que asigna el agente es la clase del dato más cercano en la memoria.

$$c_A = c_{M_i} / min_{(\vec{x}_i, c_{M_i}) \in M_i} \ \{distancia(\vec{x}, \vec{x}_i)\} \tag{3.36}$$

Esta definición permite al agente escoger una clase no mayoritaria como predicción de la clase del dato. Esto es importante cuando la zona de especialización no es homogénea o existe ruido. En caso de que el agente tenga una zona homogénea, la clase del dato más cercano corresponderá con la clase mayoritaria.

3. La clase c_A que asigna el agente es la clase del prototipo más cercano en la memoria.

$$c_A = c_k / min_{P_{c_k} \in P_c} \ \{distancia(\vec{x}, P_{c_k})\} \tag{3.37}$$

En este caso, se construye un prototipo P_{c_i} por cada clase contenida en la memoria, donde cada prototipo resume a los datos con la clase k en el agente. La elección de la clase del prototipo más cercano permite elegir la subzona dentro del agente más parecida al dato.

3.5.7. Sistema multiclasificador

Las características generales del sistema multiclasificador en este modelo son análogas al definido en la primera aproximación. El sistema se compone de un conjunto de clasificadores, donde cada clasificador es especialista en un subdominio.

El sistema multiclasificador debe especificar como se generan los nuevos clasificadores (Sección 3.5.7.1); como se seleccionan los k mejores agentes (función g de la Ecuación 3.18); y como se combinan sus resultados (función Smc de la Ecuación 3.19). La introducción de una memoria para cada agente va permitir que el sistema multiclasificador pueda explorar las características de cada agente y utilice esta información para ofrecer mejores mecanismos de combinación que los descritos en el modelo MBLF. Ya que la función de clasificación de un agente C_A ha sido mejorada en este modelo, el sistema multiclasificador va a ofrecer un abanico más amplio de posibilidades en la combinación de los k mejores agentes.

3.5.7.1. Generación del conjunto de agentes

La generación del conjunto de agentes en esta aproximación, descrita en el Algoritmo 13, va a eliminar componentes fundamentales del algoritmo LF. En este caso, se va a mantener la idea de que el agente se mueve en un entorno donde se encuentran los datos, pero se va a eliminar el componente del algoritmo LF en el que el agente transporta un dato para colocarlo en una posición correcta. En realidad, este comportamiento del algoritmo LM podía hacer tender a nuestro sistema a generar agentes especialistas en zonas con ruido, ya que los datos bien agrupados en la arena tendían a no volver a ser elegidos por los agentes.

Otro cambio importante es que la configuración del agente depende del conjunto de los n datos en su memoria, de forma que el objetivo del agente en cada iteración de su entrenamiento es optimizar la calidad de esos datos, que caracterizan su zona de especialización.

Con estos cambios, vamos a continuación a hacer algunas reflexiones sobre cuantos agentes debemos generar; que características tienen los datos de entrenamiento que van a encontrar los agentes; y como se conforman las zonas de especialización con estas características.

Conjuntos de entrenamiento para los agentes

Una modificación significativa en este modelo es el conjunto de datos disponibles para un agente. En un instante dado, un dato de entrenamiento solo va a poder servir para entrenar a un agente. Dado que el agente guarda el dato en su memoria, este deja de estar disponible para el resto de agentes mientras siga en su memoria. Si el agente suelta el dato, pasa al entorno

para que otro agente pueda usarlo. Por tanto, los datos que hay en la arena en una iteración son los datos de entrenamiento que ningún agente lleva en su memoria. Además, se cumple que el número de datos de entrenamiento que hay en la arena en un instante t es constante, ya que cuando un agente coge un dato suelta otro en la arena.

Algoritmo 13 Generación del conjunto de agentes usando enfriamiento

Entrada: Conjunto de agentes, conjunto de datos de entrenamiento
Salida: Conjunto de agentes entrenados

```
{Inicialización}
para todo dato e hacer
   Colocar e aleatoriamente en la arena
fin para
para todo agente hacer
   Situar al agente aleatoriamente en la arena
   mientras la memoria del agente no esté llena hacer
      si hay un dato en la posición del agente entonces
         Meter el dato en la memoria
         Mover el agente a una nueva posición aleatoria
      fin si
   fin mientras
fin para
{Bucle Principal}
para t=1 to t_max hacer
   para todo agente hacer
      si lugar ocupado por dato e entonces
         Obtener el dato k de la memoria que hace que la valoración total
         sea menor
         Obtener la valoración sin k y con e (V_c(k,e))
         Obtener ΔV_c como ΔV_c = V_c(k,e) − V_c
         Generar un número aleatorio u uniforme (u ∈ [0,1] )
         si ΔV_c > 0 or u < e^(−ΔV_c / T) entonces
            Soltar el dato k en la arena
            Coger el dato e
            Actualizar el agente con e
         fin si
      fin si
      T = α * T
      Mover aleatoriamente al agente a una nueva posición
   fin para
fin para
```

En el modelo MBLF, el agente "recordaba" todos los datos usados en su entrenamiento, de forma que la información de un dato usado quedaba reflejada en su prototipo, después de que el agente lo hubiese soltado. Esta característica provocaba que un dato que no encontrase una buena

agrupación fuese un dato muy usado por los agentes e, incluso, que un agente usará más de una vez el mismo dato.

Número de agentes necesarios y zona de especialización

Para poder configurar el conjunto inicial de agentes, debemos determinar dos valores que están muy relacionados entre si. Necesitamos decidir cuantos agentes vamos a generar (nA) y cuantos datos (n) vamos a almacenar en la memoria de cada agente. Además, tenemos dos parámetros fijos que van a afectar al cálculo de los dos anteriores: datos de entrenamiento (ne) y el número de clases posibles (z dentro del dominio del problema.

Al igual que en el modelo MBLF, debemos definir un conjunto suficientemente grande de agentes para intentar garantizar que el conjunto cubre el total del dominio. En el este modelo (McML), tenemos que tener en cuenta que cada agente consume n datos de entrenamiento, que dejan de estar disponibles para el resto de agentes. Aparece un nuevo parámetro, que es el número de datos que quedan en la arena (nea) después de haber inicializado a los nA agentes rellenando sus memorias. Por tanto, se debe conseguir un equilibrio entre: el número de agentes, el tamaño de la memoria de cada agente y el número de datos de entrenamiento que quedan disponibles en la arena para el entrenamiento de los agentes. Tenemos una relación directa entre estos tres parámetros y el número de datos de entrenamiento ne:

$$nea = ne - n * nA \tag{3.38}$$

Esta ecuación indica que el número de datos que quedan en la arena es el número de datos de entrenamiento menos los datos usados para inicializar a los agentes.

Otra forma de obtener el número de datos que quedan en la arena nea es calculando el valor como un tanto por ciento (p) del número de datos de entrenamiento (Ecuación 3.39),

$$nea = \frac{p}{100} * ne \tag{3.39}$$

El número de datos que se usen para llenar la memoria de los agentes (ndA) será calculado como la resta entre los originales (ne) menos los que quedan en la arena (nea) (Ecuación 3.40),

$$ndA = ne - nea \tag{3.40}$$

Al ser ndA los datos que distribuiremos en los agentes, este valor es igual a la multiplicación del número de agentes nA por el número de datos en memoria n (Ecuación 3.41).

$$ndA = nA * n \tag{3.41}$$

La Ecuación 3.41 nos permite poner a los parámetros nA y n en función uno del otro. En general, si aumentamos el número de agentes se disminuye el tamaño de su memoria y viceversa. Fijado el número de agentes, calculamos el tamaño de memoria de la forma:

$$n = \frac{ndA}{nA} \tag{3.42}$$

Análogamente, fijado el tamaño de memoria podemos definir el número de agentes de la forma:

$$nA = \frac{ndA}{n} \tag{3.43}$$

Otra forma de calcular el número de agentes es asegurar un número de agentes por cada clase (s). Podemos añadir esta información para calcular el número de agentes, que será el número de clases multiplicado por el número de agentes por cada clase:

$$nA = z * s \tag{3.44}$$

Si usamos esta fórmula para obtener el número de agentes, quedarían fijados todos los parámetros, excepto la cantidad de datos en memoria, por lo que, usando la Ecuación 3.42, obtenemos el tamaño de la memoria.

Parada del algoritmo

Las condiciones de parada en este modelo (McML) son las mismas que el anterior, ya que las modificaciones hechas no influyen en la terminación. La idea básica del algoritmo LM se mantiene, realizando t iteraciones en el entrenamiento. Como mejora, se pueden aplicar los mismos test que en el modelo MBLF. Estos tests aplicados en distintas iteraciones del entrenamiento permiten verificar si el entrenamiento va mejorando al conjunto de agentes, en cuyo caso se sigue el entrenamiento, o bien parar el entrenamiento si la efectividad del conjunto de agentes se estanca o decrece.

La gestión de la memoria en un agente permite hacer un análisis más exhaustivo de como evoluciona el sistema, viendo como se configuran las zonas dentro un agente y reajustándolo, detectando agentes poco eficientes, duplicados, zonas no cubiertas o zonas con varios agentes. En cualquiera de los casos, el entrenamiento podría agrupar o eliminar agentes y crear nuevos agentes con zonas poco cubiertas.

3.5.7.2. Multiclasificador asociado

Como ha ocurrido en el modelo MBLF, una vez generado el conjunto de agentes, debemos determinar como el sistema multiclasificador va a seleccionar a los k mejores agentes para un dato, y como va a combinar las predicciones de los k agentes para ofrecer una clasificación final. La novedad ahora es que disponemos de más información de la zona de especialización y las clases que definen al agente. Esta información va a permitir al sistema ofrecer distintas alternativas, tanto para seleccionar a los agentes como para combinar sus resultados.

Selección de los k mejores agentes para un dato

El sistema multiclasificador ha sido diseñado para generar clasificadores especialistas en zonas, por tanto los k mejores agentes serán aquellos cuyas zonas de especialización incluyan mejor al dato a clasificar. Al igual que en el modelo MBLF, la zona general del agente ha sido definida con el prototipo del agente, en este caso como media de los datos de la memoria. Para decidir cuales son los mejores agentes se necesita establecer una medida de la especialización de cada agente respecto al dato a clasificar. Esta medida puede ser el grado de pertenencia del dato al agente $F_{Z_A}(\vec{x})$, definido en la Ecuación 3.34, que obtiene un valor en el intervalo [0,1].

Por lo que, dado un dato a clasificar y un valor de k, la función de selección g va a obtener a los k agentes cuya función de pertenencia del dato sea mayor (Ecuación 3.18).

Otra cuestión que se debe resolver es el valor de k, es decir, cuantos agentes van a intervenir en la clasificación del dato. Si el valor de k es demasiado grande se habrán seleccionado agentes que no cubran al dato, por lo que sus predicciones tenderán a no ser correctas.

Se podría pensar en un valor de k distinto para cada dato a clasificar, dependiendo de la cantidad de agentes cuya distancia al dato no supere un umbral preestablecido. Es decir, exigimos una pertenencia a la zona mínima para seleccionar un agente. En cualquier caso, el método de combinación de los clasificadores se podrá encargar, como veremos a continuación, de establecer preferencias por unos u otros agentes.

En general, también podemos aplicar los criterios del modelo MBLF. Si hemos obtenido el número de agentes a partir de un número s de agentes por clase, una buena aproximación es hacer que k sea el valor de s, ya que, en general, tendremos s agentes por clase.

$$k = s$$

Si se ha utilizado la idea de aplicar un porcentaje del número de datos de

entrenamiento, podemos calcular k de la siguiente forma:

$$k = \frac{nA}{z}$$

Combinación de los k mejores agentes

Anteriormente, se ha definido, en el apartado de uso de un agente (Sección 3.5.6), como el agente puede responder a la clasificación de un dato de tres formas: devolviendo la clase mayoritaria en su memoria, la clase del dato más cercano en la memoria o la clase del prototipo más cercano al dato, usando la función para clasificar un dato.

$$C_A(\vec{x}) = c_A$$

En cualquiera de los tres casos, el sistema multiclasificador debe decidir como va a predecir la clase final (c_f) a partir de las predicciones de los k clasificadores. Al igual que en modelo MBLF, la clasificación de un dato $\vec{x}$ por el sistema multiclasificador estará definida por la combinación de las predicciones de los k clasificadores (Ecuación 3.19), de forma que la clase final c_f será el resultado de aplicar un método de combinación (Ecuación 3.20).

En general, podemos pensar en definir criterios para evaluar la calidad de un agente para clasificar un dato. La evaluación de la calidad de un agente permite aplicar, como método de combinación, un método de votación ponderada, donde el peso del voto del clasificador depende de su calidad. En el caso que nos ocupa, la calidad de un agente varía dependiendo del dato que queramos clasificar.

Definimos V_{c_i} como el valor resultante de combinar las predicciones de los k agentes respecto a la clase i, ponderando cada voto de un agente por un valor de calidad asociado.

Diremos que el voto del agente j para la clase c_i es 1 si la clase obtenida por el agente j al clasificar $\vec{x}$ es la clase i ($C_{A_j}(\vec{x}) = c_i$) , y 0 en caso contrario. Definimos este voto ($voto_{ij}$) de la forma:

$$voto_{ij} = \begin{cases} 1 & \texttt{si } C_{A_j}(x) = c_i \\ 0 & \texttt{si } C_{A_j}(x) \neq c_i \end{cases}$$

La clase final c_f será aquella cuyo valor V_{c_f} sea mayor,

$$c_f = max_{i \in [1..nc]}\{V_{c_i}\}$$

Vamos a definir cuatro maneras de combinar las predicciones de los k mejores agentes: una donde el peso de cada agentes es el mismo para todos los agentes y tres donde el peso del agente depende de un criterio de calidad.

a) La forma más sencilla de combinar los resultados es aplicar votación simple, considerando que las predicciones de los k agentes tienen el mismo peso en la clasificación. En este caso se considera que todos los agentes tienen la misma calidad. Así, la clase final del sistema multiclasificador es la clase con mayor número de votos, donde cada agente da un voto por su clase. Si V_{c_i} representa al número de votos que ha obtenido la clase i, la clase final es la clase que mayor número de votos haya obtenido,

$$V_{c_i} = \sum_{j=1}^{k} voto_{ij} \tag{3.45}$$

La elección de los k mejores clasificadores ha ido ligada a la medición de la adecuación de la zona del agente al dato. Usar votación simple puede distorsionar el resultado del sistema ya que, si k es demasiado grande, el voto de agentes poco especialistas en el dato va a tener el mismo valor que el de agentes muy apropiados.

b) El primer criterio de calidad que consideramos es el mismo que hemos aplicado para seleccionar un agente para la clasificación, es decir, la pertenencia del dato a la zona del agente. Por tanto, la calidad de un agente es proporcional a la función de pertenencia del dato $\vec{x}$ al agente ($F_{Z_A}(\vec{x})$). Como la función de pertenencia está en el intervalo [0,1], la función es el peso del voto del agente. La clase final se obtiene ponderando el voto de cada agente por su función de pertenencia, de forma que la clase cuya suma de votos ponderados sea mayor es la clase devuelta por el sistema multiclasificador.

En este caso, V_{c_i} se define como el voto total obtenidos por los k agentes para la clase i después de haber ponderado las salidas de los k agentes por sus funciones de pertenencia,

$$V_{c_i} = \sum_{j=1}^{k} voto_{ij} * f_{Z_A}(\vec{x}) \tag{3.46}$$

c) Otra forma de establecer la calidad de un agente es definir la homogeneidad de un agente respecto a las clases que lo representan. Si el agente representa a una zona donde todos los datos son de la misma clase, la probabilidad de error en la predicción de la clase es menor que si intervienen varias clases, supuesto que en ambos casos la zona de los agentes es adecuada para un dato.

Para definir el peso de agente según esta medida, se utiliza el valor del

número de datos con clase mayoritaria (N_{Cm}) dividido entre el número de datos en la memoria. Si este valor es 1 tendremos homogeneidad máxima, ya que todos los datos son de la misma clase, y por tanto, la contribución de ese agente es máxima. La clase final es la clase cuya suma de votos ponderados con esta medida sea mayor,

$$V_{c_i} = \sum_{j=1}^{k} (voto_{ij} * \frac{N_{Cm}}{n}) \tag{3.47}$$

d) Las dos medidas de calidad definidas anteriormente pueden ser utilizadas conjuntamente. El objetivo, en este caso, es tener en cuenta tanto la homogeneidad del agente como su cercanía al dato. El peso del agente se puede calcular multiplicando la proporción de datos con clase mayoritaria y la pertenencia del dato al agente,

$$V_{c_i} = \sum_{j=1}^{k} (voto_{ij} * \frac{N_{Cm}}{n} * f_{Z_A}(\vec{x})) \tag{3.48}$$

Hemos definido tres métodos para seleccionar la clase de un agente A_i, y cuatro métodos que obtienen una valoración de la clase obtenida por el agente a la hora de combinar los resultados, por tanto, combinando la forma de obtener la clase de un agente con la forma de combinación, podemos obtener doce métodos (que llamaremos C_i) diferentes para combinar las decisiones de los k mejores agentes, dependiendo de la elección que hagamos en cada grupo.

3.5.8. Experimentos

Se han implementado ocho métodos de combinación (que notaremos como C_i, para i=1...8) de los doce posibles, utilizando distintas opciones para la elección de la clase por un agente y para la valoración de la decisión, descritas en la sección anterior. El objetivo es comprobar la efectividad de este modelo y ver los efectos que producen las distintas alternativas presentadas.

El primer grupo de funciones de combinación (C_1, C_2, C_3, y C_4) utilizan la definición de la Ecuación 3.35, donde la clase que predice el agente es la clase mayoritaria entre los datos de la memoria M y aplica los cuatro métodos de combinación descritos anteriormente. Para C_1, la combinación se realiza por votación simple, obteniendo un voto por cada agente (Ecuación 3.45). Este es el método más simple definido sobre los k mejores agentes, ya que valora de igual forma a todos los agentes que intervienen. En el caso de la combinación C_2, el peso del voto de cada agente es proporcional a la

pertenencia del dato a la zona del agente (Ecuación 3.46). En el método de combinación C_3, se ha utilizado la Ecuación 3.47, que ofrece una valoración más alta a aquellos agentes que son más homogéneos. Por último, el método C_4 es el utiliza información de homogeneidad y pertenencia del dato a la zona del agente (Ecuación 3.48).

El segundo grupo de métodos (C_5, C_6) usa la definición de la clase del agente como la clase del dato en la memoria más cercano al dato a clasificar (Ecuación 3.36). Para el método C_5, se ha implementado votación simple, (Ecuación 3.45), mientras que para el método C_6, se ha elegido votación ponderada por la pertenencia del dato al agente (Ecuación 3.46).

Por último, el tercer grupo de métodos (C_7, C_8) usa la clase del prototipo de clase más cercano al dato (Ecuación 3.37) como clase predicha por el agente, implementando la combinación, al igual que en C_5 y C_6, mediante votación simple y votación ponderada por la pertenencia del dato al agente, respectivamente.

La base de datos que se ha elegido para aplicar los ocho métodos de combinación es "Pen-Based Recognition of Handwritten Digits"(Pendigits), obtenida de UCI Machine Learning Repository [8]. Esta base de datos representa información sobre dígitos escritos, con 16 atributos para cada dato; siendo el número de clases 10 (una por cada dígito). Los autores proponen un conjunto de entrenamiento con 7.494 ejemplos, y un conjunto de test de 3.498 ejemplos.

Para generar el conjunto de agentes clasificadores, $H = \{A_1, ..., A_n\}$, se deben definir los parámetros siguientes: el número de agentes clasificadores que queremos entrenar y el tamaño de la memoria asociada a cada agente. Estos son dos parámetros relacionados, que a su vez dependen de la cantidad de datos de entrenamiento. El número de agentes elegido es el resultado de dividir el número de datos de entrenamiento entre el doble del tamaño de la memoria. Este valor permite asegurar que todos los agentes pueden llenar su memoria con datos de entrenamiento y en el entorno quedan la mitad de los datos de entrenamiento para que se produzca el aprendizaje.

En el caso que nos ocupa, se ha elegido un tamaño de memoria de 20 elementos. Así, el número de agentes que entrenamos es 189. Si suponemos que las clases están uniformemente distribuidas, podemos pensar que el número de agentes entrenados para cada clase es el número de agentes entre el número de clases (en nuestro caso 18). Este valor puede ser una referencia para elegir el parámetro k, que define la cantidad de agentes más cercanos al dato.

La aplicación de distintos métodos de combinación sobre la base de datos Pendigits se refleja en las Tablas 3.4 y 3.5. En estas tablas se ha definido una fila por cada método de combinación, y cada columna refleja el número de agentes utilizados por el método de combinación, donde una casilla en la tabla representa al tanto por ciento de acierto usando un método de

combinación (fila) y un número k de agentes (columna).

	K = 1	K = 4	K = 7
C1	43.53 %	49.71 %	53.25 %
C2	43.53 %	54.80 %	56.63 %
C3	43.53 %	60.49 %	58.71 %
C4	43.53 %	60.37 %	59.86 %
C5	83.07 %	86.93 %	87.59 %
C6	83.07 %	86.30 %	87.85 %
C7	80.93 %	83.07 %	84.56 %
C8	80.93 %	83.24 %	84.19 %

Tabla 3.4: Resultado de aplicar el modelo McML para Pendigits con 187 agentes para 1, 4 y 7 agentes seleccionados

	K = 10	K = 13	K = 16
C1	57.97 %	58.23 %	58.94 %
C2	59.06 %	59.40 %	60.32 %
C3	62.12 %	62.20 %	61.92 %
C4	61.74 %	62.34 %	63.20 %
C5	86.96 %	86.99 %	86.61 %
C6	87.50 %	87.22 %	86.87 %
C7	84.04 %	84.36 %	82.79 %
C8	83.96 %	83.87 %	83.01 %

Tabla 3.5: Resultado de aplicar el modelo McML para Pendigits con 187 agentes y 10, 13 y 16 agentes seleccionados

Si resumimos la información del experimento según los tres grupos de combinación, podemos observar que para los cuatro primeros clasificadores (C_1, C_2, C_3 y C_4), que obtienen como clase del agente a la clase mayoritaria, obtienen un resultado de un 20 por ciento menos de acierto respecto a los métodos que explotan la estructura generada por el agente, tanto si se obtiene la clase del dato más cercano (C_5, C_6), como si se obtiene la clase del prototipo más cercano (C_7, C_8).

En realidad, esta diferencia se justifica porque la elección de la clase mayoritaria olvida la existencia de otras clases influyentes en la definición de la zona del agente, representada por su prototipo.

Si comparamos estos experimentos con los experimentos del modelo MBLF (sección 3.4.5) para la base de datos Pendigits, podemos observar como hemos pasado de un 96,99 % en el mejor de loa casos del modelo MBLF, a un 87,85 % en este experimento (para C_6 y $k = 7$). Recordaremos que el caso del experimento del modelo MBLF teníamos 1150 agentes, que suponían un 15 % del conjunto de datos de entrenamiento (7494 ejemplos), por lo que en ese experimento conseguíamos, con el conjunto de agentes, una reducción

del conjunto de entrenamiento del 85%. En el experimento del modelo actual (McML), el número de agentes obtenido es significativamente menor que en el modelo MBLF. En este caso, los 187 agentes suponen un 2.5% de los datos de entrenamiento, lo que significa que nuestro agente en este modelo tiene una entidad mayor de clasificador respecto a la que tenia el agente del modelo MBLF.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

4

Sistema Multiclasificador Autoorganizativo Difusos (MAD)

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

4

Sistema multiclasificador autoorganizativo difuso (MAD)

En el capítulo anterior se han desarrollado dos modelos que usan, de una u otra forma, el algoritmo LF (algoritmo 4) como marco para definir el entorno y las acciones que realizan el conjunto de agentes. El modelo McML mantiene la arena como entorno donde están los datos de entrenamiento y los agentes. En este entorno, los agentes se mueven transportando datos pero sin buscar agrupaciones para ellos.

En este capitulo se presenta un nuevo modelo basado en el modelo McML del capítulo anterior. En términos generales, el modelo que definimos va a mantener la estructura y algunos elementos del anterior, eliminando el componente de la arena del algoritmo LF. El objetivo sigue siendo definir un sistema en el que los agentes son entes autónomos que se modifican y ajustan mediante datos de entrenamiento. El agente debe decidir si el dato que se le suministra le sirve para su entrenamiento (mejora al agente) o no, devolviendo al entorno o bien el dato suministrado o bien el dato de su estructura que fue sustituido.

Para este nuevo modelo, los datos de entrenamiento van a estar disponibles para todos los agentes, de forma que un dato de entrenamiento podrá ser ofrecido a la totalidad de los agentes. De esta forma, se elimina la aleatoriedad del encuentro entre el dato y el agente, que se producía en la arena en los modelos precedentes. En este sentido, el encuentro de un dato por un agente es simulado por el sistema multiclasificador ofreciendo el dato al agente.

Respecto al uso de un agente, este modelo va a explotar la información sobre su zona de especialización, ofreciendo, al clasificar un dato, una distribución de posibilidad [106, 67] sobre cada una de las clases conocidas, lo que configura a un agente como un clasificador difuso [57]. Esta

nueva definición de la clasificación de un dato va a permitir al sistema multiclasificador poder definir mecanismos de combinación más completos, que manejen la mayor cantidad de información ofrecida por el agente. El concepto de acierto o fallo será redefinido teniendo en cuenta la totalidad de la información ofrecida por cada agente.

Para finalizar, se fijará una de las novedades más importantes que aporta este modelo, como es la inclusión de mecanismos de autoorganización en el sistema multiclasificador. Esta autoorganización del sistema consistirá en la capacidad de transformarse mediante la eliminación de agentes, su modificación, o la incorporación de nuevos agentes, con el objetivo de dirigir el proceso de generación del propio sistema. Durante el entrenamiento, se pretende que el sistema sea capaz de detectar zonas de dominio que no están siendo cubiertas por ningún agente, así como la acumulación de agentes en determinadas zonas. En definitiva, se pretende asegurar que los agentes cubren todo el dominio del problema y salvaguardar la diversidad dentro del conjunto de agentes.

A modo de resumen, el modelo que se presenta en este capítulo crea y adiestra un conjunto de agentes clasificadores difusos, detectando zonas no cubiertas y generando nuevos agentes para ellas; detecta y elimina (simplifica) aquellos agentes redundantes que clasifican de forma similar y, con el conjunto final, establece un sistema multiclasificador mediante una selección y combinación de clasificadores. Llamaremos a este modelo: "Modelo autoorganizativo difuso (modelo MAD)".

En las secciones siguientes se presentan los cambios sobre la estructura de los agentes del modelo anterior, así como la nuevas funcionalidades incorporadas. El capítulo finaliza detallando la generación del conjunto de agentes y el modelo del sistema multiclasificador obtenido.

4.1. Estructura del agente

El modelo que se presenta mantiene la estructura del agente del modelo McML. El objetivo sigue siendo tener una representación de la zona de especialización del agente, mediante una memoria con los datos que mejor caracterizan a la zona.

Sea el prototipo del agente P_A la representación canónica que resume en un solo valor la zona general del agente, y P_{c_i} el prototipo de clase que identifica, de forma única, a la subzona del agente formada con una clase c_i.

Podemos observar la estructura del agente en la Figura 4.1, que queda definida de la siguiente forma:

- $M = \{e_1, \ldots, e_n\}$, donde $e_i = (\vec{x_i}, c)$. M es la memoria del agente,

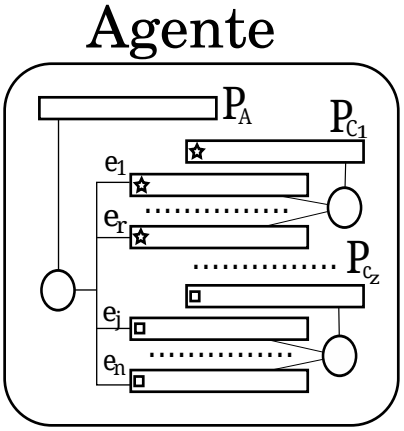

Figura 4.1: Estructura de un agente

compuesta por n datos de entrenamiento.

- $P_A \in R^m$ es el valor prototípico del agente, donde m es el número de atributos que definen el prototipo.
- $P_c = \{P_{c_1}, \ldots, P_{c_z}\}$ es el conjunto con los prototipos de las z clases que pueden intervenir en la memoria M, donde $P_{ci} = \{\vec{x}_k \ \ |(\vec{x}_k, ci) \in M\}$ es el prototipo de los datos en M que tienen clase c_i.

La eliminación de la arena del modelo hace innecesario cualquier dato referente a la posición del agente en la misma, así que esta información ya no aparece en la estructura del agente.

4.2. Criterios para la evaluación de la calidad del agente

Los agentes buscados son aquellos que están centrados en zonas del dominio bien definidas y están especializados en determinadas clases. Como ya se introdujo en los modelos anteriores, estas características pueden evaluarse mediante dos medidas: una medida sobre la varianza de los datos de la memoria, que informa sobre la calidad del prototipo del agente como representante de la zona, y la entropía de las clases del agente, como medida del nivel de especialización. Un proceso de entrenamiento del agente guiado por la mejora de estos dos valores, es decir, la calidad de su zona de especialización, modificará la configuración de su memoria.

Al igual que en los modelos precedentes, se mantiene como varianza del agente ($\bar{V}$) a la media de las varianzas de los atributos de los datos

en memoria (Ecuación 3.28) y la entropía (E) (Ecuación 3.29) de los datos de memoria. En el modelo McML, se combinaban estos dos valores para obtener la valoración del agente, usando un parámetro β que repartía la influencia de cada medida en la valoración final. El inconveniente de esta combinación lineal simple radica en que la varianza debía ser normalizada en el intervalo [0,1] para poder ser combinada con la entropía, ya que las magnitudes dispares entre varianza y entropía hacen que esta valoración sea muy dependiente del dominio de los datos.

Para evitar esta dependencia, se define una nueva valoración de la calidad de un agente (V_C), de la siguiente forma:

$$V_c = \bar{V} * (1 + E) \tag{4.1}$$

donde $V_c \in [\bar{V}, 2 * \bar{V}]$.

Esta combinación del valor promedio de varianzas con la entropía elimina el uso de un parámetro β y no obliga a normalizar la varianza. El criterio que se pretende aplicar en la valoración de la calidad es la mejora de la calidad al disminuir la entropía, frente al criterio de la mejora de la varianza para mejorar la calidad. La eliminación de una clase completa de un agente mejora sustancialmente su valoración, frente a un cambio simple de valor de las varianzas.

Cuando hay una clase en la memoria del agente, la entropía E es 0, de forma que la valoración de la calidad sólo refleja el nivel de agrupamiento de los datos, mientras que, en caso de que la entropía sea máxima ($E = 1$), el agente contiene a todas las clases posibles y la valoración de la calidad es el doble del valor de la varianza.

El peso del valor de la entropía en la valoración de la calidad puede fácilmente modificarse, sustituyendo la constante 1 por un parámetro k como se indica en la Ecuación 4.2.

$$V_c = \bar{V} * (k + E) \tag{4.2}$$

siendo $k >= 1$, donde para mayores valores de k menor será la influencia de E. En la Figura 4.2 se observa la valoración de la calidad cuando k es 1 y k es 5. El eje x representa a la varianza del agente en un rango de [0,20], el eje z representa a la entropía, y el eje y representa a la valoración de la calidad.

Tanto en el modelo McML como en el modelo que se está definiendo, la valoración de la calidad V_c es un valor a minimizar, ya que el agente será mejor cuando su varianza y su entropía son pequeñas. Es decir, la calidad del agente es mayor cuanto menor es el valor de V_c

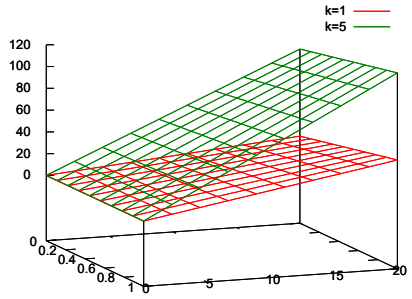

Figura 4.2: Efecto del parámetro k en el valor de la calidad del agente V_c

4.3. Creación del agente

En este modelo se mantiene la idea de que la inicialización de la zona del agente sea aleatoria. Al igual que en los dos modelos anteriores, el objetivo que se pretende conseguir es que un agente no tenga información previa de la zona donde tiene que ser especialista. Esta característica hace que el agente obtenga inicialmente una zona de especialización desconocida y aleatoria, que normalmente será muy extensa y de poca calidad, de forma que cuando el agente evolucione tienda a reducir la zona y mejorar su calidad. Para que esto ocurra, consideramos que los datos de entrenamiento están uniformemente distribuidos sobre el dominio a estudiar, y que no existe sesgo en los mismos.

Al no disponer de la arena del algoritmo LF, la inicialización de un agente se realiza mediante la selección aleatoria, con una distribución de probabilidad uniforme, de los n datos de su memoria a partir de la lista de entrenamiento, tal y como muestra el algoritmo 14.

Algoritmo 14 Inicialización de un agente: modelo MAD

Entrada: Agente
Salida: Agente con su memoria llena
1: **mientras** la memoria del agente no este llena **hacer**
2: Sacar de forma aleatoria un dato de la lista de entrenamiento
3: Meter el dato en la memoria del agente
4: **fin mientras**

4.4. Entrenamiento de un agente

El entrenamiento de un agente, al igual que en los modelos anteriores, se basa en el objetivo de especializar al agente en un dominio no predeterminado (Z_A). El entrenamiento se transforma en un proceso iterativo

de optimización, donde la función a optimizar es la valoración de la calidad de la zona de especialización del agente, medida como combinación de la varianza y la entropía (Ecuación 4.1). La optimización de la calidad se realiza minimizando el valor V_c, ya que la calidad del agente es máxima cuando su varianza y entropía son mínimas.

El entrenamiento de un agente con un dato consiste en intercambiar un dato de la memoria por el dato de entrenamiento, o por contra, no realizar ningún cambio y rechazar el dato. El sistema multiclasificador va a simular el entorno donde se mueven los datos y los agentes. El agente recibe un dato de entrenamiento y decide si mejora su zona, en cuyo caso devuelve al entorno el dato de su memoria que proporcionaba peor calidad, actualizando su estructura con el nuevo dato. Si el dato ofrecido no mejora su zona el agente lo devuelve al entorno. Se puede observar, en el algoritmo 15, este proceso.

Algoritmo 15 Entrenamiento de un agente con un dato: modelo MAD

Entrada: dato e de entrada
Salida: dato k (peor dato de la memoria), o dato e
1: Obtener el dato k de la memoria que hace que la valoración V_c sea mayor
2: Obtener la valoración sin k y con e ($V_c(k, e)$)
3: **si** $V_c(k, e) < V_c$ (Si el dato e mejora al agente) **entonces**
4: Coger el dato e
5: Actualizar el agente con el dato e
6: Devolver el dato k.
7: **si no**
8: Devolver el dato e
9: **fin si**

Este proceso de optimización local (para cada agente), se realiza simultáneamente para todos los agentes del sistema, en un algoritmo de optimización por haz de soluciones [83], por lo que en este modelo no se ha incorporado el enfriamiento definido en el modelo McML, debido a que la propia diversidad de agentes va a favorecer la salida de mínimos locales del sistema multiclasificador.

La actualización del agente con el dato se produce de la misma forma que el modelo McML, es decir, el nuevo dato cambia el prototipo del agente y el prototipo de la clase del dato, además de añadir el nuevo dato a la memoria y eliminar el peor dato. Como consecuencia de estos cambios, otro valor que se modifica en el agente es su valoración de calidad, que habrá mejorado si se ha quedado con el nuevo dato. El algoritmo de actualización es igual al del modelo McML (algoritmo 12).

4.5. Uso de un agente

El concepto de agente clasificador especialista en un subdominio del total, implica un modelo de agente en el que la adecuación de su clasificación está directamente relacionada con la adecuación del dato que se quiere clasificar a su zona de especialización. Si el dato tiene una clara pertenencia a la zona del agente, este deberá tener una aportación a la clasificación mayor que si el dato no pertenece a su zona. Incluso, si esta pertenencia es baja, el agente no debería contribuir a la clasificación del dato. Este tipo de valoración no es general del agente sino dependiente del dato y será tenida en cuenta por el multiclasificador a la hora de agregar las distintas clasificaciones.

En las siguientes secciones se fijarán los criterios para la creación de una distribución de posibilidad sobre las clases predichas por el agente.

4.5.1. Clasificación de un dato por un agente

En general, podemos afirmar que la clasificación de un dato por un agente implica que el agente debe decidir, entre las clases que lo constituyen, cual es la clase más posible para el dato a clasificar, descartando el resto de clases existentes. En el modelo McML, se han definido tres formas para elegir la clase que predice el agente: la clase mayoritaria del agente (Ecuación 3.35), la clase del dato en memoria más cercano al dato a clasificar (Ecuación 3.36) y, por último, la clase cuyo prototipo de clase es más cercano al dato (Ecuación 3.37). En realidad, el uso de cualquiera de estas tres opciones implica desechar el resto de clases como posibles soluciones y, en consecuencia, prescindir de información sobre el resto de las clases en el agente.

En este modelo, un agente clasifica un dato construyendo una distribución de posibilidad [67] sobre las clases del propio agente, configurando una clasificación difusa [57] del dato. La ventaja de esta opción es que no perdemos información sobre el resto de clases del agente y el sistema multiclasificador puede hacer un tratamiento más extenso de esta, a la hora de combinar las respuestas de los agentes.

La clasificación de un dato $\vec{x}$ por un agente A_i viene definida por una lista de pares (clase, posibilidad) para cada clase, según la siguiente ecuación:

$$C_{A_i}(\vec{x}) = [(c_1, p_{i,1}), ..., (c_z, p_{i,z})] \tag{4.3}$$

Donde $p_{i,j} \in [0, 1]$ es la posibilidad de que la clase del dato $\vec{x}$ sea c_j en el agente A_i. Como la asignación de posibilidad ha de ser sobre el total de las clases que configuran el problema, la distribución de posibilidad obtenida en la clasificación será una distribución con los valores $p_{i,j}$, de forma que

sumen uno, ya que se realiza una partición completa del espacio de clases posibles [47, 7, 85, 103]),

$$\sum_{j=1}^{z} p_{i,j} = 1$$

Para facilitar la lectura, notaremos p_i al valor de posibilidad de la clase i del un agente cualquiera, mientras que se mantendrá la notación $p_{i,j}$ si fuese necesario hacer referencia a la posibilidad de la clase en diferentes clasificaciones de diferentes agentes. Ya que el valor p_i va a representar la posibilidad de que el dato $\vec{x}$ sea de la clase i, debemos determinar en base a que criterios vamos a obtener este valor.

En general, podemos pensar que la posibilidad de una clase será mayor cuanto más cercano esté el dato a una representación de esa clase. Este criterio de cercanía, entre el dato y las clases, puede ser medido por una norma que fije la distancia entre el dato y una clase, o por una función de pertenencia directa a dicha clase.

Consideramos, en primera instancia, una función de similitud [108, 107, 82, 31] difusa, fundamentada sobre una función de distancia entre dos puntos x e y, y normalizada en el intervalo [0,1]. La similitud entre dos puntos se define como la inversa de la distancia d:

$$sim(x, y) = 1 - d(x, y) \tag{4.4}$$

Si usamos la distancia Euclidea, podemos normalizarla dividiendo la distancia entre el rango máximo de valores:

$$d(x, y) = \sqrt{\frac{(x-y)^2}{(max - min)^2}} \tag{4.5}$$

Otros tipos de funciones distancia podrían ser utilizadas, ya que dicha función sólo proporciona la posibilidad de trabajar en un espacio normado, que es el objetivo.

Una vez que se ha fijado la función distancia, se especificará, en base a la información existente en la memoria del agente, que representante se debe considerar para cada clase. Para este propósito existen varias alternativas: considerar el prototipo de la clase como valor representativo y confrontar el dato con cada prototipo, o buscar los datos internos más cercanos a nuestro dato para cada clase.

Similitud del dato al prototipo de clase

En este caso, como el agente contiene un prototipo por cada clase, que representa a la media de los datos en memoria de con esa clase, la posibilidad de una clase respecto a su prototipo ($posP_i$) se define como la similitud entre el dato a clasificar y el prototipo de la clase c_i, que representa al centro de los datos con clase i:

$$posP_i = sim(\vec{x}, P_{c_i}) \tag{4.6}$$

En la práctica, la posibilidad de una clase definida de esta forma es la similitud con el representante de la clase, que es el prototipo de clase.

El valor de la posibilidad obtenido para una clase es independiente del obtenido para el resto de clases. Si queremos que el valor $posP_i$ esté relacionado con los valores del resto de clases, vamos a normalizarlo de forma que la suma de todas las posibilidades sea 1. Obtenemos la posibilidad normalizada de una clase i (p_i) de la siguiente forma:

$$p_i = \frac{posP_i}{\sum_{j=1}^{z} posP_j} \tag{4.7}$$

donde la posibilidad para toda clase que no esté presente en la memoria del agente se define como 0.

Similitud del dato con el dato más cercano en una clase

Como una clase dentro de un agente está representada por los datos de esa clase que están en la memoria, podemos definir la posibilidad de una clase i ($posD_i$) como el máximo entre las similitudes del dato $\vec{x}$ y cada dato en memoria con clase i. En este caso, el dato más cercano en la memoria con clase i es el que nos ofrece la medida de posibilidad, con lo que la posibilidad de una clase es la similitud máxima entre $\vec{x}$ y los datos de esa clase.

$$posD_i = max_{j \in [1, d_{c_i}]} \{sim(\vec{x}, \vec{x_j})\} \tag{4.8}$$

siendo d_{c_i} el número de datos en memoria con clase i.

Normalizaremos las posibilidades del agente, transformando cada posibilidad de la misma forma que lo hicimos para la similitud al prototipo, obteniendo la posibilidad normalizada (p_i):

$$p_i = \frac{posD_i}{\sum_{i=1}^{z} posD_i} \tag{4.9}$$

El uso de funciones distancia en dominios con valores muy diversos y dispares en sus magnitudes puede originar sesgos indeseados en dichas medidas. Esto es fácilmente solucionable mediante la normalización del conjunto de entrenamiento, para obtener un conjunto donde todas las dimensiones tengan de media 0 ($\bar{x}$) y desviación típica 1 (σ). Esta normalización se hace mediante el desplazamiento de cada valor x por su media $\bar{x}$, escalado por su desviación (σ),

$$x_{norm} = \frac{x - \bar{x}}{\sigma} \tag{4.10}$$

Fijada la forma en la que se va a obtener la distribución de posibilidad sobre cada una de las clases de un agente, nos centraremos en la función del sistema multiclasificador, que es generar un conjunto de agentes especialistas en subdominios que cubran el dominio total. Esta propiedad es necesaria para garantizar que, para cualquier dato que se quiera clasificar, al menos exista un agente especialista en su zona. Tras la generación del conjunto de agentes, deberá ser el sistema multiclasificador el que se encargue se seleccionar a los agentes expertos para el dato y combinar sus decisiones para efectuar la clasificación.

Tenemos, por tanto, dos tareas bien diferenciadas que detallamos a continuación: la generación del conjunto de agentes y la definición de un mecanismo de elección y combinación de expertos en la clasificación.

4.6. Generación del conjunto de agentes

En el modelo que presentamos en este capítulo se mantiene la idea de generar un conjunto de agentes de forma que cada agente se mueve en el entono y toma decisiones respecto a su entrenamiento, independientemente de las decisiones que toman el resto de agentes.

En los modelos anteriores se ha partido de la suposición de que un conjunto de agentes, lo suficientemente grande e inicializado aleatoriamente sobre el dominio total de los datos, produce un conjunto de agentes especialistas que cubren el total del dominio. En este sentido, la cantidad de agentes en el sistema se mantiene constante, tanto en el entrenamiento de los agentes como durante su uso en la clasificación.

En este modelo se ha incorporado al sistema multiclasificador mecanismos de autoorganización en la generación de los agentes. Por un lado, el

sistema va a intentar localizar zonas del dominio no cubiertas por ningún agente. Esta detección conlleva la generación de nuevos agentes que cubran las zonas no cubiertas, de forma que, la introducción de este nuevo componente en el sistema hace que la generación de los agentes sea un proceso emergente, que se adapta a las condiciones que van creando los agentes en el entrenamiento.

Por otro lado, el procedimiento de generación de los agentes puede provocar que existan varios agentes que se hayan especializado en zonas parecidas y cuyo comportamiento sea similar. Para tener control sobre estas situaciones, se van a introducir mecanismos en el sistema multiclasificador que las detecten, de forma que agentes parecidos sean eliminados del sistema y sean sustituidos por un solo agente que los represente.

La generación de nuevos agentes para zonas no cubiertas y la sustitución de subconjuntos de agentes similares por un agente que los represente, conforma un sistema multiclasificador en el que el número de agentes va evolucionando y adaptándose a las condiciones descritas.

Detallaremos en esta sección como se va a definir la generación de los agentes con las características expuestas anteriormente. La generación de los agentes se va a desarrollar teniendo en cuenta cuatro procesos claramente diferenciados:

- Creación e inicialización del conjunto de agentes.
- Entrenamiento básico del conjunto de agentes.
- Detección de zonas no cubiertas, generando nuevos agentes.
- Reducción del conjunto de agentes, sustituyendo agentes similares por un único agente.

El algoritmo de generación de agentes basado en estos cuatro procesos es descrito, de forma general, en el Algoritmo 16, desarrollando cada uno de procesos desde la Sección 4.6.1 hasta la Sección 4.6.4.

4.6.1. Creación e inicialización del conjunto de agentes

La idea de definir un sistema multiclasificador que emerge sin información previa nos lleva, al igual que en los anteriores modelos, a tener que tomar decisiones respecto las características iniciales del conjunto de agentes. La elección del número de agentes en el sistema, el tamaño de la memoria de un agente y el método para inicializar la memoria de cada agente determinan la configuración inicial del sistema.

Algoritmo 16 Generación de agentes: modelo MAD

Entrada: cE
Salida: cA

```
 1:
 2: {Datos}
 3: cE –>  conjunto de entrenamiento
 4: cA –>  conjunto de agentes generado
 5: cAzonas –>  conjunto de agentes generado para las zonas no cubiertas
 6: cD –>  conjunto de datos devueltos en el entrenamiento básico
 7:
 8: {Algoritmo}
 9: cA = Creación de agentes con cE
10: (cA,cD) = Entrenamiento básico (cA,cE)
11: cAzonas = Generación de Agentes para zonas no cubiertas (cD,cA)
12: cA = cA ∪ cAzonas
13: cA = Reducción del conjunto de agentes (cA)
```

Número de agentes y tamaño de la memoria de un agente

La información de partida del sistema multiclasificador respecto a las características de dominio del problema son, únicamente, la cantidad de datos de entrenamiento de que se dispone y el número de clases que intervienen en el dominio (z). En este sentido, cualquier método que se proponga para calcular el número de agentes del sistema y el tamaño de la memoria de un agente (n) debe tener en cuenta, por un lado, que exista una proporcionalidad entre el número de datos de entrenamiento que se van a usar para entrenar a los agentes y los datos de entrenamiento que se van a usar para inicializar la memoria de los agentes. Por otro lado, el número de agentes debe ser lo suficientemente grande como para cubrir ampliamente las posibles subzonas del dominio, entendiendo que por lo menos debe haber una subzona por cada clase.

Estas características generales, que fueron desarrolladas en el modelo MAC dentro del la generación del conjunto de agentes (Sección 3.5.7.1), se siguen cumpliendo en modelo actual. Existen pequeñas variaciones en la terminología, como el concepto de datos que quedan para el entrenamiento, que en el modelo MAC son los datos que quedan en la arena y en este modelo, al desaparecer la arena, son los datos del conjunto de entrenamiento inicial.

Se ha desarrollado un algoritmo que describe un método para obtener el número de agentes y el número de datos en la memoria de cada agente (algoritmo 17). El método se basa en la asignación de unos valores mínimos para el número de agentes y para el número de datos en memoria. Por otro lado, el número de datos disponibles para inicializar los agentes (ndA) se va a calcular como resultado de aplicar un tanto por ciento al número de datos de entrenamiento.

Algoritmo 17 Obtención del número de agentes y tamaño de la memoria: modelo MAD

Entrada: p,s,ne, n-min
Salida: nA,n

```
 1:
 2: {Datos}
 3: p –>  porcentaje de datos para llenar la memoria de los agentes
 4: s –>  nº de agentes por clase
 5: ne –>  nº de datos de entrenamiento
 6: n-min –>  nº mínimo de datos en la memoria
 7: nA-min –>  nº mínimo de agentes
 8: ndA –>  nº de datos para inicializar los agentes
 9: ndA-min–>  nº de datos mínimo para inicializar los agentes
10: nA –>  nº de agentes final
11: n –>  nº de datos en memoria final
12:
13: {Algoritmo}
14: ndA = p * ne
15: nA-min = z * s
16: ndA-min = nA-min * n-min
17: si ndA-min <ndA entonces
18:   repetir
19:     nA = nA +1
20:     n = n +1
21:     ndA-min = n * nA
22:   hasta que ndA-min >= ndA
23:   nA = nA - 1
24:   n = n - 1
25: si no
26:   repetir
27:     nA = nA - 1
28:     n = n - 1
29:     ndA-min = n * nA
30:   hasta que ndA-min <ndA
31:   nA = nA + 1
32:   n = n + 1
33: fin si
```

Tal y como expresamos en la Ecuación 3.44, se puede calcular el número mínimo de agentes teniendo en cuenta el número de clases (z) que intervienen en el dominio. Si tengo z clases posibles se puede pensar en tener un mínimo de s agentes por cada clase. Esta estimación no garantiza que al inicializar el conjunto de agentes haya s agentes por clase, simplemente ofrece una medida relacionada con el número de clases, que en una distribución uniforme de clases y una inicialización aleatoria tiende a cumplirse. Con esta definición, el número de agentes mínimo es $z * s$.

El número de datos mínimo en memoria se puede establecer como parámetro del método. De forma general, cabe pensar en que un valor cercano a cinco es un mínimo aceptable para representar a una zona de un agente.

Si tenemos los valores mínimos para el número de agentes y el número de datos en memoria, la cantidad de datos de entrenamiento mínima ($ndA-min$) que necesitamos es el resultado de multiplicar ambos valores. Como tenemos una cantidad de datos ndA disponibles para inicializar a todos los agentes, si se cumple que $ndA-min$ es menor que ndA podemos aumentar el número de agentes mínimo y/o el tamaño de la memoria. Si $ndA-min$ es mayor que ndA estaríamos indicando que el número de agentes y su memoria sobrepasan el número de datos disponibles para su inicialización, por lo que deberemos disminuir ndA y/o el tamaño de la memoria.

El algoritmo propuesto va a ir incrementando estos valores mínimos en una unidad mientras los datos que se necesiten para inicializar a esa cantidad de agentes sea menor que el número de datos disponibles para la inicialización, o decrementando en una unidad en caso contrario. De esta forma, conseguimos ajustar la cantidad de agentes y el tamaño de la memoria al conjunto de datos para la inicialización. Una posible modificación a este algoritmo consiste en no aumentar de igual forma la cantidad de agentes y el tamaño de memoria. Si queremos primar el aumento de agentes, se podría aumentar en dos el número de agentes y en uno el tamaño de la memoria o viceversa.

Creación e inicialización del conjunto de agentes

Una vez fijado el número de agentes y el tamaño de la memoria de cada uno, se debe abordar a inicialización de la memoria de cada agente. Este proceso viene determinado por el requerimiento de que el conjunto de agentes cubra todo el dominio del problema. Una forma de conseguir esto, es que la zona de especialización de cada agente, representada por su memoria, sea inicializada con datos obtenidos aleatoriamente del conjunto de entrenamiento. Este tipo de inicialización va a a producir agentes cuyas zonas son extensas y por tanto de poca calidad. En realidad, no importa que las zonas sean extensas ya que el entrenamiento de los agentes provocará que esas zonas se vayan mejorando.

Una decisión que afecta al desarrollo del resto de fases en la generación del conjunto de agentes es si los datos de entrenamiento usados para la inicialización de los agentes se van a mantener en el conjunto de entrenamiento o se van eliminar del conjunto. En nuestro caso, se ha decidido conservar los datos de inicialización por diferentes motivos. El modelo que se está definiendo se basa en la idea de que podemos tener varios agentes especializados en zonas parecidas, ya que la última fase de la generación detecta agentes similares y reduce el conjunto. Por este motivo,

quitar del conjunto de entrenamiento el dato usado en la creación de un agente elimina la posibilidad de que otro agente pueda ser entrenado con él.

El proceso general de creación del conjunto de agentes conlleva el cálculo del número de agentes y el tamaño de la memoria, la creación de los agentes y la inicialización de la memoria de cada agente, tal y como se describe en el Algoritmo 18.

Algoritmo 18 Creación e inicialización del conjunto de agentes: modelo MAD

Entrada: cE
Salida: cA

```
 1:
 2: {Datos}
 3: cE –>  conjunto de datos de entrenamiento
 4: cA –>  conjunto de agentes
 5: nA –>  nº de agentes
 6: n –>  nº de datos en memoria
 7: p –>  porcentaje de datos para llenar la memoria de los agentes
 8: s –>  nº de agentes por clase
 9: ne –>  nº de datos de entrenamiento
10: n-min –>  nº mínimo de datos en la memoria
11:
12: {Algoritmo}
13: nA,n = Obtener Parámetros con p, s, ne, n-min
14: cA = {}
15: // Creación
16: para i = 1 to nA hacer
17:    a = crear agente (n)
18:    cA = cA ∪ { a }
19: fin para
20: // Inicilaización
21: para todo agente hacer
22:    mientras la memoria del agente no esté llena hacer
23:       Obtener de forma aleatoria un dato de cE
24:       Meter el dato en la memoria del agente
25:    fin mientras
26: fin para
```

4.6.2. Entrenamiento básico del conjunto de agentes

El entrenamiento del conjunto de agentes requiere de la especificación del modelo en el que se va a simular el movimiento autónomo de los agentes, en un entorno en el que se encuentran los datos. En los modelos anteriores se había desarrollado explícitamente este entorno con la definición de la arena del algoritmo LF. El inconveniente principal de la arena es que los movimientos aleatorios de los agentes en ese espacio pueden provocar

muchos movimientos improductivos, ya que un movimiento no garantiza pasar a una posición con un dato de entrenamiento. En este sentido, el diseño del entrenamiento en base a un número de movimientos en la arena tampoco nos da una idea del número de datos que ha encontrado un agente.

El objetivo que se plantea con esta nueva aproximación es definir una simulación del entorno en la que el objetivo original de "encontrar un dato en el entorno" se realice en un solo paso. Se va a transformar la idea de que un agente busca un dato en el entorno, de forma que va a ser el dato el que "encuentra" a un agente. En general, vamos a imaginar un entorno en el que están los agentes y un dato es lanzado en el entorno. En este punto, puede ocurrir que el dato sirva de entrenamiento a algún agente, con lo que un agente que coja el dato debe devolver al entorno un dato de su memoria; o que el dato no sirva al entrenamiento de ningún agente, en cuyo caso el dato debe seguir en el entorno.

Podemos asignar diferentes comportamientos al conjunto de agentes respecto a como va a ser tratado el dato que se lanza al entorno. Estos comportamientos son: comportamiento colaborativo, competitivo y autónomo, descritos a continuación.

Algoritmo 19 Asignación de un dato a los agentes con comportamiento colaborativo

Entrada: e, cD
Salida: cD, esDatoEscogido

```
 1:
 2: {Datos}
 3: e –>  dato de entrenamiento
 4: cD –>  conjunto de datos devueltos
 5: esDatoEscogido –>  cierto si el dato se lo queda algún agente
 6: nA –>  nº de agentes
 7: k –>  peor dato en la memoria
 8:
 9: {Algoritmo}
10: esDatoEscogido = false
11: para todo agente hacer
12:    si dato e mejora al agente entonces
13:       Actualizar el agente con e y devolver k
14:       esDatoEscogido = true
15:       cD = cD ∪ {k}
16:    fin si
17:    si No esDatoEscogido entonces
18:       cD = cD ∪ {e}
19:    fin si
20: fin para
```

Comportamiento colaborativo de los agentes respecto a un dato

Si los agentes tienen un *comportamiento colaborativo*, el dato es ofrecido a todos los agentes y cada agente decide si lo utiliza para su entrenamiento o lo devuelve al entorno. Esta nueva filosofía tiene dos consecuencias sobre el sistema: por un lado, un dato escogido por n agentes produce la devolución de n datos al entorno, el peor dato de cada agente; por otro lado, un mismo dato puede ser usado para entrenar a varios agentes. Finalmente, en caso de que ningún agente quiera el dato para su entrenamiento, el dato vuelve al entono.

El Algoritmo 19 va a gestionar un conjunto de datos devueltos (cD) al entorno. Para cada dato ofrecido a los agentes, se va añadir al conjunto de devueltos todos los datos devueltos por los agentes que hayan cogido el dato, o el propio dato si ningún agente lo ha elegido.

Algoritmo 20 Asignación de un dato a los agentes con comportamiento competitivo, aplicando el criterio de pertenencia

Entrada: e,cD
Salida: cD, esDatoEscogido

```
 1:
 2: {Datos}
 3: e –>  dato de entrenamiento
 4: cD –>  conjunto de datos devueltos
 5: esDatoEscogido –>  cierto si el dato se lo queda algún agente
 6: k –>  peor dato en la memoria
 7: pertenenciaMax –>  mayor pertenencia del dato e a cualquier agente
 8:
 9: {Algoritmo}
10: esDatoEscogido = false
11: pertenenciaMax = 0
12: {Escojo al agente con mayor pertenencia del dato}
13: para todo agente hacer
14:    p = pertenencia (e,agente)
15:    si p >pertenenciaMax entonces
16:       pertenenciaMax = p
17:       A = agente
18:    fin si
19: fin para
20: si dato e mejora al agente A entonces
21:    Actualizar el agente con e y devolver k (peor dato en la memoria)
22:    esDatoEscogido = true
23:    cD = cD ∪ {k}
24: si no
25:    cD = cD ∪ {e}
26: fin si
```

Comportamiento competitivo de los agentes respecto al dato

Si se asigna a los agentes un *comportamiento competitivo*, el sistema ofrece el dato de entrenamiento a todos los agentes y estos van a competir por él, de forma que solamente un agente será el que se quede con el dato. La competencia se puede definir en base a dos criterios: que el agente que se queda con el dato es el que ofrece mayor pertenencia del dato a su zona de especialización, o bien que el agente que se quede con el dato es el que más mejora su calidad con el dato.

El criterio de mayor pertenencia no asegura, por si mismo, que el agente de mayor pertenencia vaya a mejorar su calidad. De hecho, si el agente ya está bien formado puede rechazar entrenarse con el dato, en cuyo caso, el resto de agentes tampoco lo habrían usado. Este comportamiento se implementa en el Algoritmo 20.

Algoritmo 21 Asignación de un dato a los agentes con comportamiento competitivo, aplicando el criterio de calidad

Entrada: e, cD
Salida: cD, esDatoEscogido

```
 1:
 2: {Datos}
 3: e –>  dato de entrenamiento
 4: cD –>  conjunto de datos devueltos
 5: esDatoEscogido –>  cierto si el dato se lo queda algún agente
 6: nA –>  nº de agentes
 7: cAc –>  conjunto de agentes candidatos
 8: k –>  peor dato en la memoria
 9:
10: {Algoritmo}
11: cAc = { }
12: esDatoEscogido = false
13: para todo agente hacer
14:    si agente mejora su calidad con e entonces
15:       cAc = cAc ∪ {agente}
16:       esDatoEscogido =true
17:    fin si
18: fin para
19: si esDatoEscogido entonces
20:    A = agente del conjunto cAc cuya calidad mejora mas
21:    Actualizar el agente A con e y devolver k
22:    cD = cD ∪ {k}
23: si no
24:    cR = cR ∪ {e}
25: fin si
```

El criterio de mayor mejora de la calidad permite dirigir los datos a agentes que todavía no están bien formados. En definitiva, lo que interesa al lanzar un dato es justo entrenar a agentes que no están bien formados, por tanto, el criterio de mejora de la calidad parece adecuarse más al objetivo del entrenamiento (Algoritmo 21). Cabe notar que ambos criterios no son excluyentes, es decir, se puede pensar en la aplicación de una combinación de ambos.

Sea cual sea el criterio de competitividad, la consecuencia de este tipo de asignación es que un dato de entrenamiento solamente puede estar, en un instante dado, en un solo agente. También se cumple que la cantidad de datos en el entorno es constante, ya que cuando el dato se ofrece a los agentes estos o devuelven el propio dato, si ningún agente lo ha querido, o se devuelve el dato del agente con mayor pertenencia.

Algoritmo 22 Asignación de un dato a los agentes con un comportamiento autónomo

Entrada: e, cD
Salida: cD, esDatoEscogido

```
 1:
 2: {Datos}
 3: e –>  dato de entrenamiento
 4: cD –>  conjunto de datos devueltos
 5: esDatoEscogido –>  cierto si el dato se lo queda algún agente
 6: nA –>  nº de agentes
 7: cA –>  conjunto de agentes
 8: k –>  peor dato en la memoria
 9:
10: {Algoritmo}
11: esDatoEscogido = false
12: nA = | cA |
13: repetir
14:    si nA >0 entonces
15:       Sustraer un agente A de cA
16:       Decrementar nA
17:       si dato e mejora al agente A entonces
18:          Actualizar el agente con e y devolver k
19:          esDatoEscogido = true
20:          cD = cD ∪ {k}
21:       fin si
22:    fin si
23: hasta que esDatoEscogido o nA=0
24: si No esDatoEscogido  entonces
25:    cD = cD ∪ {e}
26: fin si
```

Comportamiento autónomo de los agentes respecto al dato

Como se ha mostrado en los dos casos anteriores, el sistema ofrece el dato a todos los agentes de forma que, o bien se queda un solo agente con el dato (competitivo), o bien se quedan k agentes con el dato (colaborativo).

Otra forma de definir el comportamiento de los agentes respecto a un dato es mantener la idea de que el agente encuentra el dato de forma fortuita, sin la intervención del resto de agentes. Este es el escenario que teníamos en el modelo basado en memoria, donde cualquier agente puede encontrar un dato, pero en un instante dado, solamente un agente tiene "conocimiento" de ese dato. En el momento en el que un agente decide tomar el dato, el resto de agentes no tiene acceso a él.

El sistema multiclasificador puede simular este comportamiento autónomo, ofreciendo secuencialmente el dato a agentes hasta que uno de los agentes lo escoja. Con este método se asegura que el dato va estar visible en el entorno hasta que algún agente lo encuentre y se lo quede. En realidad, en el peor de los casos, el sistema ofrece el dato a todos los agentes. Si ningún agente quiere el dato, este pasa al conjunto de datos devueltos; y si el dato se lo queda algún agente, este devuelve su peor dato al entorno. El Algoritmo 22 define este método.

Iteración en el entrenamiento básico

Expuestas las distintas aproximaciones que podemos seleccionar para poner a disposición de los agentes el conjunto de datos de entrenamiento, el Algoritmo 23 propone lo que entendemos por una *iteración del algoritmo en el entrenamiento básico* para el conjunto de agentes. Una iteración en el entrenamiento va a consistir en ofrecer cada dato del conjunto de entrenamiento al conjunto de agentes, de forma que, dependiendo del comportamiento asignado al conjunto de agentes, estos competirán por el dato quedándoselo el mejor agente (competitivo); se permitirá que varios agentes se queden con el dato (colaborativo); o será el primer agente al que le interese el dato el que lo elija (autónomo).

Una iteración en el entrenamiento permite, por tanto, que cada dato de entrenamiento sea usado por uno o más agentes. En general, para definir la gestión de los datos que son devueltos al entorno podemos diferenciar dos situaciones: que un dato sea elegido por algún agente o que el dato sea rechazado por todos los agentes.

Si un dato es elegido por n agentes (al menos uno), los n agentes devuelven n datos al entorno (cada agente devuelve el peor dato de su memoria). Suponiendo que entre los n datos puede haber algunos repetidos, tenemos que, por un dato que se ha introducido en el entorno se devuelven al entorno entre 1 y n. Considerando que en una iteración tenemos un conjunto

de entrenamiento cuyos datos son ofrecidos a los agentes, al finalizar la iteración tendremos un nuevo conjunto formado por los datos devueltos por los agentes y por los datos no escogidos. El algoritmo que se propone va a construir un conjunto con todos los datos devueltos en una iteración, de forma que este conjunto pasará a ser el conjunto de entrenamiento en la siguiente iteración.

Algoritmo 23 Iteración en el entrenamiento básico del conjunto de agentes

Entrada: cE
Salida: cD, mejoraAlgunAgente

```
 1:
 2: {Datos}
 3: cE ==>conjunto de datos de entrenamiento
 4: cD ==>conjunto de datos devueltos
 5: mejoraAlgunAgente ==>cierto si algún agente se queda con algún dato
 6: esDatoEscogido ==>cierto si el dato se lo queda algún agente
 7: e ==>dato de entrenamiento
 8:
 9: {Algoritmo}
10: cD = {}
11: mejoraAlgunAgente =false
12: para e=1 to número de datos en cE  hacer
13:    cD, esDatoEscogido <– Asignación de e al conjunto de agentes (e,cD)
14:    si esDatoEscogido entonces
15:       mejoraAlgunAgente = true
16:    fin si
17: fin para
```

Entrenamiento básico

El cometido de lo que se ha llamado entrenamiento básico es, partiendo de un conjunto de entrenamiento y un conjunto de agentes previamente inicializados, realzar el entrenamiento del conjunto de agentes. Este entrenamiento básico es el equivalente al entrenamiento general en los dos modelos anteriores. La realización, de forma iterativa, de lo que hemos denominado iteración del entrenamiento, provocará la adaptación emergente del conjunto de agentes.

Una iteración en el entrenamiento produce un conjunto de datos devueltos, que son datos que estaban en la memoria de los agentes o datos no elegidos por ningún agente, de forma que estos datos van a servir como nuevos datos de entrenamiento al conjunto de agentes. El entrenamiento básico va a consistir en repetir iteraciones de entrenamiento de forma que, en la iteración i, el conjunto de entrenamiento es el conjunto de datos devueltos en la iteración i-1. Para la primera iteración, el conjunto de entrenamiento será el conjunto inicial de entrenamiento.

Conforme avanza el entrenamiento, los agentes se van mejorando y, por tanto, su tendencia a coger datos disminuye, ya que la calidad de los agentes aumenta y el dato tiende a no mejorar la calidad. Esto provoca que, a medida que pasan las iteraciones, el conjunto de datos devueltos contenga, en mayor proporción, a datos que no han sido seleccionados por ningún agente.

Si en una iteración ningún dato mejora a ningún agente, el conjunto de datos devueltos será exactamente igual al conjunto de datos de entrenamiento usado en esa iteración, por lo que seguir iterando sobre el conjunto de devueltos no provocaría cambios en ningún agente. Esto es debido a que el proceso de optimización local de cada agente ha alcanzado un mínimo local. Está va a ser, por tanto, la condición de terminación del entrenamiento básico. El Algoritmo 24 describe este ciclo, donde la iteración se podrá realizar utilizando cualquiera de los tres comportamientos de agentes, definidos anteriormente para el entrenamiento con un dato.

Algoritmo 24 Entrenamiento básico del conjunto de agentes

Entrada: cE,cA
Salida: cA entrenados

```
 1:
 2: {Datos}
 3: cE "->"   conjunto de datos de entrenamiento
 4: cA ->   conjunto de agentes
 5: cD ->   conjunto de datos devueltos
 6: mejoraAlgunAgente –>   cierto si algún agente se queda con algún dato
 7:
 8: {Algoritmo}
 9: repetir
10:    (cD,mejoraAlgunAgente) = Iteración entrenamiento básico del conjunto
       de agentes (cE)
11:    cE = cD
12: hasta que No mejoraAlgunAgente
```

4.6.3. Generación de nuevos agentes para zonas no cubiertas

Cuando se realiza un entrenamiento básico de los agentes se obtiene un conjunto de agentes cuyo tamaño es igual al del conjunto creado al inicio. El conjunto obtenido mantiene a los agentes originales, con la característica de que los agentes entrenados cumplen unas pautas de calidad y están especializados en diferentes zonas del dominio de datos.

Aunque la inicialización de los agentes se produce de forma aleatoria, intentando así cubrir todo el dominio, es posible que después del entrenamiento hayan quedado zonas sin cubrir. Esto es debido a que los agentes se van especializando en zonas más pequeñas y a que pueden existir agen-

tes especializados en zonas parecidas, sin poderse garantizar que todas las zonas están cubiertas por al menos un agente.

El objetivo, una vez realizado el entrenamiento básico, es detectar las zonas no cubiertas por los agentes y generar nuevos agentes que las cubran, de forma que el nuevo conjunto de agentes estará formado por los agentes del entrenamiento básico y los agentes nuevos generados para las zonas no cubiertas.

Detección de zonas no cubiertas

Una forma de detectar las zonas que no están cubiertas por los agentes es aplicar un conjunto de test al conjunto de agentes generado. La clasificación obtendrá dos subconjuntos del conjunto de test: el conjunto de datos mal clasificados y el conjunto de los datos bien clasificados. El conjunto de los datos mal clasificados va a representar a zonas del dominio que no están siendo cubiertas por el conjunto de agentes y, por tanto, es un buen candidato para ser el conjunto de entrenamiento para un nuevo conjunto de agentes.

Debemos decidir cual va a ser el conjunto de test que se va a aplicar para encontrar las zonas no cubiertas. Una primera opción puede ser coger el conjunto de test definido en todo proceso de aprendizaje. Los datos disponibles para el aprendizaje deben ser divididos en dos subconjuntos: el conjunto de entrenamiento y el conjunto de test. Esta separación es importante para garantizar la independencia entre el proceso de aprendizaje y el uso del clasificador, ya que si se usaran datos de entrenamiento para el test, la evaluación de la respuesta del clasificador no sería independiente y, seguramente, sería mejor al haberse entrenado con esos datos.

Ya que el test que se quiere hacer en esta fase corresponde todavía al entrenamiento, no debemos usar los datos del conjunto de test original, para no mezclar unos datos con otros. Por este motivo se propone usar el conjunto de datos devueltos en el entrenamiento básico.

En la última iteración del entrenamiento básico, los agentes han mejorado su calidad, y han devuelto todos los datos de entrenamiento de esa iteración. Estos pueden ser datos que, aunque pertenezcan a la zona de algún agente, no lo mejoran o bien datos que no tienen a ningún agente que los represente. Si aplicamos el test al conjunto de datos devueltos, producido después del entrenamiento básico, obtendremos un conjunto de datos bien clasificados, normalmente los que sí pertenecían a la zona de algún agente, y un conjunto de datos mal clasificados, que posiblemente serán los datos no cubiertos por ningún agente.

En realidad, el conjunto de entrenamiento original también sirve de test. El objetivo de detectar zonas no cubiertas se plantea como la

detección de subdominios donde el conjunto de clasificadores generados en el entrenamiento básico no clasifica bien. Por tanto, cualquier dato del conjunto original que no sea bien clasificado es un elemento del conjunto de entrenamiento para la detección de zonas no cubiertas.

Generación de los nuevos agentes

El problema de generar nuevos agentes para las zonas no cubiertas es el mismo problema general de generar agentes a partir de un conjunto de entrenamiento, donde el conjunto de datos mal clasificados forma el conjunto de entrenamiento para el nuevo conjunto de agentes.

En este sentido, la obtención del conjunto de datos mal clasificados permite crear e inicializar un nuevo conjunto de agentes (Algoritmo 18) y aplicar un entrenamiento básico a este conjunto (algoritmo 24). El resultado de este entrenamiento será un conjunto de agentes especialistas en los datos mal clasificados, ya que son los datos de entrenamiento del nuevo conjunto de agentes.

Al igual que ocurría en el entrenamiento inicial, es posible que los agentes generados no cubran a todos los datos mal clasificados (los datos de su conjunto de entrenamiento). Para solucionar esta cuestión se va a repetir el proceso, es decir, se va a aplicar un test sobre los datos devueltos y se va a volver a obtener un nuevo conjunto de agentes especialistas en los datos mal clasificados por el test. Si repetimos la generación de nuevos agentes sobre los datos mal clasificados de la fase anterior, el cardinal del conjunto de datos mal clasificados tiende cero. Por tanto, la condición de parada de este ciclo será que el conjunto de mal clasificados sea vacío.

El resultado de aplicar este ciclo es que, en cada iteración, se genera un conjunto de agentes especialistas en los datos mal clasificados por conjunto de agentes de la iteración anterior, por lo que obtenemos c conjuntos de agentes. La unión de todos los conjunto de agentes generados va a formar lo que llamamos el conjunto de agentes especialistas en las zonas no cubiertas del conjunto inicial de agentes.

Si el algoritmo realiza k iteraciones, obtendremos k conjuntos de agentes, por lo que, si llamamos a cada conjunto de agentes cA_i y al conjunto total de agentes cAzonas, este se define como:

$$cAzonas = cA_1 \cup ... \cup cA_k \tag{4.11}$$

Una vez generado el conjunto de agentes para zonas no cubiertas (cAzonas), el sistema multiclasificador dispone de un conjunto total de agentes formado por la unión del conjunto de agentes del entrenamiento inicial (cA) con el conjunto recién generado (cAzonas) (Ecuación 4.12). El

Algoritmo 25 detalla este método de detección de zonas no cubiertas.

$$cA = cA \cup cAzonas \tag{4.12}$$

Algoritmo 25 Generación de agentes para zonas no cubiertas

Entrada: cE,cA
Salida: cAzonas

```
 1:
 2: {Datos}
 3: cE –>  conjunto de entrenamiento para zonas no cubiertas
 4: cA –>  conjunto de agentes inicial entrenado en la fase anterior
 5: cAzonas –>  conjunto de agentes total para las zonas no cubiertas
 6: cAi –>  conjunto de agentes generados en una iteración
 7: cBC –>  conjunto de datos bien clasificados
 8: cMC –>  conjunto de datos mal clasificados
 9:
10: {Algoritmo}
11: cAzonas = {}
12: (cBC,cMC) = Clasificación Test (cE,cA)
13: mientras |cMC| >0 hacer
14:    cAi = Creación de agentes con (cMC)
15:    (cAi,cD) = Entrenamiento básico (cAi,cMC)
16:    (cBC,cMC) = Clasificación Test (cD,cAi)
17:    cAzonas = cAzonas ∪ cAi
18: fin mientras
```

4.6.4. Reducción del conjunto agentes

Después de la ejecución de un entrenamiento inicial y de la generación de agentes para zonas no cubiertas, el sistema multiclasificador dispone de un conjunto de agentes (cA) que cubre toda la zona del dominio de datos.

Las propiedades del sistema, respecto al uso de diferentes especialistas para clasificar un dato, permiten gestionar que varios agentes sean especialista en zonas parecidas. En este sentido, el sistema multiclasificador puede elegir diferentes especialistas y combinar sus resultados.

En esta sección se plantea la reducción del conjunto de agentes, detectando los agentes que son similares entre si y eligiendo un representante. El objetivo de reducir el conjunto de agentes es poder clasificar con la misma efectividad pero con un menor número de agentes, lo que hará que la clasificación sea más rápida y consuma menos recursos.

Para ello, vamos a definir dos relaciones de similitud entre dos agentes, atendiendo a dos criterios distintos, de forma que, mediante la agregación

de ambas, se establecerá la similitud entre los agentes. Una vez que tenemos definida una relación de similaridad entre dos agentes, vamos a construir una relación de equivalencia, para formalizar el concepto de clase de equivalencia respecto a esa relación. Cada clase de equivalencia va a contener a todos los agentes que sean equivalentes entre si, cumpliéndose la relación de equivalencia entre cualquier par de agentes de la clase.

Definición de la relación de similitud entre dos agentes

Dos agentes son similares si las zonas en la que son especialistas son parecidas y/o si sus clasificaciones están correlacionadas, es decir, si clasifican de forma parecida a un conjunto de datos.

Vamos a definir dos relaciones de similitud aplicables a un par de agentes: similitud respecto a la zona de especialización y similitud respecto a la clasificación, obteniendo la similitud entre dos agentes como una agregación de ambas.

Similitud respecto a la zona de especialización

El entrenamiento de los agentes se realiza para generar agentes especialistas en subzonas del dominio, de forma que los agentes que intervienen en la clasificación son aquellos especialistas en la zona del dato a clasificar. Esta propiedad causa que aquellos agentes especialistas en la misma zona puedan tener un comportamiento similar al clasificar un dato de la zona.

Ya que la zona de un agente está representada por su prototipo general, la similitud de dos agentes A_i y A_j se puede definir en relación a la cercanía de sus prototipos. Si Usamos una medida de distancia entre dos prototipos, su similitud ($similZ_ij$) se puede calcular como:

$$similZ_ij = 1 - \frac{dist(P_{A_i}, P_{A_j})}{dMax} \tag{4.13}$$

donde $dist(P_{A_i}, P_{A_j})$ es la distancia entre los dos prototipos y $dMax$ es la distancia máxima entre los datos del agente i y los datos del agente j. La división de estos valores ofrece una distancia normalizada en el intervalo [0,1] sobre la distancia máxima de los dos agentes. Como el concepto de similitud representa a lo inverso de la distancia [106], la similitud (similZ_ij) entre dos agentes A_i y A_j se va a calcular como uno menos la distancia normalizada.

Sea Rz la relación de similitud difusa respecto a la zona de dos agentes A_i y A_j. Diremos que dos agentes (A_i, A_j) cumplen la relación de similitud $A_i Rz A_j$ en un grado $similZ_{ij}$.

El método descrito para obtener la similitud entre dos agentes respecto a su zona se presenta en el Algoritmo 26.

Algoritmo 26 Similitud de dos agentes respecto a su zona

Entrada: A_i, A_j
Salida: similZ_ij

{Datos}
A_i, A_j –> agentes
similZ_ij –> similitud entre el agente i y el j
M_{A_i} –> memoria del agente i
dMax –> distancia máxima entre un dato del agente i y otro dato del agente j

{Algoritmo}
dMax=0
para todo x $\in M_{A_i}$ **hacer**
  para todo y $\in M_{A_j}$ **hacer**
    si dist (x,y) >dMax **entonces**
      dMax = dist (x,y)
    fin si
  fin para
fin para
$similZ_ij = 1 - \frac{dist(P_{A_i}, P_{A_j})}{dMax)}$

Similitud respecto a la clasificación.

Fijado un criterio de similitud sobre las zona de especialización de dos agentes, se va a establecer un criterio de similitud basado en su forma de actuar, es decir, respecto a como clasifican. Los criterios de zona y de clasificación cubren los dos ámbitos planteados: el estructural y funcional de los agentes.

Atendiendo al criterio funcional, dos agentes actúan de forma similar si clasifican de igual forma los mismos datos. Si dos agentes obtienen la misma respuesta frente al mismo conjunto de datos, podemos eliminar del sistema a uno de ellos, por redundante, sin perder capacidad de clasificación en el sistema multiclasificador.

En general, podemos pensar que cualquier conjunto de datos de entrenamiento podría servir para comparar las clasificaciones de los agentes. En la práctica, un agente solo va a ser usado para clasificar un dato si el dato pertenece a su zona. Por tanto, parece lógico pensar en un conjunto de datos que pertenezcan a las zonas de ambos agentes. Un conjunto que cumple esta propiedad es el conjunto de datos formado por la unión de las memorias de cada agente. Este conjunto también tiene la propiedad de ser relativamente

pequeño, con lo que el proceso de cálculo de las similitudes se acorta. A su vez, es muy representativo de las zonas de los agentes, e independiente de cualquier elemento ajeno a los mismos.

Si la memoria del agente A_i y del agente A_j es respectivamente:

$$M_{A_i} = \{d_{i,1} \dots d_{i,n_i}\} \tag{4.14}$$

$$M_{A_j} = \{d_{j,1} \dots d_{j,n_j}\} \tag{4.15}$$

El conjunto de datos para calcular la similitud ($cDatosSim$) tiene $n_i * n_j$ datos, siendo n_i y n_j el tamaño de la memoria de los agentes i y j, y será la unión de la dos memorias:

$$cDatosSim = \{d_{i,1} \dots d_{i,n}, d_{j,1} \dots d_{j,n}\} \tag{4.16}$$

El algoritmo propuesto va efectuar la clasificación de cada dato de este conjunto sobre los dos agentes, y va a evaluar las diferencias en cada clasificación.

La clasificación de un dato por un agente ha sido definida en la Sección 4.5.1, como una distribución de posibilidad sobre las clases,

$$C_{A_i}(\vec{x}) = [(c_1, p_{i,1}), ..., (c_z, p_{i,z})] \tag{4.17}$$

cumpliéndose que la suma de las posibilidades es uno,

$$\sum_{i=1}^{z} p_i = 1$$

Si clasificamos un dato $\vec{x}$ del conjunto $cDatosSim$ con los agentes A_i y A_j, vamos a obtener dos distribuciones de posibilidad, que serán la predicción de cada agente para cada una de la clases:

$$C_{A_i}(\vec{x}) = [(c_1, p_{i,1}), ..., (c_z, p_{i,z})] \tag{4.18}$$

$$C_{A_j}(\vec{x}) = [(c_1, p_{j,1}), ..., (c_z, p_{j,z})] \tag{4.19}$$

La diferencia dif_k, en valor absoluto, entre la posibilidad de una clase k del agente i y del agente j ofrece el grado de disimilitud entre las predicciones de ambos agentes para esa clase.

$$dif_k = |p_{i,k} - p_{j,k}| \tag{4.20}$$

Si la diferencia es cercana a cero, los agentes son similares para ese dato en la clase k, por lo que la similitud para la clase k va estar definida como uno menos la diferencia entre las posibilidades.

$$sim_k = 1 - dif_k \tag{4.21}$$

Aplicando esta ecuación a cada clase, vamos a obtener una distribución de posibilidad de similitud ($cSim(\vec{x})$) con las similitudes en cada clase para el dato $\vec{x}$:

$$cSim(\vec{x}) = [(c_1, sim_1), ..., (c_z, sim_z)] \tag{4.22}$$

En este punto, debemos decir si los agentes tienen una clasificación similar para el dato y obtener un único valor para esa similitud. Existen diferentes medidas que se pueden aplicar para decidir cual es la similitud global para el dato. Para la agregación de toda la información en un único valor utilizaremos una T-norma cualquiera. En este caso utilizamos la T-norma del mínimo [106] para seleccionar como similitud total la similitud más pequeña entre todas las clases. Esta elección implica que nos quedamos con el peor caso.

Definiendo la similitud entre los agentes A_i y A_j para el dato $\vec{x}$ como:

$$sim(\vec{x}) = min(cSim(\vec{x})) \tag{4.23}$$

La clasificación de los agentes A_i y A_j se va a producir sobre todos los datos del conjunto $cDatosSim$, por lo que obtendremos el conjunto con las similitudes en la clasificación para cada dato $simil(cDatosSim)$

$$simil(cDatosSim) = [sim(x_1), ..., sim(x_{n_i+n_j})] \tag{4.24}$$

Este conjunto ofrece todas las similitudes para cada dato, que agregamos en un único valor que represente a la similitud entre los dos agentes.

Nuevamente podemos optar por aplicar un mecanismo estricto o relajado de elección de la similitud entre las similitudes del conjunto. Seguiremos con el criterio estricto, de forma que se define la similitud entre dos agentes respecto a la clasificación $similC_ij$ como el mínimo entre las similitudes obtenidas para cada dato.

$$similC_ij = min(simil(cDatosSim)) \tag{4.25}$$

Llamaremos Rc a la relación de similitud difusa respecto a la clasificación, ($A_i Rc A_j$), y diremos que dos agentes cumplen la relación de similitud con un grado $similC_{ij}$. El método descrito para obtener la similitud entre dos agentes se presenta en el Algoritmo 27.

Algoritmo 27 Similitud de dos agentes respecto a la clasificación

Entrada: A_i, A_j
Salida: similC_ij

{Datos}
A_i, A_j –> agentes
rc –> cierto si cumple la relación, falso en caso contrario
M_{A_i} –> memoria del agente i
cDatosSim –> conjunto con los datos de la memoria de cada agente
cSim($\vec{x}$) –> conjunto con la similitud para cada clase en el dato $\vec{x}$
simil –> conjunto con la similitudes obtenidas para cada dato
similC_ij –> similitud entre los agentes A_i, A_j

{Algoritmo}
cDatosSim = $M_{A_i} \cup M_{A_j}$
$simil = \emptyset$
para todo $\vec{x} \in$ cDatosSim **hacer**
  Clasificar $\vec{x}$ con el agente A_i ($C_{A_i}(\vec{x})$)
  Clasificar $\vec{x}$ con el agente A_j ($C_{A_j}(\vec{x})$)
  Obtener el conjunto de similitudes para cada clase ($cSim(x)$)
  Obtener la similitud general ($sim(x)$) como el mínimo del conjunto $cSim$
  Añadir la similitud de $\vec{x}$ al conjunto de similitudes ($simil = simil \cup \{sim(x)\}$)
fin para
Obtener la similitud general $similC_ij$ como el mínimo del conjunto $simil$

Agregación de ambas similitudes.

Definidas las relaciones de simililaridad Rz y Rc, se define una nueva relación de similitud (Rzc) que expresa cuando dos agentes cumplen las dos relaciones anteriores, es decir, que sean similares respecto a su zona y respecto a su clasificación. Diremos que dos agentes cumplen la relación Rzc en un grado $similZC_ij$, definido como la similitud entre dos agentes ($similZC_ij$). Se usará, de nuevo, de una T-norma, como es el mínimo entre la similitud respecto a la zona y la similitud respecto a la clasificación.

$$similZC_ij = min(similZ_ij, similC_ij) \tag{4.26}$$

El mínimo de las dos similitudes representa al peor de los casos, por lo que estamos siendo estrictos en la obtención de la valoración.

Partición del conjunto de agentes mediante la relación de similitud (clases de equivalencia)

Fijada la relación difusa de similitud Rzc entre dos agentes A_i y A_j, será esta la que va a configurar una relación de equivalencia final. Mediante el corte por un valor mínimo de similitud (Us), definiremos la relación de equivalencia R que permitirá agrupar los agentes.

Sea la relación de equivalencia R entre los agentes A_i y A_j definida como:

$$A_i R A_j = \begin{cases} cierto & \text{si } A_i Rzc A_j > Us \\ falso & \text{en otro caso} \end{cases} \tag{4.27}$$

Nuestro objetivo es obtener las clases de equivalencia sobre esa relación, es decir, obtener el cierre de la relación. Una clase de equivalencia va a contener a todos los agentes que son equivalentes entre si, por lo que podremos resumir cada conjunto de agentes que pertenezcan a la misma clase en un único agente.

Una clase de equivalencia sobre una relación R (C_h/R) contiene un conjunto de agentes que verifican las siguientes propiedades:

1. Si $A_i R A_j$ entonces A_i y A_j están en la misma clase (C_h/R) y no están en ninguna otra clase.
2. Si $A_i R A_j$ y $A_j R A_k$ entonces $A_i R A_k$.

Se cumple, además, que cualquier agente A_i cumple la relación R consigo mismo, de forma que:

$$A_i R A_i, \quad \forall A_i \tag{4.28}$$

Componiendo dicha relación consigo misma, de forma reiterada, se obtiene el cierre de la misma R^*, que definirá el conjunto de clases de equivalencia existentes. La clase de equivalencia h contiene a los k agentes que cumplen la relación:

$$C_h/R = \{A_1, \ldots, A_k\} \tag{4.29}$$

El método que obtiene todas las clases de equivalencia es descrito en el Algoritmo 28. Si tenemos f clases de equivalencia, llamaremos C al conjunto formado por todas las clases de equivalencia y C será:

$$C = \{C_1/R, \ldots, C_f/R\} \tag{4.30}$$

Algoritmo 28 Obtención de las clases de equivalencia

Entrada: C_Rzc
Salida: C

```
1:
2: {Datos}
3: C_Rzc –> conjunto con los pares de agentes que cumplen la relación
4: C –> conjunto con las clases de equivalencia obtenidas
5: cNoUsadas –> conjunto que contiene a las clases de equivalencia que no
   contienen ningún agente contenido en la clase c_i en una iteración
6: HayCambios –> es cierto si se ha modificado alguna clase en la iteración
7:
8: {Algoritmo}
9: C = C_Rzc
10: repetir
11:   para todo c_i ∈ C hacer
12:     cNoUsadas = ∅
13:     para todo c_j ∈ C and c_i ≠ c_j hacer
14:       si c_i ∩ c_j ≠ ∅ entonces
15:         c_i = c_i ∪ c_j
16:       si no
17:         cNoUsadas = cNoUsadas ∪ {c_j}
18:       fin si
19:     fin para
20:   fin para
21:   si Cardinal (cNoUsadas ∪ {c_i}) = Cardinal (cNoUsadas ∪ {c_i}) entonces
22:     HayCambios = cierto
23:   si no
24:     HayCambios = falso
25:   fin si
26:   C = cNoUsadas ∪ {c_i}
27: hasta que No hayCambios
```

El algoritmo parte de un conjunto con los pares de agentes que cumplen la relación. Este conjunto de pares se puede interpretar como el conjunto de clases de equivalencia inicial, donde cada clase está compuesta por los dos agentes que la relacionan. El algoritmo debe recorrer los pares relacionados y formar una única clase de equivalencia para aquellas parejas que tienen algún agente en común, de forma que la clase contenga a todos los agentes de esas parejas.

El nuevo conjunto de clases de equivalencia, con más agentes por clase y menos clases de equivalencia, constituye un nuevo conjunto al que aplicar el criterio general: generar una única clase para las clases del conjunto que

tengan algún elemento en común. Si repetimos este ciclo, ocurrirá, en alguna iteración, que el conjunto actual de clases no cambia. Esta circunstancia define la terminación de la generación de las clases de equivalencia.

Hemos partido de las parejas de agentes que cumplen la relación porque, aplicando la transitividad de la relación, si A_iRA_j y A_jRA_k entonces A_iRA_k, con lo que podemos obtener una clase de equivalencia con los agentes A_i, A_j y A_k. Cabe notar que es posible que haya agentes que no estén relacionados con ningún agente. En este caso, cada agente en esta situación constituye una clase de equivalencia formada únicamente por el agente.

Si llamamos $cUnicos$ al conjunto de clases formadas por un solo agente, el conjunto final de clases ($clasesEq$) estará formado por el conjunto C, devuelto por el algoritmo de generación de clases a partir de las parejas de agentes relacionados, unido al conjunto $cUnicos$,

$$clasesEq = cUnicos \cup C \tag{4.31}$$

Reducción del conjunto de agentes usando las clases de equivalencia.

Una vez conseguido el conjunto $clasesEq$, que contiene las clases de equivalencias y donde cada clase incluye a agentes equivalentes, se plantea la reducción del conjunto de agentes mediante la elección de un representante de cada clase, que será incluido en el conjunto de agentes final.

La elección del representante puede plantearse como la obtención de un nuevo agente con la información resumida de los agentes de la clase o como la elección de un agente entre los agentes de la clase. Se ha optado por la elección de un representante al azar de la clase ya que todos son equivalentes entre si.

El conjunto final de agentes, que llamaremos H, se obtiene tras calcular el conjunto con las clases de equivalencia entre los agentes y seleccionando un agente por cada una de ellas.

Si $clasesEq$ contiene n_A clases ($clasesEq = \{C_1 \dots C_{n_A}\}$), el conjunto H contiene a n_A agentes, cumpliéndose que cada agente pertenece a una clase del conjunto $clasesEq$

$$H = \{A_1 \dots A_{n_A}\} \text{ tal que } A_i \in C_i \tag{4.32}$$

4.7. Multiclasificador asociado

La generación del conjunto de agentes completa el entrenamiento del sistema multiclasificador, de forma que se dispone de un conjunto de agentes especialistas en zonas del dominio. El uso del sistema para clasificar un dato consiste en seleccionar a los mejores expertos para el dato y combinar las predicciones de estos expertos para ofrecer una predicción final.

Si llamamos g a la función que selecciona a los mejores agentes para un dato $\vec{x}$; H es el conjunto de agentes generados; y $\vec{x}$ es el dato que queremos clasificar, la función g va a seleccionar a los k mejores agentes del conjunto H para el dato $\vec{x}$, obteniendo un conjunto H^* ($H^* = \{A_1, ..., A_k\}$), donde el valor de k será calculado por la función g. El método para seleccionar a los k mejores agentes se desarrolla en la Sección 4.7.1.

$$g(\vec{x}) = H^*$$

La clasificación de un dato por el sistema multiclasificador (función Smc) se produce como mecanismo de agregación (función $Comb$) de las clasificaciones parciales de cada agente seleccionado. La combinación de las soluciones de los k agentes para obtener la clasificación final de define en la Sección 4.7.2.

$$Smc(\vec{x}) = Comb(C_{A_1}(\vec{x}), \ldots, C_{A_k}(\vec{x}))$$

4.7.1. Selección de los mejores agentes para un dato

En los modelos anteriores, la función de selección de los expertos requiere de la elección de un valor k, que representa al número de expertos que van a utilizarse para clasificar el dato. La elección del valor de k es una tarea crítica debido a que un k no adecuado puede hacer que no se estén usando los agentes más adecuados. Si k es demasiado grande se puede estar incluyendo a agentes poco expertos y si k es demasiado pequeño es posible que estemos dejando fuera de la clasificación a agentes expertos en la zona. Aunque la predicción de cada agente afecta de forma diferente en la clasificación final, dependiendo de su adecuación a la zona, lo cierto es que la inclusión o no inclusión de agentes apropiados afecta de alguna forma a la predicción final.

En el modelo que se presenta, se define una forma de seleccionar a los mejores agentes basada en la generación de valores de una distribución de posibilidad de pertenencia del dato a la zona de los agentes, y una selección de los mismos mediante una técnica de ruleta, utilizada en los algoritmos genéticos [38]. En este sentido, se va a cambiar la idea de elegir a los

k mejores agentes por la idea de: "*elegir el subconjunto de agentes cuya pertenencia acumulada del dato a esos agentes sea menor a un porcentaje*".

De forma resumida, se puede definir el método como sigue:

- Normalizar las pertenencias del dato a cada agente.
- Ordenar las pertenencias de mayor a menor.
- Obtener el valor máximo de k que hace que la suma de las mayores pertenencias normalizadas sea menor que el porcentaje dado.

Esta nueva definición causa que el número de agentes (k) seleccionados sea diferente cada vez que se clasifica un dato. La consecuencia de esta elección, respecto al número de agentes seleccionados, es que si los mejores agentes tienen una alta pertenencia del dato a ellos, el número de agentes seleccionados es menor, debido a que con menos agentes se logra sumar el porcentaje establecido. En caso contrario, se necesitan más agentes para llegar al porcentaje. Vamos a desarrollar a continuación los conceptos descritos anteriormente.

Pertenencia del dato a cada agente

La pertenencia de un dato a la zona del agente (p_{A_i}) va a ofrecer una medida de la adecuación del agente para clasificar el dato. Ya que el prototipo de un agente resume la información de la zona en un punto, la distancia entre el dato y el prototipo es una medida de la adecuación del dato a la zona del agente, de forma que, la pertenencia de un dato a la zona es mayor cuanto menor sea la distancia al prototipo. Si usamos una distancia como la Euclídea (Ecuación 4.5), podemos definir la pertenencia como el valor inverso a la distancia,

$$p_{A_i} = \begin{cases} 1, & \text{si } d(\vec{x}, P_{A_i}) = 0 \\ \frac{1}{d(\vec{x}, P_{A_i})}, & \text{si } d(\vec{x}, P_{A_i}) > 0 \end{cases} \tag{4.33}$$

Si calculamos la pertenencia del dato a cada agente y tenemos n_A agentes, obtendremos una lista P con todas las pertenencias.

$$P = [p_{A_1}, \ldots, p_{A_{n_A}}] \tag{4.34}$$

Las pertenencias calculadas anteriormente son absolutas, es decir, son independientes unas de otras. Si queremos que el valor de cada pertenencia del dato al agente esté relacionado con las pertenencias de dato al resto de agentes, podemos normalizarlas calculando la pertenencia relativa a cada agente.

Si llamamos $totP$ a la suma de las pertenencias absolutas de todos los agentes, la pertenencia relativa de cada agente (pr_{A_i}) será la pertenencia absoluta dividida entre el total.

$$totP = \sum_{i=1}^{n_A} p_{A_i} \tag{4.35}$$

$$pr_{A_i} = \frac{p_{A_i}}{totP} \tag{4.36}$$

La pertenencia relativa pr_{A_i} ofrece un valor proporcionado con el resto de pertenencias, de forma que la suma de todas las pertenencias relativas es uno.

$$\sum_{i=1}^{n_A} pr_{A_i} = 1 \tag{4.37}$$

Definimos una lista (Pr) con la pertenencia relativa de cada agente:

$$Pr = [pr_{A_1}, \ldots, pr_{A_{n_A}}] \tag{4.38}$$

Pertenencia acumulada

Los mejores agentes para un dato van a ser los agentes asociados a las pertenencias más altas (pr_{A_i}). Vamos a obtener una nueva lista Pr_{ord} con los elementos ordenados de Pr de mayor a menor, de forma que los primeros agentes de la lista serán los mejores candidatos para clasificar el dato.

Para obtener la posición en la lista ordenada de la pertenencia de cada agente A_i, definimos una función π que permuta un índice i por un índice j de la siguiente forma:

Dado un índice $i \in [1, n]$, que hace referencia al agente i, definiremos una permutación $\pi(i) = j$, donde $j \in [1, n]$. Si el índice i hace referencia a la pertenencia relativa del agente i, el valor j indica la posición del valor de pertenencia relativa del agente i dentro de la lista ordenada Pr_{ord}. Esta función cumple:

- Si la pertenencia relativa del agente A_i es mayor que la del agente A_j ($pr_{A_i} > pr_{A_j}$) entonces se cumple que $\pi(i) < \pi(j)$.
- Si $\pi(i) = j$, la función inversa $\pi^{-1}(j) = i$

La función inversa ($\pi^{-1}(j) = i$) tiene como argumento a un índice j de la lista ordenada y devuelve un índice i que corresponde al agente i, es decir,

dada una posición de la lista ordenada, nos dice a qué agente hace referencia el valor de esa posición. Siguiendo esta notación, la lista con las pertenencias ordenadas es Pr_{ord}:

$$Pr_{ord} = [pr_{\pi^{-1}(1)}, \ldots, pr_{\pi^{-1}(n_A)}] \tag{4.39}$$

La lista Pr_{ord} contiene las pertenencias ordenadas de mayor a menor, por tanto, los primeros agentes de la lista son los mejores candidatos para clasificar el dato. Nuestro objetivo es determinar cuantos agentes (k) vamos a seleccionar, de forma que, los k agentes seleccionados serán los k primeros agentes de la lista.

Dado que la suma de las pertenencias relativas es uno, la suma de los k primeros elementos de la lista Pr_{ord} es la suma acumulada de las pertenencias de los k mejores agentes. El valor acumulado para un k refleja la proporción de pertenencia que aportan los k mejores agentes en su conjunto.

Podemos establecer un valor de corte Uk, de forma que el valor acumulado más cercano al corte va a representar a los agentes seleccionados. Para ello, vamos a definir una lista con las pertenencias acumuladas (Pr_{acum}) a partir de la lista de pertenencias ordenadas Pr_{ord}, de forma que la posición i de la lista va a contener a la suma de las pertenencias de las i primeras posiciones de la lista Pr_{ord}:

$$Pr_{acum} = [pa_1, \ldots, pa_{n_A}] \tag{4.40}$$

donde

$$pa_i = \sum_{j=1}^{i} pr_{\pi^{-1}(j)} \tag{4.41}$$

Selección de agentes

Dada la lista de pertenencias acumuladas (Pr_{acum}) y un umbral de selección (Uk), obtendremos el número de agentes que intervienen en la clasificación de la siguiente forma:

$$k = max_k(pa_k < Uk) \tag{4.42}$$

El valor k es el mayor índice de la lista de pertenencias acumuladas que cumple que el valor de la pertenencia en ese índice (pa_k) es menor que el valor de corte para la selección (Uk).

El conjunto con los k mejores agentes para clasificar el dato va a elegir a los k primeros agentes representados en las k primeras posiciones de la lista de acumulados. Si definimos la función π sobre la lista de acumulados, donde $\pi(i) = j$ tendremos:

$$H^* = \{A_{\pi^{-1}(1)}, \dots A_{\pi^{-1}(k)}\} \tag{4.43}$$

donde $\pi^{-1}(i) = j$ representa al agente j que ocupa la posición i en la lista ordenada de pertenencias relativas Pr_{ord}, como ya hemos comentado.

En la Figura 4.3 se muestra un ejemplo de la obtención de los k mejores agentes. Se puede observar como las pertenencias acumuladas de los primeros agentes acumulan la mayor parte de la pertenencia total. Esto es debido a que los primeros agentes son los más cercanos al dato. En el ejemplo, un valor de corte igual a 0,8 obtiene un valor de k igual a 2. Esto significa que el valor más cercano al valor de corte lo produce la suma de las pertenencias relativas de los dos primeros agentes.

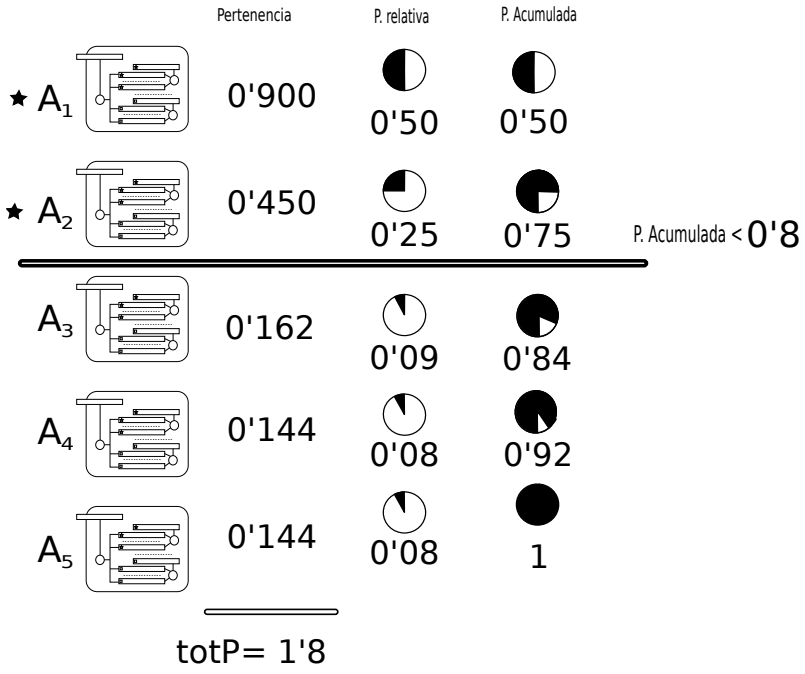

Figura 4.3: Selección de dos agentes para un valor de corte de 0,8

Si observamos el ejemplo de la Figura 4.4, el número de agentes k para el mismo valor de corte aumenta. Esto es debido a que hay más agentes con una pertenencia alta del dato, por lo que sus pertenencias relativas son menores que en el caso anterior y, por tanto, para llegar al nivel de corte necesitamos más agentes (k=3).

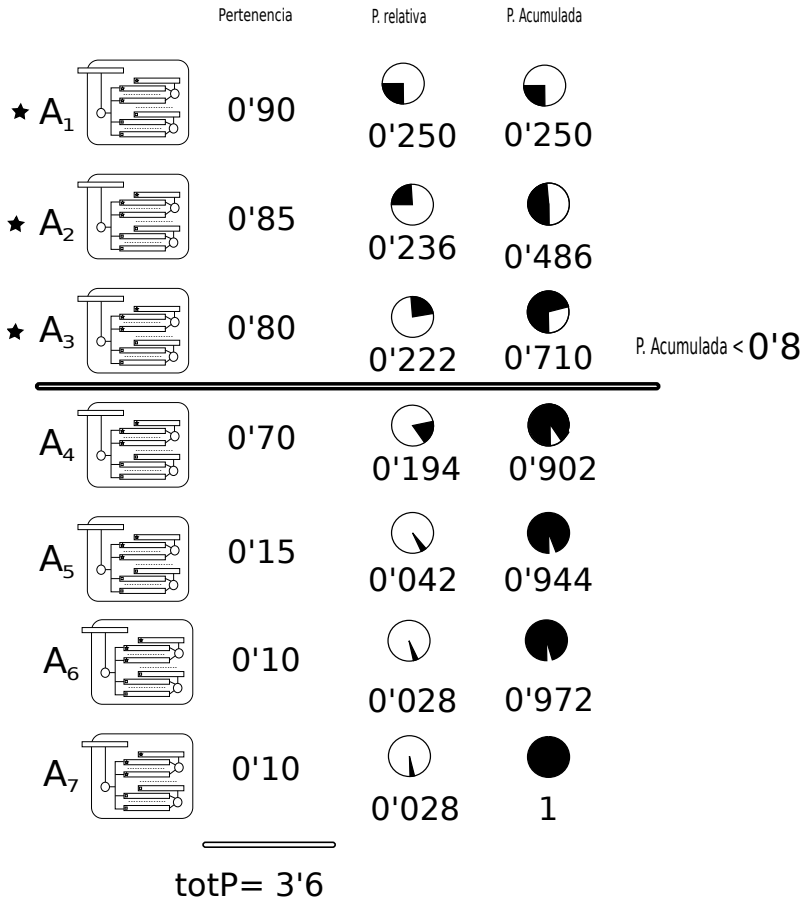

Figura 4.4: Selección de tres agentes para un valor de corte de 0,8

4.7.2. Combinación de los mejores agentes

La clasificación de un dato producida por el sistema multiclasificador va a estar definida por un método que combina las clasificaciones de cada uno de los k agentes seleccionados. Construiremos una operador de agregación, tipo OWA [104], con las distintas informaciones que proporcionan los distintos agentes. En la Sección 4.3 definimos la clasificación de un agente como una distribución de posibilidad sobre las clases, donde la posibilidad de una clase está asociada a la similitud del dato o bien al prototipo de la clase o bien al dato más cercano en la clase.

$$C_{A_i}(\vec{x}) = [(c_1, p_{i,1}), ..., (c_z, p_{i,z})]$$

La clasificación del sistema multiclasificador va a tener la misma

estructura que la clasificación de un agente, de forma que el sistema va a ofrecer la posibilidad para cada clase de que esa clase sea la que corresponda al dato. Dado que para cada clase el sistema obtiene k posibilidades (una por cada agente), la posibilidad global de una clase debe ser resultado de la combinación y agregación de estas posibilidades.

Para combinar los posibilidades de una clase se debe decidir cual va a ser la influencia de la predicción de cada agente. Una media para ponderar la influencia de un agente es su adecuación al dato. De hecho, esta medida es la que se ha usado para seleccionar a los k agentes. Por tanto, vamos a definir la posibilidad de una clase como la suma de las k posibilidades ponderadas por la importancia del agente (Ecuación 4.46).

La importancia de un agente se va a calcular como la pertenencia del dato al agente. En este caso también interesa normalizar la pertenencia para que su suma sea uno. Como vamos a usar los k mejores agentes, la pertenencia relativa (pr_i) de cada agente i se calcula como:

$$pr_i = \frac{p_i}{totPk} \tag{4.44}$$

siendo p_i la pertenencia absoluta del agente i y $totPk$ la suma de las pertenencias absolutas de los k agentes. Si consideramos, en lugar de los valores absolutos de pertenencia p_i, los valores $pr_{\pi(i)^{-1}}$ calculados anteriormente en la Ecuación 4.39 y, en lugar de $totPk$, el valor acumulado pa_k para $i = k$ según la Ecuación 4.41, la pertenencia relativa es:

$$pr_i = \frac{p_{\pi(i)^{-1}}}{pa_k} \tag{4.45}$$

Siendo p_{ij} la posibilidad para la clase j ofrecida por el agente i, la posibilidad para una clase j del sistema para los k agentes se define como:

$$p_j = \sum_{i=1}^{k} p_{ij} * pr_i \tag{4.46}$$

El sistema multiclasificador debe obtener las posibilidades p_j para cada clase. La posibilidad p_j es calculada sin tener en cuenta las posibilidades del resto de clases. Esto significa que el valor de una posibilidad no está proporcionado respecto al valor del resto de posibilidades. Para lograr esta proporcionalidad, vamos a normalizar, como en el caso de la clasificación de un agente, las posibilidades para que su suma sea uno.

Si llamamos pos_j a la nueva posibilidad para la clase j normalizada y $totPosk$ es la suma de las posibilidades de las z clases, definimos pos_j como:

$$pos_j = \frac{p_j}{totPosk} \tag{4.47}$$

donde

$$totPosk = \sum_{j=1}^{z} p_j \tag{4.48}$$

La clasificación del sistema multiclasificador obtendrá la distribución final de posibilidad como una lista de pares (c_i, pos_j) para la clase $j \in [1, z]$. Si ordenamos esta lista de pares de mayor a menor posibilidad, diremos que la clasificación del sistema multiclasificador sobre un dato obtiene esta lista ordenada de posibilidades, donde el primer par representa a la clase cuya posibilidad es mayor:

$$Smc(\vec{x}) = [(c_1, pos_1), \ldots, (c_z, pos_z)]$$

cumpliéndose que la suma de las posibilidades es uno.

La Figura 4.5 muestra un esquema de como se produce la clasificación de un dato x. En este esquema, el sistema ha seleccionado a tres agentes (k=3); cada agente ofrece una lista con las posibilidad de cada clase y la pertenencia relativa del dato al agentes, de forma que el sistema multiclasificador obtiene una nueva distribución de clases, usando las distribuciones de los agentes junto con sus pertenencias relativas.

4.8. Evaluación del sistema multiclasificador

La evaluación de la clasificación del sistema se realiza, como en cualquier algoritmo de clasificación, aplicando el clasificador a un conjunto de test y contabilizando los aciertos y fallos. El conjunto de test es obtenido como un subconjunto de los datos iniciales del problema, de forma que los datos iniciales se descomponen en dos subconjuntos disjuntos: entrenamiento y test.

Cada dato $e = (\vec{x}, c)$ lleva asociado un vector $\vec{x}$ y la clase c correcta para $\vec{x}$, de forma que un test sobre $\vec{x}$ comprueba si la clase predicha por el clasificador coincide con la clase del dato.

Ya que la clasificación de un dato por el sistema multiclasificador devuelve una lista con las posibilidades de cada clase ($Smc(\vec{x}) = [(c_1, pos_1), \ldots, (c_z, pos_z)]$), se va a definir un valor de acierto asociado a la clasificación como:

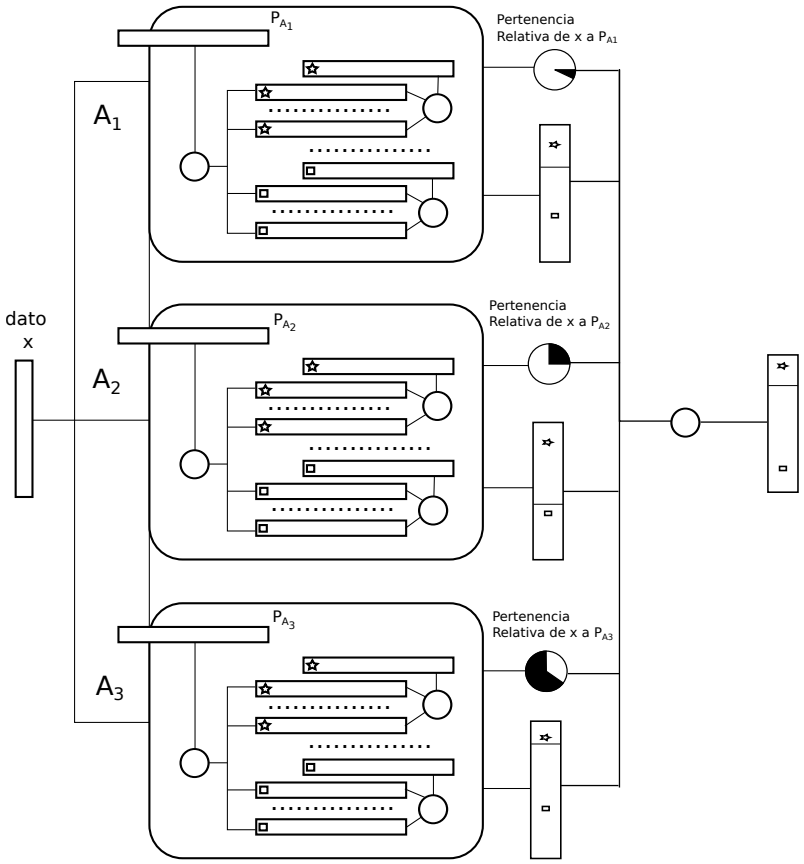

Figura 4.5: Combinación de tres agentes para clasificar un dato x

$$Acierto(\vec{x}, c) = pos_c \tag{4.49}$$

donde pos_c es la posibilidad para la clase c, que es la correcta, en la distribución $Smc(\vec{x})$. El valor pos_c ofrece información del grado de acierto en la clasificación del dato. Fijado el valor de acierto que obtiene el sistema multiclasificador, definimos el valor de fallo del mismo como la suma de las posibilidades de todas las demás clases distintas a la clase c. Al estar ante una distribución normalizada, el nivel de fallo será:

$$Fallo((\vec{x}, c)) = 1 - Acierto((\vec{x}, c)). \tag{4.50}$$

Si usamos el sistema multiclasificador para clasificar los datos de un conjunto de test ($cTest$), podemos definir la calidad de la clasificación ($CalidadTest$) como la media del valor de acierto para el total del conjunto

de test. Este valor representa al grado de acierto medio de la clasificación sobre el conjunto de test:

$$CalidadTest(cTest) = \frac{\sum_{(\vec{x},c)\in cTest} Acierto((\vec{x},c))}{Cardinal(cTest)} \tag{4.51}$$

Con motivo de comparar la calidad de nuestro sistema multiclasificador con los sistemas no difusos, establecemos un valor no difuso de acierto o fallo, mediante el corte por un valor mínimo de acierto.

Diremos que el sistema multiclasificador ha clasificado correctamente a un dato $\vec{x}$ si la posibilidad obtenida por el sistema para la clase correcta es mayor que un umbral de acierto Uok. Este umbral nos indica como de estrictos queremos ser a la hora de interpretar el acierto en la predicción de una clase. La función $Correcto$ se define como:

$$Correcto(Smc((\vec{x}),c)) \begin{cases} 1 & \text{si } Acierto((\vec{x},c)) > Uok \\ 0 & \text{en otro caso} \end{cases} \tag{4.52}$$

Si queremos saber el porcentaje de acierto del sistema multiclasificador sobre el conjunto de test con n datos, sumaremos los aciertos obtenidos para cada dato, dividiéndolos entre el número de datos del conjunto de test. Llamaremos $Test$ a la función que aplica el test a cada dato y obtiene el porcentaje de aciertos, donde n es el cardinal del conjunto de test.

$$Test(cTest) = \frac{\sum_{i=1}^{n} Correcto(smc(\vec{x}_i,c))}{n} \tag{4.53}$$

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

5

Experimentos

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

5
Experimentos

Los experimentos que se van a realizar están caracterizados por los parámetros que han sido definidos en el modelo MAC. Podemos distinguir entre los parámetros que van a ser iguales para todos los experimentos y aquellos que van a variar. A continuación detallamos cada uno de ellos.

Los tres parámetros que afectan al agente van a mantener su valor en todos los experimentos. El método para obtener la posibilidad de cada clase al clasificar un dato va a ser la similitud con el dato más cercano (Ecuación 4.9); la similitud entre los datos de un agente se calcula con la distancia Euclídea normalizada (Ecuación 4.5) y el valor k de la Ecuación 4.2, que aumenta o disminuye el peso de la entropía será 1.

Los parámetros del sistema multiclasificador que se mantendrán constantes son: El número mínimo de agentes por clase, que serán cinco; el valor mínimo para considerar que se ha acertado (Uok de la Ecuación 4.52), cuyo valor es 0.95; el umbral de selección de agentes (Uk de la Ecuación 4.42) que será 0.1.

El modelo ha sido implementado usando la estrategia de asignación de datos a los agentes llamada colaborativa (Sección 4.6.2). Esta estrategia ofrece un dato de entrenamiento a todos los agentes, permitiendo que más de un agente elija el dato.

Los parámetros que van a variar en los experimentos son tres: la memoria mínima de un agente, usada en el algoritmo para el cálculo del número de agentes y el tamaño de la memoria (Algoritmo 17), y que por defecto será 3; y el umbral de similitud (Us), utilizado para decidir si dos agentes son equivalentes (Ecuación 4.27), y que será 0.8; y el porcentaje de datos de entrenamiento usados para crear e inicializar los agentes, que por defecto será el 30 por ciento.

En el apéndice se podrán observar los diferentes conjuntos de agentes

obtenidos para cada uno de los experimentos, donde se podrá visualizar el contenido de cada agente.

5.1. Datos con clases bien definidas

Base de datos D2C5: datos con una clase por zona y tantas zonas como clases

Para verificar el sistema multiclasificador utilizaremos inicialmente aquel caso más simple para un conjunto de datos de entrenamiento, donde las zonas que representan a las clases son disjuntas, es decir, cada zona contiene a una sola clase. Se ha generado un conjunto de datos, que llamaremos D2C5, que contiene datos con dos atributos (x_1,x_2) y cinco clases. Los valores de los atributos han sido distribuidos uniformemente, atendiendo a las normales descritas en la Tabla 5.1, tal y como muestra la Figura 5.1.

Distribución	$n(\bar{x}, \delta)$ para x_1	$n(\bar{x}, \delta)$ para x_2
clase = 0	(10,3)	(10,3)
clase = 1	(90,3)	(10,3)
clase = 2	(50,5)	(50,5)
clase = 3	(10,3)	(90,3)
clase = 4	(90,3)	(90,3)

Tabla 5.1: Distribución normal para los datos D2C5

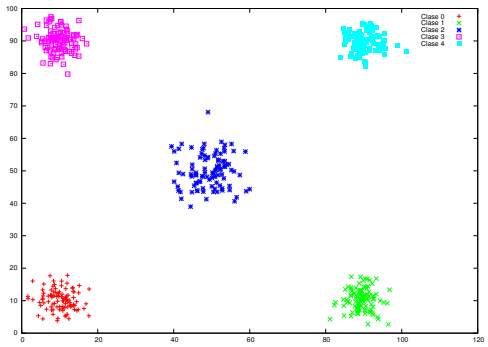

Figura 5.1: Datos para el conjunto D2C5

Respecto a los parámetros del sistema, se han realizado cuatro experimentos, variado el umbral del similitud Us y el valor mínimo de la memoria de un agente m. Los resultados de estos experimentos se muestran en la Tabla 5.2, que refleja en cada fila un experimento. En las columnas de la tabla se representa: el valor Us;el número mínimo de datos en la memoria

(m); el número de agentes en el entrenamiento básico (nA EB); el número de agentes en la detección de zonas (nA Zonas); el número de agentes en la reducción (nA Reducción); y el tanto por ciento de acierto conseguido al aplicar el conjunto de test al conjunto final de agentes (Acietro en Test).

Us	M	nA EB	nA Zonas	nA Reducción	Acierto en Test
0.80	3	28	28	12	100 %
0.20	3	28	28	7	100 %
0.80	7	21	21	6	100 %
0.20	7	21	21	5	100 %

Tabla 5.2: Resultados para los datos D2C5

En los cuatro experimentos podemos observar que el número de agentes obtenido en el entrenamiento básico es el mismo que en la generación de zonas no cubiertas. Esto es debido a que el dominio inicial está muy bien delimitado para cada clase, por lo que, después del entrenamiento básico todas las zonas están cubiertas por los agentes y la detección de zonas no genera nuevos agentes. También podemos observar como el porcentaje de acierto es el mismo en todos los experimentos, llegando al 100 %.

Vamos a analizar a continuación como afecta la variación de dos parámetros en los agentes generados. Cuando el tamaño de la memoria es pequeño (m=3) el número de agentes en las dos primeras fases es mayor que si la memoria es grande (m=7). Esto es debido a que el cálculo del número de agentes y el número agentes por clase se realiza adaptando el valor de la memoria al número de datos disponible en la creación de agentes.

Un valor elevado del umbral de similitud (Us=0.8) va a exigir a los agentes ser muy parecidos para considerar que son similares por los que obtendremos más agentes en la reducción. En los dos primeros experimentos pasamos de 12 agentes a 7, variando este umbral de 0.8 a 0.2. Podemos observar como en el cuarto experimento se obtienen 5 agentes, justo un agente por cada zona original.

El sistema multiclasificador obtenido para el primer experimento, con un valor de 3 para el número de datos mínimo en memoria y un valor de 0.8 para el umbral de similitud, se encuentra detallado en el Apéndice A.1.

A continuación, vamos a mostrar como han evolucionado los agentes en el entrenamiento básico. En la Figura 5.2 se reflejan los prototipos de los agentes justo después de haberse creado, antes del entrenamiento básico. Al ser una inicialización aleatoria, los agentes contienen datos de diferentes zonas, por los que sus prototipos no representan zonas reales del dominio.

En la primera iteración del entrenamiento básico, después de haber ofrecido todos los datos de entrenamiento a todos los agentes, los agentes ya están situados en las cinco zonas que definen el dominio. En esta iteración, todos los agentes han sido modificados por algún dato al menos una vez.

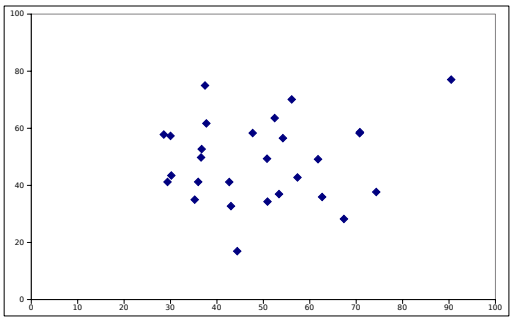

Figura 5.2: Agentes creados para D2C5

En la segunda iteración en el entrenamiento básico, solamente dos agentes cambian su configuración con algún dato y es en la tercera iteración cuando ningún agente cambia y se termina el entrenamiento básico.

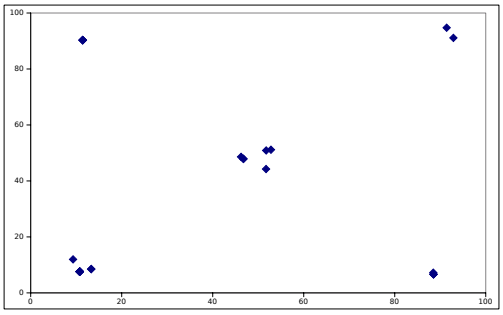

Figura 5.3: Agentes obtenidos en la primera iteración del EB para D2C5

Respecto a la reducción de agentes, mostramos en la Tabla 5.3 cuales han sido las 12 clases de equivalencia obtenidas al reducir los 28 agentes del primer experimento. Si enumeramos los agentes del 0 al 27, podemos observar, en cada fila, el agente obtenido y la clase de equivalencia del agente.

Agente Final	Clase de equivalencia
17	(17)
18	(18)
19	(19)
27	(27)
12	(12, 4)
24	(24)
8	(8)
14	(14)
16	(16, 20, 7, 25, 15)
0	(0, 1)
23	(23, 11, 21, 13)
2	(2, 3, 5, 6, 9, 10, 22, 26)

Tabla 5.3: Clases de equivalencia en la reducción para los datos D2C5

Base de datos D2C4 1c2z: datos con una clase por zona y dos zonas con la misma clases

Los datos del experimento anterior reflejan la mejor situación: zonas bien separadas y cada zona de una clase. En este nuevo experimento vamos a añadir dos zonas distintas para una misma clase. En los algoritmos clásicos, una situación de este tipo podría inducir a formar una agrupación para la clase, cuyo centro sería el centro situado entre las dos zonas. Dado que los agentes son especialistas en zonas, se van a crear diferentes agentes para cada una de las dos zonas con la misma clase.

El conjunto de datos generado, que llamaremos D2C4 1c2z, contiene datos con dos atributos y cuatro clases, distribuidos como en el ejemplo D2C5, con la diferencia de que la zona superior derecha representa a la misma clase que la zona inferior izquierda. Al igual que en el caso anterior, los datos han sido distribuidos según la normal de la Tabla 5.4. y se pueden observar en la Figura 5.4, donde en cada fila se ha representado una de las zonas generadas, conteniendo la primera zona y la última zona a datos de la misma clase (clase 0).

Distribución	$n(\bar{x}, \delta)$ para x_1	$n(\bar{x}, \delta)$ para x_2
clase = 0	(10,3)	(10,3)
clase = 1	(90,3)	(10,3)
clase = 2	(50,5)	(50,5)
clase = 3	(10,3)	(90,3)
clase = 0	(90,3)	(90,3)

Tabla 5.4: Distribución normal para los datos D2C4 1c2z

Se han usado los parámetros originales descritos en la introducción del capítulo y los resultados obtenidos se pueden observar en la Tabla 5.5. Por un lado, el número de agentes en la detección de zonas se mantiene igual a

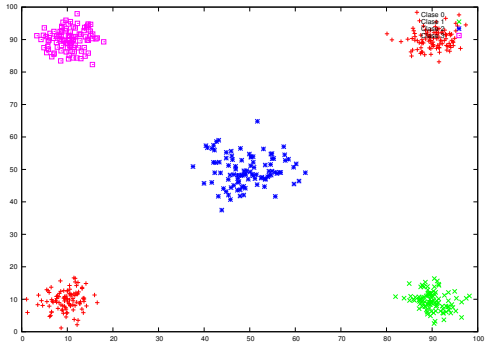

Figura 5.4: Datos para el conjunto D2C4 1c2z

los agentes obtenidos en el entrenamiento inicial, lo que significa que, al igual que en el caso anterior, que el primer conjunto de agentes ya cubre todas las zonas. Esta idea se refuerza observando que el acierto en el conjunto de test sobre cada uno de los conjuntos de agentes de cada fase es también del 100 %.

D2C5	EB	Zonas	Reducción
Número de Agentes	24	24	9
Acierto en Test	100 %	100 %	100 %

Tabla 5.5: Resultados para los datos D2C4 1c2z

En la Figura 5.5 se reflejan los prototipos de los agentes después de la fase de reducción, y la composición de estos agentes se puede examinar en el Apéndice A.2. Se puede observar como las zonas originales tienen a agentes que las cubren. Se cumple, además, que cada agente contiene a una sola clase, la clase de la zona del agente.

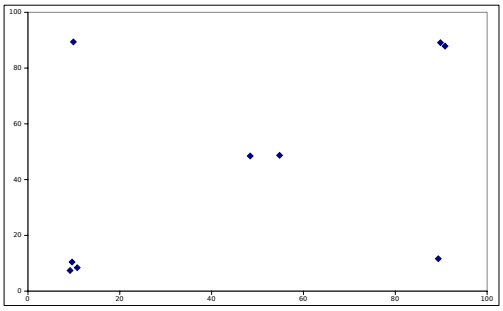

Figura 5.5: Agentes después de aplicar la reducción para D2C4 1c2z

5.2. Datos con clases que no están bien definidas

En esta sección vamos a estudiar dos conjuntos de datos con zonas que contienen diferentes clases. El primer conjunto mantiene una estructura parecida de zonas a los ejemplos anteriores, excepto que en una zona vamos a generar datos de diferentes clases. El segundo experimento va a usar un conjunto de datos generado por Ludmila Kuncheva [57] en el que hay zonas con mezcla de clases. En este caso podremos comparar nuestros resultados con los resultados de su experimento.

Base de datos D2C5 1z2clases: datos con una zona con dos clases mezcladas

El objetivo de este experimento es observar el comportamiento del sistema cuando hay una zona que contiene datos mezclados de dos clases. En este caso, se han generado cuatro zonas, tres de las cuales cuentan con datos de una sola clase y la cuarta contiene datos de dos clases. La distribución usada para generar los datos (Tabla 5.6) obtiene una zona derecha que contiene a dos clases (la clase 3 y 4). Esto significa que el intervalo [83,90] se van a concentrar la mayoría de datos mezclados. Podemos observar los datos generados con esta distribución en la Figura 5.6.

Distribución	$n(\bar{x}, \delta)$ para x_1	$n(\bar{x}, \delta)$ para x_2
clase = 0	(10,3)	(10,3)
clase = 1	(50,5)	(50,5)
clase = 2	(10,3)	(90,3)
clase = 3	(83,7)	(10,7)
clase = 4	(90,7)	(10,7)

Tabla 5.6: Distribución normal para los datos D2C5 1z2clases

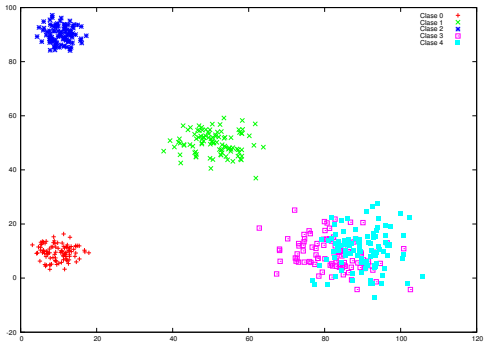

Figura 5.6: Datos para el conjunto D2C5 1z2clases

Podemos observar en la Tabla 5.7 los resultados obtenidos aplicando los parámetros iniciales. En primer lugar, y como novedad en estos

experimentos, la fase de detección de nuevas zonas obtiene nuevos agentes. El entrenamiento básico genera 28 agentes y la detección de zonas genera 3 nuevos agentes, con lo que el total de agentes en la segunda fase es 31.

D2C5 z2clases	EB	Zonas	Reducción
Número de Agentes	28	38	7
Acierto en Test	60 %	83 %	83 %

Tabla 5.7: Resultados para los datos D2C5 1z2clases

Si observamos el acierto en cada fase, vemos que, para los agentes de la fase inicial, se ha conseguido un 60 % de acierto. Analizando la estructura de los agentes en esa fase, se detecta que se han generado agentes cuyos datos son de las clases: 0, 1 y 4 y no existe ningún agente cuyos datos sean de la clase 2 y 3. La razón de este comportamiento es que uno de los parámetros que influyen en la función de la calidad es la entropía, por lo que los agentes tienden a especializarse en zonas con una sola clase, desechando los datos de clases diferentes. Como los agentes son autónomos y no tienen información de las zonas de los demás agentes, varios agentes se van especializando en las mismas zonas mientras que algunas zonas quedan sin cubrir.

La fase de detección de zonas no cubiertas permite corregir esta tendencia, de forma que se van a generar 10 nuevos agentes, siete de ellos representado a la clase 3 y tres agentes para la clase 2. El test sobre el nuevo conjunto de agentes proporciona un incremento en el acierto del 23 %. El 17 % de fallo lo podemos achacar a la zona común en la que se han generado datos de las dos clases.

Por último cabe destacar que la fase de reducción ha obtenido 7 agentes, manteniendo el índice de acierto de los 38 agentes de la fase anterior. El contenido de estos agentes está disponible en el Apéndice A.3.

Base de Datos Cone Torus de Kuncheva: datos con clases mezcladas

Otro conjunto de datos es Cone Torus, generado y usado por Ludmila Kuncheva en [57]. Los datos tienen dos dimensiones y tres clases; se han generado 400 datos de entrenamiento y 400 datos de test, de forma que el 25 % corresponden a la clase 1, el 50 % a la clase 2 y el 25 % a la clase 3. Los datos son accesibles en [58].

Estos datos han sido generados usando tres distribuciones: en forma de cono, de medio toro y una distribución normal. Podemos observar que hay zonas con diferentes clases en el gráfico que dibuja cada punto (Figura 5.7). La clase 1 está claramente separada de las clases 2 y 3 aunque hay algún dato de la clase 3 en su zona. Respecto a las clases 2 y 3 podemos observar que tienen una amplia zona en común (la zona central); la clase 2 se expande hacia la derecha y la clase 3 hacia arriba.

En la Tabla 5.8 podemos observar los resultados obtenidos. Al igual

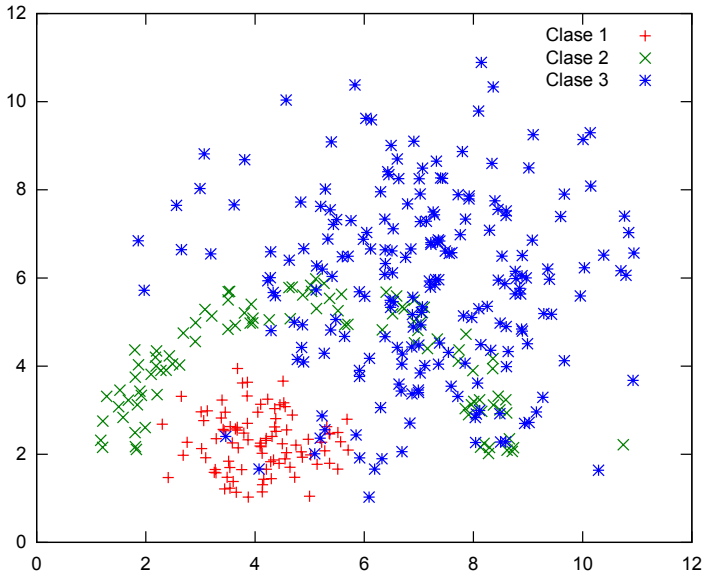

Figura 5.7: Datos para el conjunto Cone Torus

que en el experimento anterior, se van a generar nuevos agentes en la fase de detección de nuevas zonas. En el entrenamiento básico se generan 18 agentes, 29 agentes en la detección de zonas (11 nuevos en zonas no cubiertas) y, la última fase, reduce el conjunto a 17 agentes. Los agentes finales generados se detallan en el Apéndice A.4

Cone Torus	EB	Zonas	Reducción
Número de Agentes	18	29	17
Acierto en Test	63 %	68 %	76 %

Tabla 5.8: Resultados para los datos Cone Torus

En la tabla de resultados podemos observar como se ha pasado de una efectividad del 68 % en el conjunto con las zonas cubiertas hasta una efectividad del 76 % en el conjunto reducido. La explicación a la ganancia de efectividad está basada en el uso del sistema al clasificar. Al haber menos agentes, la pertenencia relativa de cada agente sube, por lo que se seleccionarán menos agentes para cada dato. Esto significa que solo los agentes muy influyentes participan en la clasificación.

El análisis del contenido de los agentes en cada fase respecto a las clases que lo componen ha sido definido en la Tabla 5.9. Se ha definido una columna para cada posible conjunto de clases encontrado en el conjunto de agentes. En general, se han encontrado agentes con cada una de las clases posibles y también se han encontrado agentes con dos clases (la 2 y la 3). Cada fila contiene información de las clases que contienen los agentes en

una determinada fase. Cabe notar que la fila 3 contienen a la suma de los agentes generados en el entrenamiento básico y los generados en la detección de zonas (fila 1 y 2).

	c = 1	c = 2	c = 3	c = 2 y 3
EB = 18	2	8	2	6
Zonas = 11	1	8	2	0
EB + Zonas = 29	3	16	4	6
Reducción = 17	2	6	6	3

Tabla 5.9: Composición de los agentes en las tres fases para los datos Cone Torus

A continuación se va analizar, en la fase de detección de zonas, como evolucionan los datos no cubiertos que sirven de entrenamiento para la generación de zonas (Figura 5.8) y cuales son las clases que componen cada uno de los agentes generados en cada ciclo de detección de nuevas zonas (Tabla 5.10).

Zonas = 11	c = 1	c = 2	c = 3	c = 2 y 3
Zona 1 = 5	0	3	2	0
Zona 2 = 4	0	0	4	0
Zona 3 = 1	1	0	0	0
Zona 4 = 1	0	1	0	0

Tabla 5.10: Clases de los agentes en cada zona detectada para los datos Cone Torus

La Figura 5.8 (a) representa a los datos que han sido mal clasificados por los agentes generados en el entrenamiento básico y que serán los datos de entrenamiento para la primera detección de zonas. Podemos comprobar como hay zonas de los datos originales que no aparecen, ya que son las zonas donde se han especializado los agentes del entrenamiento básico. Estas zonas corresponden a la mayoría de los datos de clase 1, a los de clase 2 de la zona izquierda y a datos de clase 3 de la zona superior derecha.

En la Figura 5.8 (b) se observan los datos mal clasificados después de obtener un nuevo conjunto de agentes para la zona primera. Observamos como en la zona central hay menos datos de las clases en las que se han especializado estos agentes: las clases 2 (tres agentes) y 3 (dos agentes).

En la Figura 5.8 (c) comprobamos los datos mal clasificados para la zona segunda. En esta zona, según la Tabla 5.10, se han generado 4 agentes, todos de la clase 3. Se puede observar como, gracias a estos nuevos agentes, han desaparecido todos los datos mal clasificados de la clase 3. En este gráfico quedan algunos datos de la clase 1 y una zona muy compacta de la clase 2.

En la tercera zona generada se ha obtenido un agente de la clase 1 que va a cubrir a todos los datos de esta clase, por lo que en la Figura 5.8 (d) observamos que solo aparecen los datos de la clase 2.

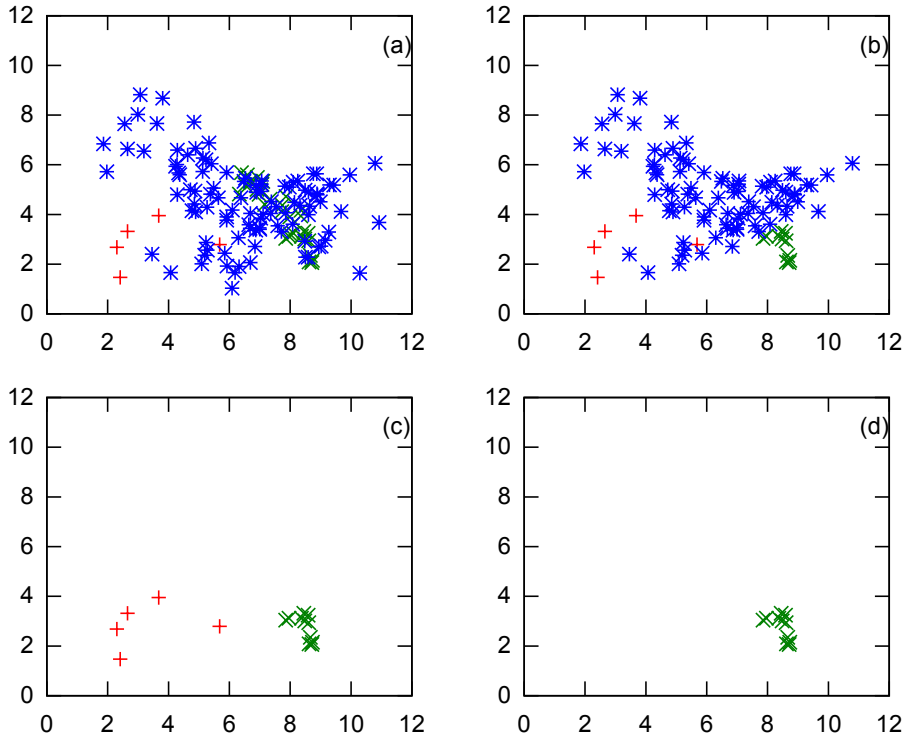

Figura 5.8: Datos en el ciclo de detección de zonas no cubiertas para Cone Torus

La cuarta zona generada obtiene un agente de la clase 2 que cubre a todos los datos que no habían sido bien clasificados (Figura 5.8 (d)), por lo que el ciclo de generación de zonas termina obteniendo los 11 agentes generados en las 4 iteraciones.

5.3. Bases de datos del repositorio UCI

En esta sección se van mostrar los resultados de los experimentos con tres bases de datos clásicas, que han sido usadas en la literatura con diferentes algoritmos de clasificación. Las bases de datos se pueden encontrar en el repositorio UCI y son: Iris, Pendigits y Wine. Se ha reflejado, en la Tabla 5.11, el número de datos disponible de cada una; el número de atributos; el número de clases; y, por último, la cantidad de datos usados para el entrenamiento y para el test del conjunto de datos original.

Comenzamos el experimento con la base de datos Iris. Los valores de los parámetros variables usados son: para la memoria mínima, 3; como umbral similitud, 0.8; y el porcentaje de datos para la inicialización ha sido un 50 %. Podemos observar en la Tabla 5.12 cuales han sido los resultados del

	Nº datos	Nº atributos	Nº clases	Nº Entrena	Nº Test
Iris	150	4	3	106	44
Wine	178	13	3	128	50
Pendigits	10992	16	10	7494	3498

Tabla 5.11: Características de Iris, Pendigits y Wine

experimento con estos parámetros. En la fase de entrenamiento básico se han obtenido 16 agentes, con un tanto por ciento de acierto en el conjunto de test del 66%. La causa de que este valor sea relativamente bajo es que hay zonas en las que no se ha especializado ningún agente. Esto se observa claramente al comprobar que con 4 nuevos agentes, obtenidos en la fase de detección de zonas, el sistema ofrece un porcentaje de acierto del 98%. Cabe destacar que en la fase de reducción se ha pasado de 20 agentes a 3 agentes, manteniendo el porcentaje de acierto en un 98%. Por tanto, el sistema multiclasificador final contiene tres agentes que cubren al dominio general. En el Apéndice A.5 se puede observar la composición de los tres agentes del sistema.

Iris	EB	Zonas	Reducción
Número de Agentes	16	20	3
Acierto en Test	66%	98%	98%

Tabla 5.12: Resultados para Iris

El experimento con la base de datos Wine se ha realizado con una memoria mínima de 4 datos; un umbral de similitud de 0.95; y un porcentaje del 30% de los datos para la inicialización de los agentes. En este caso, se ha aumentado el umbral de similitud debido a que un umbral menor reducía a tres el número de agentes, decreciendo el porcentaje de acierto respecto a la fase de detección de zonas. Los resultados para estos parámetros los podemos consultar en la Tabla 5.13. En este caso, la detección de zonas genera un nuevo agente e incrementa en un 6% el acierto en el test, pasando del 90% al 96%. La fase de reducción obtiene 5 agentes de los 10 de la anterior fase, manteniendo su porcentaje de acierto. Los agentes finales se pueden consultar en el Apéndice A.7.

Wine	EB	Zonas	Reducción
Número de Agentes	9	10	5
Acierto en Test	90%	96%	96%

Tabla 5.13: Resultados para Wine

Por último, el experimento con los datos Pendigits ha obtenido un 80% de acierto sobre 14 agentes obtenidos en la reducción, usando como parámetros: 11 datos para la memoria; 0.95 para el umbral de similitud y el 8% de los datos para la inicialización. Los agentes obtenidos se muestran en

el Apéndice A.6.

Resultados de otros algoritmos sobre las bases de datos: Cone Torus, Iris, Pendigits y Wine

Para finalizar este capítulo, se han realizado diferentes experimentos, usando algoritmos conocidos, sobre las bases de datos Cone Torus, Iris, Pendigits y Wine. Nuestro objetivo es mostrar los resultados de estos experimentos y compararlos con los resultados obtenidos con el modelo MAC propuesto. Se ha usado la herramienta weka [40], que implementa diferentes métodos de clasificación y se han escogido los algoritmos Bl1 [1] y K* [19]. El algoritmo Bl1 usa una medida simple de distancia para encontrar los datos de entrenamiento mas cercanos al dato a clasificar, y predice la clase del dato más cercano. K* es un clasificador basado en instancias donde la clase de dato se obtiene obteniendo las instancias similares al dato, según alguna función de similaridad.

La tabla 5.14 muestra los resultados de estos algoritmos y el modelo MAC. Podemos observar como en el caso de las bases de datos Iris y Wine nuestro modelo iguala o mejora a los dos métodos. Teniendo en cuenta que el conjunto de agentes para Iris y Wine es muy reducido (3 y 5 agentes respectivamente), podemos concluir que nuestro método, no solo mejora a los anteriores sino que la clasificación se efectúa bastante más eficientemente. En el caso de Cone Torus y Pendigits el resultado de nuestro algoritmo es inferior al de los otros métodos. En este caso, debemos tener en cuenta que el número de agentes es muy bajo (17 y 14 respectivamente), y que los datos son complejos: en el caso de Cone Torus tenemos zonas con datos de dos clases mezclados y, en el caso de Pendigits, tenemos diez clases diferentes.

	IB1	K*	modelo MAD
Cone Torus	84,75 %	87,25 %	76 %, con 17 agentes
Iris	96,00 %	96,00 %	98 %, con 3 agentes
Pendigits	97,22 %	96,54 %	80 %, con 14 agentes
Wine	96,00 %	92,00 %	96 %, con 5 agentes

Tabla 5.14: Resultados usando los algoritmos Bl-1, K* y el modelo MAD

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

6

Conclusiones y Trabajos Futuros

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

6

Conclusiones y trabajos futuros

6.1. Compendio de aportaciones

El objetivo principal de este trabajo fue enunciado en la introducción como **diseñar y definir un sistema multiclasificador formado por un conjunto de agentes clasificadores autónomos, que se especialicen en zonas específicas del dominio estudiado**. Para alcanzarlo se plantearon un conjunto de tres subobjetivos: estudiar y analizar el estado del conocimiento de los sistemas multiclasificadores; definir el sistema multiclasificador con múltiples agentes especialistas; y probar y validar el sistema multiclasificador ideado.

Referente al estado del conocimiento en el ámbito de los sistemas multiclasificadores, se ha realizado una síntesis de las características más relevantes del mismo, focalizando su presentación en la forma de obtener los clasificadores básicos y la manera de combinarlos. Se han presentado los algoritmos más extendidos para la construcción de distintos conjuntos de datos de aprendizaje, como eje para los algoritmos de obtención de clasificadores básicos homogéneos y desacoplados en sus clasificaciones. También se han expuesto las metodologías y técnicas usuales para la combinación de los resultados de las clasificaciones básicas, mediante algoritmos de fusión y selección de clasificadores. Se han estudiado los modelos en el ámbito de la generación explicita de especialistas y la selección dinámica de los expertos, y se ha enmarcado nuestra propuesta en este ámbito. De esta forma, el primer subobjetivo del trabajo ha quedado completado, dando una visión general sobre los elementos relevantes para nuestra propuesta, abandonando una intención enciclopedista y exhaustiva del tema, fuera de los objetivos perseguidos.

Cumplimentado el primer subobjetivo, presentamos una propuesta inicial fundamentada en las técnicas de agrupamiento de hormigas. Tras

una primera exposición de dichas técnicas, se **propone extender la funcionalidad de los agentes transportadores con la funcionalidad de la clasificación**. Este enfoque obliga a redefinir la estructura del agente transportador, así como su forma de actuar para lograr su especialización. El agente transportador pasa a ser un agente clasificador y transportador, manteniendo la capacidad de agrupamiento de los datos y formando un clasificador entrenado con los datos que van conformando su zona de especialización y no están bien agrupados en la arena. Después del entrenamiento, un agente obtiene un representante de su zona de especialización e información de la clase que más veces ha encontrado en sus datos de entrenamiento, configurando un clasificador binario sobre el mismo. Este enfoque proporciona agentes desacoplados en sus decisiones, configurados de forma autónoma debido al encuentro aleatorio de los datos.

El experimento realizado sobre este modelo se ha limitado a una implementación muy simple de cada agente, cuya estructura contiene a la media de los datos escogidos y la clase mayoritaria de esos datos; y una combinación de las respuestas de los agentes basada en la obtención de la clase mayoritaria de los k agentes más cercanos al dato, con igual peso en la clasificación final. La capacidad de aprendizaje de cada agente es muy limitada y, por tanto, el sistema resultante ha obtenido mejores resultados cuando tenemos un número elevado de agentes. Con esto se ha conseguido un sistema de reducción del conjunto inicial de datos de entrenamiento, al cual se ha aplicado la selección de los K vecinos más cercanos para clasificar un nuevo dato, donde el vecino más cercano está referido al agente cuyo prototipo está mas cercano. Comparando los resultados con la aplicación de algoritmo del vecino mas cercano al conjunto inicial de datos, podemos observar que los resultados son similares. Por tanto, el campo de aplicación de esta propuesta queda limitado a la reducción del conjunto de entrenamiento, donde cada agente (su prototipo) representaría a un nuevo dato del nuevo conjunto de entrenamiento reducido y la clase mayoritaria representa a la clase asociada al prototipo.

La segunda aproximación que se propone mantiene la idea de un espacio en el que se mueven los agentes y estos deciden si los datos encontrados van a servir para su entrenamiento, eliminándose el componente de agrupamiento de los datos en el espacio. La agrupación de datos del algoritmo original en realidad producía un efecto colateral, en el sentido de que lo que buscamos es un modelo de exploración en el que los agentes encuentren datos y tomen decisiones respecto a ellos; la agrupación solo sirve para que datos bien agrupados no puedan ser usados como datos de entrenamiento para nuevos agentes. Respecto a la estructura de los agentes básicos, se observa que la información asociada a cada uno de los agentes es pobre, por lo que se introduce una mejora en la misma. En esta aproximación se **añade al agente una memoria y una medida de su calidad**. Esta memoria, o almacén de datos, define la zona de especialización del agente y las clases que intervienen en la zona, permitiendo que los agentes puedan decidir sobre varias clases distintas. La calidad del agente se va a medir en

base a las características de los datos que lo forman y a la diversidad de sus clases, produciéndose el entrenamiento de un agente si el nuevo dato mejora su calidad.

Los dos modelos anteriores tienen deficiencias comunes. Por un lado, al ser generados los agentes sin información previa, no podemos asegurar que el conjunto de agentes va a cubrir el dominio general. Tampoco se establecen mecanismos de evaluación posterior del sistema para detectar zonas no cubiertas y reorganizar los agentes. Otro aspecto a destacar en estos modelos es que no se define un criterio de terminación del entrenamiento. Este se basa en la repetición, un número fijo de iteraciones, de movimientos de los agentes en el espacio de datos.

El modelo final que se propone intenta solucionar las deficiencias anteriores. En este modelo se mantiene la idea de la generación de agentes autónomos que se mueven en un entorno donde se encuentran los datos y que toman decisiones respecto a los datos encontrados. En realidad, no se necesita de un espacio real en el que colocar los datos sino que se puede simular un entorno en el que los agentes encuentran datos, los procesan, y los devuelven al entorno. En este caso, los agentes siguen siendo inicializados con datos aleatorios, de forma que se define una primera fase de entrenamiento en la que se genera un primer conjunto de agentes. El sistema entonces debe detectar si todas las zonas del dominio son cubiertas por este conjunto de agentes y, en caso contrario, lanzar un proceso de creación de nuevos agentes en las zonas no cubiertas. La detección de estas zonas no cubiertas y la generación de agentes que las cubran supone una mejora sustancial respecto a las dos primeras aproximaciones.

Otra mejora importante respecto a los dos primeros modelos es que el sistema multiclasificador es capaz de evaluar a los agentes obtenidos después del entrenamiento básico y de la detección de zonas. Los agentes obtenidos en este punto cubren todas las zonas pero es posible que tengamos diferentes agentes que representan a la misma zona. Esto es debido a que los agentes se crean y entrenan sin información global del sistema, de forma autónoma e independiente del resto. La definición de una relación de similaridad entre agentes ha permitido establecer una categorización de los mismos por clases de equivalencia. Esta similaridad es definida teniendo en cuenta dos características: la configuración del agente con respecto al subdominio que define y la clasificación que realiza. Esta relación de similaridad permite definir un **mecanismo de simplificación del conjunto de agentes** que obtiene un representante por cada clase de equivalencia. El conjunto final de agentes se forma con el representante de cada clase de equivalencia, logrando mantener la efectividad del conjunto de agentes sobre el conjunto de test. Este resultado permite una reducción muy significativa del conjunto de agentes, incrementando de forma sustancial su rendimiento en el proceso de explotación del sistema, así como facilitando el uso optimizado de los recursos.

Con el objetivo de explotar la información estructural del agente, se amplía la información que ofrece un agente al clasificar un dato introduciendo una **respuesta difusa del clasificador**, que presenta una distribución de posibilidad sobre el conjunto de clases posibles. Este enfoque proporciona mucha mayor riqueza de matices, frente al clásico clasificador binario. Para establecer esta clasificación difusa se fijan las funciones que definen los distintos valores de pertenencia.

Otra aportación del modelo final está referida a la simulación del entorno y como los agentes interactúan con los datos. Se han definido tres modelos de gestión de los datos respecto a los agentes. El modelo colaborativo, en el que en un momento dado un dato es visible por todos lo agentes, y cada agente decide si debe utilizarlo para su entrenamiento. Esta asignación de datos tiene la ventaja de que permitimos a todos los agentes disponer de la misma información, pero tiene el inconveniente de que el sistema puede tender a especializar a agentes exactamente en las mismas zonas. Con el modelo competitivo se pretende que un dato solo pueda servir de entrenamiento al agente que mejora más su calidad con dicho dato. Esta estrategia permite que los agentes en los primeros instantes del entrenamiento se especialicen en zonas diferentes, pero se pierde la posibilidad de uso del dato por más de un agente. Por ultimo, el modelo autónomo es el que más se parece a los modelos de las aproximaciones iniciales. En este caso, un dato es ofrecido secuencial y aleatoriamente a los agentes, de forma que el primer agente al que el dato mejore su calidad es el que se lo queda, tal y como ocurría en las aproximaciones iniciales cuando los agentes se encontraban un dato en la arena.

Por otro lado, se han definido condiciones de terminación del entrenamiento basadas en que los agentes dejen de mejorar su calidad con los los datos. También se establece una condición de terminación para la detección de zonas que está referida a que todos los datos encuentren a algún agente que los clasifique correctamente.

Con respecto a la combinación de clasificadores básicos, se evoluciona de una metodología inicial basada en técnicas como los k-vecinos más cercanos, a la propuesta final basada en **un ranking de los clasificadores básicos, que son seleccionados para la clasificación, con aportaciones ponderadas de cada uno de ellos**.

El modelo propuesto desarrolla y alcanza el segundo subobjetivo, definiendo un sistema multiclasificador cuyos agentes básicos son especialistas en dominios específicos de los datos. Su construcción y simplificación garantiza la independencia y diversidad de los mismos.

Como características generales podemos señalar que las principales ventajas de este método son que los agentes clasificadores son simples e independientes en sus actuaciones. Esta característica permite afrontar problemas con gran cantidad de datos de una forma paralela y distribuida. Así mismo, un agente es un agente activo, que se mueve en el espacio

de datos y es capaz de seleccionar los datos que le son útiles para su entrenamiento. Respecto a los datos de entrenamiento, el modelo permite la incorporación de nuevos datos de entrenamiento en diferentes momentos del aprendizaje del sistema. Estas capacidades hacen que el aprendizaje de los agentes sea un proceso incremental y evaluable en cualquier momento del entrenamiento, lo que permite al sistema multiclasificador redirigir el entrenamiento para generar un conjunto de agentes que cumplan el cubrimiento de zonas.

6.2. Trabajos futuros

La realización de este trabajo permite establecer varias líneas de investigación y trabajos futuros en relación a los resultados obtenidos. Nos centraremos en el modelo MAD que es el modelo final obtenido en este trabajo. Comprobada la validez del modelo y los resultados obtenidos, hay varias medidas que se podrían revisar. La primera de ellas es la medida de calidad del agente, ya que la integración de la varianza y la entropía resulta demasiado simple. El efecto que provoca es que los agentes tiendan a desprenderse de datos de distinta clase para disminuir su entropía. Un estudio más detallado de diferentes formas de agregación de esta información podría mejorar el rendimiento de los agentes, permitiendo con más frecuencia la coexistencia de varias clases en un mismo agente.

El estudio de los efectos de los distintos modelos de asignación de datos a los agentes en el entrenamiento, así como el uso de información previa de zonas para inicializar los agentes, es otro de los trabajos futuros que puede servir para la mejora de su creación y una disminución del número de iteraciones en el proceso de descubrimiento de zonas de datos no analizados correctamente.

Otra cuestión, que en este trabajo no ha sido considerada en la creación y selección de los agentes, es el uso de la información conflictiva que pueden tener. Frente a una explotación de los agentes que son buenos clasificadores, olvidamos a los malos clasificadores, desaprovechando una información valiosa. La incorporación de esa información sobre el mal rendimiento de unos determinados agentes, para aumentar el rendimiento del sistema, es una línea de trabajo que quedaría abierta.

De igual forma, la incorporación de funciones de agregación tipo OWA en la selección del conjunto de agentes, podría generalizar el mecanismo de selección y agregación de los mismos, facilitando el uso de nuevos mecanismos y algoritmos específicos para dominios concretos.

Uno de los resultados más relevantes de este trabajo es la simplificación del conjunto de agentes. Este proceso se realiza en la fase final del multiclasificador para reducir el número de agentes, mejorando la eficiencia

del sistema multiclasificador. El estudio de los efectos de aplicar esta reducción, ya no sólo en la fase final, sino en cada una de las fases del algoritmo, permitirá observar si se producen mejoras en el rendimiento temporal del sistema de aprendizaje al tratar con menos agentes en cada momento, así como observar como afecta esta reducción a la calidad de la clasificación. Dentro del mismo ámbito, se podría contemplar la posibilidad de definir un prototipo representante de cada una de las clases de equivalencia, en lugar de la selección de un miembro de de la misma.

En general, el potencial de este modelo se encuentra en definir un sistema multiclasificador en el que se siga explotando la capacidad de evaluación y adaptación de los agentes mientras se están entrenando. En este sentido, un cambio importante puede ser que los parámetros del modelo se adapten a cada agente, de forma que, no solo el agente configure su zona de especialización sino también los parámetros que afectan al resto de su entrenamiento y al uso que el sistema hace del agente.

Desde el punto de vista de la eficiencia del algoritmo propuesto, un desarrollo para que la ejecución en paralelo sea efectiva produciría una mejora del rendimiento en el aprendizaje de los multiclasificadores. La introducción de técnicas de multiproceso, tanto para su desarrollo en GPU's como en múltiples CPU's, puede reducir drásticamente los tiempos de proceso en la fase de aprendizaje, permitiendo una respuesta más interactiva, además de permitir aumentar drásticamente el tamaño del conjunto de datos. Desde este punto de vista, el estudio de la integración de diferentes de fuentes distribuidas, así como de sus sistemas multiclasificadores inducidos, abren un camino para una posible planificación jerárquica de multiclasificadores.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Apéndice A

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Apéndice

A

Sistemas multiclasificadores generados

A.1. Conjunto de agentes para D2C5

Sistema Multiclasificador para datos D2C5

Ag_0	P_0:['46.748', '47.876'] V_0:['0.490', '0.742'] V_c:0.62 $\overline{V}$:0.62 E:0.00	
	C_2	P_{0_2}:['46.748', '47.876'] V:['0.490', '0.742']
	M	([46.9553228661, 46.7324297657], '2') ([45.8911037588, 48.6453779806], '2') ([45.8363994487, 48.6909053781], '2') ([47.6412371852, 48.2844370688], '2') ([47.476549936, 48.277906475], '2') ([46.6887496675, 46.6275349072], '2')

Conjunto de Agentes Clasificadores	
Ag_2	P_2:['10.766', '7.595'] V_2:['0.186', '0.182'] V_c:0.18 $\overline{V}$:0.18 E:0.00
	C_0: P_{2_0}:['10.766', '7.595'] V:['0.186', '0.182']
	M: ([10.639824886, 8.18751891158], '0') ([11.1027489546, 7.90287145395], '0') ([10.6852301558, 7.45795815123], '0') ([9.98630200439, 7.23047626542], '0') ([10.8083048927, 6.93938747657], '0') ([11.3752422982, 7.85059157659], '0')
Ag_8	P_8:['53.101', '51.412'] V_8:['0.483', '0.188'] V_c:0.34 $\overline{V}$:0.34 E:0.00
	C_2: P_{8_2}:['53.101', '51.412'] V:['0.483', '0.188']
	M: ([53.477843454, 51.2978717785], '2') ([52.3420984315, 50.580631613], '2') ([53.9595933569, 51.8697681787], '2') ([53.3783537735, 51.2750243069], '2') ([51.9884799682, 51.6877894867], '2') ([53.4603127681, 51.7602230612], '2')
Ag_{12}	P_12:['13.295', '8.509'] V_12:['0.217', '0.034'] V_c:0.13 $\overline{V}$:0.13 E:0.00
	C_0: P_{12_0}:['13.295', '8.509'] V:['0.217', '0.034']
	M: ([13.6361592551, 8.74140421627], '0') ([13.5519451638, 8.44406406106], '0') ([13.0148007663, 8.38309190693], '0') ([12.8774555485, 8.21091205528], '0') ([14.0018722351, 8.58088726774], '0') ([12.6884444265, 8.69337919513], '0')

Conjunto de Agentes Clasificadores		
Ag_{14}	P_14:['9.283', '11.733'] V_14:['0.191', '0.224'] V_c:0.21 $\overline{V}$:0.21 E:0.00	
	C_0	P_{14_0}:['9.283', '11.733'] V:['0.191', '0.224']
	M	([8.97395198023, 12.5059974675], '0') ([8.52311425551, 11.4374849174], '0') ([9.65393467733, 12.2294965649], '0') ([9.58547087878, 11.6629322613], '0') ([9.7691208628, 11.2793486749], '0') ([9.18967631054, 11.2805550803], '0')
Ag_{16}	P_16:['11.435', '90.306'] V_16:['0.104', '0.350'] V_c:0.23 $\overline{V}$:0.23 E:0.00	
	C_3	P_{16_3}:['11.435', '90.306'] V:['0.104', '0.350']
	M	([11.4458940887, 90.8049853652], '3') ([11.280571959, 90.6307324352], '3') ([10.9150884299, 89.7106874415], '3') ([11.7467944124, 90.2242094371], '3') ([11.9048134306, 91.0616207019], '3') ([11.3143659604, 89.4048204132], '3')
Ag_{17}	P_17:['92.207', '90.621'] V_17:['0.343', '0.370'] V_c:0.36 $\overline{V}$:0.36 E:0.00	
	C_4	P_{17_4}:['92.207', '90.621'] V:['0.343', '0.370']
	M	([92.0751681312, 91.3186095342], '4') ([92.3864859742, 90.6876941354], '4') ([92.4380481322, 89.7641226202], '4') ([93.0612760935, 91.4317091972], '4') ([91.1008295509, 90.4549294659], '4') ([92.1791942481, 90.0666832819], '4')

Conjunto de Agentes Clasificadores	
Ag_{18}	P_18:['51.724', '44.248'] V_18:['0.814', '0.459'] V_c:0.64 $\overline{V}$:0.64 E:0.00
	C_2: P_{18_2}:['51.724', '44.248'] V:['0.814', '0.459']
	M: ([50.8041486486, 44.4570265031], '2') ([52.7634652392, 43.9798420273], '2') ([50.4456076448, 43.5125738536], '2') ([52.7353000507, 44.5735372858], '2') ([51.4196376114, 43.5139926878], '2') ([52.1787995002, 45.4529271285], '2')
Ag_{19}	P_19:['45.281', '49.083'] V_19:['0.337', '0.550'] V_c:0.44 $\overline{V}$:0.44 E:0.00
	C_2: P_{19_2}:['45.281', '49.083'] V:['0.337', '0.550']
	M: ([44.8614539234, 50.5526867577], '2') ([44.7181783194, 48.2714765751], '2') ([45.8911037588, 48.6453779806], '2') ([45.8326602984, 49.4125267248], '2') ([45.8363994487, 48.6909053781], '2') ([44.5447348496, 48.9241388407], '2')
Ag_{23}	P_23:['88.470', '7.216'] V_23:['0.063', '0.186'] V_c:0.12 $\overline{V}$:0.12 E:0.00
	C_1: P_{23_1}:['88.470', '7.216'] V:['0.063', '0.186']
	M: ([88.3805420709, 7.69052527652], '1') ([88.9897497204, 7.08256966086], '1') ([88.4525299233, 7.16119719089], '1') ([88.1676278794, 7.79638555808], '1') ([88.3658273367, 6.50062216218], '1') ([88.4639085774, 7.06292825653], '1')

Conjunto de Agentes Clasificadores		
Ag_{24}	P_24:['51.748', '50.887'] V_24:['0.199', '0.271'] V_c:0.24 $\overline{V}$:0.24 E:0.00	
	C_2	P_{24_2}:['51.748', '50.887'] V:['0.199', '0.271']
	M	([51.0923156899, 50.3349295792], '2') ([52.3420984315, 50.580631613], '2') ([51.9884799682, 51.6877894867], '2') ([51.4470316059, 51.375644223], '2') ([52.1657410597, 50.3138914819], '2') ([51.4518540155, 51.0302294467], '2')
Ag_{27}	P_27:['91.407', '94.741'] V_27:['0.450', '0.339'] V_c:0.39 $\overline{V}$:0.39 E:0.00	
	C_4	P_{27_4}:['91.407', '94.741'] V:['0.450', '0.339']
	M	([90.227412894, 95.08871564], '4') ([90.7429144586, 94.8816800187], '4') ([91.7954089415, 93.5092852282], '4') ([91.8140326168, 95.063223533], '4') ([91.9824275652, 94.6472684228], '4') ([91.8785788939, 95.2571486543], '4')

A.2. Conjunto de agentes para D2C4 1c2z

Sistema Multiclasificador para datos D2C4 1c2z		
Ag_0	P_0:['10.731', '8.411'] V_0:['0.273', '0.168'] V_c:0.22 $\overline{V}$:0.22 E:0.00	
	C_0	P_{0_0}:['10.731', '8.411'] V:['0.273', '0.168']
	M	([10.4203663053, 8.55915356996], '0') ([10.0606930512, 9.19905869779], '0') ([10.3461777423, 8.69616074745], '0') ([10.6321405812, 8.14358356391], '0') ([11.2253825834, 8.29810622059], '0') ([11.7134473474, 8.05987171922], '0') ([10.7171050684, 7.91976976373], '0')

Conjunto de Agentes Clasificadores	
Ag_8	P_8:['9.611', '10.433'] V_8:['0.073', '0.101'] V_c:0.09 $\overline{V}$:0.09 E:0.00
	C_0: P_{8_0}:['9.611', '10.433'] V:['0.073', '0.101']
	M: ([9.21580360647, 10.3395138633], '0') ([9.74363180426, 10.9728622738], '0') ([9.43483523447, 10.8204884718], '0') ([9.93397543145, 10.1123263696], '0') ([9.95483188839, 10.1059702673], '0') ([9.33175154033, 10.4551911338], '0') ([9.66293468605, 10.2235035604], '0')
Ag_{12}	P_12:['89.841', '89.112'] V_12:['0.110', '0.297'] V_c:0.20 $\overline{V}$:0.20 E:0.00
	C_0: P_{12_0}:['89.841', '89.112'] V:['0.110', '0.297']
	M: ([89.5955238815, 89.7402079675], '0') ([90.2215372952, 88.4380746368], '0') ([90.1684747087, 89.2143418898], '0') ([89.6912567579, 89.7182807254], '0') ([89.7719841867, 88.8623294298], '0') ([89.2691713057, 89.5004617915], '0') ([90.1667306833, 88.3114190517], '0')
Ag_{13}	P_13:['9.167', '7.399'] V_13:['0.154', '0.364'] V_c:0.26 $\overline{V}$:0.26 E:0.00
	C_0: P_{13_0}:['9.167', '7.399'] V:['0.154', '0.364']
	M: ([8.61822537564, 8.25989573398], '0') ([8.82455960172, 6.64683178559], '0') ([9.72154045137, 6.95302727578], '0') ([9.66181143122, 7.39129628984], '0') ([8.9630846902, 6.70287710644], '0') ([9.37287981154, 7.83733195456], '0') ([9.00856098655, 8.00318952068], '0')

Conjunto de Agentes Clasificadores

Ag_{16}	P_16:['54.798', '48.697'] V_16:['0.484', '0.403'] V_c:0.44 $\overline{V}$:0.44 E:0.00	
	C_2	P_{16_2}:['54.798', '48.697'] V:['0.484', '0.403']
	M	([55.9604352948, 48.7305303814], '2') ([54.8759815953, 49.4683583216], '2') ([54.242507634, 48.1279191116], '2') ([54.2232682287, 49.4479670049], '2') ([53.9620401295, 47.7228146053], '2') ([55.62262367, 49.1305122596], '2') ([54.6991162949, 48.2525642176], '2')
Ag_{17}	P_17:['9.901', '89.391'] V_17:['0.203', '0.192'] V_c:0.20 $\overline{V}$:0.20 E:0.00	
	C_3	P_{17_3}:['9.901', '89.391'] V:['0.203', '0.192']
	M	([9.21513752674, 89.6188412393], '3') ([10.0084931415, 88.8611384416], '3') ([10.4698640272, 89.9191638259], '3') ([9.31539332702, 89.804083679], '3') ([10.1096306432, 88.8638792661], '3') ([9.81712727274, 88.964358481], '3') ([10.372359042, 89.7084860169], '3')
Ag_{18}	P_18:['89.361', '11.623'] V_18:['0.061', '0.075'] V_c:0.07 $\overline{V}$:0.07 E:0.00	
	C_1	P_{18_1}:['89.361', '11.623'] V:['0.061', '0.075']
	M	([89.2631346126, 11.3183937385], '1') ([89.3914518761, 11.7567055367], '1') ([89.1484925394, 11.6041221925], '1') ([89.0495366942, 11.1211405724], '1') ([89.7996702147, 11.860511483], '1') ([89.2554204787, 11.8608484111], '1') ([89.6212886133, 11.8376896131], '1')

Conjunto de Agentes Clasificadores

Ag_{19}

P_19:['90.837', '87.869']
V_19:['0.326', '0.181']
V_c:0.25 $\overline{V}$:0.25 E:0.00

C_0 | P_{19_0}:['90.837', '87.869']
V:['0.326', '0.181']

M
([90.1667306833, 88.3114190517], '0')
([90.2456125308, 87.8799412759], '0')
([90.2215372952, 88.4380746368], '0')
([91.5776280236, 87.4461030985], '0')
([91.2261825821, 88.1959271927], '0')
([90.9585253889, 87.2795676655], '0')
([91.4612316974, 87.5342994845], '0')

Ag_{21}

P_21:['48.404', '48.473']
V_21:['0.325', '0.220']
V_c:0.27 $\overline{V}$:0.27 E:0.00

C_2 | P_{21_2}:['48.404', '48.473']
V:['0.325', '0.220']

M
([48.3576918999, 48.2442919485], '2')
([47.325714964, 48.2442584267], '2')
([47.9880473498, 48.0655673111], '2')
([49.049732376, 48.5740656842], '2')
([49.0493355954, 48.3925458909], '2')
([48.7190157553, 49.565139099], '2')
([48.340796291, 48.2261809094], '2')

A.3. Conjunto de agentes para D2C5 z2clases

Sistema Multiclasificador para datos D2C5 z2clases

Ag_0	P_0:['55.151', '47.730'] V_0:['0.150', '0.209'] V_c:0.18 $\overline{V}$:0.18 E:0.00	
	C_1	P_{0_1}:['55.151', '47.730'] V:['0.150', '0.209']
	M	([55.8328319287, 48.2065202511], '1') ([54.9043933878, 48.1738235404], '1') ([55.3660901285, 47.6026185135], '1') ([54.6991046556, 47.6668308245], '1') ([55.2820663493, 46.8429230893], '1') ([54.8207380248, 47.8896538456], '1')
Ag_4	P_4:['86.105', '13.162'] V_4:['0.339', '0.535'] V_c:0.44 $\overline{V}$:0.44 E:0.00	
	C_4	P_{4_4}:['86.105', '13.162'] V:['0.339', '0.535']
	M	([86.5015703459, 12.135831552], '4') ([86.223622527, 13.4502714125], '4') ([86.1680680825, 14.1955362788], '4') ([86.9789945369, 12.7151587508], '4') ([85.2288990552, 13.8696486249], '4') ([85.5313649973, 12.6074533975], '4')
Ag_7	P_7:['47.846', '52.120'] V_7:['0.342', '0.163'] V_c:0.25 $\overline{V}$:0.25 E:0.00	
	C_1	P_{7_1}:['47.846', '52.120'] V:['0.342', '0.163']
	M	([48.1568911301, 52.1954941045], '1') ([48.6412533004, 52.6269821174], '1') ([46.8532360238, 52.0252760069], '1') ([47.8126616848, 51.4157846574], '1') ([48.2115910992, 51.922712494], '1') ([47.3980461307, 52.5340458707], '1')

Conjunto de Agentes Clasificadores

Ag_8

P_8:['50.629', '50.076']
V_8:['0.280', '0.117']
V_c:0.20 $\overline{V}$:0.20 E:0.00

C_1 | P_{8_1}:['50.629', '50.076']
V:['0.280', '0.117']

M
([50.6648701078, 49.857070336], '1')
([51.6130740563, 50.1839707425], '1')
([50.3737795767, 50.5674538767], '1')
([50.3219783107, 50.3135205828], '1')
([49.9268757889, 49.4909032445], '1')
([50.8739332538, 50.0427728114], '1')

Ag_{25}

P_25:['11.614', '8.916']
V_25:['0.069', '0.128']
V_c:0.10 $\overline{V}$:0.10 E:0.00

C_0 | P_{25_0}:['11.614', '8.916']
V:['0.069', '0.128']

M
([11.6814208113, 8.6080315884], '0')
([11.7443448065, 9.29704961236], '0')
([11.4476885406, 9.01124170135], '0')
([11.6842746083, 8.707132573], '0')
([11.9857582494, 9.42466243384], '0')
([11.1421946897, 8.44896625736], '0')

Ag_{28}

P_28:['11.539', '88.713']
V_28:['0.049', '0.091']
V_c:0.07 $\overline{V}$:0.07 E:0.00

C_2 | P_{28_2}:['11.539', '88.713']
V:['0.049', '0.091']

M
([11.1469693116, 88.3319407875], '2')
([11.673523473, 88.3583981995], '2')
([11.6353951918, 88.8852538478], '2')
([11.4352879922, 88.6572606108], '2')
([11.8530926068, 89.1834026735], '2')
([11.4883947232, 88.8639092842], '2')

Conjunto de Agentes Clasificadores		
Ag_{32}	P_32:['82.741', '4.667'] V_32:['0.989', '0.185'] V_c:0.59 $\overline{V}$:0.59 E:0.00	
	C_3	P_{32_3}:['82.741', '4.667'] V:['0.989', '0.185']
	M	([81.4571262506, 4.79317626014], '3') ([82.1166730947, 4.48174653669], '3') ([84.0455254761, 5.07023231406], '3') ([81.8637528452, 5.28465251063], '3') ([83.8861554284, 4.02070370964], '3') ([83.0792281674, 4.35195378671], '3')

A.4. Conjunto de agentes para Cone Torus

Sistema Multiclasificador para datos Cone Torus

Ag_0	P_0:['3.855', '5.035'] V_0:['0.015', '0.010'] V_c:0.01 $\overline{V}$:0.01 E:-0.00	
	C_2	P_{0_2}:['3.855', '5.035'] V:['0.015', '0.010']
	M	([3.9097386, 4.976959], '2') ([3.9585281, 4.9811198], '2') ([3.8062239, 5.2054389], '2') ([3.6390149, 4.922783], '2') ([3.9621044, 5.0888096], '2')
Ag_3	P_3:['6.578', '6.541'] V_3:['0.037', '0.016'] V_c:0.03 $\overline{V}$:0.03 E:-0.00	
	C_3	P_{3_3}:['6.578', '6.541'] V:['0.037', '0.016']
	M	([6.3832117, 6.6382273], '3') ([6.8538501, 6.6590084], '3') ([6.5075631, 6.6126142], '3') ([6.7568553, 6.4668323], '3') ([6.387953, 6.3258495], '3')
Ag_4	P_4:['7.102', '7.436'] V_4:['0.033', '0.022'] V_c:0.03 $\overline{V}$:0.03 E:-0.00	
	C_3	P_{4_3}:['7.102', '7.436'] V:['0.033', '0.022']
	M	([7.0270292, 7.2801793], '3') ([7.1388878, 7.2878708], '3') ([6.7932218, 7.679427], '3') ([7.2626583, 7.4979229], '3') ([7.2882391, 7.4348223], '3')

Conjunto de Agentes Clasificadores

Ag_8	P_8:['8.053', '2.922'] V_8:['0.007', '0.003'] V_c:0.01 $\overline{V}$:0.01 E:0.61	
	C_3	P_{8_3}:['8.067', '2.913'] V:['0.001', '0.004']
	C_2	P_{8_2}:['8.031', '2.936'] V:['0.015', '0.001']
	M	([8.0531274, 2.9117289], '3') ([7.909734, 2.8986327], '2') ([8.1513045, 2.9741956], '2') ([8.1144162, 2.9926238], '3') ([8.034083, 2.8331834], '3')

Ag_9	P_9:['4.846', '5.743'] V_9:['0.029', '0.006'] V_c:0.02 $\overline{V}$:0.02 E:-0.00	
	C_2	P_{9_2}:['4.846', '5.743'] V:['0.029', '0.006']
	M	([5.0360977, 5.6938995], '2') ([4.6797044, 5.7926742], '2') ([4.8736606, 5.614154], '2') ([5.0188513, 5.8331223], '2') ([4.6219898, 5.7819808], '2')

Ag_{10}	P_10:['8.187', '2.168'] V_10:['0.009', '0.008'] V_c:0.01 $\overline{V}$:0.01 E:0.46	
	C_3	P_{10_3}:['8.041', '2.266'] V:['0.000', '0.000']
	C_2	P_{10_2}:['8.224', '2.144'] V:['0.004', '0.007']
	M	([8.1941004, 2.2459403], '2') ([8.0405941, 2.2655481], '3') ([8.2767904, 2.016279], '2') ([8.1298364, 2.1602591], '2') ([8.2946483, 2.1543313], '2')

Conjunto de Agentes Clasificadores	
Ag_{11}	$P_1$1:['3.666', '2.568'] $V_1$1:['0.006', '0.003'] V_c:0.00 $\overline{V}$:0.00 E:-0.00
	C_1 | P_{11_1}:['3.666', '2.568'] V:['0.006', '0.003']
	M | ([3.5454649, 2.6278953], '1') ([3.6400652, 2.5755502], '1') ([3.6751717, 2.6106941], '1') ([3.6721929, 2.5464273], '1') ([3.7953519, 2.4797522], '1')
Ag_{12}	$P_1$2:['1.564', '2.249'] $V_1$2:['0.093', '0.019'] V_c:0.06 $\overline{V}$:0.06 E:-0.00
	C_2 | P_{12_2}:['1.564', '2.249'] V:['0.093', '0.019']
	M | ([1.2071254, 2.1566809], '2') ([1.8098514, 2.1721824], '2') ([1.8276974, 2.110271], '2') ([1.8021354, 2.4898699], '2') ([1.1744105, 2.3179261], '2')
Ag_{16}	$P_1$6:['6.818', '4.427'] $V_1$6:['0.021', '0.007'] V_c:0.02 $\overline{V}$:0.01 E:0.46
	C_3 | P_{16_3}:['6.793', '4.402'] V:['0.023', '0.006']
	C_2 | P_{16_2}:['6.919', '4.526'] V:['0.000', '0.000']
	M | ([6.8703774, 4.4331323], '3') ([6.6062014, 4.4265519], '3') ([7.0006828, 4.4765722], '3') ([6.9191721, 4.5259658], '2') ([6.6954408, 4.2736786], '3')
Ag_{17}	$P_1$7:['2.284', '3.934'] $V_1$7:['0.019', '0.007'] V_c:0.01 $\overline{V}$:0.01 E:-0.00
	C_2 | P_{17_2}:['2.284', '3.934'] V:['0.019', '0.007']
	M | ([2.1883739, 4.0005539], '2') ([2.0973292, 3.8125502], '2') ([2.2752366, 3.9018136], '2') ([2.4882519, 4.0499348], '2') ([2.3716253, 3.9059353], '2')

Conjunto de Agentes Clasificadores

Ag_{18}	P_18:['8.706', '2.580'] V_18:['0.052', '0.066'] V_c:0.06 $\overline{V}$:0.06 E:-0.00	
	C_3	P_{18_3}:['8.706', '2.580'] V:['0.052', '0.066']
	M	([9.0193801, 2.7213107], '3') ([8.9378651, 2.6987017], '3') ([8.4844957, 2.9194507], '3') ([8.5915561, 2.2839986], '3') ([8.4969523, 2.2768632], '3')
Ag_{20}	P_20:['8.032', '4.098'] V_20:['0.079', '0.028'] V_c:0.05 $\overline{V}$:0.05 E:-0.00	
	C_2	P_{20_2}:['8.032', '4.098'] V:['0.079', '0.028']
	M	([7.990712, 3.8970466], '2') ([7.7217892, 4.1118234], '2') ([8.3572423, 4.1707392], '2') ([8.3536588, 3.9428363], '2') ([7.7354892, 4.3665027], '2')
Ag_{24}	P_24:['6.835', '3.451'] V_24:['0.023', '0.006'] V_c:0.01 $\overline{V}$:0.01 E:-0.00	
	C_3	P_{24_3}:['6.835', '3.451'] V:['0.023', '0.006']
	M	([6.9911024, 3.4579468], '3') ([6.9952639, 3.4008001], '3') ([6.6376861, 3.5906218], '3') ([6.8738841, 3.3692929], '3') ([6.6763513, 3.435897], '3')
Ag_{25}	P_25:['8.771', '4.803'] V_25:['0.037', '0.026'] V_c:0.03 $\overline{V}$:0.03 E:-0.00	
	C_3	P_{25_3}:['8.771', '4.803'] V:['0.037', '0.026']
	M	([8.8969354, 4.7930885], '3') ([8.5988516, 4.8990214], '3') ([8.488783, 4.9757367], '3') ([8.9909268, 4.5069047], '3') ([8.8814685, 4.8407031], '3')

Conjunto de Agentes Clasificadores	
Ag_{26}	$P_2$6:['4.369', '5.923'] $V_2$6:['0.019', '0.082'] V_c:0.05 $\overline{V}$:0.05 E:-0.00
	C_3 | P_{26_3}:['4.369', '5.923'] V:['0.019', '0.082']
	M | ([4.3732729, 5.5969289], '3') ([4.6255967, 6.4043327], '3') ([4.2804789, 6.0080734], '3') ([4.3328178, 5.6688904], '3') ([4.2340653, 5.9354573], '3')
Ag_{27}	$P_2$7:['2.454', '2.490'] $V_2$7:['0.021', '0.585'] V_c:0.30 $\overline{V}$:0.30 E:-0.00
	C_1 | P_{27_1}:['2.454', '2.490'] V:['0.021', '0.585']
	M | ([2.3007487, 2.6832279], '1') ([2.4095598, 1.4712458], '1') ([2.6506405, 3.3151118], '1')
Ag_{28}	$P_2$8:['8.722', '2.106'] $V_2$8:['0.000', '0.002'] V_c:0.00 $\overline{V}$:0.00 E:-0.00
	C_2 | P_{28_2}:['8.722', '2.106'] V:['0.000', '0.002']
	M | ([8.7099461, 2.0638757], '2') ([8.7331557, 2.1471811], '2')

A.5. Conjunto de agentes para Iris

Sistema Multiclasificador para datos IRIS

Ag_0	P_0:['4.575', '3.175', '1.375', '0.200'] V_0:['0.012', '0.002', '0.007', '0.000'] V_c:0.01 $\overline{V}$:0.01 E:-0.00	
	C_0	P_{0_0}:['4.575', '3.175', '1.375', '0.200'] V:['0.012', '0.002', '0.007', '0.000']
	M	([4.4, 3.2, 1.3, 0.2], '0') ([4.7, 3.2, 1.3, 0.2], '0') ([4.6, 3.1, 1.5, 0.2], '0') ([4.6, 3.2, 1.4, 0.2], '0')
Ag_{10}	P_10:['6.575', '3.050', '4.475', '1.400'] V_10:['0.012', '0.013', '0.007', '0.005'] V_c:0.01 $\overline{V}$:0.01 E:-0.00	
	C_1	P_{10_1}:['6.575', '3.050', '4.475', '1.400'] V:['0.012', '0.013', '0.007', '0.005']
	M	([6.6, 3.0, 4.4, 1.4], '1') ([6.7, 3.1, 4.4, 1.4], '1') ([6.6, 2.9, 4.6, 1.3], '1') ([6.4, 3.2, 4.5, 1.5], '1')
Ag_{16}	P_16:['6.375', '2.900', '5.575', '1.975'] V_16:['0.002', '0.015', '0.002', '0.032'] V_c:0.01 $\overline{V}$:0.01 E:-0.00	
	C_2	P_{16_2}:['6.375', '2.900', '5.575', '1.975'] V:['0.002', '0.015', '0.002', '0.032']
	M	([6.4, 2.8, 5.6, 2.1], '2') ([6.4, 3.1, 5.5, 1.8], '2') ([6.3, 2.9, 5.6, 1.8], '2') ([6.4, 2.8, 5.6, 2.2], '2')

A.6. Conjunto de agentes para Pendigits

Sistema Multiclasificador para datos IRIS

Ag_0

P_0:['31.727', '83.636', '66.909', '100.000', '85.091', '79.636', '52.000', '56.000', '93.455', '43.545', '99.182', '17.909', '52.273', '2.545', '0.000', '0.545']
V_0:['21.289', '9.322', '19.355', '0.000', '25.174', '12.231', '25.818', '11.636', '18.612', '9.521', '4.149', '7.901', '4.198', '7.157', '0.000', '0.975']
V_c:11.08 $\overline{V}$:11.08 E:-0.00

C_3 — P_{0_3}:['31.727', '83.636', '66.909', '100.000', '85.091', '79.636', '52.000', '56.000', '93.455', '43.545', '99.182', '17.909', '52.273', '2.545', '0.000', '0.545']
V:['21.289', '9.322', '19.355', '0.000', '25.174', '12.231', '25.818', '11.636', '18.612', '9.521', '4.149', '7.901', '4.198', '7.157', '0.000', '0.975']

M	([32.0, 87.0, 74.0, 100.0, 82.0, 75.0, 45.0, 53.0, 87.0, 43.0, 100.0, 18.0, 54.0, 0.0, 0.0, 3.0], '3')
	([30.0, 78.0, 64.0, 100.0, 90.0, 82.0, 62.0, 57.0, 95.0, 42.0, 100.0, 17.0, 52.0, 4.0, 0.0, 0.0], '3')
	([41.0, 82.0, 64.0, 100.0, 83.0, 77.0, 46.0, 53.0, 93.0, 42.0, 100.0, 19.0, 52.0, 6.0, 0.0, 0.0], '3')
	([33.0, 85.0, 69.0, 100.0, 78.0, 78.0, 53.0, 56.0, 100.0, 44.0, 93.0, 16.0, 48.0, 1.0, 0.0, 0.0], '3')
	([30.0, 83.0, 61.0, 100.0, 93.0, 81.0, 53.0, 59.0, 89.0, 48.0, 100.0, 19.0, 56.0, 0.0, 0.0, 0.0], '3')
	([30.0, 86.0, 65.0, 100.0, 84.0, 85.0, 58.0, 63.0, 100.0, 45.0, 98.0, 21.0, 51.0, 6.0, 0.0, 0.0], '3')
	([28.0, 82.0, 74.0, 100.0, 87.0, 77.0, 46.0, 51.0, 88.0, 48.0, 100.0, 18.0, 52.0, 0.0, 0.0, 0.0], '3')
	([27.0, 86.0, 70.0, 100.0, 77.0, 77.0, 50.0, 53.0, 96.0, 40.0, 100.0, 19.0, 52.0, 6.0, 0.0, 0.0], '3')
	([32.0, 88.0, 65.0, 100.0, 83.0, 76.0, 50.0, 55.0, 94.0, 38.0, 100.0, 12.0, 52.0, 0.0, 0.0, 1.0], '3')
	([26.0, 79.0, 69.0, 100.0, 92.0, 85.0, 56.0, 56.0, 90.0, 42.0, 100.0, 15.0, 51.0, 0.0, 0.0, 2.0], '3')
	([40.0, 84.0, 61.0, 100.0, 87.0, 83.0, 53.0, 60.0, 96.0, 47.0, 100.0, 23.0, 55.0, 5.0, 0.0, 0.0], '3')

Ag_{32}

P_{32}:['31.000', '74.273', '54.273', '100.000', '77.909', '75.091', '57.818', '40.455', '23.273', '12.545', '0.000', '6.909', '50.182', '5.364', '100.000', '0.727']
V_{32}:['7.273', '9.289', '28.198', '0.000', '10.083', '5.537', '13.421', '10.612', '20.744', '16.612', '0.000', '11.174', '0.512', '14.231', '0.000', '3.107']
V_c:9.42 $\overline{V}$:9.42 E:-0.00

C_2 — P_{32_2}:['31.000', '74.273', '54.273', '100.000', '77.909', '75.091', '57.818', '40.455', '23.273', '12.545', '0.000', '6.909', '50.182', '5.364', '100.000', '0.727']
V:['7.273', '9.289', '28.198', '0.000', '10.083', '5.537', '13.421', '10.612', '20.744', '16.612', '0.000', '11.174', '0.512', '14.231', '0.000', '3.107']

M	([36.0, 72.0, 47.0, 100.0, 78.0, 76.0, 55.0, 38.0, 17.0, 8.0, 0.0, 5.0, 50.0, 13.0, 100.0, 0.0], '2')
	([30.0, 76.0, 53.0, 100.0, 82.0, 74.0, 66.0, 39.0, 31.0, 12.0, 0.0, 3.0, 50.0, 3.0, 100.0, 0.0], '2')
	([30.0, 78.0, 62.0, 100.0, 81.0, 77.0, 59.0, 46.0, 25.0, 19.0, 0.0, 11.0, 50.0, 5.0, 100.0, 0.0], '2')
	([32.0, 73.0, 59.0, 100.0, 81.0, 71.0, 54.0, 36.0, 16.0, 9.0, 0.0, 2.0, 50.0, 2.0, 100.0, 0.0], '2')
	([30.0, 74.0, 60.0, 100.0, 77.0, 74.0, 54.0, 43.0, 19.0, 16.0, 0.0, 12.0, 52.0, 9.0, 100.0, 0.0], '2')
	([29.0, 71.0, 55.0, 100.0, 74.0, 75.0, 59.0, 39.0, 30.0, 9.0, 0.0, 8.0, 49.0, 0.0, 100.0, 2.0], '2')
	([26.0, 80.0, 57.0, 100.0, 72.0, 75.0, 54.0, 44.0, 24.0, 18.0, 0.0, 11.0, 50.0, 8.0, 100.0, 0.0], '2')
	([32.0, 72.0, 57.0, 100.0, 76.0, 71.0, 56.0, 35.0, 21.0, 7.0, 0.0, 4.0, 51.0, 0.0, 100.0, 6.0], '2')
	([32.0, 71.0, 54.0, 100.0, 82.0, 77.0, 62.0, 43.0, 24.0, 17.0, 0.0, 8.0, 50.0, 6.0, 100.0, 0.0], '2')
	([29.0, 72.0, 45.0, 100.0, 78.0, 78.0, 60.0, 41.0, 25.0, 12.0, 0.0, 4.0, 50.0, 6.0, 100.0, 0.0], '2')
	([35.0, 78.0, 48.0, 100.0, 76.0, 78.0, 57.0, 41.0, 24.0, 11.0, 0.0, 8.0, 50.0, 7.0, 100.0, 0.0], '2')

Ag_2

P_2:['21.545', '94.636', '0.000', '54.727', '11.364', '10.182', '61.727', '0.000', '99.636', '31.636', '97.818', '76.818', '53.636', '100.000', '8.455', '77.364']
V_2:['30.248', '10.959', '0.000', '6.926', '14.777', '7.603', '5.471', '0.000', '0.231', '4.231', '9.603', '12.512', '17.504', '0.000', '18.248', '16.413']
V_c:9.67 $\overline{V}$:9.67 E:-0.00

C_0 — P_{2_0}:['21.545', '94.636', '0.000', '54.727', '11.364', '10.182', '61.727', '0.000', '99.636', '31.636', '97.818', '76.818', '53.636', '100.000', '8.455', '77.364']
V:['30.248', '10.959', '0.000', '6.926', '14.777', '7.603', '5.471', '0.000', '0.231', '4.231', '9.603', '12.512', '17.504', '0.000', '18.248', '16.413']

M	([23.0, 97.0, 0.0, 53.0, 20.0, 6.0, 66.0, 0.0, 100.0, 35.0, 98.0, 84.0, 53.0, 100.0, 9.0, 81.0], '0')
	([23.0, 100.0, 0.0, 58.0, 8.0, 12.0, 58.0, 0.0, 99.0, 29.0, 100.0, 75.0, 56.0, 100.0, 14.0, 78.0], '0')
	([17.0, 94.0, 0.0, 57.0, 16.0, 10.0, 64.0, 0.0, 100.0, 32.0, 98.0, 80.0, 55.0, 100.0, 14.0, 73.0], '0')
	([26.0, 92.0, 0.0, 49.0, 10.0, 4.0, 61.0, 0.0, 100.0, 31.0, 95.0, 76.0, 50.0, 100.0, 4.0, 83.0], '0')
	([27.0, 97.0, 0.0, 57.0, 6.0, 14.0, 58.0, 0.0, 100.0, 30.0, 100.0, 72.0, 57.0, 100.0, 11.0, 74.0], '0')
	([12.0, 95.0, 0.0, 53.0, 8.0, 12.0, 60.0, 0.0, 99.0, 31.0, 100.0, 73.0, 56.0, 100.0, 7.0, 78.0], '0')
	([26.0, 92.0, 0.0, 54.0, 13.0, 12.0, 63.0, 0.0, 99.0, 30.0, 100.0, 74.0, 61.0, 100.0, 13.0, 80.0], '0')
	([16.0, 90.0, 0.0, 57.0, 13.0, 12.0, 62.0, 0.0, 100.0, 30.0, 90.0, 76.0, 45.0, 100.0, 3.0, 77.0], '0')
	([17.0, 100.0, 0.0, 56.0, 12.0, 10.0, 63.0, 0.0, 100.0, 36.0, 100.0, 81.0, 51.0, 100.0, 1.0, 79.0], '0')
	([19.0, 91.0, 0.0, 52.0, 10.0, 10.0, 63.0, 0.0, 99.0, 32.0, 100.0, 75.0, 56.0, 100.0, 7.0, 80.0], '0')
	([31.0, 93.0, 0.0, 56.0, 9.0, 10.0, 61.0, 0.0, 100.0, 32.0, 95.0, 79.0, 50.0, 100.0, 10.0, 68.0], '0')

Ag_{35}

P_{35}:['100.000', '100.000', '54.909', '83.636', '21.455', '57.182', '0.000', '27.818', '16.818', '0.000', '73.182', '3.636', '61.545', '24.727', '7.818', '14.455']
V_{35}:['0.000', '0.000', '13.719', '9.686', '9.884', '5.421', '0.000', '1.967', '15.058', '0.000', '12.149', '7.322', '19.702', '13.471', '9.421', '15.339']
V_c:8.32 $\overline{V}$:8.32 E:-0.00

C_6 — P_{35_6}:['100.000', '100.000', '54.909', '83.636', '21.455', '57.182', '0.000', '27.818', '16.818', '0.000', '73.182', '3.636', '61.545', '24.727', '7.818', '14.455']
V:['0.000', '0.000', '13.719', '9.686', '9.884', '5.421', '0.000', '1.967', '15.058', '0.000', '12.149', '7.322', '19.702', '13.471', '9.421', '15.339']

M	([100.0, 100.0, 62.0, 88.0, 26.0, 61.0, 0.0, 30.0, 14.0, 0.0, 68.0, 2.0, 63.0, 23.0, 12.0, 13.0], '6')
	([100.0, 100.0, 51.0, 85.0, 18.0, 58.0, 0.0, 28.0, 17.0, 0.0, 78.0, 1.0, 68.0, 20.0, 8.0, 12.0], '6')
	([100.0, 100.0, 56.0, 80.0, 23.0, 54.0, 0.0, 25.0, 19.0, 0.0, 77.0, 5.0, 70.0, 22.0, 11.0, 16.0], '6')
	([100.0, 100.0, 56.0, 83.0, 22.0, 57.0, 0.0, 29.0, 11.0, 0.0, 68.0, 0.0, 62.0, 24.0, 6.0, 11.0], '6')
	([100.0, 100.0, 56.0, 78.0, 24.0, 53.0, 0.0, 27.0, 11.0, 0.0, 74.0, 3.0, 56.0, 23.0, 6.0, 14.0], '6')
	([100.0, 100.0, 51.0, 87.0, 18.0, 60.0, 0.0, 29.0, 20.0, 0.0, 71.0, 2.0, 58.0, 28.0, 8.0, 17.0], '6')
	([100.0, 100.0, 60.0, 86.0, 26.0, 58.0, 0.0, 28.0, 16.0, 0.0, 73.0, 9.0, 56.0, 31.0, 3.0, 16.0], '6')
	([100.0, 100.0, 49.0, 86.0, 17.0, 59.0, 0.0, 28.0, 18.0, 0.0, 76.0, 6.0, 61.0, 29.0, 7.0, 17.0], '6')
	([100.0, 100.0, 56.0, 80.0, 23.0, 55.0, 0.0, 26.0, 22.0, 0.0, 76.0, 4.0, 62.0, 23.0, 10.0, 13.0], '6')
	([100.0, 100.0, 54.0, 85.0, 21.0, 57.0, 0.0, 27.0, 23.0, 0.0, 75.0, 7.0, 57.0, 29.0, 3.0, 23.0], '6')
	([100.0, 100.0, 53.0, 82.0, 18.0, 57.0, 0.0, 29.0, 14.0, 0.0, 69.0, 1.0, 64.0, 20.0, 12.0, 7.0], '6')

Conjunto de Agentes Clasificadores

Ag_{66}

$P_6$6:['59.636', '93.273', '55.727', '63.545', '85.545', '20.091', '12.909', '0.000', '2.545', '33.545', '78.636', '61.455', '99.818', '98.364', '10.273', '94.000']
$V_6$6:['225.140', '33.835', '92.380', '34.066', '93.884', '27.901', '34.264', '0.000', '38.066', '6.793', '102.959', '8.430', '0.331', '7.868', '74.017', '24.727']
V_c:50.29 $\overline{V}$:50.29 E:-0.00

C_8	P_{66_8}:['59.636', '93.273', '55.727', '63.545', '85.545', '20.091', '12.909', '0.000', '2.545', '33.545', '78.636', '61.455', '99.818', '98.364', '10.273', '94.000'] V:['225.140', '33.835', '92.380', '34.066', '93.884', '27.901', '34.264', '0.000', '38.066', '6.793', '102.959', '8.430', '0.331', '7.868', '74.017', '24.727']
M	([77.0, 83.0, 47.0, 72.0, 78.0, 27.0, 22.0, 0.0, 0.0, 30.0, 68.0, 60.0, 100.0, 100.0, 20.0, 90.0], '8') ([48.0, 96.0, 62.0, 65.0, 88.0, 27.0, 21.0, 0.0, 21.0, 33.0, 79.0, 67.0, 100.0, 100.0, 0.0, 85.0], '8') ([67.0, 99.0, 62.0, 62.0, 97.0, 13.0, 0.0, 0.0, 7.0, 34.0, 100.0, 59.0, 100.0, 100.0, 10.0, 93.0], '8') ([65.0, 93.0, 68.0, 73.0, 92.0, 23.0, 13.0, 0.0, 0.0, 38.0, 87.0, 65.0, 100.0, 100.0, 5.0, 98.0], '8') ([76.0, 97.0, 64.0, 62.0, 100.0, 15.0, 14.0, 0.0, 0.0, 35.0, 90.0, 59.0, 98.0, 100.0, 0.0, 98.0], '8') ([33.0, 90.0, 57.0, 56.0, 91.0, 20.0, 9.0, 0.0, 0.0, 33.0, 76.0, 64.0, 100.0, 100.0, 4.0, 96.0], '8') ([42.0, 99.0, 55.0, 61.0, 71.0, 24.0, 18.0, 0.0, 0.0, 30.0, 64.0, 61.0, 100.0, 94.0, 9.0, 100.0], '8') ([76.0, 100.0, 59.0, 66.0, 89.0, 23.0, 9.0, 0.0, 0.0, 31.0, 80.0, 62.0, 100.0, 96.0, 2.0, 86.0], '8') ([42.0, 83.0, 60.0, 53.0, 89.0, 11.0, 13.0, 0.0, 0.0, 33.0, 77.0, 57.0, 100.0, 92.0, 18.0, 100.0], '8') ([61.0, 96.0, 46.0, 68.0, 71.0, 22.0, 11.0, 0.0, 0.0, 34.0, 67.0, 63.0, 100.0, 100.0, 25.0, 93.0], '8') ([69.0, 90.0, 33.0, 61.0, 75.0, 16.0, 12.0, 0.0, 0.0, 38.0, 77.0, 59.0, 100.0, 100.0, 20.0, 95.0], '8')

Ag_{98}

$P_9$8:['51.000', '92.364', '9.727', '76.727', '47.909', '37.455', '36.000', '0.000', '1.000', '22.636', '56.000', '55.455', '100.000', '91.182', '31.182', '99.091']
$V_9$8:['52.545', '26.413', '102.198', '21.835', '26.810', '6.793', '70.182', '0.000', '4.364', '20.959', '48.727', '17.157', '0.000', '15.058', '121.421', '3.174']
V_c:33.60 $\overline{V}$:33.60 E:-0.00

C_8	P_{98_8}:['51.000', '92.364', '9.727', '76.727', '47.909', '37.455', '36.000', '0.000', '1.000', '22.636', '56.000', '55.455', '100.000', '91.182', '31.182', '99.091'] V:['52.545', '26.413', '102.198', '21.835', '26.810', '6.793', '70.182', '0.000', '4.364', '20.959', '48.727', '17.157', '0.000', '15.058', '121.421', '3.174']
M	([55.0, 92.0, 17.0, 78.0, 55.0, 38.0, 50.0, 0.0, 0.0, 21.0, 56.0, 55.0, 100.0, 92.0, 25.0, 100.0], '8') ([67.0, 100.0, 18.0, 81.0, 44.0, 39.0, 35.0, 0.0, 0.0, 28.0, 53.0, 57.0, 100.0, 88.0, 42.0, 94.0], '8') ([54.0, 83.0, 11.0, 68.0, 44.0, 32.0, 30.0, 0.0, 0.0, 27.0, 49.0, 59.0, 100.0, 90.0, 21.0, 100.0], '8') ([47.0, 100.0, 27.0, 81.0, 57.0, 37.0, 26.0, 0.0, 0.0, 23.0, 56.0, 53.0, 100.0, 90.0, 40.0, 98.0], '8') ([46.0, 92.0, 0.0, 75.0, 48.0, 39.0, 44.0, 0.0, 7.0, 19.0, 52.0, 57.0, 100.0, 92.0, 37.0, 100.0], '8') ([53.0, 93.0, 3.0, 71.0, 47.0, 34.0, 23.0, 0.0, 0.0, 29.0, 59.0, 63.0, 100.0, 100.0, 9.0, 98.0], '8') ([48.0, 94.0, 0.0, 76.0, 53.0, 39.0, 47.0, 0.0, 3.0, 14.0, 70.0, 46.0, 100.0, 88.0, 28.0, 100.0], '8') ([43.0, 84.0, 0.0, 77.0, 48.0, 37.0, 28.0, 0.0, 1.0, 25.0, 68.0, 57.0, 100.0, 97.0, 21.0, 100.0], '8') ([44.0, 93.0, 4.0, 83.0, 43.0, 42.0, 40.0, 0.0, 0.0, 16.0, 48.0, 52.0, 100.0, 86.0, 49.0, 100.0], '8') ([44.0, 95.0, 1.0, 72.0, 39.0, 36.0, 38.0, 0.0, 0.0, 24.0, 49.0, 57.0, 100.0, 90.0, 34.0, 100.0], '8') ([60.0, 90.0, 26.0, 82.0, 49.0, 39.0, 35.0, 0.0, 0.0, 23.0, 56.0, 54.0, 100.0, 90.0, 37.0, 100.0], '8')

Ag_{97}

$P_9$7:['46.364', '87.909', '98.727', '100.000', '86.455', '95.364', '36.364', '77.091', '59.909', '54.727', '93.636', '29.455', '53.636', '2.182', '0.000', '2.636']
$V_9$7:['30.413', '17.174', '5.471', '0.000', '40.612', '11.504', '42.231', '12.992', '37.719', '38.198', '31.140', '15.157', '19.868', '6.876', '0.000', '19.140']
V_c:20.53 $\overline{V}$:20.53 E:-0.00

C_5	P_{97_5}:['46.364', '87.909', '98.727', '100.000', '86.455', '95.364', '36.364', '77.091', '59.909', '54.727', '93.636', '29.455', '53.636', '2.182', '0.000', '2.636'] V:['30.413', '17.174', '5.471', '0.000', '40.612', '11.504', '42.231', '12.992', '37.719', '38.198', '31.140', '15.157', '19.868', '6.876', '0.000', '19.140']
M	([40.0, 88.0, 95.0, 100.0, 88.0, 100.0, 40.0, 77.0, 72.0, 61.0, 100.0, 33.0, 58.0, 7.0, 0.0, 0.0], '5') ([43.0, 87.0, 100.0, 100.0, 96.0, 93.0, 47.0, 72.0, 53.0, 48.0, 91.0, 34.0, 58.0, 2.0, 0.0, 0.0], '5') ([53.0, 91.0, 100.0, 100.0, 86.0, 100.0, 28.0, 86.0, 58.0, 64.0, 98.0, 32.0, 60.0, 0.0, 0.0, 1.0], '5') ([54.0, 88.0, 100.0, 100.0, 85.0, 94.0, 38.0, 80.0, 59.0, 59.0, 83.0, 29.0, 47.0, 1.0, 0.0, 0.0], '5') ([42.0, 79.0, 93.0, 100.0, 89.0, 96.0, 39.0, 78.0, 51.0, 43.0, 100.0, 33.0, 56.0, 3.0, 0.0, 0.0], '5') ([47.0, 88.0, 98.0, 100.0, 81.0, 93.0, 31.0, 76.0, 61.0, 54.0, 100.0, 31.0, 56.0, 7.0, 0.0, 0.0], '5') ([43.0, 87.0, 100.0, 100.0, 77.0, 98.0, 29.0, 77.0, 60.0, 59.0, 95.0, 32.0, 54.0, 4.0, 0.0, 0.0], '5') ([42.0, 96.0, 100.0, 100.0, 80.0, 98.0, 32.0, 76.0, 58.0, 60.0, 86.0, 28.0, 48.0, 0.0, 0.0, 14.0], '5') ([52.0, 87.0, 100.0, 100.0, 92.0, 95.0, 44.0, 77.0, 71.0, 51.0, 94.0, 22.0, 47.0, 0.0, 0.0, 5.0], '5') ([54.0, 84.0, 100.0, 100.0, 80.0, 88.0, 29.0, 72.0, 59.0, 49.0, 94.0, 23.0, 51.0, 0.0, 0.0, 1.0], '5') ([40.0, 92.0, 100.0, 100.0, 97.0, 94.0, 43.0, 77.0, 57.0, 54.0, 89.0, 27.0, 55.0, 0.0, 0.0, 8.0], '5')

Ag_{50}

$P_5$0:['0.000', '62.000', '42.909', '75.545', '80.364', '95.909', '100.000', '98.909', '91.364', '73.545', '83.000', '47.909', '75.545', '22.636', '81.000', '0.000']
$V_5$0:['0.000', '19.636', '37.174', '10.612', '23.686', '14.810', '0.000', '4.446', '9.686', '2.793', '13.636', '2.628', '4.248', '2.777', '15.636', '0.000']
V_c:10.11 $\overline{V}$:10.11 E:-0.00

C_1	P_{50_1}:['0.000', '62.000', '42.909', '75.545', '80.364', '95.909', '100.000', '98.909', '91.364', '73.545', '83.000', '47.909', '75.545', '22.636', '81.000', '0.000'] V:['0.000', '19.636', '37.174', '10.612', '23.686', '14.810', '0.000', '4.446', '9.686', '2.793', '13.636', '2.628', '4.248', '2.777', '15.636', '0.000']
M	([0.0, 63.0, 41.0, 77.0, 79.0, 94.0, 100.0, 100.0, 85.0, 74.0, 75.0, 48.0, 71.0, 22.0, 82.0, 0.0], '1') ([0.0, 71.0, 39.0, 79.0, 82.0, 100.0, 100.0, 95.0, 93.0, 71.0, 87.0, 46.0, 75.0, 22.0, 84.0, 0.0], '1') ([0.0, 65.0, 51.0, 79.0, 84.0, 100.0, 100.0, 94.0, 92.0, 70.0, 83.0, 46.0, 73.0, 22.0, 78.0, 0.0], '1') ([0.0, 64.0, 43.0, 73.0, 79.0, 91.0, 100.0, 100.0, 92.0, 75.0, 84.0, 49.0, 78.0, 23.0, 74.0, 0.0], '1') ([0.0, 54.0, 47.0, 68.0, 83.0, 90.0, 100.0, 100.0, 93.0, 74.0, 86.0, 48.0, 76.0, 23.0, 79.0, 0.0], '1') ([0.0, 58.0, 38.0, 78.0, 73.0, 98.0, 100.0, 100.0, 92.0, 74.0, 84.0, 49.0, 77.0, 23.0, 80.0, 0.0], '1') ([0.0, 64.0, 46.0, 77.0, 81.0, 100.0, 100.0, 99.0, 96.0, 72.0, 88.0, 45.0, 74.0, 19.0, 85.0, 0.0], '1') ([0.0, 64.0, 36.0, 72.0, 73.0, 90.0, 100.0, 100.0, 89.0, 74.0, 82.0, 47.0, 78.0, 21.0, 76.0, 0.0], '1') ([0.0, 63.0, 55.0, 75.0, 84.0, 97.0, 100.0, 100.0, 91.0, 75.0, 81.0, 50.0, 77.0, 25.0, 81.0, 0.0], '1') ([0.0, 58.0, 42.0, 78.0, 90.0, 99.0, 100.0, 100.0, 87.0, 75.0, 78.0, 50.0, 76.0, 25.0, 84.0, 0.0], '1') ([0.0, 58.0, 34.0, 75.0, 76.0, 96.0, 100.0, 100.0, 95.0, 75.0, 85.0, 49.0, 76.0, 24.0, 88.0, 0.0], '1')

Conjunto de Agentes Clasificadores

Ag_{49}

$P_4$9:['34.455', '100.000', '18.545', '78.091', '0.000', '47.273', '37.000', '34.545', '96.000', '36.091', '98.273', '62.091', '84.636', '30.909', '78.545', '0.000']
$V_4$9:['39.884', '0.000', '18.975', '17.537', '0.000', '10.198', '31.818', '8.430', '22.182', '12.992', '7.107', '23.537', '13.868', '7.174', '35.702', '0.000']
V_c:15.59 $\overline{V}$:15.59 E:-0.00

C_4	P_{49_4}:['34.455', '100.000', '18.545', '78.091', '0.000', '47.273', '37.000', '34.545', '96.000', '36.091', '98.273', '62.091', '84.636', '30.909', '78.545', '0.000'] V:['39.884', '0.000', '18.975', '17.537', '0.000', '10.198', '31.818', '8.430', '22.182', '12.992', '7.107', '23.537', '13.868', '7.174', '35.702', '0.000']
M	([42.0, 100.0, 20.0, 80.0, 0.0, 47.0, 39.0, 38.0, 100.0, 37.0, 98.0, 66.0, 83.0, 31.0, 83.0, 0.0], '4') ([21.0, 100.0, 22.0, 74.0, 0.0, 45.0, 34.0, 36.0, 100.0, 33.0, 97.0, 62.0, 84.0, 31.0, 77.0, 0.0], '4') ([32.0, 100.0, 19.0, 74.0, 0.0, 42.0, 44.0, 31.0, 100.0, 44.0, 93.0, 66.0, 85.0, 33.0, 81.0, 0.0], '4') ([43.0, 100.0, 10.0, 75.0, 0.0, 45.0, 46.0, 34.0, 100.0, 35.0, 93.0, 60.0, 88.0, 29.0, 78.0, 0.0], '4') ([34.0, 100.0, 24.0, 84.0, 0.0, 51.0, 28.0, 36.0, 93.0, 35.0, 100.0, 66.0, 91.0, 32.0, 84.0, 0.0], '4') ([30.0, 100.0, 17.0, 87.0, 0.0, 52.0, 38.0, 28.0, 99.0, 34.0, 100.0, 71.0, 85.0, 36.0, 78.0, 0.0], '4') ([35.0, 100.0, 25.0, 77.0, 0.0, 49.0, 35.0, 34.0, 98.0, 31.0, 100.0, 56.0, 82.0, 27.0, 66.0, 0.0], '4') ([38.0, 100.0, 18.0, 81.0, 0.0, 52.0, 36.0, 38.0, 95.0, 42.0, 100.0, 60.0, 83.0, 31.0, 85.0, 0.0], '4') ([27.0, 100.0, 14.0, 76.0, 0.0, 44.0, 41.0, 33.0, 94.0, 35.0, 100.0, 63.0, 76.0, 34.0, 76.0, 0.0], '4') ([39.0, 100.0, 21.0, 77.0, 0.0, 47.0, 27.0, 37.0, 84.0, 36.0, 100.0, 60.0, 88.0, 29.0, 86.0, 0.0], '4') ([38.0, 100.0, 14.0, 74.0, 0.0, 46.0, 39.0, 35.0, 93.0, 35.0, 100.0, 53.0, 86.0, 27.0, 70.0, 0.0], '4')

Ag_{18}

$P_1$8:['99.545', '86.455', '63.727', '99.364', '45.273', '67.273', '87.455', '74.182', '92.091', '66.455', '86.636', '29.000', '55.273', '0.636', '0.000', '5.000']
$V_1$8:['2.066', '35.339', '28.017', '4.050', '32.017', '14.198', '37.702', '28.694', '18.083', '22.248', '30.959', '11.636', '40.198', '2.050', '0.000', '17.273']
V_c:20.28 $\overline{V}$:20.28 E:-0.00

C_9	P_{18_9}:['99.545', '86.455', '63.727', '99.364', '45.273', '67.273', '87.455', '74.182', '92.091', '66.455', '86.636', '29.000', '55.273', '0.636', '0.000', '5.000'] V:['2.066', '35.339', '28.017', '4.050', '32.017', '14.198', '37.702', '28.694', '18.083', '22.248', '30.959', '11.636', '40.198', '2.050', '0.000', '17.273']
M	([100.0, 100.0, 56.0, 93.0, 40.0, 65.0, 85.0, 78.0, 94.0, 70.0, 86.0, 32.0, 51.0, 5.0, 0.0, 0.0], '9') ([100.0, 91.0, 69.0, 100.0, 53.0, 63.0, 85.0, 70.0, 90.0, 63.0, 80.0, 23.0, 44.0, 0.0, 0.0, 7.0], '9') ([100.0, 80.0, 54.0, 100.0, 42.0, 64.0, 92.0, 80.0, 96.0, 76.0, 94.0, 35.0, 62.0, 0.0, 0.0, 8.0], '9') ([95.0, 88.0, 59.0, 100.0, 51.0, 67.0, 100.0, 81.0, 98.0, 65.0, 92.0, 28.0, 59.0, 0.0, 0.0, 4.0], '9') ([100.0, 89.0, 62.0, 100.0, 47.0, 67.0, 91.0, 79.0, 94.0, 67.0, 91.0, 31.0, 63.0, 0.0, 0.0, 3.0], '9') ([100.0, 78.0, 70.0, 100.0, 38.0, 76.0, 83.0, 75.0, 86.0, 59.0, 83.0, 27.0, 58.0, 0.0, 0.0, 11.0], '9') ([100.0, 90.0, 64.0, 100.0, 45.0, 68.0, 81.0, 70.0, 85.0, 61.0, 81.0, 27.0, 52.0, 0.0, 0.0, 9.0], '9') ([100.0, 84.0, 63.0, 100.0, 39.0, 67.0, 86.0, 70.0, 93.0, 71.0, 85.0, 33.0, 58.0, 0.0, 0.0, 2.0], '9') ([100.0, 85.0, 67.0, 100.0, 56.0, 66.0, 96.0, 79.0, 90.0, 65.0, 86.0, 28.0, 54.0, 0.0, 0.0, 11.0], '9') ([100.0, 80.0, 67.0, 100.0, 42.0, 73.0, 80.0, 70.0, 89.0, 64.0, 79.0, 25.0, 45.0, 1.0, 0.0, 0.0], '9') ([100.0, 86.0, 70.0, 100.0, 45.0, 64.0, 83.0, 64.0, 98.0, 70.0, 96.0, 30.0, 62.0, 1.0, 0.0, 0.0], '9')

Ag_{19}

$P_1$9:['20.818', '96.727', '20.273', '65.182', '57.364', '34.000', '26.000', '0.000', '0.091', '27.182', '2.818', '76.091', '43.091', '96.545', '100.000', '97.273']
$V_1$9:['12.694', '20.017', '30.926', '10.331', '18.595', '17.818', '19.818', '0.000', '0.083', '24.694', '9.967', '20.446', '12.992', '10.066', '0.000', '6.017']
V_c:13.40 $\overline{V}$:13.40 E:-0.00

C_5	P_{19_5}:['20.818', '96.727', '20.273', '65.182', '57.364', '34.000', '26.000', '0.000', '0.091', '27.182', '2.818', '76.091', '43.091', '96.545', '100.000', '97.273'] V:['12.694', '20.017', '30.926', '10.331', '18.595', '17.818', '19.818', '0.000', '0.083', '24.694', '9.967', '20.446', '12.992', '10.066', '0.000', '6.017']
M	([20.0, 96.0, 18.0, 65.0, 53.0, 30.0, 27.0, 0.0, 0.0, 32.0, 0.0, 75.0, 44.0, 100.0, 100.0, 96.0], '5') ([26.0, 97.0, 28.0, 70.0, 59.0, 35.0, 22.0, 0.0, 0.0, 23.0, 8.0, 71.0, 40.0, 100.0, 100.0, 100.0], '5') ([22.0, 100.0, 25.0, 63.0, 54.0, 26.0, 22.0, 0.0, 0.0, 37.0, 2.0, 82.0, 49.0, 96.0, 100.0, 95.0], '5') ([17.0, 100.0, 12.0, 65.0, 55.0, 42.0, 29.0, 0.0, 0.0, 20.0, 8.0, 74.0, 43.0, 98.0, 100.0, 100.0], '5') ([20.0, 91.0, 25.0, 67.0, 54.0, 32.0, 19.0, 0.0, 0.0, 32.0, 0.0, 80.0, 49.0, 92.0, 100.0, 100.0], '5') ([16.0, 100.0, 10.0, 59.0, 57.0, 35.0, 27.0, 0.0, 0.0, 29.0, 2.0, 86.0, 45.0, 96.0, 100.0, 96.0], '5') ([17.0, 100.0, 17.0, 69.0, 60.0, 36.0, 30.0, 0.0, 0.0, 20.0, 0.0, 73.0, 41.0, 91.0, 100.0, 92.0], '5') ([18.0, 100.0, 23.0, 69.0, 57.0, 39.0, 29.0, 0.0, 0.0, 27.0, 0.0, 75.0, 40.0, 94.0, 100.0, 99.0], '5') ([22.0, 94.0, 26.0, 62.0, 64.0, 31.0, 34.0, 0.0, 0.0, 27.0, 5.0, 72.0, 41.0, 100.0, 100.0, 97.0], '5') ([27.0, 100.0, 18.0, 65.0, 52.0, 36.0, 20.0, 0.0, 0.0, 27.0, 6.0, 77.0, 45.0, 95.0, 100.0, 96.0], '5') ([24.0, 86.0, 21.0, 63.0, 66.0, 32.0, 27.0, 0.0, 1.0, 25.0, 0.0, 72.0, 37.0, 100.0, 100.0, 99.0], '5')

Ag_{56}

$P_5$6:['69.000', '64.000', '54.909', '100.000', '1.909', '75.727', '54.364', '39.000', '77.091', '0.000', '3.455', '10.818', '35.909', '51.455', '100.000', '83.364']
$V_5$6:['94.909', '9.273', '103.537', '0.000', '9.355', '24.198', '38.413', '6.909', '53.174', '0.000', '21.521', '18.149', '56.992', '19.339', '0.000', '37.686']
V_c:30.84 $\overline{V}$:30.84 E:-0.00

C_8	P_{56_8}:['69.000', '64.000', '54.909', '100.000', '1.909', '75.727', '54.364', '39.000', '77.091', '0.000', '3.455', '10.818', '35.909', '51.455', '100.000', '83.364'] V:['94.909', '9.273', '103.537', '0.000', '9.355', '24.198', '38.413', '6.909', '53.174', '0.000', '21.521', '18.149', '56.992', '19.339', '0.000', '37.686']
M	([67.0, 66.0, 56.0, 100.0, 10.0, 74.0, 59.0, 38.0, 70.0, 0.0, 0.0, 15.0, 29.0, 54.0, 100.0, 82.0], '8') ([78.0, 67.0, 65.0, 100.0, 0.0, 75.0, 56.0, 37.0, 84.0, 0.0, 2.0, 10.0, 35.0, 52.0, 100.0, 86.0], '8') ([63.0, 65.0, 35.0, 100.0, 1.0, 69.0, 61.0, 36.0, 79.0, 0.0, 0.0, 15.0, 33.0, 54.0, 100.0, 86.0], '8') ([83.0, 63.0, 48.0, 100.0, 0.0, 76.0, 61.0, 40.0, 82.0, 0.0, 3.0, 9.0, 38.0, 55.0, 100.0, 91.0], '8') ([56.0, 63.0, 71.0, 100.0, 1.0, 81.0, 48.0, 42.0, 76.0, 0.0, 0.0, 13.0, 27.0, 56.0, 100.0, 86.0], '8') ([72.0, 68.0, 42.0, 100.0, 0.0, 68.0, 61.0, 34.0, 79.0, 0.0, 1.0, 16.0, 50.0, 53.0, 100.0, 88.0], '8') ([57.0, 68.0, 52.0, 100.0, 4.0, 71.0, 48.0, 37.0, 63.0, 0.0, 0.0, 13.0, 27.0, 49.0, 100.0, 78.0], '8') ([61.0, 59.0, 58.0, 100.0, 0.0, 84.0, 44.0, 42.0, 80.0, 0.0, 8.0, 5.0, 33.0, 44.0, 100.0, 70.0], '8') ([86.0, 59.0, 56.0, 100.0, 0.0, 81.0, 48.0, 42.0, 71.0, 0.0, 11.0, 8.0, 45.0, 52.0, 100.0, 88.0], '8') ([63.0, 62.0, 67.0, 100.0, 5.0, 79.0, 52.0, 41.0, 73.0, 0.0, 0.0, 2.0, 32.0, 42.0, 100.0, 75.0], '8') ([73.0, 64.0, 54.0, 100.0, 0.0, 75.0, 60.0, 40.0, 91.0, 0.0, 13.0, 13.0, 46.0, 55.0, 100.0, 87.0], '8')

Conjunto de Agentes Clasificadores

Ag_{58}

$P_5$8:['100.000', '98.727', '52.909', '99.818', '15.182', '80.909', '11.636', '48.636', '54.545', '51.727', '77.364', '24.727', '44.727', '0.636', '0.091', '4.545']
$V_5$8:['0.000', '1.653', '10.446', '0.149', '18.876', '6.810', '27.868', '8.595', '33.884', '13.653', '23.686', '6.380', '20.926', '2.050', '0.083', '15.339']
V_c:11.90 $\overline{V}$:11.90 E:-0.00

C_5	$P_{5_{8_2}}$:['100.000', '98.727', '52.909', '99.818', '15.182', '80.909', '11.636', '48.636', '54.545', '51.727', '77.364', '24.727', '44.727', '0.636', '0.091', '4.545'] V:['0.000', '1.653', '10.446', '0.149', '18.876', '6.810', '27.868', '8.595', '33.884', '13.653', '23.686', '6.380', '20.926', '2.050', '0.083', '15.339']

M	([100.0, 97.0, 48.0, 100.0, 12.0, 80.0, 9.0, 42.0, 51.0, 53.0, 80.0, 28.0, 49.0, 0.0, 0.0, 5.0], '5') ([100.0, 97.0, 56.0, 100.0, 22.0, 83.0, 18.0, 50.0, 53.0, 47.0, 68.0, 22.0, 37.0, 0.0, 0.0, 6.0], '5') ([100.0, 100.0, 54.0, 99.0, 15.0, 80.0, 20.0, 47.0, 64.0, 56.0, 80.0, 24.0, 46.0, 0.0, 0.0, 2.0], '5') ([100.0, 97.0, 49.0, 100.0, 11.0, 82.0, 10.0, 51.0, 54.0, 52.0, 76.0, 29.0, 47.0, 5.0, 0.0, 0.0], '5') ([100.0, 100.0, 54.0, 100.0, 17.0, 85.0, 15.0, 50.0, 55.0, 47.0, 85.0, 22.0, 50.0, 0.0, 0.0, 6.0], '5') ([100.0, 100.0, 48.0, 100.0, 9.0, 75.0, 8.0, 52.0, 61.0, 56.0, 85.0, 25.0, 49.0, 0.0, 0.0, 5.0], '5') ([100.0, 100.0, 54.0, 99.0, 16.0, 79.0, 0.0, 51.0, 48.0, 51.0, 76.0, 27.0, 47.0, 1.0, 1.0, 0.0], '5') ([100.0, 98.0, 54.0, 100.0, 13.0, 83.0, 11.0, 49.0, 53.0, 55.0, 73.0, 27.0, 46.0, 0.0, 0.0, 5.0], '5') ([100.0, 98.0, 57.0, 100.0, 23.0, 83.0, 12.0, 47.0, 43.0, 45.0, 77.0, 24.0, 45.0, 1.0, 0.0, 0.0], '5') ([100.0, 100.0, 51.0, 100.0, 11.0, 81.0, 9.0, 51.0, 57.0, 55.0, 73.0, 22.0, 37.0, 0.0, 0.0, 14.0], '5') ([100.0, 99.0, 57.0, 100.0, 18.0, 79.0, 16.0, 45.0, 61.0, 52.0, 78.0, 22.0, 39.0, 0.0, 0.0, 7.0], '5')

Ag_{29}

$P_2$9:['0.000', '92.455', '42.182', '100.000', '66.091', '71.182', '55.364', '32.455', '41.636', '0.000', '16.545', '32.727', '50.091', '41.909', '100.000', '43.545']
$V_2$9:['0.000', '18.612', '12.331', '0.000', '9.174', '7.603', '13.868', '15.702', '12.050', '0.000', '5.157', '12.744', '10.810', '8.810', '0.000', '12.612']
V_c:8.72 $\overline{V}$:8.72 E:-0.00

C_7	P_{29_7}:['0.000', '92.455', '42.182', '100.000', '66.091', '71.182', '55.364', '32.455', '41.636', '0.000', '16.545', '32.727', '50.091', '41.909', '100.000', '43.545'] V:['0.000', '18.612', '12.331', '0.000', '9.174', '7.603', '13.868', '15.702', '12.050', '0.000', '5.157', '12.744', '10.810', '8.810', '0.000', '12.612']

M	([0.0, 82.0, 37.0, 100.0, 65.0, 73.0, 54.0, 36.0, 42.0, 0.0, 16.0, 32.0, 48.0, 42.0, 100.0, 44.0], '7') ([0.0, 96.0, 40.0, 100.0, 60.0, 66.0, 58.0, 27.0, 41.0, 0.0, 20.0, 35.0, 54.0, 42.0, 100.0, 46.0], '7') ([0.0, 91.0, 40.0, 100.0, 65.0, 69.0, 62.0, 29.0, 48.0, 0.0, 16.0, 31.0, 51.0, 40.0, 100.0, 48.0], '7') ([0.0, 96.0, 42.0, 100.0, 66.0, 73.0, 55.0, 36.0, 44.0, 0.0, 20.0, 31.0, 50.0, 44.0, 100.0, 46.0], '7') ([0.0, 98.0, 44.0, 100.0, 69.0, 73.0, 52.0, 32.0, 37.0, 0.0, 16.0, 39.0, 51.0, 48.0, 100.0, 49.0], '7') ([0.0, 90.0, 47.0, 100.0, 65.0, 69.0, 48.0, 31.0, 37.0, 0.0, 14.0, 34.0, 49.0, 40.0, 100.0, 39.0], '7') ([0.0, 94.0, 41.0, 100.0, 71.0, 77.0, 53.0, 39.0, 42.0, 0.0, 15.0, 32.0, 47.0, 42.0, 100.0, 40.0], '7') ([0.0, 89.0, 40.0, 100.0, 64.0, 70.0, 55.0, 27.0, 38.0, 0.0, 15.0, 30.0, 57.0, 36.0, 100.0, 39.0], '7') ([0.0, 91.0, 40.0, 100.0, 64.0, 72.0, 54.0, 36.0, 45.0, 0.0, 13.0, 30.0, 45.0, 42.0, 100.0, 45.0], '7') ([0.0, 96.0, 43.0, 100.0, 70.0, 70.0, 59.0, 29.0, 39.0, 0.0, 19.0, 39.0, 52.0, 45.0, 100.0, 44.0], '7') ([0.0, 94.0, 50.0, 100.0, 68.0, 71.0, 59.0, 35.0, 45.0, 0.0, 18.0, 27.0, 47.0, 40.0, 100.0, 39.0], '7')

A.7. Conjunto de agentes para Wine

Sistema Multiclasificador para datos IRIS

Ag_0	P_0:['-0.825', '-0.865', '-1.480', '0.152', '-1.140', '-0.545', '-0.295', '-0.478', '-0.408', '-0.917', '0.363', '0.958', '-0.640'] V_0:['0.145', '0.032', '0.019', '0.083', '0.067', '0.210', '0.063', '0.022', '0.104', '0.222', '0.070', '0.196', '0.050'] V_c:0.10 $\overline{V}$:0.10 E:-0.00	
	C_1	P_{0_1}:['-0.825', '-0.865', '-1.480', '0.152', '-1.140', '-0.545', '-0.295', '-0.478', '-0.408', '-0.917', '0.363', '0.958', '-0.640'] V:['0.145', '0.032', '0.019', '0.083', '0.067', '0.210', '0.063', '0.022', '0.104', '0.222', '0.070', '0.196', '0.050']
	M	([-0.77, -1.09, -1.58, 0.07, -1.57, -0.27, -0.02, -0.68, -0.93, -0.15, 0.73, 1.23, -0.8], '1') ([-0.59, -0.6, -1.36, 0.36, -1.09, -0.13, -0.1, -0.27, -0.19, -0.95, 0.37, 1.36, -0.28], '1') ([-1.46, -0.83, -1.33, 0.46, -1.02, -0.47, -0.41, -0.44, -0.1, -1.4, -0.02, 1.02, -0.85], '1') ([-0.48, -0.94, -1.65, -0.28, -0.88, -1.31, -0.65, -0.52, -0.41, -1.17, 0.37, 0.22, -0.63], '1')

Ag_2	P_2:['0.490', '2.250', '0.068', '0.728', '-0.705', '-1.075', '-1.408', '1.452', '-0.740', '1.115', '-1.395', '-1.302', '-0.343'] V_2:['0.212', '0.194', '0.108', '0.058', '0.065', '0.086', '0.004', '0.172', '0.181', '0.051', '0.152', '0.041', '0.081'] V_c:0.11 $\overline{V}$:0.11 E:-0.00	
	C_2	P_{2_2}:['0.490', '2.250', '0.068', '0.728', '-0.705', '-1.075', '-1.408', '1.452', '-0.740', '1.115', '-1.395', '-1.302', '-0.343'] V:['0.212', '0.194', '0.108', '0.058', '0.065', '0.086', '0.004', '0.172', '0.181', '0.051', '0.152', '0.041', '0.081']
	M	([0.92, 2.86, 0.33, 0.36, -0.39, -0.95, -1.39, 1.37, -0.9, 1.22, -1.39, -1.26, -0.06], '2') ([0.94, 1.73, -0.36, 1.01, -0.88, -1.58, -1.52, 1.37, -0.75, 0.73, -0.78, -1.24, -0.77], '2') ([0.24, 2.46, -0.14, 0.85, -0.53, -0.85, -1.37, 2.11, -0.07, 1.31, -1.57, -1.63, -0.11], '2') ([-0.14, 1.95, 0.44, 0.69, -1.02, -0.92, -1.35, 0.96, -1.24, 1.2, -1.84, -1.08, -0.43], '2')

Ag_3	P_3:['0.490', '-0.555', '0.740', '-0.850', '-0.355', '0.695', '0.805', '-0.378', '0.003', '0.068', '0.485', '0.357', '1.498'] V_3:['0.100', '0.011', '0.131', '0.065', '0.119', '0.114', '0.082', '0.028', '0.070', '0.097', '0.154', '0.128', '0.068'] V_c:0.09 $\overline{V}$:0.09 E:-0.00	
	C_0	P_{3_0}:['0.490', '-0.555', '0.740', '-0.850', '-0.355', '0.695', '0.805', '-0.378', '0.003', '0.068', '0.485', '0.357', '1.498'] V:['0.100', '0.011', '0.131', '0.065', '0.119', '0.114', '0.082', '0.028', '0.070', '0.097', '0.154', '0.128', '0.068']
	M	([0.97, -0.6, 0.18, -1.08, -0.81, 0.5, 0.73, -0.52, 0.38, 0.27, 0.86, 0.4, 1.81], '0') ([0.39, -0.39, 1.15, -0.83, 0.1, 1.13, 1.19, -0.36, 0.12, 0.45, 0.51, 0.32, 1.65], '0') ([0.09, -0.67, 0.69, -0.44, -0.18, 0.26, 0.4, -0.52, -0.25, -0.35, 0.73, -0.15, 1.12], '0') ([0.51, -0.56, 0.94, -1.05, -0.53, 0.89, 0.9, -0.11, -0.24, -0.1, -0.16, 0.86, 1.41], '0')

Ag_7	P_7:['0.863', '0.618', '0.188', '0.728', '-0.478', '-0.933', '-1.278', '0.650', '-0.605', '1.700', '-1.382', '-1.395', '-0.290'] V_7:['0.175', '0.224', '0.074', '0.123', '0.114', '0.116', '0.001', '0.014', '0.053', '0.197', '0.076', '0.007', '0.025'] V_c:0.09 $\overline{V}$:0.09 E:-0.00	
	C_2	P_{7_2}:['0.863', '0.618', '0.188', '0.728', '-0.478', '-0.933', '-1.278', '0.650', '-0.605', '1.700', '-1.382', '-1.395', '-0.290'] V:['0.175', '0.224', '0.074', '0.123', '0.114', '0.116', '0.001', '0.014', '0.053', '0.197', '0.076', '0.007', '0.025']
	M	([0.53, 1.33, 0.44, 1.17, 0.1, -0.76, -1.25, 0.63, -0.3, 1.04, -1.13, -1.51, -0.03], '2') ([0.43, 0.74, 0.08, 0.69, -0.6, -0.55, -1.24, 0.79, -0.58, 1.55, -1.79, -1.43, -0.35], '2') ([1.01, 0.31, -0.21, 0.85, -0.74, -1.47, -1.32, 0.47, -0.95, 2.08, -1.13, -1.34, -0.46], '2') ([1.48, 0.09, 0.44, 0.2, -0.67, -0.95, -1.3, 0.71, -0.59, 2.13, -1.48, -1.3, -0.32], '2')

Ag_9	P_9:['-0.725', '0.760', '0.940', '0.605', '-0.460', '-0.915', '-0.205', '2.070', '-0.435', '-0.325', '0.350', '-0.520', '-0.640'] V_9:['0.664', '0.270', '1.166', '0.164', '0.397', '0.006', '0.046', '0.002', '0.235', '0.051', '0.740', '0.122', '0.023'] V_c:0.30 $\overline{V}$:0.30 E:-0.00	
	C_1	P_{9_1}:['-0.725', '0.760', '0.940', '0.605', '-0.460', '-0.915', '-0.205', '2.070', '-0.435', '-0.325', '0.350', '-0.520', '-0.640'] V:['0.664', '0.270', '1.166', '0.164', '0.397', '0.006', '0.046', '0.002', '0.235', '0.051', '0.740', '0.122', '0.023']
	M	([-1.54, 0.24, 2.02, 0.2, 0.17, -0.84, 0.01, 2.03, -0.92, -0.55, 1.21, -0.17, -0.49], '1') ([0.09, 1.28, -0.14, 1.01, -1.09, -0.99, -0.42, 2.11, 0.05, -0.1, -0.51, -0.87, -0.79], '1')

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Referencias Bibliográficas

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Referencias

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

[2] M. Aksela. Comparison of classifier selection methods for improving committee performance. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems: 4th international workshop, MCS 2003*, Lecture Notes in Computer Science, pages 184–93. Springer, 2003.

[3] E. Alpaydin. Voting over multiple condensed nearest neighbors. *Artificial Intelligence Review*, 11(1-5):115–132, 1997.

[4] F. Aurenhmmer. Voronoi diagrams - a survey of a fundamental geometric data structure. *Computing Surveys*, 23(3):345–405, September 1991.

[5] W. Baxt. Improving the accuracy of an artificial neural network using multiple differently trained networks. *Neural Computation*, 4(5):772–780, 1992.

[6] J. Bezdek, T. Reichherzer, G. Lim, and Y. Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 28(1):67–79, 1998.

[7] N.R. Paland J.C. Bezdek. On cluster validity for the fuzzy c-means model. *Ieee Transactions On Fuzzy Systems*, 3(3):370–379, August 1995.

[8] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. http://archive.ics.uci.edu/ml/, 1998.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–40, 1996.

[11] C. Brodley. Addressing the selective superiority problem: Automatic algorithm /model class selection. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 17–24, 1993.

[12] F .Brucker, F. Benites, and E. Sapozhnikova. An empirical comparison of flat and hierarchical performance measures for multi-label classification with hierarchy extraction. *Knowledge-based and Intelligent Information and Engineering Systems, Pt I: 15th International Conference, Kes 2011*, 6881:German Res Ctr Artificial Intelligence (DFKI) GmbH; KES Int; UnivEOLEOLKaiserslautern, Ctr Computat & Math Modeling (CM2); Univ Kaiserslautern,EOLEOLInst Integrated Sensor Syst; Univ

Kaiserslautern, Comp Sci Dept, ChairsEOLEOLKnowledge–Based Syst; Univ Kaiserslautern, Elect & Comp Engn Dept,EOLEOLIntegrated Sensor Syst, 2011.

[13] E. Castillo, D. Cerrillo, and L. Jimenez. Agentes clasificadores especialistas: Definición y uso en sistemas multiclasificadores. *Congreso Español de Metaheuriísticas, Algoritmos Evolutivos y Bioinspirados MAEB'04, Córdoba*, 2004.

[14] E. Castillo, M. Delgado, and L. Jimenez. Agentes clasificadores: Una nueva metodología para la construcción de sistemas multiclasificadores. *Primer congreso iberoamericano de algoritmos evolutivos y bioinspirados AEB'02, Mérida*, 2002.

[15] P. Chan and S. Stolfo. Experiments on multistrategy learning by meta-learning. In *In Proc. Second Intl. Conference on Info. and Knowledge Mgmt*, pages 314–323, 1993.

[16] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *Proceedings of the 12th International Conference on Machine Learning*, pages 90–98. Morgan Kaufmann, 1995.

[17] P. Chan and S. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8:5–28, 1996.

[18] C. Chang. Finding prototypes for nearest neighbor classiers. *EEE Transactions on Computers*, 23(11):1179–1184, 1974.

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

[20] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Geometric Learning Algorithms*, 13(1):21–27, 1967.

[21] B. Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *IEEE Computer Society Press*, 1991.

[22] P. Datta and D. Kibler. Learning symbolic prototypes. In *Proc. 14th International Conference on Machine Learning*, pages 75–82. Morgan Kaufmann, 1997.

[23] J. Deneubourg, S. Aron, S. Goss, J. Pasteels, and G. Duerinck. The dynamics of collective sorting: Robotlike ants and ant-like robots. In *In Proc. 1st Int. Conf. Simulation of Adaptive Behaviour*, pages 356–363, 1990.

[24] J. Deneubourg and N. Franks. Self -organizing nest construction in ants: individual worker behaviour and the nest's dynamics. In *Animat Bahavior*, volume 54, pages 779–796, 1997.

[25] L. Didaci and G. Giacinto. Dynamic classifier selection by adaptive k-nearest neighbor rule,â [online]. available: http://ce.diee.unica.it/en/publications/papers-prag/mcs-conference-19.pdf. In *MCS 2004, LNCS 3077*, pages 174–183, 2004.

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

[27] P. Domingos. Rule induction and instance-based learning: A unified approach. In *IJCAI*, pages 1226–1232, 1995.

[28] B. Efron. Bootstrap methods: another look at the jackknife. *The Annals os Statistics*, 7(1):1–26, 1979.

[29] S. P. Engelson and M. Koppel. Identifying the information contained in a flawed theory. In *ICML*, pages 131–138, 1996.

[30] M. Dos Santos Eulanda, R. Sabourin, and P. Maupin. A dynamic overproduce-and-choose strategy for the selection of classifier ensembles. *Pattern Recognition*, 41(10):2993–3009, 2008.

[31] B. Farhadinia and A.I. Ban. Developing new similarity measures of generalized intuitionistic fuzzy numbers and generalized interval-valued fuzzy numbers from similarity measures of generalized fuzzy numbers. *Mathematical and Computer Modelling*, 57(3-4):812–825, February 2013.

[32] Center for the Study of Complex Systems. University of Michigan. Swarm development group, 2012. `http://swarm.org/` [Online; Accedida el 18-February-2013].

[33] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann, 1996.

[34] G. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Computers*, 18:431–433, 1972.

[35] G. Giacinto. Adaptive selection of image classifiers. *Electronics Letters*, 36(5):420–422., 2000.

[36] G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9-10):699–707, 2001.

[37] G. Giacinto, F. Roli, and G. Fumera. Selection of classifiers based on multiple classifier behaviour. In *Proc. of the Joint international workshops on Syntactical and Structural Pattern Recognition & Statistical Pattern Recognition*.

[38] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996.

[39] M. Gressmann, G. Palm, and O. Lohlein. Surround view pedestrian detection using heterogeneous classifier cascades. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1317 –1324, oct. 2011.

[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. In *SIGKDD Explorations* [40].

[41] L. Hansen and P. Salomon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.

[42] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–16, 1968.

[43] N. Holden and A. Freitas. Improving the performance of hierarchical classification with swarm intelligence. *Evolutionary Computation, Machine Learning and Data Mining In Bioinformatics, Proceedings*, 4973:48–60, 2008.

[44] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[45] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *Ieee Transactions On Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002.

[46] J.M. Keller, M.R. Gray, and J.A. Gives. A fuzzy k-nearest neighbor algorithm. *Ieee Transactions On Systems Man and Cybernetics*, 15(4):580–585, 1985.

[47] T. Kenesei and J. Abonyi. Hinging hyperplane based regression tree identified by fuzzy clustering and its application. *Applied Soft Computing*, 13(2):782–792, February 2013.

[48] D. Kibler and D. Aha. Learning representative exemplars of concepts. In *Fourth International Workshop on Machine Learning*, pages 24–30, 1987.

[49] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598):671â680, 1983.

[50] J. Kittler. Fusion of information in pattern classification. In *Artificial Intelligence and Pattern Recognition Techniques for Computer Vision*, pages 1–8, 1998.

[51] J. Kittler. Pattern classification: Fusion of information. In S Singh, editor, *Proc. Intern. Conf. on Advances in Pattern Recognition*, pages 13–22, London, 1998. Springer.

[52] J. Kittler. Multiple expert fusion. In W Gaul and H Locarek-Junge, editors, *Studies in Classification, Data Analysis and Knowledge Organisation: Classification in the Information Age*, pages 21–28, Berlin, 1999. Springer.

[53] D. Klein, k. Toutanova, H.T. Ilhan, S.D Kamvar, and C.D. Manning. Combining heterogeneous classifiers for word-sense disambiguation. In *Proceedings of the ACL-02 Workshop on Word Sense Disambiguation: Recent Successes and Future Directions - Volume 8*, WSD '02, pages 74–80, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[54] A.H.R. Ko, R. Sabourin, and Jr. Alceu Souza Britto. From dynamic classifier selection to dynamic ensemble selection. *Pattern Recogn.*, 41:1718–1731, May 2008.

[55] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

[56] M. Kubat and M. Cooperson. Voting nearest-neighbor subclassifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 503–510. Morgan Kaufmann, San Francisco, CA, 2000.

[57] L.I Kubcheva. *Fuzzy Classifier Design*. Studies in Fuzziness and Soft Computing. Physica-Verlag HD, 2000.

[58] L.I Kubcheva. Artificial data, 2010. `http://pages.bangor.ac.uk/~mas00a/activities/artificial_data.htm` [Online; Accedida el 17-February-2013].

[59] L. I. Kuncheva and C.J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[60] L.I. Kuncheva. Clustering-and-selection model for classifier combination. pages 185–188, 2000.

[61] L.I. Kuncheva. Switching between selection and fusion in combining classifiers: an experiment. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(2):146–156, 2002.

[62] L.I. Kuncheva, J.C. Bezdek, and R.P.W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34:299–314, 2001.

[63] L.I. Kuncheva and L.C. Jain. Designing classifier fusion systems by genetic algorithms. *IEEE-EC*, 4(4):327, November 2000.

[64] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[65] E. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In MA MIT Press, Cambridge, editor, *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3*, page 501â508, 1994.

[66] E. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proc. of Third Intl. Conf. on Simulation of Adaptive Behavior*, pages 499–508. MIT Press., 1994.

[67] L.Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100:9–34, 1999.

[68] G. Manjunath, M.N. Murty, and D. Sitaram. A practical heterogeneous classifier for relational databases. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3316 –3319, aug. 2010.

[69] D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann, 1997.

[70] M. Meilă and M.I. Jordan. Learning fine motion by markov mixtures of experts. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1003–1009. The MIT Press, 1996.

[71] R. Meir. Bias, variance and the combination of least squares estimators. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 295–302. The MIT Press, 1995.

[72] C. J. Merz. Dynamical selection of learning algorithms. In D. Fisher and H. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics, 5*. Springer Verlag, 1995.

[73] C.J. Merz. *Classification and Regression by Combining Models*. PhD thesis, University of California, Irvine, 1998.

[74] N. Metropolis, A. Rosenbluth, M.N. Rosenbluth, A. Teller, and E.Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21 (6):1087, 1953.

[75] D. Mladenic and M. Grobelnik. Feature selection on hierarchy of web documents. *Decision Support Systems*, 35(1):PII S0167–9236(02)00097–0, April 2003.

[76] S. Venturini N.Monmarché. Antclass: discovery of clusters in numeric data by an hybridization of an ant colony with the kmeans algorithm.

[77] C. Pinciroli, V. Trianni, R. OâGrady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G.Caro, F.Ducatelle, M. Birattari, L. Gambardella, and M. Dorigo. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6:271–295, 2012.

[78] Andreas L. Prodromidis, Philip K. Chan, and Salvatore J. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. In *Advances of Distributed Data Mining*. AAAI Press, 2000.

[79] Andreas L. Prodromidis and Salvatore J. Stolfo. A comparative evaluation of meta-learning strategies over large and distributed data sets. In *In Workshop on Meta-learning, Sixteenth Intl. Conf. Machine Learning*, pages 18–27. Stefan Institute, 1999.

[80] Andreas L. Prodromidis and Salvatore J. Stolfo. Minimal cost complexity pruning of meta-classifiers. In *Proc. Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, page 979, 1999.

[81] J.R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1996.

[82] T.A. Runkler. Merging partitions using similarities of anchor subsets. *Synergies of Soft Computing and Statistics For Intelligent Data Analysis*, 190:CajAstur Bank; Univ Konstanz; Univ Magdeburg; German Soc Comp Sci;EOLEOLEuropean Soc Fuzzy Logic & Technol, 2013.

[83] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[84] D. Ruta and B. Gabrys. Classifier selection for majority voting. *Information Fusion*, 6(1):63–81, 2005.

[85] M. Sabzekar and M. Naghibzadeh. Fuzzy c-means improvement using relaxed constraints support vector machines. *Applied Soft Computing*, 13(2):881–890, February 2013.

[86] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276, 1991.

[87] E.R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

[88] R.E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann, 1997.

[89] B.W. Silverman and M.C. Jones. Fix,e, and hodges,jl, (1951) - an important contribution to nonparametric discriminant-analysis and density-estimation - commentary on fix and hodges (1951). *International Statistical Review*, 57(3):233–247, December 1989.

[90] D.B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proc. 11th International Conference on Machine Learning*, pages 293–301. Morgan Kaufmann, 1994.

[91] S. Y. Sohn and H. W. Shin. Experimental study for the comparison of classifier combination methods. *Pattern Recogn.*, 40:33–40, January 2007.

[92] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. The MIT Press, 1996.

[93] S. Stolfo, A.L. Prodromidis, S. Tselepis, Wenke Lee, Wei Fan, and Philip K. Chan. Jam: Java agents for meta-learning over distributed databases. In *In Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, pages 74–81. AAAI Press, 1997.

[94] A. X. Sun, E. P. Lim, W. K. Ng, and J. Srivastava. Blocking reduction strategies in hierarchical text classification. *Ieee Transactions On Knowledge and Data Engineering*, 16(10):1305–1308, October 2004.

[95] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 4th edition, 2008.

[96] V.r Tresp and M. Taniguchi. Combining estimators using non-constant weighting functions. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 419–426. The MIT Press, 1995.

[97] D. Wettschereck and TG. Dietterich. An experimental comparison of nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning*, 19(1):5–28, 1995.

[98] D. Wilson. Asymptotic properties of nearest neighbour rules using edited data. *IEEE Transactions on Systems, Man and Cybernetics*, 2:408–421, 1972.

[99] D.R. Wilson and T.R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

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

[101] D.H. Wolpert. Combining generealizers using partitions of the learning set. In L. Nadel and D. Stein, editors, *Lectures in Complex Systems*. Addison-Wesley, 1993.

[102] K. Woods, W.P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-19(4):405–10, April 1997.

[103] X.L.L. Xie and G. Beni. A validity measure for fuzzy clustering. *Ieee Transactions On Pattern Analysis and Machine Intelligence*, 13(8):841–847, August 1991.

[104] R.R. Yager. Families of owa operators. *Fuzzy Sets and Systems*, 59(2):125 – 148, 1993.

[105] B. Yuan and R. Liu. Multiple classifiers combination by clustering and selection. *Information Fusion*, pages 163–168, 2001.

[106] L. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[107] L. Zadeh. Similarity relations and fuzzy orderings. *Information Sciences*, 3(2):177–&, 1971.

[108] L. Zadeh. Fuzzy logic equals computing with words. *Ieee Transactions On Fuzzy Systems*, 4(2):103–111, May 1996.

[109] Y. Zhang. Selecting typical instances in instance-based learning. In *Proceedings of the Ninth International Machine Learning Workshop*, pages 470–479, 1992.

[110] Y. Zhang and L. Zhu. Integration of heterogeneous classifiers for intrusion detection. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–145 –V5–147, aug. 2010.

[111] Y. Zhao, R. Schwartz, J. Sroka, and J. Makhoul. Hierarchical mixtures of experts methodology applied to continuous speech recognition. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 859–865. The MIT Press, 1995.

[112] Y. Zhao, R. Schwartz, J. Sroka, and J. Makhoul. Hierarchical mixtures of experts methodology applied to continuous speech recognition. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 859–865. The MIT Press, 1995.

Grupo de Investigación ORETO
Universidad de Castilla-La Mancha

Ciudad Real, Abril de 2013