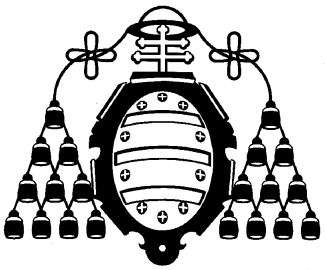
UNIVERSIDAD DE OVIEDO



Departamento de Informática

Aprendizaje de funciones de valoración a partir de ordenaciones

Tesis Doctoral

Autor
Jorge Díez Peláez

Directores
Antonio Bahamonde Rionda
Oscar Luaces Rodríguez
Juan José del Coz Velasco

Noviembre de 2002

Quiero dedicar este trabajo a mi familia, especialmente a Ana y a Guillermo, por el tiempo que no hemos podido pasar juntos

Agradecimientos

Debo agradecer al Centro de Inteligencia Artificial de la Universidad de Oviedo la colaboración prestada, especialmente a Antonio Bahamonde, Juan José del Coz y Oscar Luaces por su inestimable ayuda y a Jaime Alonso porque el paralelismo de nuestros trabajos ha logrado que sean mejores. Agradezco también al SERIDA, especialmente a Juan José Mangas la deferencia que ha tenido al suministrarme los datos necesarios para realizar el análisis de la calidad sensorial de la sidra. También debo agradecer al SERIDA que decidiese tenerme en plantilla durante parte de la realización de esta tesis en su sección de *mejora genética* (CENSYRA). A mis compañeros y jefe de sección (Félix Goyache) les agradezco mucho su ayuda en los buenos y en los malos momentos. Este trabajo ha sido financiado en parte por el Ministerio de Ciencia y Tecnología a través del proyecto de referencia TIC2001-3579, cofinanciado mediante el fondo FEDER.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Presentación

1.1. Introducción

El Aprendizaje Automático es una parte de la Inteligencia Artificial que se ocupa de conseguir que sistemas computerizados sean capaces de sintetizar el conocimiento necesario para desarrollar una tarea sin necesidad de interacción humana [Mitchell, 1997]. Esto se puede abordar *enseñando* al sistema mediante un **conjunto de ejemplos** que representa las pautas de comportamiento a seguir para la realización de dicha tarea y que suele denominarse conjunto de entrenamiento. En esta memoria nos centraremos en este tipo de aprendizaje a partir de ejemplos; de aprendizaje inductivo. Los tipos de problemas a los que se enfrenta suelen ser de dos tipos:

Simbólicos o de clasificación, cuando se trata de automatizar una tarea en la que el resultado está definido por una opción de un conjunto finito de alternativas. En este caso los ejemplos del conjunto de entrenamiento vendrán etiquetados con la opción a que dan lugar; el mecanismo de aprendizaje debe extraer el conocimiento necesario para elegir la alternativa adecuada en función de los valores de los atributos que describen los ejemplos.

Continuos o de regresión, en los que los sistemas deben aprender a valorar numéricamente objetos. El conjunto de entrenamiento de este tipo de problemas está compuesto por la descripción de los objetos y la valoración o calificación numérica que se les ha otorgado.

En esta memoria se presenta un nuevo sistema de aprendizaje denominado $\mathcal{LACE}$ ($\mathcal{L}$earning to $\mathcal{A}$ssess by $\mathcal{C}$omparisons $\mathcal{E}$xamples), que aprende a valorar objetos a partir de conjuntos de entrenamiento formados por descripciones de objetos que **no incluyen su valoración**, por carecer de ella. El aprendizaje se efectúa a partir de compa-

raciones entre **pares de objetos** que indican qué objeto es mejor, en algún sentido, en la comparación. La solución obtenida por el algoritmo será una función capaz de otorgar valoraciones numéricas a cualquier objeto del dominio. La función estará representada por un conjunto de reglas. La virtud de esta función es su coherencia con las comparaciones; es decir, asigna valores menores a los peores objetos y mayores a los mejores.

La peculiaridad de nuestro algoritmo reside en que parte de un tipo de conocimiento, pares de objetos ordenados, y trata de llegar a otro, reglas de regresión para valorar numéricamente esos mismos objetos. Lo habitual en los sistemas de aprendizaje es utilizar como *alimento* el mismo conocimiento que queremos aprender. En nuestro caso no es así, partimos de un conocimiento inferior (el orden) y tratamos de llegar a un conocimiento superior (la valoración).

1.2. Motivación

En esta sección comentaremos las circunstancias que nos han llevado a considerar que es necesaria la existencia de algoritmos capaces de aprender funciones de valoración a partir de comparaciones. La aplicación de este tipo de algoritmos tiene mucha importancia para resolver problemas de la vida real en los que se pretende valorar la calidad objetos sin tener una metodología de valoración diseñada.

Como se indica en [Díez et al., 2002b], la valoración de la calidad es una materia compleja: se pretende condensar todas las características deseables de un objeto en un solo número. Sin embargo, frecuentemente, este número no refleja estrictamente un valor absoluto, sino una calidad relativa del objeto respecto a otros de su misma especie. Esto es especialmente cierto en objetos de origen biológico; su calidad depende de un grupo, no siempre bien definido, de propiedades multisensoriales que son el resultado de su composición química, su estructura física, la interacción de ambas y la manera en que son percibidos por los sentidos humanos [Meilgaard et al., 1991]. Esta situación se vuelve más compleja cuando se considera el grado de calidad de los productos alimenticios desde el punto de vista de los expertos o de los consumidores. Al no existir una especificación detallada del grado de calidad, los expertos pueden adoptar un perfil de la misma que excede considerablemente el esperado por el consumidor [Bollen et al., 2002]. Los consumidores demandan que la calidad de los productos pueda ser percibida mediante los sentidos primarios a través de atributos simples. Esto da lugar a que la literatura refleje desacuerdo entre la valoración de la calidad por el consumidor o por el experto [Bollen et al., 2002][Goyache et al., 2001a].

Sin embargo, la industria alimenticia necesita abastecer los mercados con productos de calidad uniforme para satisfacer las demandas del consumidor de una calidad normalizada. Además, si es posible, los productores alimenticios querrían conocer cuáles son los parámetros principales (químicos y físicos) de la valoración de la calidad desde el punto de vista del consumidor, para así mejorar la aceptación de sus

productos.

La manera más sencilla de construir procedimientos automáticos para calcular la valoración de los objetos es hacerse con un conjunto de valoraciones representativas y aplicar un algoritmo de aprendizaje automático de funciones de regresión como `CUBIST` [Quinlan, 2002], `M5` [Quinlan, 1992], `M5'` [Wang y Witten, 1997], `Safe` (System to Acquire Functions from Examples) [Quevedo y Bahamonde, 1999] o `Bets` (Best Examples in Training Sets) [del Coz et al., 1999]. Sin embargo, nuestra experiencia con objetos de origen biológico [Goyache et al., 2001b] [Bahamonde et al., 2001] [Goyache et al., 2001a] [Albertí et al., 2002] [Díez et al., 2002a] [Díez et al., 2001a] [Díez et al., 2001b] [Díez et al., 2002c] [Quevedo et al., 2001] nos ha demostrado que la complejidad de la tarea de valoración hace que la capacidad para repetir las evaluaciones humanas tienda a ser baja. Por lo tanto, la fiabilidad del conjunto de entrenamiento es pobre a pesar de que los expertos hayan sido entrenados exhaustivamente y tengan una gran experiencia como calificadores [Kusabs et al., 1998].

Los expertos y los consumidores son perfectamente capaces de escoger un objeto frente a otro, sin embargo, suelen errar cuando se les pide que valoren un objeto con un número. Existe el llamado *efecto lote* que perjudica la valoración; los expertos calificadores tratan de puntuar los diferentes objetos con un sentido relativo, comparándolos con los restantes del lote. Por eso, un objeto rodeado de otros peores, probablemente tendrá una puntuación más alta que si se presenta rodeado de objetos mejores. Aunque se puede encontrar inaceptable la variabilidad individual en la valoración absoluta de la calidad de un objeto, la posición relativa obtenida en el lote es bastante constante. Este es uno de los aspectos que trata de explotar nuestro sistema.

1.3. Clasificación de los sistemas de aprendizaje

Antes de dar una introducción más detallada de las características principales del sistema de aprendizaje que se presenta en esta memoria, conviene situarlo en el lugar que le corresponde dentro de las distintas categorías de sistemas de aprendizaje automático existentes. Por ello es interesante conocer las diferentes clasificaciones que pueden establecerse entre los sistemas de aprendizaje atendiendo a diversos factores:

- **Dependiendo de si durante el entrenamiento se utiliza o no la información que se pretende predecir**:
  - *Aprendizaje supervisado.* Los sistemas que aplican este tipo de aprendizaje conocen la categoría que quieren predecir y guían el aprendizaje en la dirección adecuada haciendo comprobaciones periódicas de la calidad (supervisando) de la solución calculada hasta el momento.
  - *Aprendizaje no supervisado.* En este caso, a los sistemas no se les suministra la categoría de cada ejemplo de entrenamiento y por tanto no pueden supervisar el conocimiento inferido.

- **Dependiendo del tipo de los datos o indicaciones no estructuradas que el sistema emplee para realizar su labor de aprendizaje**. En este caso se pueden distinguir dos tipos de sistemas:
  - *Sistemas de aprendizaje a partir de ejemplos.* Esta clase de algoritmos, mediante procesos inductivos, extrae el conocimiento del problema a partir de las descripciones parciales sobre el dominio que constituye cada uno de los ejemplos. Los ejemplos o instancias estarán descritos por un conjunto de atributos que representan las variables fundamentales del problema.
  - *Sistemas de aprendizaje por analogía.* En este caso el conocimiento extraído será de tipo deductivo, basado en las relaciones de semejanza que se puedan establecer entre los elementos que representen el dominio de partida.

- **En función de la clase de problemas que pueden resolver**, o dicho de otra forma, basándose en los resultados que producen los algoritmos. De acuerdo a este criterio se puede diferenciar:
  - Los *sistemas clasificadores de categorías discretas*, es decir, aquellos que aprenden a clasificar los elementos del dominio en un número finito de clases simbólicas.
  - Los *sistemas que predicen valores numéricos continuos.* Estos sistemas generalmente sintetizan funciones que se encargan de relacionar las posibles entradas del dominio con las salidas numéricas correspondientes.

- **Dependiendo de los elementos que se empleen para expresar el conocimiento aprendido**:
  - Los *sistemas basados en redes de neuronas artificiales.* Este tipo de sistemas tratan de reproducir el funcionamiento de las redes de neuronas humanas, a pesar de lo cual producen un conocimiento que no es humanamente inteligible.
  - Los *algoritmos que expresan el conocimiento mediante un grupo de instancias o ejemplos.* Estos sistemas basados en el recuerdo de instancias, o IBL (*Instance-Based Learning*), término acuñado por Aha en [Aha, 1990] [Aha et al., 1991], memorizan prototipos de casos que se le suministran durante el entrenamiento y los emplean para predecir un comportamiento en situaciones similares.
  - Los *sistemas que sintetizan un conocimiento inteligible a través de generalizaciones simbólicas.* Habitualmente suelen emplear dos tipos de estructuras para representar el conocimiento que extraen: o bien *árboles de decisión*, formados por nodos con las condiciones que se deben cumplir y con las predicciones en sus hojas; o bien *reglas de clasificación*, expresadas mediante

cláusulas de lógica proposicional donde las premisas son condiciones sobre los atributos de los ejemplos y el consecuente la predicción a realizar si se cumplen todas esas premisas.

- **De acuerdo al mecanismo de evaluación que emplee el sistema para aplicar el conocimiento sintetizado**:
  - *Por ajuste exacto*, de forma que para realizar una predicción han de cumplirse todas las condiciones. Si lo expresamos en términos de reglas, deben ser ciertas todas las premisas de la regla para poder aplicarla y predecir la clase que aparezca en su consecuente.
  - *Por mínima distancia*. En este caso no es necesario que se cumplan todas las premisas, sino que es suficiente con que sean las condiciones más próximas, según la métrica utilizada, a los valores de entrada.

- **Según el grado de legibilidad** para el ser humano del conocimiento aprendido. En este sentido podemos distinguir dos tipos de sistemas:
  - Los algoritmos que, aunque resuelven eficazmente los problemas planteados, sintetizan un *tipo de conocimiento que no resulta comprensible para la mente humana*. La utilidad de estos sistemas puede ser aceptable para resolver determinados tipos de problemas, sin embargo, no resultan convenientes cuando los problemas a los que se aplican requieren explicaciones sobre las decisiones que toman.
  - Los sistemas en los que la representación del conocimiento que inducen es *humanamente inteligible*. El grado de utilidad de estos sistemas es lógicamente mayor, ya que no solamente servirán para realizar sus tareas de predicción habituales, sino que también pueden resultar provechosos para enseñar a resolver el problema que el sistema ha aprendido y para ofrecer explicaciones sobre cada una de las decisiones que adoptan.

- Por último, se pueden clasificar los sistemas de aprendizaje **de acuerdo a la geometría de las regiones de decisión que generan**:
  - Regiones de decisión definidas como *hiperrectángulos en el espacio de atributos* (lados son paralelos a los ejes).
  - Regiones formadas por lados oblicuos, es decir, representables mediante *polígonos irregulares multidimensionales* (hiperpolígonos irregulares).
  - *Áreas de geometría irregular*. Las regiones de decisión de este tipo estarán delimitadas por contornos irregulares.

1.4. Características fundamentales de $\mathcal{LACE}$

En la sección anterior se han mostrado diferentes aspectos a los que se puede atender para clasificar sistemas de aprendizaje. En esta sección se tratará de enmarcar $\mathcal{LACE}$, el algoritmo objeto de esta memoria, en las clases que le correspondan, en función de sus características.

Tipo de aprendizaje

Nuestro sistema no encaja dentro de la definición de aprendizaje supervisado ni no supervisado. El conjunto de entrenamiento que se le suministra a $\mathcal{LACE}$ no contiene la valoración de los objetos, así que, teniendo en cuenta que el objetivo final del algoritmo es ofrecer funciones de valoración, no se trata de aprendizaje supervisado. Sin embargo, las comparaciones entre objetos dan una indicación de la función valoración que se debe aplicar a los objetos, pudiéndose además, comprobar la calidad de la solución mediante el número de comparaciones erradas, con lo que se puede supervisar el aprendizaje. Por tanto, diremos que $\mathcal{LACE}$ aplica un tipo de aprendizaje *semi-supervisado*.

Datos que emplea el sistema para realizar su labor de aprendizaje

$\mathcal{LACE}$ es un sistema de aprendizaje basado en ejemplos, puesto que extrae el conocimiento del problema a partir de conjuntos de ejemplos de entrenamiento. No obstante, cabe destacar que éstos no responden al esquema convencional (formado por la descripción del objeto y su categoría o valoración), sino que están formados por un conjunto de *comparaciones*. Cada comparación está formada por la descripción de dos objetos, sin incluir sus valoraciones, considerándose como *objeto peor*, o menos valorado, aquel cuya descripción se encuentra situada en la parte izquierda de la comparación. Más adelante (sección 3.1) se dará una explicación más detallada del formato de los conjuntos de entrenamiento que utiliza el sistema.

Clase de problemas que puede resolver

El algoritmo presentado resuelve problemas en los que se precisa dar una valoración a los objetos que se le presentan. En ese sentido puede considerarse un sistema de predicción de valores numéricos. Sin embargo, a diferencia de otros sistemas de este tipo, $\mathcal{LACE}$ no trata de encontrar, en primera instancia, valoraciones para los objetos lo más parecidas posible a las que darían los expertos. Dadas las características del conjunto de entrenamiento, formado por comparaciones, cada problema puede tener un número infinito de soluciones correctas, ya que existen infinitas funciones capaces de reproducir la ordenación que representa el conjunto. Lo que busca el algoritmo es obtener una función de valoración que minimice el error en la ordenación de objetos no vistos por el algoritmo durante el aprendizaje. La escala de las valoraciones producidas

por $\mathcal{LACE}$ pueden diferir, y seguramente lo harán, del rango de valores con el que valoraría un experto humano los objetos del dominio. El algoritmo puede usar un proceso posterior para adaptar el rango de valores que produce como valoraciones a la escala habitual con la que se califican los objetos del problema. Todas estas circunstancias se describirán mejor en la sección 3.3.

Elementos empleados para expresar el conocimiento aprendido

Este sistema representa el conocimiento sintetizado a través de generalizaciones simbólicas. Estas generalizaciones tienen la estructura de reglas de valoración, como la siguiente:

$$f(u) \leftarrow att_1 = v_1 \wedge att_2 = v_2 \wedge ... \wedge att_n = v_n \tag{1.1}$$

donde u representa el vector que describe las características del objeto presentado para su valoración. En esta estructura se pueden diferenciar dos partes: el consecuente, $f(u)$, que representa la función de valoración para la regla, y el conjunto de antecedentes, en forma conjuntiva, que representa un punto en el espacio de los atributos. El conjunto de antecedentes puede, eventualmente, estar vacío si el algoritmo considera que una única función es suficiente para la valoración de todos los objetos.

Mecanismo de aplicación del conocimiento sintetizado

Las reglas de valoración inducidas por $\mathcal{LACE}$ son aplicadas por el criterio de mínima distancia, es decir, para valorar un nuevo objeto, el sistema utilizará la función de valoración de la regla que se encuentre más próxima al objeto, en términos de distancia euclídea. Si consideramos las reglas de valoración como casos representativos de una determinada región del espacio de atributos, en la que el algoritmo genera una función de valoración adecuada, la aplicación por mínima distancia pretende determinar la *región* a la que pertenece el objeto para aplicarle la función correspondiente. La idea que subyace es que los expertos posiblemente emplearán distintas funciones de valoración dependiendo de la región del dominio a la que pertenezca el objeto. En la Figura 1.1 se muestra un ejemplo de la aplicación del criterio de mínima distancia.

Grado de legibilidad del conocimiento aprendido

Como ya se ha comentado, $\mathcal{LACE}$ representa el conocimiento mediante generalizaciones simbólicas, concretamente mediante reglas. Esto hace que la solución ofrecida por este sistema sea humanamente inteligible. Es más, dado que no se requiere de los expertos que nos proporcionen valoraciones numéricas, las reglas que $\mathcal{LACE}$ produce pueden ser analizadas por los mismos para estudiar su verosimilitud. En muchos casos estos análisis a posteriori ayudan a los propios expertos en aspectos como la unidad de criterios de valoración o la simplificación de los mismos. En un intento por conseguir que la solución sea más compacta y legible, $\mathcal{LACE}$ dispone de mecanismos para reducir el número de reglas producidas.

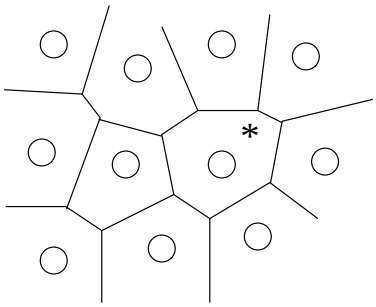


Figura 1.1: Regiones de decisión de las reglas durante la evaluación. En la figura se muestran las reglas (◯) y el ámbito de aplicación de las mismas (regiones de Voronoi). También se puede apreciar como el objeto ($*$) es clasificado por la regla central derecha al encontrarse situado en su región de decisión.

Geometría de las regiones de decisión generadas

Dado que $\mathcal{LACE}$ induce reglas que se corresponden con puntos en el espacio de atributos y que el mecanismo de evaluación se establece por mínima distancia, las regiones de decisión serán áreas de geometría irregular. Puede verse un ejemplo de las regiones de decisión que se producen en la Figura 1.1.

1.5. Objetivos de $\mathcal{LACE}$

Los objetivos principales que han guiado el diseño del algoritmo son fundamentalmente tres, que se enumeran a continuación:

- **Obtener un sistema de valoración en aquellos problemas en los que no hay una metodología definida al respecto o si la hay no es suficientemente fiable.** Aquellos problemas en los que se pretende aprender a valorar objetos y no existen expertos entrenados en la valoración ni una metodología adecuada no pueden ser abordados mediante técnicas de regresión ni mediante algoritmos de aprendizaje de categoría continua, puesto que no existe tal categoría. Estos problemas sí podrán ser resueltos por $\mathcal{LACE}$, ya que lo único que precisa es un conjunto de comparaciones sin valoraciones precisas. Partiendo de esto se puede obtener una escala de valoración mediante reglas bien definidas.
- **Sintetizar el conocimiento del problema mediante un pequeño conjunto de reglas de valoración.** Partiendo del conjunto de entrenamiento se tratará de encontrar regiones del espacio de ejemplos que agrupen objetos que puedan ser valorados correctamente con una única función.

- **Conseguir una eficacia aceptable.** Uno de los aspectos más importantes de los sistemas de aprendizaje es su eficacia, esto es, la capacidad que tienen para extraer el conocimiento necesario para resolver satisfactoriamente el problema planteado. La estimación del error cometido por $\mathcal{LACE}$ no puede estar directamente basada en el error en las valoraciones, ya que no se conocen las calificaciones reales de los objetos. En su lugar, la eficacia de $\mathcal{LACE}$ se ha de estimar en función del número de comparaciones efectuadas correctamente. Se entiende que si el sistema produce valoraciones que permiten ordenar correctamente el mayor número de objetos, estas mismas valoraciones, una vez adaptadas a la escala del problema, deben estar próximas a las que los objetos deben tener.

1.6. Estructura de la memoria

En el resto de esta memoria se describirán con detalle todas y cada una de las partes de las que consta $\mathcal{LACE}$. Comenzaremos comentado el estado del arte del aprendizaje a partir de comparaciones en el Capítulo 2. A continuación daremos una visión global del algoritmo en el Capítulo 3. Posteriormente, en el Capítulo 4, veremos como $\mathcal{LACE}$ busca la mejor ubicación en el espacio de atributos para sus reglas de valoración. Después de ubicar las reglas es necesario asignarles funciones de valoración; esto se verá en el Capítulo 5. Se buscarán después regiones del espacio de ejemplos en las que pueda utilizarse una única función de valoración, lo que permitirá eliminar reglas innecesarias, Capítulo 6. El Capítulo 7 mostrará cómo se nivelan las funciones de valoración de las reglas, consiguiendo que mejore la precisión del sistema. Los dos últimos capítulos servirán para mostrar los resultados experimentales obtenidos por este algoritmo así como las conclusiones que se pueden extraer de este trabajo. Para finalizar, se incluye un apéndice en el que se describe la complejidad temporal del algoritmo.

Capítulo 2

Otros enfoques del problema

2.1. Introducción

La mayor parte de las investigaciones en el campo del Aprendizaje Automático se centran en aprender a clasificar. Sin embargo hay ciertos problemas en los que es preferible *ordenar* a clasificar. Un ejemplo puede ser la *recuperación de información*; cuando se realiza una búsqueda por determinadas palabras, los documentos que cumplen el criterio de búsqueda deben presentarse al usuario en un orden. Este orden puede establecerse a partir de la solución aportada por un algoritmo de aprendizaje que usa un conjunto de entrenamiento con categorías continuas.

Una ventaja del aprendizaje a partir de comparaciones es que el conocimiento de partida es más fácil de obtener, siendo también más fiable. Evidentemente, es más sencillo decidir qué objeto es mejor comparando dos, que asignar una valoración a los mismos.

Por este motivo, varios autores han desarrollado algoritmos que aprenden a partir de comparaciones. En este capítulo comentaremos distintos enfoques propuestos para la solución de este tipo de problemas. Veremos que la meta principal de estos algoritmos es aprender a escoger entre dos alternativas, es decir, aprender a preferir. Este objetivo es diferente al nuestro, ya que lo que nosotros pretendemos es, partiendo de las comparaciones, sintetizar una o varias funciones que sean capaces de otorgar valoraciones a los objetos. Nuestro enfoque tiene, por tanto, más aplicaciones, puesto que podremos valorar cualquier objeto individualmente, con lo que conoceremos implícitamente si es preferible a otros o no.

2.2. Predicados de preferencia

En [Utgoff y Saxena, 1987] y [Utgoff y Clouse, 1991] se comenzó a hablar de los *predicados de preferencia*. Se comenta en estos artículos que en determinados problemas es más importante saber escoger entre dos alternativas que conocer su valoración absoluta. La valoración absoluta nos llevaría también a poder escoger la mejor alternativa, cogeríamos la que mayor valoración tenga. Sin embargo, resulta más sencillo calcular la valoración relativa de ambas alternativas entre sí. Opinan los autores que se puede encontrar una función de evaluación que satisfaga las condiciones impuestas mediante métodos estándar y los denominan *métodos de preferencia de estados* (state preference methods). Asumen que un estado x viene descrito por una conjunción de d características (todas ellas numéricas) representadas como un vector d-dimensional $F(x)$. Indican que la función de evaluación $H(x)$ se representa como una combinación lineal $W^T F(x)$, donde W es el vector de pesos. De esta manera, sabiendo que el estado C es mejor que el B, debe cumplirse que $H(C) > H(B)$. Los autores definen entonces el *predicado de preferencia* $P(x, y)$, que es cierto si y solo si $H(x) > H(y)$. Este predicado puede desarrollarse de la siguiente manera:

$$\begin{gathered} P(C, B) \\ H(C) > H(B) \\ W^T F(C) > W^T F(B) \\ W^T (F(C) - F(B)) > 0 \end{gathered}$$

Conociendo la diferencia entre ambos estados, lo único desconocido es el vector de pesos W. Realizando este desarrollo para todas las comparaciones conocidas, resulta un sistema de inecuaciones lineales. Así, para cada diferencia, el vector de pesos irá modificándose en un porcentaje hacia la dirección indicada por la diferencia.

2.3. El jugador artificial de Backgammon

Desde el punto de vista de esta memoria, el componente clave de este jugador es un mecanismo implementado como una red neuronal simétrica formada por dos subredes separadas, una para cada objeto de la comparación [Tesauro, 1989]. El objetivo de esta red neuronal es ser capaz de decidir entre dos movimientos cuál es el mejor. Tesauro fuerza a que ambas subredes tengan los mismos pesos excepto en la capa de salida, donde se multiplican por -1 (ver Figura 2.1). Esto le permite garantizar dos propiedades importantes para el problema:

- La comparación de dos posiciones *no debe ser ambigua*. Es decir, si se le presenta a la red la comparación (a, b) y a resulta mejor que b, ese mismo resultado debe darse si se le presenta la comparación (b, a).

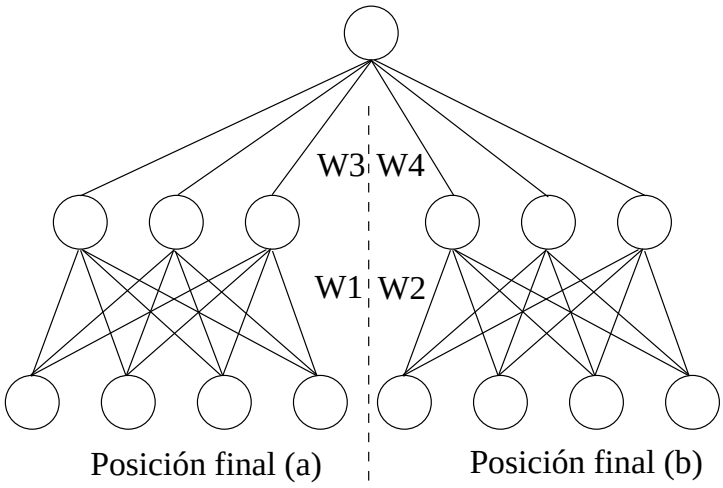


Figura 2.1: Red diseñada por Tesauro para jugar al Backgammon. El conocimiento de partida de esta red es la situación del tablero resultante tras realizar dos posibles movimientos. Estas dos situaciones se comparan y a la red se le indica que la situación a es mejor (más ventajosa) que la b. Los grupos de pesos tienen relaciones de simetría ($W_1 = W_2$). En la capa de salida los pesos tienen signos diferentes ($W_3 = -W_4$).

- Las comparaciones deben ser *transitivas*. Es decir, si la red concluye que a es mejor que b y que b es mejor que c, entonces debe concluir también que a es mejor c.

La capa de entrada de la red recibe la descripción de la situación del tablero resultante tras realizar dos posibles movimientos (a y b) en una partida de Backgammon. La capa de salida devolverá 1 si a es una mejor situación que b ó 0 si b es mejor que a. Si prescindiésemos de la capa de salida obtendríamos la valoración otorgada por la red a cada jugada. Esto es muy interesante, ya que partiendo de comparaciones es capaz de aprender una función de valoración. Sin embargo, únicamente es capaz de aprender *una* función.

2.4. 1ARC y CIBL

Ya en la década de los 90, se presentó el sistema 1ARC y varias evoluciones del mismo [Broos y Branting, 1994] [Branting y Broos, 1997]. Este algoritmo se basa en la idea de que para un usuario u, sus preferencias vienen dadas por la relación binaria P_u, tal que $P_u(S_1, S_2)$, que es cierta cuando u prefiere S_1 antes que S_2. Los autores tratan esta

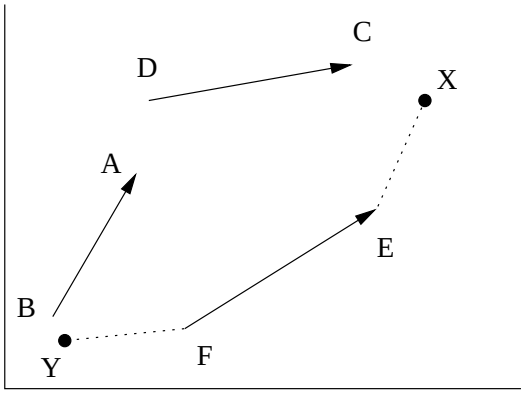


Figura 2.2: 1ARC escoge a X frente a Y porque la instancia más cercana es el arco $\overleftarrow{EF}$ que representa que E es preferible a F. La diferencia entre $\overleftarrow{XY}$ y $\overleftarrow{EF}$ es la suma de las distancias Euclídeas representadas por las líneas punteadas. Dado que X está más cerca de E e Y de F, entonces X se prefiere a Y.

comparación como un arco en el espacio de ejemplos que representa el vector diferencia entre S_1 y S_2. Todos los arcos resultantes de las comparaciones se proyectan en el espacio de ejemplos, con lo que ante una nueva comparación, aplicando una versión del *vecino más cercano*, se puede predecir la decisión tomada por el usuario. Por ejemplo, el conjunto de preferencias $P_u(A,B), P_u(C,D), P_u(E,F)$ puede representarse como se muestra en la Figura 2.2 mediante los arcos $\overleftarrow{AB}$, $\overleftarrow{CD}$ y $\overleftarrow{EF}$ (donde $\overleftarrow{AB} \equiv P_u(A,B)$).

Dado un nuevo par de objetos X e Y, la distancia de $\overleftarrow{XY}$ a $\overleftarrow{EF}$ es la suma de las distancias $dist(Y,F) + dist(X,E)$, calculando la distancia entre dos puntos mediante la métrica Euclídea. Se calculará la distancia de $\overleftarrow{XY}$ e $\overleftarrow{YX}$ respecto a todos los arcos. Si $\overleftarrow{XY}$ se encuentra más cerca de algún arco que $\overleftarrow{YX}$, entonces se preferirá X a Y; sino, Y será el objeto escogido.

Una extensión de 1ARC es el algoritmo CIBL. La diferencia entre ambos algoritmos consiste en que CIBL construye un camino entre los dos nuevos ejemplos que se presentan para su comparación conectando varios arcos. CIBL utiliza el algoritmo de Dijkstra [Aho et al., 1974] para encontrar el camino de mínimo coste, asumiendo que el camino desde el principio hasta el fin de un arco tiene coste cero y el resto de porciones que forman el camino tienen un coste igual a su distancia Euclídea.

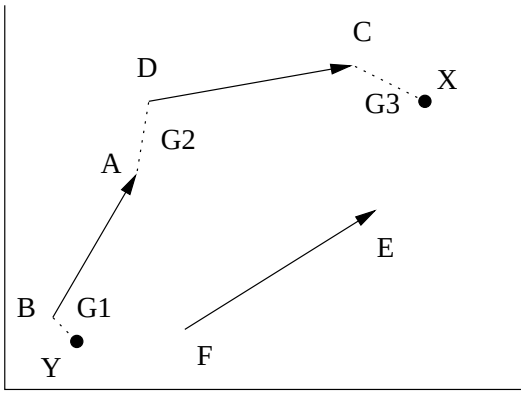


Figura 2.3: `CIBL` escoge a X frente a Y porque el camino de menor coste que los une es el formado por las instancias $\overleftarrow{AB}$ y $\overleftarrow{CD}$. Como Y está más próximo a B y X a C, entonces este sistema determina que X es mejor que Y.

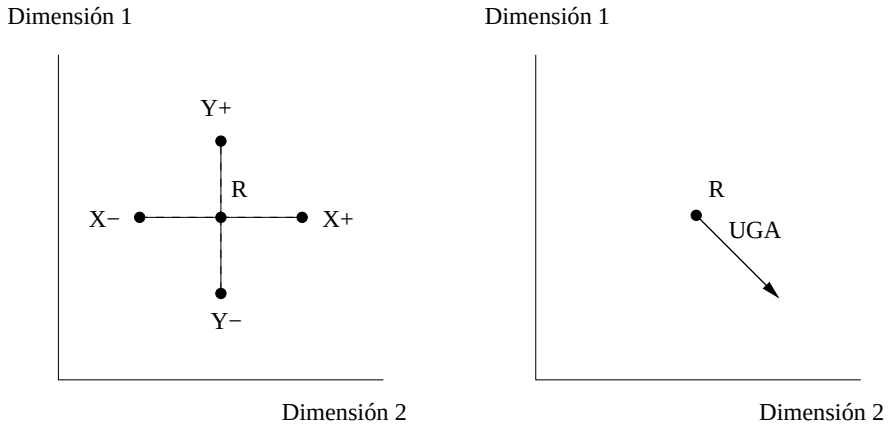


Figura 2.4: Cálculo de la UGA correspondiente al punto R. Se determina la relevancia y la polaridad de cada dimensión (imagen izquierda). Si el usuario indica $P_u(Y-, Y+)$ y $P_u(X+, X-)$, entonces la UGA resultante tiene la pendiente $(1, -1)$ partiendo de R (imagen derecha).

2.5. UGAMA

En [Branting, 1999] se presenta una nueva mejora del `1ARC`. La primera idea aportada en este artículo es la adquisición de las *Unit Gradient Approximations* (UGAs). Dado un punto R, se trata de ver la relevancia de los atributos y el signo de su pendiente. Para ello se crean para cada dimensión del problema dos nuevos puntos, que tendrán la misma situación que R pero para esa dimensión un punto tendrá un incremento δ y el otro un decremento de la misma cantidad. Se le pide al usuario que elija entre uno de esos dos puntos (objetos); esto nos dará la dirección (aumento o disminución) de ese atributo para aumentar la calidad de los objetos. Si el usuario no es capaz de decidirse por alguno de los dos puede considerarse ese atributo como irrelevante. En la Figura 2.4 podemos ver el punto R y las proyecciones correspondientes al aumentar o disminuir cada atributo. Si el usuario indica $P_u(Y-, Y+)$ y $P_u(X+, X-)$, entonces la UGA resultante tiene la pendiente $(1, -1)$ partiendo de R. El autor indica que *una única UGA* será suficiente como conjunto de entrenamiento para el `1ARC` si trata de predecirse el comportamiento de una función lineal. En el artículo se ve que los resultados de dicho sistema mejoran utilizando las UGAs. Aunque las UGAs pueden ser buenos representantes de la solución de un problema, es difícil construirlas en situaciones de la vida real. En la evaluación de la calidad sensorial de la sidra natural (que veremos en el capítulo de resultados) tenemos sidras descritas mediante 64 atributos. Partiendo de una sidra es imposible hacerla variar δ en cada atributo para calcular la pendiente del mismo, ya que la sidra se forma tras un proceso natural. Se podría intentar encontrar dos sidras que tuviesen 63 atributos en común y uno de ellos distinto, sin embargo la probabilidad de hallar sidras que cumplan esta circunstancia es casi nula. Otro inconveniente que presenta esta solución es que no todos los problemas pueden resolverse mediante una única función lineal.

Branting indica una solución para este inconveniente. Si la función que se trata de predecir no es lineal, en alguna parte aparecerá un punto de inflexión. Calculando varias UGAs podemos llegar a una situación como la que se presenta en la parte izquierda de la Figura 2.5. En [Branting, 1999] se comenta el algoritmo `UGAMA` que trata de solventar esta situación. Agrupa las UGAs con igual pendiente en clases, y después para cada una de ellas calcula la media de las UGAs que contiene. Con esto habrá realizado el paso *unión* en la figura. Posteriormente se alinean o nivelan las UGAs resultantes, enfrentándolas en la zona de inflexión.

La solución aportada en este artículo es una solución parcial, ya que aunque se acierten las comparaciones en las que ambos objetos estén al mismo lado del punto de inflexión, no se sabe qué sucederá cuando en una comparación se presente un objeto de cada lado.

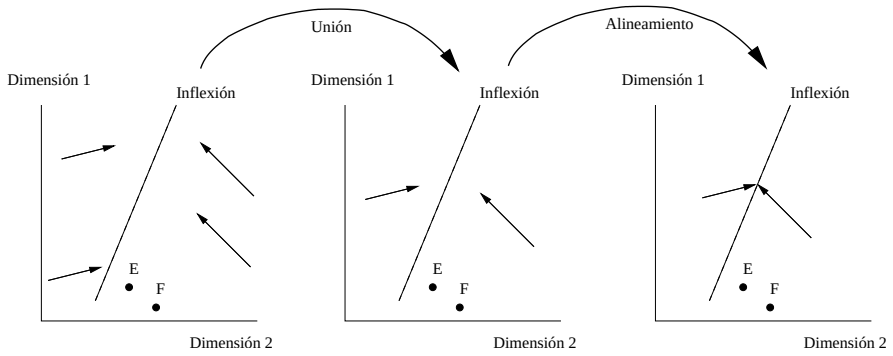


Figura 2.5: En la figura se ve un ejemplo del funcionamiento de UGAMA. En la parte izquierda, el par (E, F) sería clasificado erróneamente por encontrarse más cerca de UGAs de pendiente diferente. Se hace necesaria la unión de las UGAs con igual pendiente y una alineación posterior en los puntos de inflexión, para así delimitar correctamente el ámbito de aplicación de las mismas.

2.6. Combinando ordenaciones diferentes

En [Cohen et al., 1999], se presenta otro enfoque para aprender a ordenar objetos. Se define un predicado de preferencia $PREF(u, v)$ que devuelve una medida numérica (entre 0 y 1) que representa lo cierto que es que u es preferible a v. Este predicado será utilizado para comparar los objetos de un conjunto y, posteriormente, establecer un orden entre los mismos que se ajuste, lo más posible, a los resultados obtenidos al aplicar el predicado de preferencia a todos los pares posibles del conjunto de objetos. La calidad de un objeto viene dada por una función f que se utiliza para definir una *función de preferencia primitiva* R_f tal que

$$R_f(u, v) = \begin{cases} 1 & \text{si } f(u) > f(v) \\ 0 & \text{si } f(u) < f(v) \\ \frac{1}{2} & \text{en otro caso} \end{cases}$$

Veámoslo con un ejemplo. Si escribimos una palabra en un buscador de Internet, tendremos un listado de enlaces a páginas web que se nos muestran en un orden. En este caso la función f puede entenderse como la posición que ocupa la página en la ordenación. El valor de la primera página será -1, el de la segunda -2 y así sucesivamente. De esta manera, se cumplirá que $R_f(\textit{primera,segunda}) = 1$. Si utilizamos el mismo criterio de búsqueda en otro buscador, probablemente obtendremos otra ordenación, lo que dará lugar a otra función f. Si usamos n buscadores tendremos varias funciones de ordenación $(f_1, \ldots, f_n)$ que darán lugar a varias funciones de pre-

ferencia primitivas $(R_1, \ldots, R_n)$. Combinando estas últimas se obtiene la función de preferencia común para la búsqueda

$$PREF(u, v) = \sum_{i=1}^{n} w_i R_i(u, v)$$

donde w es un vector de pesos y w_i debe entenderse como la importancia que se otorga a R_i al construir la función de preferencia $PREF$. Ésta podrá ser utilizada para obtener una ordenación lo más coherente posible con todas las demás. El cálculo de esta nueva ordenación es un problema *NP*-duro, ya que la transitividad del predicado aprendido no está garantizada en absoluto. Si se muestra la nueva ordenación de páginas, generada con $PREF$, al usuario del buscador, es posible que visite en primer lugar la página colocada en tercera posición. Esto debe interpretarse como que la función de preferencia calculada *debería preferir la tercera página a la primera y a la segunda*. Estos fallos son utilizados para recalcular el vector de pesos w, lo que da lugar a una nueva función de preferencia.

2.7. Clasificación mediante Round Robin

Finalmente, en un trabajo reciente [Fürnkranz, 2002] se presenta una manera de solucionar problemas de categoría simbólica con múltiples clases mediante la construcción de varios clasificadores binarios que aprenden a diferenciar entre pares de clases. La apropiada combinación de todos ellos permite crear un clasificador único para todas las clases del problema. En ocasiones las dos clases empleadas para la síntesis de alguno de estos clasificadores binarios no se corresponden con las clases originales del problema. De hecho Fürnkranz expone diferentes métodos para la elaboración de los conjuntos de entrenamiento necesarios para la construcción de los clasificadores binarios.

El propio autor señala que los mecanismos teóricos de la clasificación basada en round robin no solamente están restringidos al aprendizaje de categorías discretas, sino que pueden tener relación con el aprendizaje a partir de comparaciones entre pares de objetos.

Capítulo 3

Visión general de $\mathcal{LACE}$

3.1. Conjunto de entrenamiento

Habitualmente, el conocimiento de partida utilizado por los sistemas de aprendizaje basados en ejemplos se estructura en conjuntos de entrenamiento en los que se describen las características de cada ejemplo. En caso de utilizar técnicas de aprendizaje supervisado la descripción de cada ejemplo contiene, además, el valor de la característica o características que se desean aprender a predecir. En este sentido se puede considerar que $\mathcal{LACE}$ es un algoritmo de aprendizaje *semi-supervisado*, puesto que los conjuntos de entrenamiento se componen de pares de ejemplos para los que no disponemos de la valoración de la característica a aprender, aunque si sabemos cual es, según el criterio de los expertos, el mejor y el peor valorado.

En la Tabla 3.1 se muestra la estructura típica de un conjunto de ejemplos tradicional para aprendizaje supervisado. Este conjunto contiene la descripción de los ejemplos y la clase (o valoración, si se trata de un problema de categoría continua) de cada uno de ellos.

En los problemas de calificación numérica, en los que un grupo de expertos valora los objetos, puede haber discrepancias en cuanto a la valoración *absoluta* que otorgan. Sin embargo, si los expertos pudieran hacer valoraciones relativas respecto a otros objetos no existirían dichas discrepancias. Aquí subyace el hecho de que para un experto es más sencillo ordenar una serie de objetos (valoración relativa) que otorgarles una puntuación (valoración absoluta). Aprovechando esta circunstancia, se construyen los conjuntos de entrenamiento de $\mathcal{LACE}$ formados por comparaciones entre pares de objetos. La información es más fiable ya que se prescinde del elemento que contiene mayor grado de inconsistencia, la calificación numérica absoluta. El esquema de este tipo de conjuntos puede verse en la Tabla 3.2.

	Atributos			
Ejemplos	Atributo_1	...	Atributo_n	**Clase**
1	$\text{valor}_{1,1}$	...	$\text{valor}_{1,n}$	clase_1
2	$\text{valor}_{2,1}$	...	$\text{valor}_{2,n}$	clase_2
...	...	...	...	...
m	$\text{valor}_{m,1}$	...	$\text{valor}_{m,n}$	clase_m

Tabla 3.1: Esquema general de un conjunto de entrenamiento tradicional. Cada ejemplo empleado para extraer el conocimiento de un problema está descrito por un conjunto de atributos y su correspondiente clase. En los problemas de predicción numérica o regresión, la clase es un número real. En los problemas de clasificación, la clase es una etiqueta de un conjunto finito de posibilidades.

	Atributos					
	Peor			**Mejor**		
Comparaciones	Atributo_1	...	Atributo_n	Atributo_1	...	Atributo_n
1	$\text{valor}_{1,1}$	...	$\text{valor}_{1,n}$	$\text{valor}_{i_1,1}$	...	$\text{valor}_{i_1,n}$
2	$\text{valor}_{2,1}$	...	$\text{valor}_{2,n}$	$\text{valor}_{i_2,1}$	...	$\text{valor}_{i_2,n}$
...	...	...	...	...	...	...
m	$\text{valor}_{m,1}$	...	$\text{valor}_{m,n}$	$\text{valor}_{i_m,1}$	...	$\text{valor}_{i_m,n}$

Tabla 3.2: Esquema general de un conjunto de entrenamiento de $\mathcal{LACE}$. Cada ejemplo es una comparación entre dos objetos. La parte izquierda del ejemplo será la descripción del objeto que se considera *peor* dentro de la comparación. En la parte derecha se colocará el objeto considerado *mejor*. No se tiene ninguna indicación de la valoración numérica de los objetos.

3.2. Motivación geométrica

El elemento más destacable de la información contenida en el conjunto de entrenamiento es la *relación entre la valoración de los dos objetos que integran la comparación*, que se registra explícitamente en el formato de los datos de entrada y da lugar a la justificación geométrica del algoritmo, que se expone en esta sección.

Sean u y v vectores que describen las características de dos objetos que los expertos han comparado, resultando que u es *peor* que v; en símbolos, $u < v$. Por tanto, se busca una función f tal que $f(u) < f(v)$. Si se asume que f tiene un comportamiento lineal, al menos en las cercanías de los vectores, se tiene que encontrar un vector w tal que

$$f_w(u) = u \cdot w < v \cdot w = f_w(v) \tag{3.1}$$

donde $\cdot$ denota el producto escalar entre vectores.

Desde el punto de vista geométrico, la función f_w representa la distancia al hiperplano $u \cdot w = 0$; es decir, el hiperplano de vectores perpendiculares a w. Si se busca un w considerando únicamente vectores normalizados (es decir $\|w\| = 1$), la diferencia más amplia entre los valores de $f_w(u)$ y $f_w(v)$ se alcanza cuando w es el vector normalizado en la dirección de $(v - u)$. De hecho,

$$\begin{aligned} f_w(v-u) &= (v-u)\cdot w \le \|v-u\|\cdot\|w\| = \|v-u\| = \\ &= \underbrace{\left(\frac{(v-u)}{\|v-u\|}\cdot\frac{(v-u)}{\|v-u\|}\right)}_{1}\cdot\|v-u\| = \\ &= (v-u)\cdot\frac{(v-u)}{\|v-u\|} = f_{\frac{(v-u)}{\|v-u\|}}(v-u) \end{aligned} \tag{3.2}$$

En general, se parte de una familia de *comparaciones*

$$\{u_i < v_i : i = 1, \ldots, n\} \tag{3.3}$$

y se quiere inducir una función f, con un comportamiento lineal en su ámbito de aplicación que, se espera, sea capaz de distinguir el mayor número de veces posible que u_i es peor que v_i, porque $f(u_i) < f(v_i)$. El algoritmo que se propone, utiliza la intuición geométrica introducida en esta sección como punto de partida para la búsqueda de la función f. Por tanto, la tarea principal del algoritmo será combinar las pautas locales sugeridas por cada comparación suministrada en el conjunto de entrenamiento.

El conjunto de entrenamiento, compuesto de comparaciones (3.3), da lugar a un conjunto de $2 \cdot n$ pares de vectores:

$$\left\{\left(\frac{v_i - u_i}{\|v_i - u_i\|}, u_i\right) : i = 1, \ldots, n\right\} \cup \left\{\left(\frac{v_i - u_i}{\|v_i - u_i\|}, v_i\right) : i = 1, \ldots, n\right\} \tag{3.4}$$

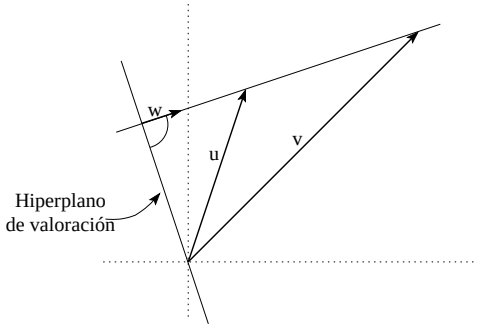


Figura 3.1: Dados dos objetos representados por los vectores u y v, si u es peor que v, el vector normalizado en la dirección de la diferencia, $w = \frac{v-u}{\|v-u\|}$, define un hiperplano. La distancia desde el hiperplano a los objetos es una función de valoración local válida.

Así, el par (w, u) debe entenderse como una sugerencia de regla de valoración indicando lo siguiente:

> **Si** z se encuentra en las proximidades de u
> **entonces** su valoración es $f_w(z)$.

Dado que $f_w(z) = z \cdot w$, habitualmente identificaremos w con la *función* lineal f_w. De la misma manera, consideraremos a u como la *condición* de la regla. Utilizando una notación abreviada, la regla antes mencionada será:

$$w \leftarrow u. \tag{3.5}$$

En general, intentaremos conseguir una función de valoración f definida por partes en todo el espacio de atributos. En otras palabras, la función de valoración final vendrá dada por una lista de reglas

$$(w_1 \leftarrow u_1), (w_2 \leftarrow u_2), \ldots, (w_m \leftarrow u_m), \tag{3.6}$$

que deben ser evaluadas mediante el criterio de mínima distancia. La función f que resulte finalmente inducida trabajará como se muestra a continuación, para un vector arbitrario z:

$$f(z) = w_k \cdot z, \quad \text{si } \|z - u_k\| \leq \|z - u_j\|, \forall j = 1, \ldots, m \tag{3.7}$$

3.3. Solución propuesta

Existen distintas técnicas capaces de aprender a valorar objetos a partir de un conjunto que contenga muestras de valoraciones. Con la regresión lineal se puede calcular la

función que más se ajusta los ejemplos suministrados; el objetivo será maximizar la correlación entre la valoración propuesta por la función y la valoración indicada para cada ejemplo. Sin embargo, no todos los problemas tiene soluciones lineales; en esas situaciones puede calcularse una estimación logarítmica capaz de obtener la recta o la curva exponencial que mejor se ajuste a los datos presentados. La aplicación de estas técnicas precisa un conocimiento, a priori, de las características del problema con el que no siempre se cuenta. Otra alternativa es emplear algoritmos de aprendizaje de reglas de regresión capaces de ofrecer buenas soluciones ante datos que definen funciones complejas. Estos sistemas tratan también de aproximar lo más posible la valoración propuesta a la valoración indicada en el conjunto de entrenamiento.

Pero, ¿qué sucede cuando en el conjunto de entrenamiento no se dispone de la valoración que deben tener los objetos?. En nuestro caso el conjunto de entrenamiento contiene comparaciones entre objetos cuya calificación numérica desconocemos; la comparación únicamente indica qué objeto es mejor (en términos relativos, con respecto al otro objeto de la comparación) y su situación en el espacio de los ejemplos. Las indicaciones que se pueden extraer de esta información nos permitirán inducir funciones de valoración que permitan reproducir las ordenaciones relativas con éxito.

La solución que $\mathcal{LACE}$ ofrecerá para cada problema será, obviamente, una de las infinitas soluciones existentes. Esto es así, ya que el conocimiento de partida (conjunto de entrenamiento) está expresando comparaciones de objetos, nunca su valoración. Esto supone que el conjunto de comparaciones refleja múltiples funciones de valoración adecuadas (el sistema sintetiza una de ellas, que en general, no coincidirá con la función de valoración real del problema). Todo esto quedará más claro con el siguiente ejemplo. Supongamos un problema cuya función de valoración *real* sea la definida en la Figura 3.2(b), es decir, $f(x) = x^2$. Teniendo en cuenta que siempre manejamos valores normalizados en el intervalo $[0, 1]$, los objetos tienen una mayor valoración a medida que aumenta el valor del atributo x. Aprendiendo a partir de las comparaciones, cualquiera de las funciones de la Figura 3.2(a), 3.2(b) o 3.2(c) sería una solución válida, puesto que todas son capaces de reproducir con acierto las ordenaciones del conjunto de entrenamiento. El problema que presentan las funciones de la segunda fila de la Figura 3.2 (d, e y f) es conceptualmente el mismo, salvo que en este caso la valoración aumenta a medida que disminuye el valor de x. La última fila de funciones (g, h e i) presenta un problema ligeramente más complejo, en el que los objetos alcanzan las mayores calificaciones cuando el atributo x presenta valores en torno a 0,5. En este caso, cualquier función creciente en $[0, 0{,}5)$, decreciente en $(0{,}5, 1]$ y con ambas partes del dominio de la función simétricas, resulta adecuada para reproducir los órdenes parciales del conjunto de entrenamiento. Las funciones de valoración de este tipo son muy habituales en controles de calidad.

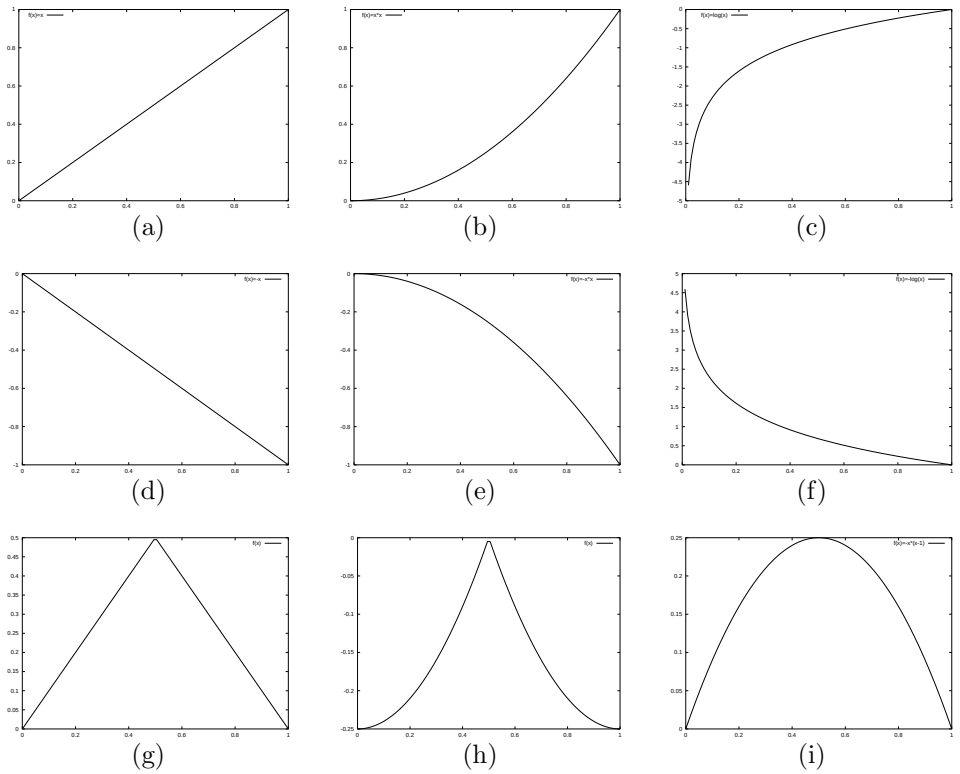


Figura 3.2: Cada fila de imágenes se corresponde con tres posibles soluciones para un mismo problema. En las gráficas (a-c) la valoración aumenta a medida que se incrementa el valor del atributo x. Cualquiera de las tres soluciones es válida, ya que lo importante es el signo de la pendiente. Las gráficas (d-f) muestran tres posibles soluciones para un problema en el que al aumentar el valor de x se decrementa su valoración. Las tres últimas gráficas presentan soluciones a un problema en el que hasta el 0,5 el valor de la x incrementa la valoración y a partir de dicho valor hace que se decremente. En este caso no basta con conocer el signo de las pendientes, es necesario que sean simétricas.

3.4. Esquema general del algoritmo $\mathcal{LACE}$

En esta sección se presenta el mecanismo general de aprendizaje de reglas de valoración que emplea el sistema $\mathcal{LACE}$. Este algoritmo parte de un conjunto de comparaciones entre pares de objetos y retorna una o varias reglas de valoración, que pueden considerarse como una función de calificación numérica definida por partes.

El primer paso del algoritmo consiste en reescribir el conjunto de comparaciones, previamente normalizadas en el intervalo $[0, 1]$ en un conjunto de pares como los descritos en la Ecuación (3.4) que, conceptualmente, indican:

$$\textit{función} \leftarrow \textit{condición}$$

Este conjunto de pares podría constituir una solución al problema, puesto que estamos ya ante un conjunto de reglas de valoración. Sin embargo, esta solución no es adecuada, puesto que presenta algunos inconvenientes que se detallan a continuación:

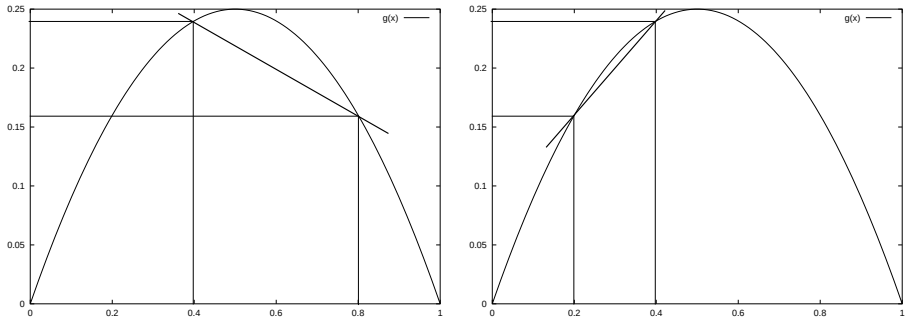
- El *sobreajuste* al conjunto de entrenamiento es, probablemente, el inconveniente más evidente de esta aproximación.
- El *número de pares* resultante, esto es, de reglas, es demasiado numeroso. Si se parte de n comparaciones tendremos $2 \cdot n$ reglas.
- Es probable que se obtengan *reglas contradictorias* debido a que el número de comparaciones que constituyen el conjunto de partida no se corresponde, en general, con el número de objetos comparados en el problema. Si tenemos N objetos diferentes podríamos llegar a tener $\frac{N \cdot (N-1)}{2}$ comparaciones que darían lugar a $N \cdot (N-1)$ reglas. Esto sucede debido a que cada objeto participa, generalmente, en varias comparaciones, dando lugar a múltiples reglas de valoración posibles para cada objeto.

  Los problemas que se derivan de esta circunstancia pueden observarse en el ejemplo de la Figura 3.3, donde tenemos objetos descritos por un único atributo x, y donde vamos a suponer que la función de valoración real es $f(x) = -x(x-1)$. En estas condiciones podríamos tener una comparación entre un objeto cuyo valor de x fuese 0,4 y otro con $x = 0{,}8$. Teniendo en cuenta que $f(0{,}8) < f(0{,}4)$, la reescritura propuesta en (3.4) da lugar a las reglas:

$$\begin{array}{rcl} -x & \leftarrow & x = 0{,}4 \\ -x & \leftarrow & x = 0{,}8 \end{array}$$

  Si, además, disponemos de la comparación entre los objetos $x = 0{,}4$ y $x = 0{,}2$ obtendremos las reglas:

$$\begin{array}{rcl} x & \leftarrow & x = 0{,}4 \\ x & \leftarrow & x = 0{,}2 \end{array}$$



(a) (b)

Figura 3.3: En la figura se pueden ver dos comparaciones: 0,8 < 0,4 y 0,2 < 0,4. En ambas comparaciones se tiene que $u < v$ utilizando como función de valoración $f(x) = -x(x-1)$ en el rango $[0, 1]$. Para un mismo punto (0,4) tenemos que en una comparación (0,4, 0,8) la diferencia es negativa (a) y en la otra (0,4, 0,2) es positiva (b).

> El hecho de que el objeto con $x = 0{,}4$ aparezca en estas dos comparaciones nos conduce a obtener dos reglas contradictorias, ya que para misma condición ($x = 0{,}4$) aplican funciones opuestas ($f(x) = x$ y $f(x) = -x$).
>
> Este inconveniente se agrava aún más en el caso particular de que el conjunto de comparaciones contenga incoherencias. Por ejemplo, podríamos tener un par que indicase que el objeto u es peor que el v ($u < v$) y otro que indicase lo contrario ($v < u$). En este caso obtendríamos dos reglas contradictorias cuyas condiciones de aplicación vendrían dadas por los valores de los atributos de u y otras dos, también contradictorias, cuyas condiciones vendrían dadas por v.

Por todos estos motivos es necesario tratar de reducir el número de reglas sugeridas por el conjunto de comparaciones. Desde un punto de vista global, la idea central del algoritmo $\mathcal{LACE}$ es la de localizar grupos (clusters) de objetos en el espacio de atributos para, posteriormente, calcular funciones lineales de valoración para cada grupo (véase la figura 3.4). Esta idea se sustenta en la suposición de que en el vecindario de un objeto se utilizará una misma función de valoración.

El mecanismo utilizado en el sistema $\mathcal{LACE}$ para detectar grupos de objetos es el *Growing Neural Gas* (`GNG`) [Fritzke, 1995], una variante de los mapas auto-organizados de Kohonen (`SOM`) [Kohonen, 1995]. A diferencia de éstos, la topología de la red de nodos no es una estructura rígida con un número de nodos y conexiones definido a priori, sino que durante el proceso de entrenamiento se modifica de forma dinámica, aumentando o disminuyendo el número de nodos y conexiones en función de la distribución de ejemplos en el espacio del problema.

Algoritmo 3.1 Algoritmo que aprende a valorar a partir de un conjunto de comparaciones entre pares de objetos ($\mathcal{LACE}$).

Algoritmo LACE(*Comparaciones*): Reglas de valoración
Ejemplos = REESCRIBIREJEMPLOS(*Comparaciones*);

// hacer grupos (clusters) con las condiciones de los ejemplos ...
Celdas = GNG(*Comparaciones*);

// transformar las celdas en reglas de valoración ...
Reglas = CONVERTIR(*Celdas*);

CALCULARFUNCIONESLOCALES(*Reglas*, *Comparaciones*, *Ejemplos*);
Regiones = OBTENERREGIONES(*Reglas*, *Comparaciones*, *Ejemplos*);
ELIMINARREGLASIRRELEVANTES(*Regiones*, *Ejemplos*);
AJUSTARFUNCIONESGLOBALMENTE(*Regiones*, *Comparaciones*, *Ejemplos*);
FunciónDefecto = CALCULARFUNCIÓNÚNICA(*Comparaciones*);
si (PRECISIÓN(*FunciónDefecto*) $\geq$ PRECISIÓN(REGLASDE(*Regiones*))) **entonces**
  retorna *FunciónDefecto*;
si no
  retorna REGLASDE(*Regiones*);
fin si

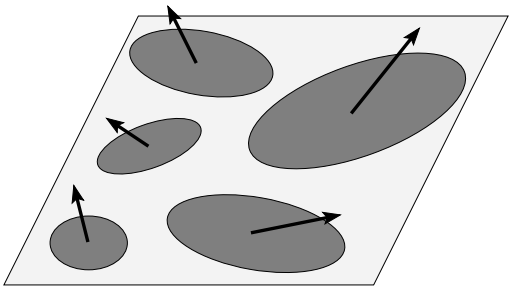


Figura 3.4: Los grupos de funciones parciales representan, en cada nodo, un entorno en el espacio de atributos de los objetos a ser valorados (coloreados en oscuro en la figura) y un vector apuntando en la dirección en la que se medirán las valoraciones. En otras palabras, el mapa representa un conjunto de reglas de valoración que se aplicarán por el criterio de mínima distancia.

La función GNG (véase Algoritmo 3.1) agrupa los objetos de las comparaciones atendiendo a los valores de los atributos que los describen y devuelve un conjunto de celdas y conexiones que estarán distribuidas por el espacio de ejemplos. Posteriormente, mediante la función CONVERTIR, las celdas se transformarán en reglas de valoración, donde su *condición* será su posición en el espacio de ejemplos y su *función* se representará mediante un vector n-dimensional, donde cada valor se corresponde con el peso de cada atributo o, en otras palabras, con el coeficiente de cada variable en la función. Así, cada celda de la red `GNG` será una regla de valoración.

Tras la construcción de la red `GNG`, el procedimiento CALCULARFUNCIONESLOCALES se ocupará de calcular una primera aproximación a la función en cada celda, es decir, al consecuente de cada regla. Estas funciones irán ajustándose a medida que se buscan las *regiones* del problema (OBTENERREGIONES), en las que se agruparán varias reglas/celdas que tendrán una misma función de valoración. Una vez agrupadas las celdas en regiones se aplica un proceso de selección, eliminándose aquellas reglas innecesarias (ELIMINARREGLASIRRELEVANTES). Finalmente, se llevará a cabo un ajuste de las funciones entre regiones con objeto de aumentar la eficacia en la ordenación de pares de objetos que no pertenezcan a una misma región y, por tanto, deban ser valorados por funciones diferentes (AJUSTARFUNCIONESGLOBALMENTE). Este último procedimiento dará lugar a las reglas finales.

Antes de retornar este conjunto final de reglas, se comprueba la precisión que se alcanza en reescritura con una única función de valoración. Si se alcanza una precisión mayor o igual que con el conjunto de reglas de valoración, entonces se retornará una única función.

La complejidad temporal de este algoritmo es bastante aceptable. En el peor de los casos su complejidad es $O(r^2 \cdot a \cdot n \log n)$ (ver Apéndice A), que es la correspondiente a la fase OBTENERREGIONES. La r se corresponde con el número de reglas (que como máximo es 100) y la a representa el número de atributos, que es menor que el número de comparaciones n.

3.5. Fases del algoritmo

Como se puede apreciar en el Algoritmo 3.1, $\mathcal{LACE}$ es un sistema modular que consta, fundamentalmente, de cuatro partes bien diferenciadas. En esta sección se comenta brevemente cada una de ellas, haciendo una descripción más detallada en los capítulos siguientes.

Ajuste de condiciones

La primer fase del algoritmo se encarga de agrupar ejemplos que se encuentran próximos. Para esta tarea se utiliza `GNG`, un algoritmo de agrupamiento reconocido por su eficiente adaptación a la topología de los problemas. Al finalizar esta fase tendremos

una red cuyas celdas o nodos actúan como representantes de un *cluster* de ejemplos del conjunto de entrenamiento.

Cálculo de funciones locales

Considerando el vector de referencia de cada grupo de ejemplos obtenido mediante el algoritmo de agrupamiento como las condiciones de una regla, esta segunda fase buscará, para cada regla, la función que clasifique más *comparaciones locales* correctamente. Las comparaciones locales son aquellas en las que ambos objetos de la comparación son valoradas por la misma regla. El número de comparaciones locales de cada regla será, en general, muy pequeño, así que, durante esta fase no se pretende obtener funciones de valoración finales; únicamente se busca una indicación del signo de las pendientes de las mismas.

Reducción del número de funciones

Basándonos en la primera aproximación a las reglas sugeridas por la fase anterior, el algoritmo intenta unir reglas próximas asignándoles la misma función de valoración. La función que se les asignará será la que maximice el número de comparaciones locales correctas. En caso de que la nueva función no supere el umbral de aciertos estimado para la unión que se está tratando, las reglas no se unirán. De esta manera se van conformando regiones de celdas que, conceptualmente, son grupos de reglas de valoración con condiciones diferentes pero con la misma función.

Una vez que el algoritmo ha construido regiones, podemos reducir el número de celdas en cada una de ellas de forma homóloga a como un algoritmo de aprendizaje basado en instancias reduce el número de instancias memorizadas para ser posteriormente utilizadas en la clasificación. En nuestro caso las instancias son las reglas de valoración y la clase es la función asociada a cada una de ellas. Esto nos permite utilizar una adaptación simplificada de una reconocida familia de algoritmos de reducción de instancias, DROP [Wilson y Martinez, 2000].

Ajuste global de funciones

La última fase del algoritmo pretende ajustar las pendientes de las funciones de valoración asociadas a cada región para mejorar la eficacia en la comparación entre objetos que pertenezcan a distintas regiones y, por tanto, sean valorados por funciones diferentes. Esta fase se encarga de nivelar las funciones corrigiendo ligeramente sus pendientes y añadiéndoles un término independiente, innecesario para las comparaciones entre objetos de una misma región, pero clave cuando las comparaciones de un par de objetos se realizan con funciones de regiones diferentes.

3.6. Generación de números aleatorios

A lo largo de las diferentes fases del algoritmo se utiliza, frecuentemente, la generación de números aletorios. Por esta razón es importante contar con un generador de números aleatorios eficiente.

En la implementación del algoritmo se ha utilizado el generador de números pseudoaleatorio *Mersenne Twister* [Matsumoto y Nishimura, 1998]. Este generador, utiliza los *números primos de Mersenne*. Dado un número n, se dice que se trata de un número primo de Mersenne cuando $2^n - 1$ es primo. En el diseño del generador se han tenido en cuenta los defectos de otros generadores. Tiene un periodo de $2^{19937} - 1$ y se asegura la propiedad de equidistribución 623-dimensional. Además, es cuatro veces más rápido que la llamada `rand()` de la librería estándar `ANSI-C`.

Otra ventaja de este generador es que, a diferencia de `rand()`, es independiente del sistema operativo utilizado, es decir, genera los mismos números aleatorios en operativos diferentes.

Capítulo 4

Ajuste de condiciones

4.1. Introducción

Como se ha explicado anteriormente, la solución de $\mathcal{LACE}$ consiste en un conjunto de reglas de valoración que se aplican por mínima distancia. Así, la función asociada a una determinada regla se utilizará para valorar los objetos que estén más próximos a esa regla que a las otras, puesto que cabe esperar que objetos parecidos sean valorados por una misma función. Es importante, pues, establecer adecuadamente las condiciones de aplicación de las reglas, que vendrán definidas por un vector de coordenadas en el espacio de los atributos. Para calcular estos vectores utilizaremos un mecanismo de *clustering* o agrupamiento, que nos permitirá obtener un *vector de referencia* representativo de cada grupo de ejemplos similares.

En este capítulo se comentan algunos algoritmos de agrupamiento susceptibles de ser usados para la construcción de un conjunto preliminar de vectores de referencia, que serán usados posteriormente como antecedentes de las reglas. En particular, se detalla el funcionamiento del *Growing Neural Gas* (`GNG`), puesto que es el que finalmente se utiliza en $\mathcal{LACE}$.

4.2. Convenciones de notación y propiedades

Antes de comenzar con la descripción de los distintos algoritmos presentamos la notación utilizada. Los algoritmos que se describen en este capítulo se basan en la construcción de una estructura de red formada por celdas o nodos donde se almacenan determinadas propiedades. Las redes están formadas por un conjunto de N celdas: $\mathcal{A} = \{c_1, c_2, \ldots, c_N\}$. Cada celda c tiene asociado un *vector de referencia* $w_c \in \mathbb{R}^n$,

que indica su posición (*receptive field center*) en el espacio de ejemplos. Los vectores de referencia se van adaptando a medida que se van presentando ejemplos de entrenamiento al algoritmo.

Entre las celdas que componen la red existe un conjunto (que puede ser vacío) de *conexiones de vecindad* ($\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$). Estas conexiones carecen de peso y son simétricas ($(i,j) \in \mathcal{C} \Leftrightarrow (j,i) \in \mathcal{C}$). Se utilizan en algunos métodos para propagar la adaptación del vector de referencia de una celda hacia sus vecinas, de esta forma, para una celda c, denotamos con N_c al conjunto de sus vecinas directas topológicamente ($N_c = \{i \in \mathcal{A} | (c,i) \in \mathcal{C}\}$), es decir, aquellas que se encuentran conectadas con c mediante un arco.

Asumimos que los ejemplos se generan de acuerdo a una función de densidad de probabilidad continua, $p(\xi), \xi \in \mathbb{R}^n$, o proceden de un conjunto de entrenamiento finito, $\mathcal{D} = \{\xi_1, \xi_2, \ldots, \xi_M\}, \xi_i \in \mathbb{R}^n$.

Dado un ejemplo ξ, la celda *ganadora*, $s(\xi)$, de entre las celdas del conjunto $\mathcal{A}$, se define como la celda cuyo vector de referencia es el más cercano a ξ ($s(\xi) = \arg\min_{c \in \mathcal{A}} \|\xi - w_c\|$), donde $\|\cdot\|$ denota la norma Euclídea del vector.

Dado un conjunto de celdas donde cada una tiene asociado un vector de referencia en $\mathbb{R}^n$, la *región de Voronoi* de la celda c (V_c) se define como el conjunto de puntos de $\mathbb{R}^n$ para los cuales w_c es el vector de referencia más cercano ($V_c = \{\xi \in \mathbb{R}^n | s(\xi) = c\}$).

4.3. Clasificación de los algoritmos

A continuación se muestra una clasificación de los algoritmos de agrupamiento en función de algunas características del mecanismo de aprendizaje y/o de las peculiaridades de la red a la que dan lugar [Fritzke, 1997]. Para mostrar de forma gráfica la diferencia en los resultados obtenidos por cada algoritmo se muestran en esta sección una serie de figuras en las que se superpone la silueta del espacio de ejemplos con el dibujo de las redes de celdas obtenidas por cada uno de los algoritmos. Estas figuras se han tomado de [Fritzke, 1997] y [Fritzke, 2002].

- Dentro de la categoría de *Hard Competitive Learning* (HCL) se enmarcan los algoritmos donde cada ejemplo determina la adaptación de una sola celda (la ganadora). La inicialización de los vectores de referencia de las celdas de partida puede llevar a resultados totalmente diferentes. Estos algoritmos tienen el conocido problema de las *celdas muertas*: una celda que nunca resulte ganadora mantendrá su posición indefinidamente. Este inconveniente puede evitarse si, en lugar de adaptar únicamente la celda ganadora, se adaptan también (en menor medida) las celdas cercanas. Se dice de los algoritmos que trabajan de esta manera que pertenecen a la categoría de *Soft Competitive Learning* (SCL). Esta manera de efectuar las adaptaciones de las celdas hace decrecer la dependencia de la solución obtenida con respecto a la inicialización de las celdas de partida.

- Los algoritmos en los que la adaptación de las celdas se realiza una vez que ya han sido presentados todos los ejemplos de entrenamiento se conocen como algoritmos de adaptación por *lote*. Los algoritmos de adaptación *on-line* son aquellos en los que la adaptación de las celdas se efectúa inmediatamente después de la presentación de cada ejemplo.

- Existen algoritmos que tienen una *estructura fija de red*. Son aquellos algoritmos en los que las conexiones entre las celdas de la red tienen unas determinadas restricciones, como vecindades constantes a lo largo de todo el aprendizaje.

- Finalmente, hay algoritmos en los que el número de celdas (N) es constante durante todo el proceso de entrenamiento mientras que hay otros en los que el número de celdas varía a medida que avanza el aprendizaje. Evidentemente, la capacidad de adaptación de una red aumenta con un número de celdas variable.

4.4. Algunos algoritmos de agrupamiento

4.4.1. El algoritmo k-means

El algoritmo `k-means` [MacQueen, 1967] calcula el vector de referencia de una celda como la media de las posiciones de los ejemplos para los cuales ha sido la ganadora. Así, en un instante t, el vector de referencia de la celda c será:

$$w_c(t) = \frac{\xi_1^c, \xi_2^c, \ldots, \xi_t^c}{t} \tag{4.1}$$

En la Figura 4.1 se puede ver un ejemplo de la evolución del algoritmo `k-means` en un problema artificial. La Figura 4.2 muestra los resultados finales obtenidos por este mismo algoritmo tras haber sido entrenado con otros problemas artificiales. Se aprecia en las figuras que, aunque la distribución de los vectores de referencia a variado, en los lugares en los que inicialmente había agrupaciones, siguen quedando restos de dichos grupos. Esta es una de las consecuencias de utilizar la técnica de adaptar únicamente la celda ganadora.

4.4.2. Self-organizing Feature Map

Los mapas auto-organizados (`SOM`) de Kohonen [Kohonen, 1995] son mallas con un número de celdas y de conexiones fijo durante todo el proceso de entrenamiento. El ámbito de adaptación sobre la vecindad de la celda ganadora y el grado de adaptación decrecen a medida que aumenta el número de ejemplos presentados.

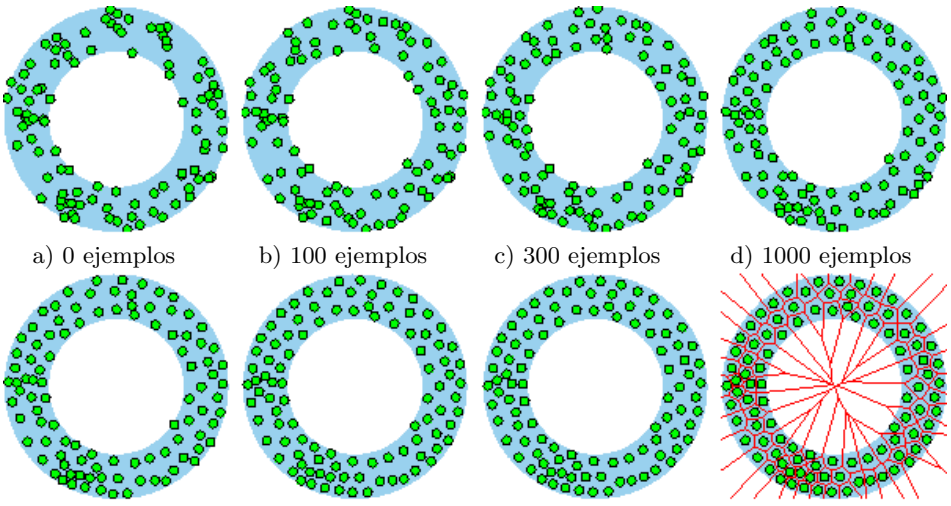
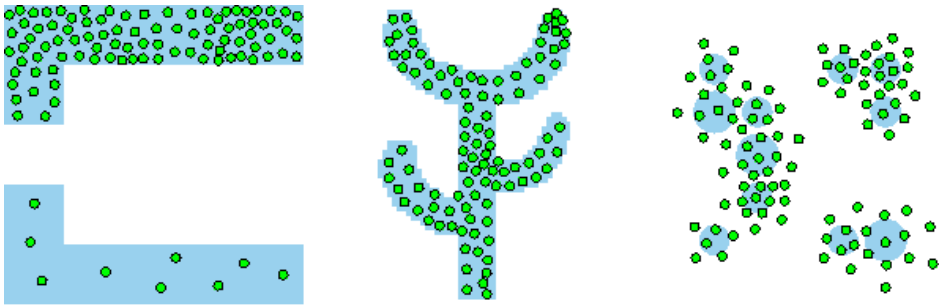


Figura 4.1: Evolución del `k-means` ante un anillo con distribución uniforme. El número de celdas permanece constante durante todo el aprendizaje. Aunque pueda parecer que el número de celdas se incrementa, esto no es así. Ocurre, simplemente, que en la figura a), muchas de las celdas se encuentran superpuestas. a) Estado Inicial. b-f) Estados intermedios. g) Estado final. h) Regiones de Voronoi correspondientes al estado final. Se puede ver que la distribución final de los vectores de referencia presenta una agrupación similar a la inicial (esta circunstancia se aprecia mejor en la parte inferior izquierda, donde la densidad de los vectores es mayor que en otras zonas).



a) b) c)

Figura 4.2: Resultados de la evolución del `k-means` tras la lectura de 40000 ejemplos para tres problemas con distribuciones de probabilidad diferentes. a) La distribución es uniforme en ambas zonas sombreadas, pero la densidad es 10 veces mayor en la parte superior. b) La distribución es uniforme en la zona sombreada. c) El radio de cada una de las 11 circunferencias indica la desviación estándar de la función de densidad Gaussiana con la que se generaron los ejemplos. Todas las funciones de densidad Gaussianas tienen, a priori, la misma probabilidad.

La distancia entre las celdas de la malla se utiliza para determinar el grado en que una celda es adaptada. Si representamos la malla como una matriz de celdas,

$$\mathcal{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots \\ a_{2,1} & a_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{4.2}$$

la distancia entre las celdas $r = a_{km}$ y $s = a_{ij}$ se calcula como:

$$d(r, s) = |i - k| + |j - m| \tag{4.3}$$

así que para calcular el grado de adaptación de la celda r, tras la presentación de un ejemplo para el que s es la ganadora, el grado de adaptación se obtiene con la siguiente expresión:

$$h_{rs} = \exp\left(\frac{-d(r, s)^2}{2\sigma^2}\right) \tag{4.4}$$

donde la desviación estándar, σ, de la campana de Gauss varía a lo largo del entrenamiento de acuerdo a la expresión:

$$\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{t/t_{max}} \tag{4.5}$$

para un valor inicial σ_i y un valor final σ_f. El parámetro t, que se utiliza como metáfora del tiempo, es un contador que se incrementa con la presentación de cada ejemplo, hasta alcanzar un total de t_{max} presentaciones.

El mecanismo de entrenamiento de este modelo de red es el siguiente:

1. Se inicializa el conjunto $\mathcal{A}$ conteniendo $N = N_1 \times N_2$ celdas. Cada celda c contiene un vector de referencia $w_c \in \mathbb{R}^n$, inicializado aleatoriamente de acuerdo a $\mathrm{P}(\xi)$. El conjunto de conexiones $\mathcal{C}$ se inicializa de forma que las celdas queden conectadas formando una malla de N_1 filas y N_2 columnas. Se inicializa el parámetro t a cero.

2. Se toma un ejemplo ξ.

3. Se determina la celda ganadora: $s = s(\xi) = \arg\min_{c \in \mathcal{A}} \|\xi - w_c\|$

4. Se adaptan las celdas de la malla, modificando su vector de referencia w_r según la expresión
$$\Delta w_r = \epsilon(t) h_{rs}(\xi - w_r), \tag{4.6}$$
donde $\epsilon(t) = \epsilon_i \left(\frac{\epsilon_f}{\epsilon_i}\right)^{t/t_{max}}$, siendo ϵ_i el grado de adaptación inicial y ϵ_f el final.

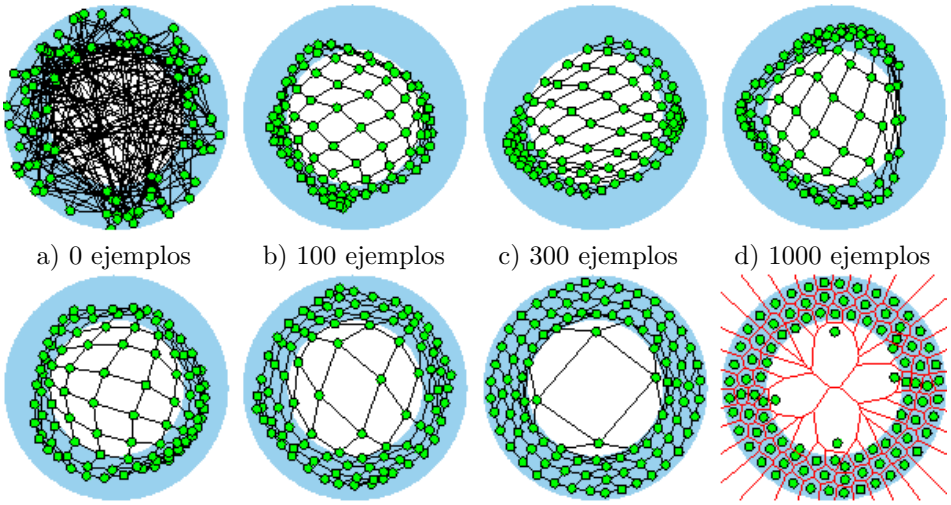
5. Se incrementa el parámetro t.

6. Si $t < t_{max}$ continuar en el paso 2.

En las Figuras 4.3 y 4.4 se muestran soluciones de los mapas auto-organizados a varios problemas. Los parámetros utilizados han sido: $\sigma_i = 3{,}0$, $\sigma_f = 0{,}1$, $\epsilon_i = 0{,}5$, $\epsilon_f = 0{,}005$, $t_{max} = 10000$ y $N_1 = N_2 = 10$. En la Figura 4.3, se puede ver que las red evoluciona rápidamente al principio, sin embargo, las restricciones impuestas a la red (topología fija y número de celdas constante), hacen que la red necesite muchos ejemplos para adaptarse realmente a la solución del problema. Las mismas dificultades pueden apreciarse en en Figura 4.4.

4.4.3. Neural Gas con Competitive Hebbian Learning

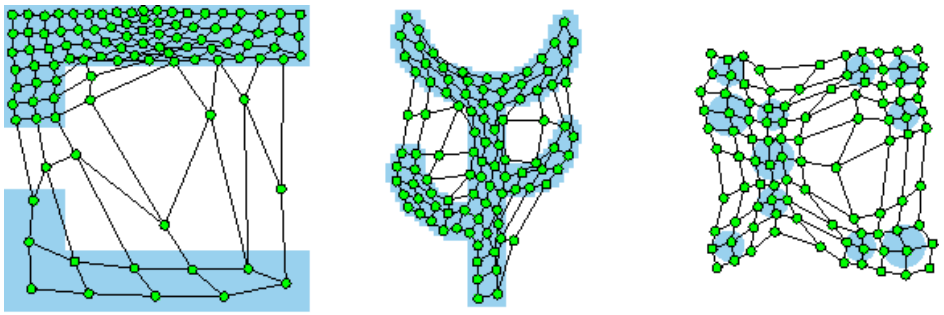
A partir del algoritmo *Neural Gas* [Martinetz y Schulten, 1991] y de la idea del *Competitive Hebbian Learning* [Martinetz, 1993] nace un algoritmo denominado *Topology-Representing Networks* [Martinetz y Schulten, 1994]. Este algoritmo es una extensión del *Neural Gas* en el que se permite la creación y destrucción de arcos. Los arcos se crean entre la celda ganadora de un ejemplo y la segunda más cercana y se destruyen cuando llevan un determinado tiempo sin que sus dos celdas asociadas sean las más cercanas a un ejemplo. El funcionamiento de este algoritmo es el siguiente:



a) 0 ejemplos b) 100 ejemplos c) 300 ejemplos d) 1000 ejemplos

e) 2500 ejemplos f) 10000 ejemplos g) 40000 ejemplos h) Reg. de Voronoi

Figura 4.3: Evolución del SOM ante un anillo con distribución uniforme. a) Estado Inicial. b-f) Estados intermedios. g) Estado final. h) Regiones de Voronoi correspondientes al estado final. Al tener un grado de adaptación alto al principio, las adaptaciones iniciales son muy fuertes.



a) b) c)

Figura 4.4: Resultados de la evolución del SOM tras la lectura de 40000 ejemplos para tres problemas con distribuciones de probabilidad diferentes (descritas en la Figura 4.2).

1. Se inicializa el conjunto $\mathcal{A}$ conteniendo N celdas: $\mathcal{A} = \{c_1, c_2, \ldots, c_N\}$. Cada celda c_i contiene un vector de referencia $w_{c_i} \in \mathbb{R}^n$ inicializado aleatoriamente de acuerdo a $\mathrm{P}(\xi)$. El conjunto de conexiones $\mathcal{C}$ inicialmente se encuentra vacío. Se inicializa el parámetro t a cero.

2. Se toma un ejemplo ξ.

3. Se ordenan todos las celdas de $\mathcal{A}$ de acuerdo a su distancia respecto a ξ. Se busca la secuencia de índices $(i_0, i_1, \ldots, i_N)$ tal que w_{i_0} es el vector de referencia más cercano a ξ, w_{i_1} es el segundo más cercano y w_{i_k} $(k = 0, 1, \ldots, N-1)$ es un vector de referencia tal que k vectores w_j existen con $\|\xi - w_j\| < \|\xi - w_k\|$, es decir, se ordenan los vectores de referencia por su distancia a ξ. Denotaremos como $k_i(\xi, \mathcal{A})$ al número k asociado con el vector de referencia w_i.

4. Se adaptan los vectores: $\Delta w_i = \epsilon(t) \cdot h_\lambda(k_i(\xi, \mathcal{A})) \cdot (\xi - w_i)$, con las siguientes dependencias temporales:

$$\begin{aligned} \lambda(t) &= \lambda_i \left(\lambda_f/\lambda_i\right)^{t/t_{max}} \\ \epsilon(t) &= \epsilon_i \left(\epsilon_f/\epsilon_i\right)^{t/t_{max}} \\ h_\lambda(k) &= \exp(-k/\lambda(t)) \end{aligned}$$

5. Si no existe una conexión (arco) entre las celdas representadas por i_0 e i_1 (ganadora y segunda más cercana), entonces se crea: $\mathcal{C} = \mathcal{C} \cup \{(i_0, i_1)\}$.

6. Se asigna a la conexión entre la celda i_0 y la i_1 una *vejez* de cero: $\mathrm{vejez}_{(i_0,i_1)} = 0$, y se incrementa la vejez de todas las conexiones que parten de i_0 en una unidad: $\mathrm{vejez}_{(i_0,i)} = \mathrm{vejez}_{(i_0,i)} + 1,\ (\forall i \in N_{i_0})$.

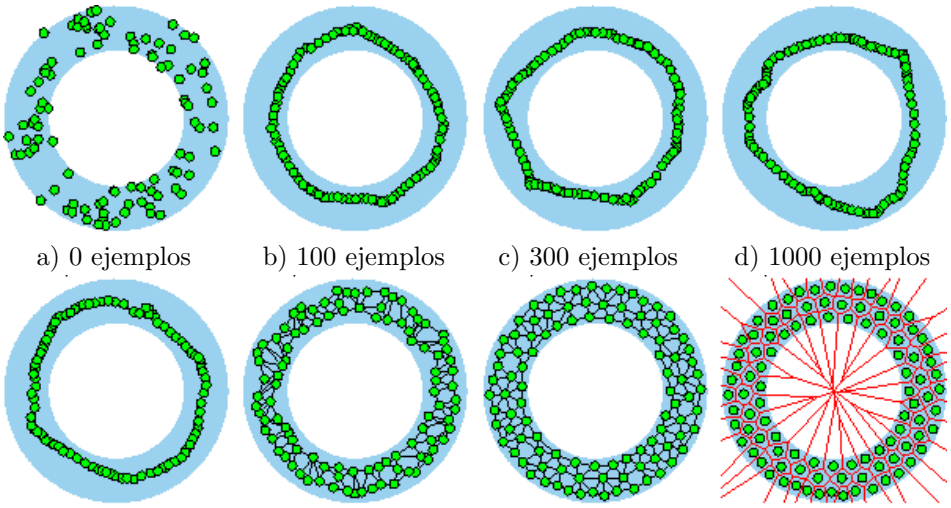
7. Se eliminan los arcos con vejez superior a la máxima permitida, $T(t)$, donde $T(t) = T_i(T_f/T_i)^{t/t_{max}}$.

8. Se incrementa el parámetro t.

9. Si $t < t_{max}$ continuar en el paso 2.

Los parámetros dependientes del tiempo tienen unos valores iniciales $(\lambda_i, \epsilon_i, T_i)$ y otros finales $(\lambda_f, \epsilon_f, T_f)$ que deben se suministrados. En las Figuras 4.5 y 4.6 se pueden ver las redes resultantes de aplicar este algoritmo a los problemas artificiales que hemos venido utilizando en esta sección. Las ejecuciones se han realizado con los siguientes parámetros: $\lambda_i = 10$, $\lambda_f = 0{,}01$, $\epsilon_i = 0{,}5$, $\epsilon_f = 0{,}005$, $T_i = 20$, $T_f = 200$ $N = 100$ y $t_{max} = 40000$. Vemos que se adapta rápidamente al domino del problema y llega a una solución óptima. Su principal problema es que al tener un número constante de celdas, se necesita tener un conocimiento *a priori* del problema o probar con distinto número de celdas.

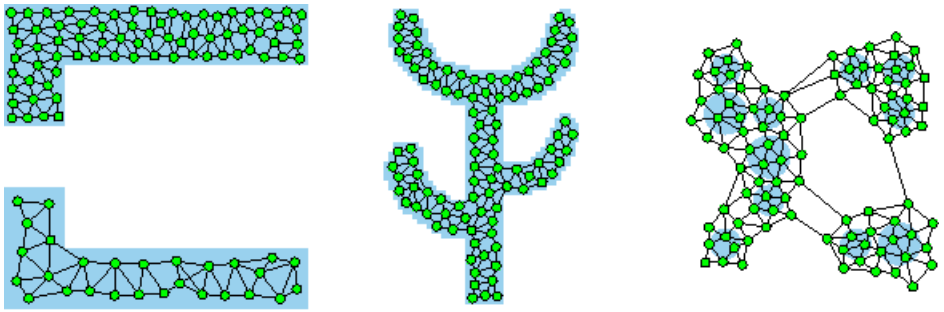
Ajuste de condiciones



a) 0 ejemplos b) 100 ejemplos c) 300 ejemplos d) 1000 ejemplos

e) 2500 ejemplos f) 10000 ejemplos g) 40000 ejemplos h) Reg. de Voronoi

Figura 4.5: Evolución del *Neural Gas con Competitive Hebbian Learning* ante un anillo con distribución uniforme. a) Estado Inicial. b-f) Estados intermedios. g) Estado final. h) Regiones de Voronoi correspondientes al estado final. Las celdas se mueven de acuerdo al algoritmo *Neural Gas*. Los arcos se crean siguiendo la filosofía del Competitive Hebbian Learning y son eliminados si las celdas conectadas por los arcos no resultan ganadoras durante un tiempo.



a) b) c)

Figura 4.6: Resultados de la evolución del *Neural Gas with Competitive Hebbian Learning* tras la lectura de 40000 ejemplos para tres problemas con distribuciones de probabilidad diferentes (descritas en la Figura 4.2).

4.4.4. Growing Cell Structure

Este método [Fritzke, 1994b] tiene una diferencia destacable con respecto a los métodos ya descritos: el número de celdas varía durante el proceso de aprendizaje. Para determinar el lugar en el que se debe insertar una nueva celda, se mide el error de cada celda respecto a los ejemplos que ha agrupado durante el aprendizaje. Se considera error la distancia media a la que queda la celda respecto de los ejemplos que resulta ganadora. La celda que más error tenga será la que precise ayuda de una nueva celda. La red que se va construyendo durante el entrenamiento debe estar formada únicamente por elementos básicos a los que Fritzke denomina *simplex*. Dependiendo de la dimensión de la red (k) estos elementos serán líneas (para $k = 1$), triángulos (para $k = 2$) o tetraedros (para $k = 3$). El funcionamiento de este modelo de red es el siguiente:

1. Dada la dimensión k, se inicializa $\mathcal{A}$ con $k+1$ celdas: $\mathcal{A} = \{c_1, \ldots, c_{k+1}\}$ que contienen un vector de referencia $w_{c_i} \in \mathbb{R}^n$ inicializado aleatoriamente de acuerdo a $\mathrm{P}(\xi)$. El conjunto de conexiones $\mathcal{C}$ se inicializa de manera que cada celda esté conectada con las demás (de esta manera se forma el *simplex* para la dimensión k dada).

2. Se toma un ejemplo ξ.

3. Se determina la celda ganadora: $s = \arg\min_{c\in\mathcal{A}} \|\xi - w_c\|$.

4. Se incrementa el error de la celda ganadora, s, con el cuadrado de la distancia entre el ejemplo y el vector de referencia de dicha celda:

$$\Delta E_s = \|\xi - w_s\|^2. \tag{4.7}$$

5. Se adapta el vector de referencia de la celda ganadora

$$\Delta w_s = \epsilon_b(\xi - w_s) \tag{4.8}$$

   y de sus vecinas

$$\Delta w_i = \epsilon_n(\xi - w_i),\ \forall i \in N_s. \tag{4.9}$$

6. Si el número de ejemplos analizados es un entero múltiplo del parámetro λ debe insertarse una nueva celda de la siguiente manera:

   - Determinar la celda q con el máximo error: $q = \arg\max_{c\in\mathcal{A}} E_c$.
   - Determinar la celda f de entre las vecinas de q cuyos vectores de referencia, w_f y w_q respectivamente, estén lo más alejados posible.
   - Insertar una celda r en el punto medio entre q y f: $\mathcal{A} = \mathcal{A} \cup \{r\}$ y $w_r = (w_q + w_f)/2$.

- Añadir arcos que conecten la celda r con q y f, y eliminar el arco que une q con f: $\mathcal{C} = \mathcal{C} \cup \{(r,q),(r,f)\} \setminus \{(q,f)\}$. Para hacer que la red vuelva a estar formada por elementos *simplex*, la celda r debe conectarse con todos los vecinos comunes de q y f, es decir, deben añadirse a $\mathcal{C}$ los arcos entre r y las celdas que formen el conjunto $N_q \cap N_f$.
- Decrementar el error de todas las celdas vecinas de r:

$$\Delta E_i = -\alpha(E_i/|N_i|),\ (\forall i \in N_r). \tag{4.10}$$

- Inicializar el error de la nueva celda r como la media del error de sus vecinas:

$$E_r = \frac{1}{|N_r|} \sum_{i \in N_r} E_i. \tag{4.11}$$

7. Se decrementa el error de todas las celdas en una proporción fija de β:

$$\Delta E_c = -\beta E_c,\ \forall c \in \mathcal{A}. \tag{4.12}$$

8. Si el criterio de parada aún no se ha cumplido debe volverse a paso 2.

El criterio de parada de este algoritmo puede establecerse en función del número de *simplex* de la red y/o del error cometido por las celdas. En [Fritzke, 1994b] se comenta un método para eliminar celdas superfluas según el cual deben eliminarse periódicamente aquellas celdas en las que $\tilde{p} = h_c/|V_c|$ es menor que un umbral dado η. $|V_c|$ representa el número de ejemplos contenidos en su región de Voronoi y h_c es la frecuencia relativa con la que ha sido la celda más cercana a los ejemplos presentados (*relative signal frequency*).

En las Figuras 4.7 y 4.8 pueden verse resultados ofrecidos por el *growing cell structure*. Los parámetros utilizados en las ejecuciones han sido: $\lambda = 200$, $\epsilon_b = 0{,}06$, $\epsilon_n = 0{,}002$, $\alpha = 1{,}0$ y $\beta = 0{,}0005$. Observando las figuras, vemos que, dada la rigidez de la red, le cuesta mucho adaptarse completamente a la topología del problema. Por esta misma razón, en las figuras se ve que hay celdas cuya eliminación supondría una mejora en la solución del problema, sin embargo, se conservan para mantener la estructura de *simplex*.

4.4.5. Growing Neural Gas

Al igual que en el método anterior, en el *growing neural gas* (`GNG`) [Fritzke, 1994a, Fritzke, 1995] el número de celdas de la red varía durante el aprendizaje. El mecanismo por el que este número se incrementa es el propuesto en el *growing cell structures* [Fritzke, 1994b] y la gestión de los arcos sigue el criterio del *competitive Hebbian learning* [Martinetz, 1993]. Al igual que en el *growing cell structures*, para determinar el lugar en el que se debe insertar una nueva celda, se mide el error de cada celda

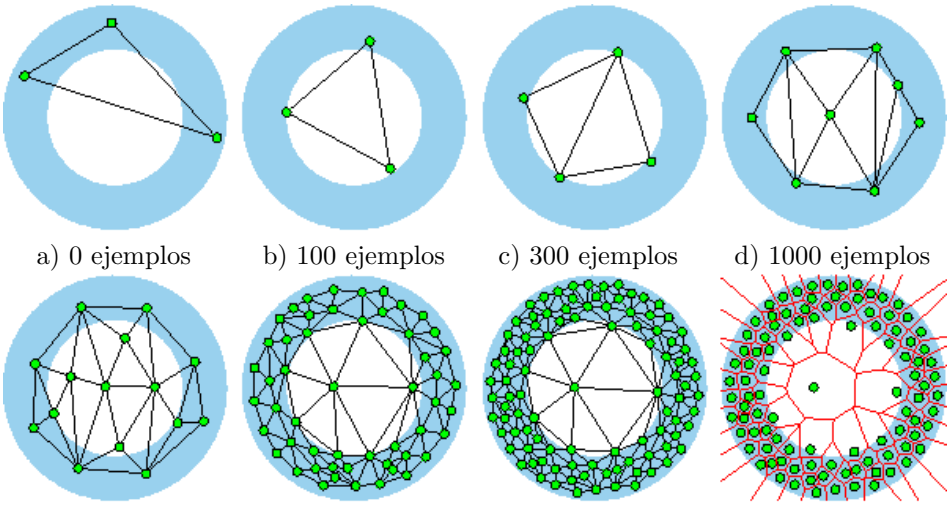


Figura 4.7: Evolución del GCS ante un anillo con distribución uniforme. a) Estado Inicial. b-f) Estados intermedios. g) Estado final. h) Regiones de Voronoi correspondientes al estado final. Se ve como la rigidez de la red perjudica su adaptación.

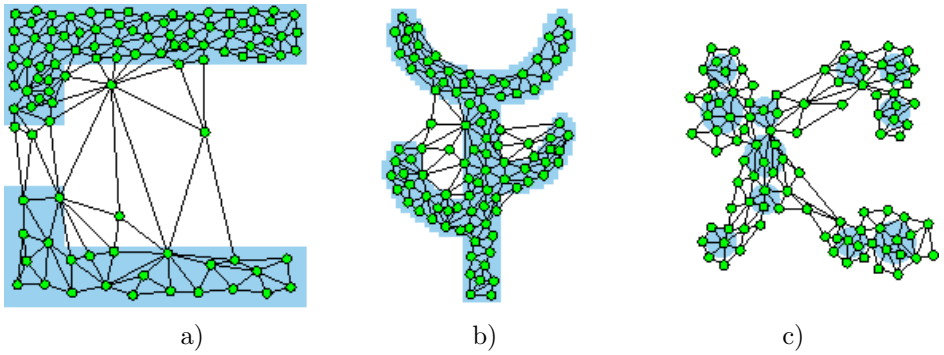


Figura 4.8: Resultados de la evolución del GCS tras la lectura de 40000 ejemplos para tres problemas con distribuciones de probabilidad diferentes (descritas en la Figura 4.2).

respecto a los ejemplos que ha agrupado durante el aprendizaje. La celda que más error tenga será la que precise ayuda de una nueva celda. El funcionamiento de este algoritmo es el siguiente:

1. Se inicializa el conjunto $\mathcal{A}$ con dos celdas: $\mathcal{A} = \{c_1, c_2\}$, cada una de ellas con su correspondiente vector de referencia $w_{c_i} \in \mathbb{R}^n$ inicializado aleatoriamente de acuerdo a $\mathrm{P}(\xi)$. Inicialmente, el conjunto de conexiones $\mathcal{C}$ se encuentra vacío.

2. Se toma un ejemplo ξ.

3. Se determina la celda ganadora ($s_1 = \arg\min_{c\in\mathcal{A}}\|\xi - w_c\|$) y la segunda más cercana a ξ ($s_2 = \arg\min_{c\in\mathcal{A}\setminus\{s_1\}}\|\xi - w_c\|$).

4. Si no existe una conexión entre las celdas representadas por s_1 y s_2, entonces debe ser creada: $\mathcal{C} = \mathcal{C} \cup \{(s_1, s_2)\}$.

5. Se asigna al arco (s_1, s_2) una *vejez* de cero, $\mathrm{vejez}_{(s_1,s_2)} = 0$, y se incrementa la vejez de todos los arcos que parten de s_1: $\mathrm{vejez}_{(s_1,i)} = \mathrm{vejez}_{(s_1,i)} + 1,\ \forall i \in N_{s_1}$.

6. El error de la celda ganadora se incrementa en el cuadrado de la distancia entre dicha celda y el ejemplo presentado:

$$\Delta E_{s_1} = \|\xi - w_{s_1}\|^2. \tag{4.13}$$

7. Se adapta el vector de referencia de la celda ganadora

$$\Delta w_{s_1} = \epsilon_b(\xi - w_{s_1}) \tag{4.14}$$

   y de sus vecinas

$$\Delta w_i = \epsilon_n(\xi - w_i),\ \forall i \in N_{s_1}. \tag{4.15}$$

8. Se eliminan todos los arcos que tenga una vejez superior a v_{max}. Si con la eliminación de los arcos resulta alguna celda sin conexiones entonces se elimina la celda.

9. Si el número de ejemplos analizados es un entero múltiplo del parámetro λ debe insertarse una nueva celda de la siguiente manera:

   - Determinar la celda q con el máximo error: $q = \arg\max_{c\in\mathcal{A}} E_c$.
   - Determinar la celda f de entre las vecinas de q con el máximo error: $f = \arg\max_{c\in N_q} E_c$
   - Insertar una celda r en el punto medio entre q y f: $\mathcal{A} = \mathcal{A} \cup \{r\}$ y $w_r = (w_q + w_f)/2$.
   - Añadir arcos que conecten la celda r con q y f, y eliminar el arco que une q con f: $\mathcal{C} = \mathcal{C} \cup \{(r, q), (r, f)\}$ y $\mathcal{C} = \mathcal{C}\setminus\{(q, f)\}$.

- Decrementar el error de q y f:

$$\Delta E_q = -\alpha E_q \quad (4.16)$$
$$\Delta E_f = -\alpha E_f \quad (4.17)$$

- Inicializar el error de la nueva celda r como la media del error de las celdas q y f:

$$E_r = (E_q + E_f)/2. \quad (4.18)$$

10. Se decrementa el error de todas las celdas en una proporción fija β:

$$\Delta E_c = -\beta E_c, \ \forall c \in \mathcal{A} \quad (4.19)$$

11. Si el criterio de parada aún no se ha cumplido debe volverse a paso 2.

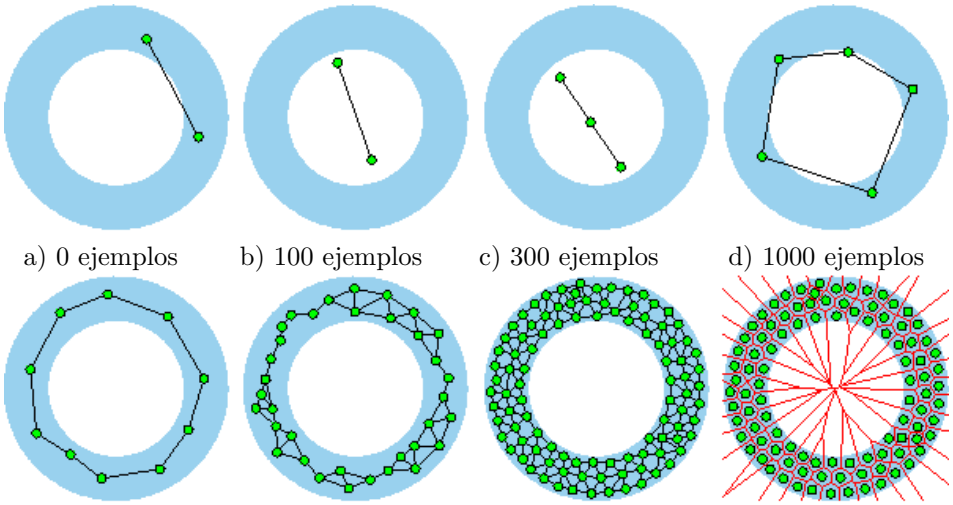
El criterio de parada de este algoritmo puede consistir en haber alcanzado el número de celdas deseado o que la suma del error de todas las celdas de la red haya alcanzado un determinado valor. En las Figuras 4.9 y 4.10 pueden verse resultados obtenidos por el *growing neural gas*. Los parámetros utilizados en las ejecuciones han sido: $\lambda = 300$, $\epsilon_b = 0{,}05$, $\epsilon_n = 0{,}0006$, $\alpha = 0{,}5$, $\beta = 0{,}0005$, $N = 100$ y $v_{max} = 88$. Se puede apreciar en las figuras que este algoritmo consigue llegar a una solución aproximada rápidamente. Además, la solución aportada tras analizar un número de ejemplos aceptable, es casi perfecta, adaptándose perfectamente a la topología del problema.

4.5. Algoritmo de agrupamiento de $\mathcal{LACE}$

Si analizamos los algoritmos de agrupamiento descritos en las secciones anteriores podemos extraer el siguiente resumen (sintetizado en la Tabla 4.1):

Algoritmo	CL	On-line	Estructura fija	Nº de celdas variable	CHL
K-means	Hard	√			
SOM	Soft	√	√		
NG con CHL	Soft	√			√
GCS	Soft	√	√	√	
GNG	Soft	√		√	√

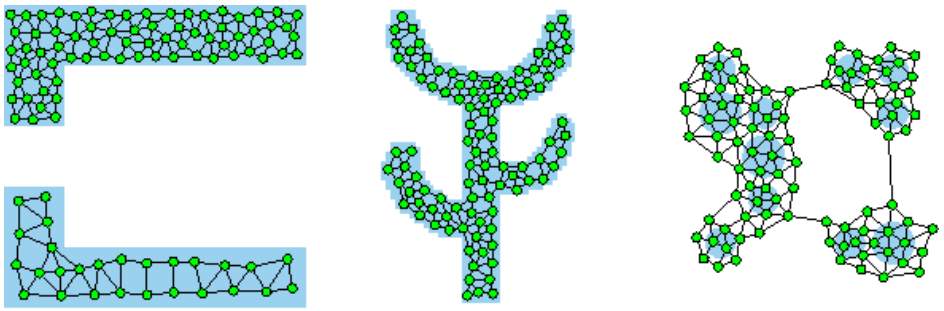
Tabla 4.1: Características principales de los algoritmos de agrupamiento presentados. *CL* se refiere a *Competitive Learning* y *CHL* a *Competitive Hebbian Learning*.



a) 0 ejemplos b) 100 ejemplos c) 300 ejemplos d) 1000 ejemplos

e) 2500 ejemplos f) 10000 ejemplos g) 40000 ejemplos h) Reg. de Voronoi

Figura 4.9: Evolución del GNG ante un anillo con distribución uniforme. a) Estado Inicial. b-f) Estados intermedios. g) Estado final. h) Regiones de Voronoi correspondientes al estado final. Puede verse que esta red, desde un principio, adquiere la forma del problema.



a) b) c)

Figura 4.10: Resultados de la evolución del GNG tras la lectura de 40000 ejemplos para tres problemas con distribuciones de probabilidad diferentes (descritas en la figura 4.2).

k-means. Se trata de un algoritmo en el que toda la adaptación se hace sobre la celda que resulta ganadora (*hard competitive learning*), lo que hace que los resultados producidos sean muy dependientes del proceso de inicialización. El número de celdas de la red debe darse a priori y permanece constante durante el entrenamiento, lo que implica que se debe tener un buen conocimiento del problema o que se deben realizar varios procesos de entrenamiento con distinto número de celdas, para acercarse a la solución adecuada.

Self-organizing feature map. Este tipo de red adapta la celda ganadora y las vecinas (*soft competitive learning*) tras cada presentación. Sin embargo, adolece de los mismos problemas que el `k-means`, puesto que la topología y el número de celdas son fijos y deben ser indicados inicialmente. En las Figuras 4.3 y 4.4 se puede apreciar que, en ocasiones, muchas celdas pueden quedar finalmente situadas en posiciones localizadas entre grupos de ejemplos.

Neural gas con competitive Hebbian learning Al igual que en los algoritmos anteriores, tiene el problema de que el número de celdas es constante. Sin embargo, el uso del *competitive Hebbian learning* [Martinetz, 1993] hace que la topología de la red varíe durante el aprendizaje, logrando que se adapte mejor a la geometría del problema.

Growing cell structures Esta red tiene un número de celdas variable, lo que permite una mejor adaptación al problema, sin embargo la fuerte restricción que tiene la topología de la red, que sólo puede estar constituida por elementos *simplex*, hace que algunas celdas se puedan quedar en zonas intermedias (Figuras 4.7 y 4.8), tal como pasaba con el *Self-Organizing feature map*.

Growing neural gas El número de celdas de esta red varía a medida que avanza el proceso de entrenamiento. Al igual que en el *growing cell structures*, se van añadiendo celdas en los lugares donde hay más densidad de ejemplos por celda. Además, utiliza la idea del *competitive Hebbian learning*, con lo que la topología de la red se adapta fácilmente a la geometría del problema, logrando que la red tenga una distribución espacial similar a la de los ejemplos (Figuras 4.9 y 4.10).

Existen otros métodos de agrupamiento que no se han citado aquí pero que, en general, presentan los inconvenientes ya mencionados de contar con un número de celdas prefijado e invariable; esto implica que se va a detectar un número determinado de grupos que no necesariamente va a corresponderse con el verdadero número de grupos existentes en el conjunto de ejemplos.

De los métodos analizados, el más flexible parece el `GNG`, puesto que se adapta mejor y más rápido a la geometría de cada problema, tanto en el número de celdas como en la posición de éstas; esto se debe a que el `GNG` da a las celdas más libertad desde el punto de vista geométrico para desplazarse por el espacio de atributos y posicionarse en un lugar más adecuado.

Algoritmo 4.1 Función que se utiliza para la búsqueda de la mejor situación para las condiciones de las reglas.

```
Función Gng (Comparaciones): Red
Red = InicializarRed();
InicializarParámetrosDeAprendizaje();
NumCompPeriodo = 0;
repetir
  para (cada comparación C de Comparaciones) hacer
    para (E ∈ {ObjetoPeor(C), ObjetoMejor(C)}) hacer
      CeldaGanadora = ObtenerCeldaMásCercana(Red, E);
      CeldaSegunda = ObtenerSegundaCeldaMásCercana(Red, E);
      ActualizarArcos(Red, CeldaGanadora, CeldaSegunda);
      IncrementarError(CeldaGanadora, E);
      AdaptarGanadora(CeldaGanadora, E);
      AdaptarVecinas(VecinasDe(CeldaGanadora), E);
      EliminarArcos(Red);
      NumCompPeriodo = NumCompPeriodo + 1;
      si (NumCompPeriodo = λ) entonces
        InsertarCelda(Red);
        NumCompPeriodo = 0;
      fin si
      ActualizarErrores(Red);
    fin para
  fin para
hasta que (CriterioDeParada(Comparaciones,Red))
retorna Red;
```

El pseudocódigo del GNG adaptado para su uso en el sistema $\mathcal{LACE}$ puede verse en el Algoritmo 4.1. Dado que el conjunto de entrenamiento está constituido por comparaciones entre pares de objetos, la adaptación del algoritmo GNG toma los ejemplos por parejas y aplica de forma iterativa el mismo proceso (GNG estándar de Fritzke) a ambos.

Los primeros pasos del algoritmo GNG inicializan la estructura de la red con dos celdas no conectadas entre si, además de los parámetros del algoritmo con los siguientes valores (sugeridos por Fritzke):

$\lambda = 100$**.** Cada λ comparaciones se busca la mejor posición para insertar una nueva celda.

$\epsilon_b = 0{,}05$**.** Es el grado de adaptación de la celda ganadora.

$\epsilon_n = 0{,}0006$**.** Grado de adaptación de las celdas vecinas de la celda ganadora.

$\alpha = 0{,}5$**.** Indica el grado de decremento del error de las celdas entre las cuales se ha insertado la nueva celda.

$\beta = 0{,}0005$**.** Tasa de decremento del error de todas las celdas tras la presentación de un ejemplo.

$v_{max} = 88$**.** Vejez máxima permitida a un arco. Los arcos que superan esa vejez son eliminados.

Con los parámetros de aprendizaje ya inicializados comienza el proceso de entrenamiento, que consiste en presentar el conjunto de comparaciones a la red. Cada comparación está compuesta por dos objetos ($comparación = (u, v)$ tal que $u < v$), así que deben tratarse ambos. Se toma el primero de los objetos, u, de la comparación y se busca la celda más cercana al mismo (*CeldaGanadora*) y la segunda más cercana (*CeldaSegunda*). Si entre ambas celdas hay un arco, su vejez se establece a cero y si no existe se crea uno con vejez cero; a continuación se incrementa la vejez de todos los arcos que parten de *CeldaGanadora* (incluido el que la une con *CeldaSegunda*) en una unidad.

En el siguiente paso se incrementa el error de la celda ganadora y se adaptan los vectores de referencia de la ganadora y de sus vecinas, tratando de aproximarlos al vector que describe el objeto u que está siendo procesado. La adaptación de la celda ganadora será mucho mayor ($\epsilon_b >> \epsilon_n$) que la de sus vecinas.

Posteriormente, el procedimiento EliminarArcos se encarga de ver qué arcos tienen una vejez superior a un determinado umbral (v_{max}) y los suprime. La eliminación de arcos puede, eventualmente, dejar celdas sin conexión alguna, en cuyo caso son también destruidas.

Cada λ objetos presentados se inserta una nueva celda utilizando el procedimiento InsertarCelda. Este procedimiento selecciona las celdas q y f, donde q es la celda

Algoritmo 4.2 Función que determina el momento en el que la red dejará de ser entrenada.

```
Función CriterioDeParada (Comparaciones, Red): CIERTO/FALSO
// se pone el conjunto de comparaciones locales de cada celda a vacío
Clocales_c = ∅      ∀c ∈ CeldasDe(Red);
para (cada comparación C de Comparaciones) hacer
  Mejor = ObjetoMejor(C);
  Peor = ObjetoPeor(C);
  CeldaMejor = ObtenerCeldaMásCercana(Red, Mejor);
  CeldaPeor = ObtenerCeldaMásCercana(Red, Peor);
  si (CeldaMejor = CeldaPeor) entonces
    Clocales_CeldaMejor = Clocales_CeldaMejor ∪ C;
  fin si
fin para
para (cada celda C de CeldasDe(Red)) hacer
  si (Clocales_c = ∅) entonces
    retorna CIERTO;
  fin si
fin para
retorna FALSO;
```

con mayor error acumulado de toda la red y f es la vecina de q con mayor error acumulado. La nueva celda se inserta entre q y f con un vector de referencia en el punto medio de ambas. Además, se elimina el arco entre q y f, se decrementa su error en una fracción de α y se crean arcos desde ellas a la nueva celda. El error de la nueva celda se inicializa con la media de los errores de q y f. Finalmente, se actualiza el error de todas las celdas de la red, decrementándolo en una fracción de β.

Este mismo proceso debe realizarse con el segundo objeto de la comparación. Cuando ambos objetos han sido presentados se procesa la siguiente comparación. El entrenamiento finalizará cuando se cumpla un determinado criterio de parada, que definimos en la sección siguiente.

4.5.1. El criterio de parada

El criterio utilizado para detener el entrenamiento está basado en algunas características que la red debe tener para las siguientes fases de $\mathcal{LACE}$, por lo que deben hacerse algunas consideraciones.

El uso del `GNG` nos permite obtener un conjunto de vectores de referencia o prototipos que representan a grupos de objetos. Estos prototipos se van a utilizar para definir las condiciones de aplicación de las funciones de valoración que se calcularán posteriormente por lo que, a priori, parece lógico pensar que cuanto mayor sea el nú-

mero de prototipos, mayor es el detalle con el que se definen las regiones de aplicación de las funciones.

Sin embargo, para calcular la primera aproximación a las funciones asociadas a los prototipos no es suficiente con conocer los objetos a los que representa, puesto que no disponemos de su valoración, sino que hace falta conocer el resultado de comparaciones. En otras palabras, el número de prototipos no debe ser demasiado alto o de lo contrario, la probabilidad de encontrar comparaciones en las que ambos objetos estén en la misma región de Voronoi será demasiado baja. Evitando esta circunstancia, el algoritmo dispondrá de la información necesaria (*comparaciones locales*) para obtener una primera versión de las funciones de valoración. Entendemos por comparaciones locales aquellas en las que ambos objetos pertenecen a la misma región de Voronoi: dada la comparación $C = (u, v)$ tal que $u < v$ y una red formada por las celdas $\mathcal{A} = \{c_1, c_2, \ldots, c_N\}$, se dice que C es una *comparación local* si ambos objetos de la comparación tienen la misma celda ganadora, es decir:

$$\arg\min_{c\in\mathcal{A}}\|u - w_c\| = \arg\min_{c\in\mathcal{A}}\|v - w_c\| \tag{4.20}$$

De las consideraciones anteriores se deduce que para establecer la condición de parada del `GNG` es necesario alcanzar un compromiso entre el grado de detalle en el establecimiento de las condiciones y la fiabilidad de las funciones de valoración preliminares.

El criterio de parada que se propone pretende dejar que la red evolucione añadiendo todas las celdas que el `GNG` estime oportuno para adaptarse a la topología del problema, siempre que las celdas tengan comparaciones locales. La comprobación del número de comparaciones locales de cada celda se hará tras la presentación completa del conjunto de comparaciones. Esto permitirá que las celdas insertadas en las etapas intermedias del entrenamiento tengan la oportunidad de situarse en posiciones adecuadas en el espacio de ejemplos, puesto que pueden darse situaciones como la que refleja la Figura 4.11, en la que el entrenamiento podría verse detenido prematuramente, dando lugar a una red poco apropiada.

El Algoritmo 4.2 muestra en pseudocódigo el detalle de la implementación del criterio de parada del `GNG`. El procedimiento consiste en inicializar el conjunto de comparaciones locales de cada celda a vacío y para cada objeto de cada comparación del conjunto de entrenamiento se busca la celda más próxima, esto es, su prototipo. Si ambos objetos comparten el mismo prototipo, estamos ante una comparación local que se incluirá en el conjunto de comparaciones locales de la celda ganadora. Tras presentar todas las comparaciones se comprueba que todas las celdas tengan su conjunto de comparaciones locales distinto de vacío. Si es así, la red puede seguir creciendo, puesto que el criterio de parada no se cumple. En otro caso se debe dar por finalizado el entrenamiento de la red.

Ajuste de condiciones

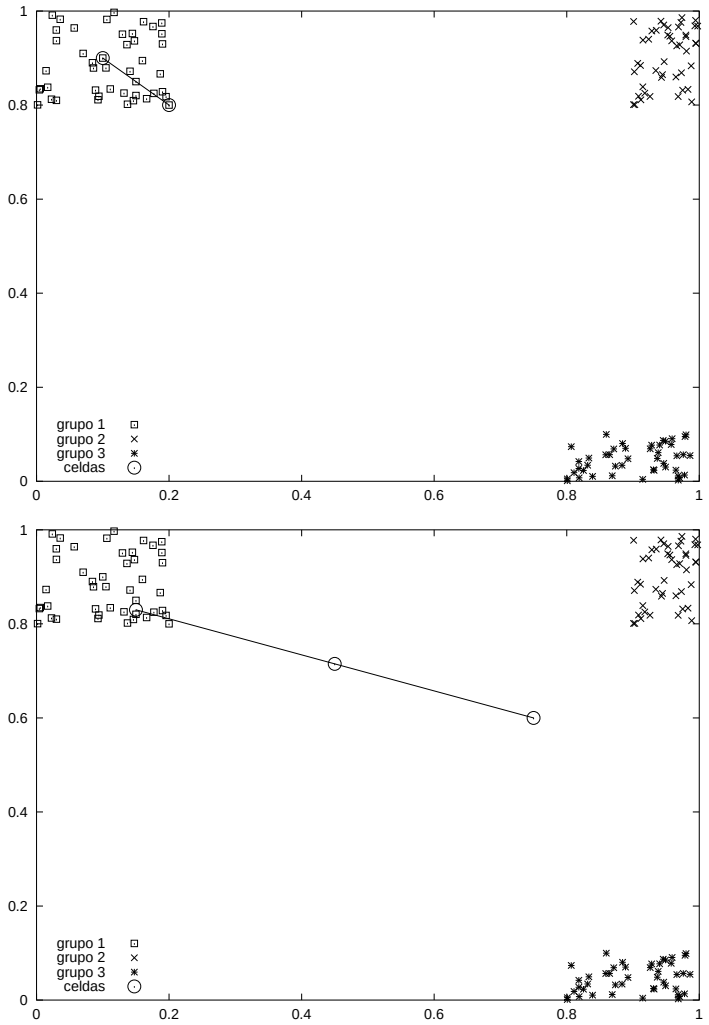


Figura 4.11: En la figura superior se ve una situación en la cual hay tres grupos bien diferenciados de ejemplos. El algoritmo `GNG` ha situado casualmente las dos celdas iniciales en el grupo de ejemplos de la esquina superior izquierda. Después de haber presentado a la red λ comparaciones (figura inferior) una celda se mantiene en el grupo inicial y la otra tiende a aproximarse a los otros dos grupos de la derecha; en ese instante se inserta una nueva celda entre ambas. Si se aplicase en este momento el criterio de parada la nueva celda no *representaría* a ninguna comparación, de manera que el entrenamiento de la red finalizaría dando una solución muy poco apropiada.

Valor	Subatr. Rojo	Subatr. Verde	Subatr. Azul
Rojo	1	0	0
Verde	0	1	0
Azul	0	0	1

Tabla 4.2: Codificación de los atributos discretos. Se crean tantos subatributos como valores puede tomar el atributo. Dado un valor del atributo, su codificación consiste en asignar un 1 al subatributo correspondiente y 0 a los demás subatributos.

4.5.2. Valores desconocidos y atributos simbólicos

En ocasiones, sucede que se desconoce alguno de los valores de los atributos de un ejemplo. Estos valores se denominan *valores desconocidos*. El tratamiento que hacemos de estos valores es el siguiente. Cuando se quiere calcular la distancia de una celda a un ejemplo que tienen algún valor desconocido, la distancia será la que resulte ignorando el atributo en cuestión. Las celdas que deban adaptarse por un ejemplo de estas características, se adaptarán únicamente en los atributos para los cuales el ejemplo tiene valores conocidos.

Aunque la implementación de $\mathcal{LACE}$, no admite, de momento, atributos simbólicos, ya nos hemos planteado la solución de esta circunstancia. Una primera aproximación a la codificación de atributos discretos de forma numérica, podría ser mediante la utilización de números enteros consecutivos para representar cada posible valor del atributo. Sin embargo, si empleamos la métrica Euclídea nos encontraremos con que la distancia entre un par de valores vendrá marcada única y exclusivamente por la propia ordenación de los mismos, lo cual carece de todo fundamento.

En [Kohonen, 1995], se comenta una solución para resolver este problema: hacer equidistante cada par de valores del atributo. En la práctica consistiría en sustituir el atributo discreto original, por tantos nuevos atributos numéricos como valores pueda tomar el de partida. Los nuevos atributos (subatributos) sólo tomarán dos valores posibles, 1 ó 0. En la Tabla 4.2, se muestra un ejemplo de conversión de un atributo discreto (*Color*) con tres valores: *rojo*, *verde* y *azul*. En las filas puede observarse un ejemplo de cómo se convertiría una instancia con cada valor. Esta codificación garantiza que la distancia entre cualquier par de valores sea la misma.

En [del Coz y Bahamonde, 1999] se hace una adaptación de esta idea en la que los valores de los subatributos van auto-organizándose en el tiempo, de manera que en función del conocimiento, algunos valores de un atributo discreto en las celdas estarán más cercanos que otros. Esto produce un efecto *borroso* en los atributos discretos, de manera que las celdas de la red no tendrán un valor concreto para esos subatributos, sino que su valor reflejará el *grado de pertenencia* que tienen con cada uno de los valores posibles. Esta idea ha sido también aplicada con éxito en sistemas como `INNER` [Luaces, 1999] [Luaces et al., 1999] y `BETS` [del Coz, 2000] [del Coz et al., 1999].

Capítulo 5

Cálculo de funciones locales

5.1. Introducción

Tras aplicar el algoritmo de agrupamiento tenemos grupos de ejemplos con características similares. Cada grupo está representado por un prototipo o vector de referencia que, como ya se ha comentado en capítulos anteriores, constituye el conjunto de antecedentes de una regla cuya aplicación indica la función que se debe usar para calificar objetos.

En este capítulo se aborda el problema de encontrar, precisamente, el consecuente de cada una de esas reglas, esto es, las funciones de valoración que van a ser aplicadas a los objetos comprendidos en la región de Voronoi de cada prototipo. Esta función debe ser adecuada, desde el punto de vista local, a dicha región. Por tanto, el criterio de calidad de las funciones ha de establecerse con respecto a objetos comprendidos en la región de Voronoi del prototipo y, dado que no disponemos de sus valoraciones, sólo podemos aspirar a reproducir con el mayor acierto posible las ordenaciones comprendidas en la región. Así, para cada prototipo vamos a buscar una función de valoración lineal que maximice el porcentaje de aciertos de ordenación en su región de Voronoi considerando únicamente las comparaciones locales, en las que ambos objetos pertenecen a la misma región.

En la Sección 3.2 se mostró como, dados dos objetos representados por los vectores u y v, si u es peor que v, el vector normalizado en la dirección de la diferencia $w = \frac{v-u}{\|v-u\|}$ define un hiperplano donde la distancia desde éste a los objetos representa una función de valoración local válida. Supongamos que tenemos un problema donde los objetos son descritos mediante dos atributos. Si representamos las diferencias normalizadas en un gráfico (como en la Figura 5.1), entonces sabemos que la solución que buscamos es la recta que, pasando por el origen de coordenadas, separa la región

donde están estos puntos de donde no están. En realidad, se necesita la hipótesis, a menudo sobreentendida, de que el objeto descrito por un vector de coordenadas cero es el peor posible. Si describimos el hiperplano (en este caso recta) como

$$\sum_{i=0}^{d-1}(a_i X_i) = 0 \tag{5.1}$$

donde d es el número de atributos, X_i es la variable i-ésima y a_i es el coeficiente i-ésimo, entonces la valoración de los objetos será su distancia respecto a dicho hiperplano

$$\text{valoración}(u) = \sum_{i=0}^{d-1}(a_i u_i) \tag{5.2}$$

siendo u_i el atributo i-ésimo que describe el objeto u.

5.2. Usando la media de las diferencias

Una primera aproximación a la solución podría ser el hiperplano perpendicular al vector media de las diferencias normalizadas. Así, siguiendo con el ejemplo anterior, si la media de los vectores diferencia fuese el vector (a, b), la solución sería la recta $ax + by = 0$. Sin embargo, si la distribución de las diferencias no es uniforme, la media quedará desviada dando lugar a un hiperplano también desviado con respecto al que representa una solución adecuada.

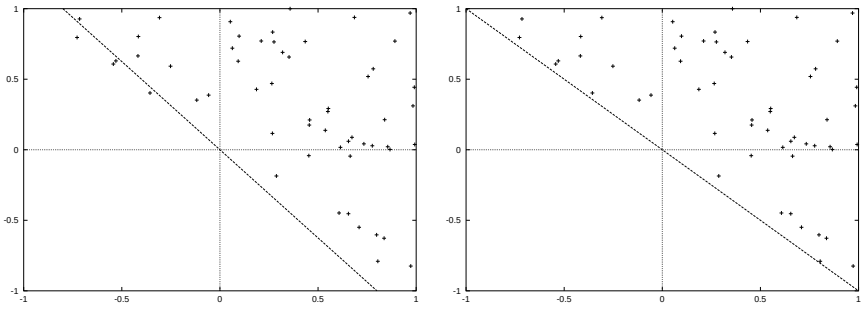
La Figura 5.1(a) muestra un conjunto de diferencias normalizadas y el hiperplano perpendicular al vector media. Se puede observar que la división que efectúa esta recta no deja a un lado todas las diferencias. En 5.1(b) se puede ver una solución óptima.

En general, la media es una aproximación razonable a la solución, pero debe ser mejorada para intentar subsanar la desviación que le ocasiona la falta de uniformidad en la distribución de comparaciones.

5.3. Método de búsqueda de la mejor función

Puesto que la media no conduce a la mejor solución en determinadas circunstancias, es necesario utilizar otro método de cálculo de las funciones de valoración. El método empleado está inspirado en el algoritmo de aprendizaje de categorías simbólicas `OC1` [Murthy et al., 1994], cuya adaptación a $\mathcal{LACE}$ se describe a continuación. Este método, a su vez, está tomado del sistema `CART` [Breiman et al., 1984].

Dados dos objetos u y v tales que $u < v$ y dado un hiperplano como el de la Ecuación (5.1), la valoración de cada objeto se calcula tal como se indica en (5.2),



(a) (b)

Figura 5.1: La recta propuesta como solución utilizando la media se desvía ligeramente de la solución correcta (a). En (b) puede verse una mejor aproximación puesto que la solución que se busca es la recta que, pasando por el origen de coordenadas, separa la región del espacio donde están las diferencias del resto.

con lo que la función de valoración consigue un acierto si otorga a u una valoración menor que a v, es decir, si ocurre que:

$$\sum_{i=0}^{d-1}(a_i u_i) < \sum_{i=0}^{d-1}(a_i v_i); \tag{5.3}$$

$$\sum_{i=0}^{d-1}(a_i v_i) - \sum_{i=0}^{d-1}(a_i u_i) > 0; \tag{5.4}$$

$$\sum_{i=0}^{d-1}(a_i (v_i - u_i)) > 0; \tag{5.5}$$

Considerando las diferencias normalizadas entre cada par de objetos que constituyen cada una de las comparaciones del conjunto de entrenamiento, el número de aciertos en ordenación de la función definida por un hiperplano de coeficientes $(a_i : i = 0, \ldots, d-1)$ es igual al número de veces que se cumple que

$$\sum_{i=0}^{d-1}(a_i x_i) > 0$$

donde $(x_0, \ldots, x_{d-1})$ representa una diferencia normalizada.

De manera alternativa, despejando a_m de la inecuación, se considera un acierto cuando se cumple que

$$a_m > \frac{-(a_0 x_0 + \ldots + a_{m-1} x_{m-1} + a_{m+1} x_{m+1} + \ldots + a_{d-1} x_{d-1})}{x_m} \tag{5.6}$$

si $x_m > 0$, o cuando se cumple que

$$a_m < \frac{-(a_0x_0 + \ldots + a_{m-1}x_{m-1} + a_{m+1}x_{m+1} + \ldots + a_{d-1}x_{d-1})}{x_m} \tag{5.7}$$

si $x_m < 0$. Si $x_m = 0$ el número de aciertos será el número de veces que ocurra que

$$a_0x_0 + \ldots + a_{m-1}x_{m-1} + a_{m+1}x_{m+1} + \ldots + a_{d-1}x_{d-1} > 0 \tag{5.8}$$

Esta alternativa para comprobar si se acierta o falla una comparación constituye la idea básica empleada por el algoritmo `OC1` para subdividir el espacio de ejemplos y efectuar labores de clasificación. En nuestro caso, este enfoque nos indica los valores entre los que deben moverse los coeficientes que definen el hiperplano de valoración para maximizar el número de aciertos. Por ejemplo, si tenemos que $x_m > 0$, $a_m = 0{,}15$ y $\frac{-(\ldots)}{x_m} = 0{,}27$ entonces estamos ante un fallo, puesto que el coeficiente $a_m < 0{,}27$ cuando debería ser mayor; pero sabemos que si modificamos a_m asignándole, digamos, el valor 0,28 ($0{,}28 > 0{,}27$) la comparación será acertada.

Así, para optimizar una función, lo que se hará será ir optimizando cada coeficiente, fijados los demás. Sin embargo, hay que ser cuidadosos con estas modificaciones, puesto que es posible que al asignar el valor 0,28 a a_m para acertar una comparación hayamos provocado que se fallen otras comparaciones que con $a_m = 0{,}15$ eran acertadas. Para el coeficiente a_m se contabilizarán los aciertos según las ecuaciones (5.6), (5.7) y (5.8). Si consideremos

$$\frac{-(a_0x_0 + \ldots + a_{m-1}x_{m-1} + a_{m+1}x_{m+1} + \ldots + a_{d-1}x_{d-1})}{x_m} = C, \tag{5.9}$$

entonces tendremos que buscar un a_m que maximice el número de aciertos en el conjunto de diferencias teniendo en cuenta que

$$\text{es acierto si } \begin{cases} a_m > C & \text{si} \quad x_m > 0 \\ a_m < C & \text{si} \quad x_m < 0 \end{cases} \tag{5.10}$$

y que si $x_m = 0$ entonces el valor de a_m no influye en el acierto o fallo de la comparación.

Para mayor claridad llamaremos c_i al cociente de la ecuación (5.9), cuando para considerar un acierto en la comparación i-ésima, el coeficiente a_m deba ser menor que dicho cociente, y lo denotaremos como C_i cuando a_m deba ser mayor. Para un determinado coeficiente se nos puede plantear la situación siguiente:

$$c_1 < C_7 < C_2 < C_4 < c_5 < c_0 < c_3 < c_6$$

donde las *ces* están colocadas de menor a mayor. En un caso como este parece que la mejor ubicación para un valor a_m será entre C_4 y c_5, puesto que deja más C_i a la izquierda y c_i a la derecha.

5.3.1. Valores desconocidos

Ya se ha comentado la posible existencia de valores desconocidos en los ejemplos. Si uno de los dos ejemplos de una comparación tiene valores desconocidos, la diferencia normalizada de la comparación para esos atributos deberá tomar un valor nulo, es decir cero. Consideramos que al no conocer uno de los dos valores que dan lugar a la diferencia no podemos cuantificar en que grado están distantes y si la diferencia es favorable a uno u otro ejemplo. Por esta razón decidimos considerar que no hay diferencias entre los dos objetos de la comparación para ese atributo.

Esto se ajusta perfectamente al desarrollo matemático expuesto en este capítulo. Cuando se quiera optimizar un coeficiente y nos encontremos con un valor cero en la diferencia normalizada para el atributo relacionado, se aplicará la ecuación (5.8), luego, el valor desconocido no afectará al cálculo de la función.

5.4. Aplicación del método

En los Algoritmos 5.1 y 5.2, puede verse la aplicación de esta teoría al sistema de aprendizaje $\mathcal{LACE}$. Después de buscar una buena colocación para las reglas el sistema llama al procedimiento CalcularFuncionesLocales. Como ya se ha comentado, en este paso se trabaja únicamente con las comparaciones locales, por esto, lo primero que se hace es obtener las comparaciones locales de cada regla mediante el procedimiento CalcularLocalesDeCadaRegla.

Una vez que se tienen las comparaciones de cada regla, se calcula su función de valoración. Al igual que en [Murthy et al., 1994], se realizará el cálculo de la función R veces. La primera vez se tomará como punto de partida de la media de las diferencias normalizadas y se irá optimizando la función hasta que no haya mejoría en el número de aciertos. Las veces siguientes se partirá de funciones aleatorias que se optimizarán también en busca de la mejor solución posible. En $\mathcal{LACE}$, el valor de R utilizado es 2, ya que consideramos que la media es una buena aproximación a la solución y, por tanto, la optimización de la misma suele dar los mejores resultados. Dejamos un segundo intento mediante la optimización de una función aleatoria por si estuviésemos ante un conjunto de diferencias normalizadas muy descompensado, en el que la media, posiblemente, no nos llevaría a la mejor solución.

Finalmente, la función de valoración que se asigna a una regla, será aquella que mayor número de comparaciones locales haya clasificado correctamente, independientemente de que dicha función haya sido obtenida partiendo de la media de diferencias normalizadas o de una función aleatoria.

La parte más compleja de este procedimiento se desarrolla en el método OptimizarFunción. En el Algoritmo 5.2 se muestra su descripción detallada. Como puede observarse, la optimización de la función finaliza cuando no se alcanzan mejoras en el número de aciertos.

Algoritmo 5.1 Procedimiento encargado del cálculo de la función de valoración de cada regla a partir de las comparaciones locales de cada una.

```
Procedimiento CalcularFuncionesLocales(Reglas,Comparaciones,Ejemplos)
CalcularLocalesDeCadaRegla(Reglas,Comparaciones,Ejemplos);
para cada regla r de Reglas hacer
  Diferencias = Diferencias(Locales_r);
  para i desde 1 hasta R hacer
    si (i = 1) entonces
      Función_r = Media(Diferencias);
      aciertos = Aciertos_r = CalcularAciertos(Función_r, Diferencias);
    si no
      función = Aleatoria();
      aciertos = CalcularAciertos(función, Diferencias);
    fin si
    repetir
      aci = aciertos;
      función = OptimizarFunción(Diferencias, función, aciertos);
    hasta que (aci = aciertos)
    si (aciertos > Aciertos_r) entonces
      Función_r = función;
      Aciertos_r = aciertos;
    fin si
  fin para
fin para
```

Algoritmo 5.2 Función que se ocupa de optimizar la función de valoración para una regla dadas las diferencias normalizadas que clasifica.

```
Función OptimizarFunción(Diferencias, función, aciertos):función
Mejora = CIERTO;
mientras (Mejora) hacer
  // PASO 1: perturbación determinística
  repetir
    para cada coeficiente c de función hacer
      valor_c = OptimizarCoeficiente(c, Diferencias, aciertosOP);
      si (aciertosOP ≥ aciertos) entonces
        aciertos = aciertosOP;
        función_c = valor_c;
      fin si
    fin para
  hasta que (aci = aciertos)
  // PASO 2: perturbación no determinística
  Mejora = FALSO;
  para i desde 1 hasta J hacer
    vector = Aleatorio(función);
    funAux = OptimizarAlfa(vector, función, Diferencias, aci);
    si (aci >aciertos) entonces
      aciertos = aci;
      función = funAux;
      i = J;
      Mejora = CIERTO;
    si no
      i = i + 1;
    fin si
  fin para
fin mientras
```

Para tratar de optimizar una función se siguen las dos etapas definidas por los autores del OC1. El primer paso, denominado *perturbación determinística*, selecciona para cada coeficiente el valor que maximice el número de aciertos. Para seleccionar el mejor valor, la función OptimizarCoeficiente sigue el método explicado en la sección anterior. Esta optimización (coeficiente a coeficiente) finalizará cuando, tras mejorarlos, no se detecte aumento en el número de aciertos respecto de la optimización de coeficientes anterior. Existen diferentes estrategias para abordar esta optimización. En [Murthy et al., 1994], los autores dicen haber experimentado con las siguientes estrategias:

- *Seq*, se optimizan los coeficientes en orden de aparición,
- *Best*, optimiza primero el coeficiente que consigue un mayor número de aciertos, y
- *R-50*, el orden de optimización de los coeficientes es aleatorio y cada coeficiente se optimiza 50 veces.

Comentan los autores que ninguno de estos métodos o estrategias de optimización destaca sobre los demás, así que en nuestro sistema empleamos la estrategia secuencial por su sencillez.

El segundo paso se denomina *perturbación no determinística*. La perturbación determinística ha podido llevarnos a alcanzar un máximo local, con lo que aun teniendo una buena solución, es posible que no hayamos obtenido la mejor solución. Por esta razón, en el segundo paso, se genera un vector aleatorio que se sumará al vector de coeficientes que alcanzaba el máximo local. La idea es dar un salto aleatorio en el espacio de funciones con la esperanza de que la nueva función sea mejor que la vieja, para así, tratar de obtener la función que alcanza el máximo de esa región. La suma del vector aleatorio provoca un incremento distinto para cada coeficiente, sin embargo, no suele dar buenos resultados por sí misma. Es necesario indicar el grado de incremento que se quiere en cada coeficiente, que se calcula como el producto de cada coeficiente por el factor α. Así que el número de aciertos de la nueva función será igual al número de veces que se cumple la siguiente desigualdad:

$$\sum_{i=0}^{d-1} ((a_i + \alpha r_i)x_i) > 0 \tag{5.11}$$

siendo $(x_0, \ldots, x_{d-1}) \in \mathcal{E}$, conjunto de diferencias normalizadas, y r el vector generado de forma aleatoria. Tenemos los coeficientes a_i en un máximo local y hemos generado el vector r, así que nos queda únicamente calcular el valor de α para el que se maximiza el número de aciertos. Partiendo de la ecuación anterior resulta que el α óptimo es aquel que mayor número de veces cumple las siguientes desigualdades:

$$\alpha \quad > \quad \frac{-ax}{rx} \quad \text{si } rx > 0$$

$$\alpha \quad < \quad \frac{-ax}{rx} \quad \text{si } rx < 0$$

Desarrollando la ecuación (5.11) resulta que el número de aciertos se contabiliza como el número de veces que se cumple que $ax + \alpha rx > 0$. Si $rx = 0$ entonces tenemos que debe cumplirse $ax > 0$ para que sea acierto, luego en el caso en que $rx = 0$ el valor de α no influye en que se acierte o falle la diferencia. Si consideramos $-ax/rx = C$ tendremos que cuando $rx > 0$, α debe ser mayor que C para que sea acierto y cuando $rx < 0$, entonces para que se produzca un acierto α debe ser menor que C. Si representamos por *ces* minúsculas aquellas para las que α debe ser menor y por mayúsculas las del caso contrario, estaríamos ante el mismo problema resuelto anteriormente (Sección 5.3), en el que se busca para α un valor que deje a su izquierda el mayor número de mayúsculas posibles y a su derecha el mayor número de minúsculas posibles (si se ordena de menor a mayor).

La perturbación no determinística se repite un máximo de J veces. Si en una de ellas se encuentra una función mejor, entonces se vuelve a realizar la perturbación determinística en busca de la mejor función de esa región. Si después de los J intentos no se mejora, se da por finalizada la búsqueda de la mejor función. Los autores de `OC1` establecen el valor de J a 5 por defecto.

5.5. Máquinas de Soporte Vectorial

Las máquinas de soporte vectorial (`SVM`) [Cristianini y Shawe-Taylor, 2000], introducidas por Vladimir Vapnik en [Vapnik, 1998], tratan de aprender reglas de decisión lineales, $h(\vec{x}) = signo\{\vec{w} \cdot \vec{x} + b\}$, descritas por un vector de pesos $\vec{w}$ y un umbral b. Las `SVM` aprenden a partir de un conjunto de ejemplos de entrenamiento, $S_n = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$, donde $\vec{x}_i \in \mathbb{R}^N$ son las descripciones de los ejemplos e $y_i \in \{-1, +1\}$ es la clase que indica la pertenencia o no a una categoría. Dado un conjunto de entrenamiento S_n en el que las dos categorías son linealmente separables, las `SVM` buscan el hiperplano con la máxima distancia Euclídea a los ejemplos de ambas clases. Esta distancia se conoce como el margen δ (ver Figura 5.2).

Thorsten Joachims ha publicado un estudio [Joachims, 2001b] en el que analiza las características de los problemas de clasificación de textos y justifica teóricamente la calidad de los resultados obtenidos mediante `SVM`. Indica que los problemas de clasificación de textos se caracterizan

- por tener un número muy alto de atributos (pudiendo llegar a 30000 si se toman las palabras como atributos),
- porque los ejemplos se encuentran muy esparcidos en el espacio,
- por un uso heterogéneo de los términos, ya que en lenguaje natural pueden utilizarse diferentes palabras para expresar los mismos conceptos,

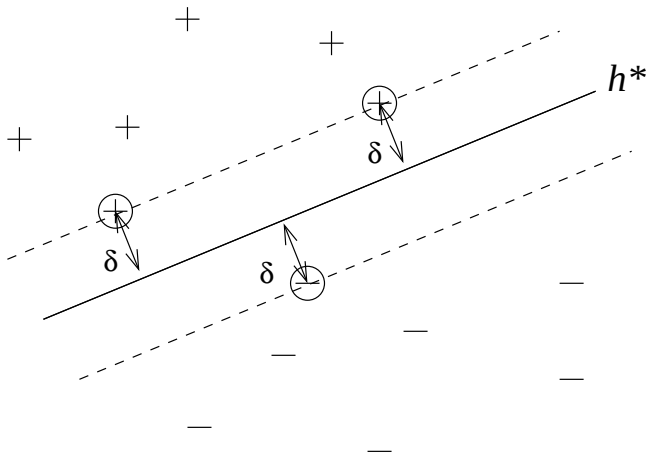


Figura 5.2: Problema de clasificación en dos dimensiones. El hiperplano h^* separa los ejemplos positivos y negativos con el mayor margen posible δ. Los ejemplos a distancia δ del hiperplano se conocen como *vectores de soporte* (marcados con círculos).

- por el alto grado de redundancia (diferentes características relevantes en la tarea de clasificación se ven reflejadas a menudo en un texto), y
- porque la distribución de la frecuencia de las palabras de un texto es muy dispar, ya que un grupo reducido de palabras se repite muy frecuentemente, mientras que la mayoría de las palabras aparecen pocas veces en el texto.

Joachims enuncia un teorema acerca del cálculo del límite del error esperado de `SVM` en la clasificación de textos (demostrado en [Joachims, 2001a]) del que se extrae la idea de que un amplio margen δ, con pocos errores en el conjunto de entrenamiento, conduce a un error de predicción esperado bajo.

El problema que resolvemos mediante la adaptación del `OC1` puede ser resuelto también mediante las máquinas de soporte vectorial. La idea es etiquetar las diferencias normalizadas que se tienen con la clase positiva (Figura 5.1) y esas mismas diferencias multiplicadas por -1 etiquetarlas como clase negativa, de manera que se afronta un problema de separación lineal de clases [Hastie et al., 2002]. La máquina de soporte vectorial deberá buscar el hiperplano que separa ambas clases.

Este tipo de problemas es esencialmente diferente de los problemas de clasificación de textos. La descripción de los ejemplos suele ser bastante menor de 50 atributos, los ejemplos no suelen estar muy esparcidos, no tiene el problema de los términos heterogéneos y no tiene por qué haber diferencias significativas en la frecuencia de aparición del valor de los atributos. En vista de estas diferencias no se puede concluir

que las máquinas de soporte vectorial tengan un buen funcionamiento ante el problema que le proponemos. Es más, se han realizado diversas pruebas (ver sección 8.3.4) en las que se observa que el error en la clasificación es mayor utilizando `SVM` y las diferencias aumentan a medida que se añade ruido al problema. Por tanto, aunque las `SVM` son aplicables a este tipo de problemas, dadas las características del mismo los resultados no son tan buenos como con la adaptación del `OC1`.

Capítulo 6

Reducción del número de funciones

6.1. Introducción

En el capítulo anterior se explicó cómo a partir de vectores de referencia o prototipos en el espacio de ejemplos se forman reglas de valoración locales. El número de funciones obtenidas es igual al número de prototipos, es decir, se tiene una función por cada regla de valoración. Por tanto, hasta esa fase el número de funciones obtenidas es bastante elevado.

En este capítulo se expondrá cómo se ha resuelto la tarea de reducir el número de funciones de valoración. La idea central es buscar regiones formadas por la agrupación de una o varias reglas de forma que todas ellas puedan compartir una misma función de valoración. Ésto nos permitirá obtener una solución más compacta, mucho más comprensible y facilitará el trabajo de la fase siguiente, en la que se hace un ajuste común de todas las funciones obtenidas.

6.2. Método de reducción

Pretendemos incluir en una misma región aquellas reglas que clasifiquen una porción del espacio de ejemplos en la que las comparaciones locales puedan ser correctamente clasificadas mediante una única función lineal. Un problema similar ha sido resuelto exitosamente en [Díez et al., 2002d], donde se presenta una modificación del algoritmo `RISE` (*Rule Induction from a Set of Exemplars*) [Domingos, 1994] [Domingos, 1995] [Domingos, 1996]. Dicho algoritmo trabaja con problemas de categoría simbólica. `RISE`

Algoritmo 6.1 Algoritmo RISE. Se parte de reglas puntuales, que son generalizadas posteriormente.

Algoritmo RISE($Ejemplos$): *Reglas*
Reglas = *Ejemplos*;
mientras (se incremente $Precision(Reglas)$) **hacer**
  para (cada regla R de *Reglas*) **hacer**
    Buscar el ejemplo E más cercano a R que aún no esté cubierto por R y que sea de su misma clase;
    R' = GeneralizaciónMásEspecífica(R, E);
    Reglas$'$ = *Reglas* con R reemplazada por R';
    si ($Precision(Reglas') \geq Precision(Reglas)$) **entonces**
      Reemplaza *Reglas* por *Reglas*$'$;
      si (R' es idéntica a otra regla de *Reglas*) **entonces**
        Elimina R' de *Reglas*;
      fin si
    fin si
  fin para
fin mientras

inicialmente transforma todos los ejemplos en reglas puntuales (cubren un único punto del espacio de entrada) y partiendo de ese conjunto de reglas generado, comienza a generalizarlas en paralelo. Para cada regla busca el ejemplo más cercano (de los contenidos en el conjunto de entrenamiento) que tenga la misma clase que la regla y que aún no esté cubierto por ella. Si mejora la calidad del conjunto de reglas, entonces la generalización es aceptada (ver Algoritmos 6.1 y 6.2). Dado que cada ejemplo ha generado una regla, si se utilizasen todas ellas para calcular la precisión del conjunto, entonces el error sería cero. Este inconveniente se solventa excluyendo, para la clasificación de cada ejemplo, aquella regla generada por el mismo, a no ser que la regla ya haya sido generalizada y cubra además a otros ejemplos.

La solución planteada en RISE puede ser adaptada al problema con que nos enfrentamos en $\mathcal{LACE}$, a pesar de carecer de clases. En nuestro sistema, en lugar de generalizar reglas, lo que pretendemos es agruparlas formando regiones que puedan compartir una misma función. Inicialmente cada regla dará lugar a una región, ya que todas tienen funciones diferentes. El proceso posterior consistirá en tratar de unir iterativamente las dos regiones más cercanas y si la calidad del conjunto aumenta se aceptará la unión. El proceso continuará hasta que no se puedan seguir haciendo uniones de calidad.

En RISE, el cálculo de la precisión se hace mediante una reescritura (con la particularidad antes comentada). En nuestro caso no es posible calcular el fallo en reescritura. Las funciones que tenemos en este punto del algoritmo están calculadas teniendo en cuenta solamente las comparaciones locales que debe valorar cada función, y sin te-

Algoritmo 6.2 Generalización más específica. Se extiende la regla lo necesario para que cubra al ejemplo. Se considera que el $Antecedente_i = Cierto$ si $Ejemplo_i = Regla_i \vee Regla_{i,inferior} \leq Ejemplo_i \leq Regla_{i,superior}$.

Función GeneralizaciónMásEspecífica(*Regla*, *Ejemplo*): *Regla*
para (cada atributo i) **hacer**
  si ($Antecedente_i = Verdadero$) **entonces**
    no se hace nada
  si no si (i es simbólico $\wedge Ejemplo_i \neq Regla_i$) **entonces**
    $Antecedente_i = Verdadero$
  si no si (i es numérico $\wedge Ejemplo_i > Regla_{i,superior}$) **entonces**
    $Regla_{i,superior} = Ejemplo_i$
  si no si (i es numérico $\wedge Ejemplo_i < Regla_{i,inferior}$) **entonces**
    $Regla_{i,inferior} = Ejemplo_i$
  fin si
fin para

ner en consideración aquellas comparaciones que implican a dos reglas de valoración distintas (*comparaciones cruzadas*). Por esta razón las funciones no se encuentran *niveladas*, es decir, las valoraciones propuestas por las funciones pueden estar en escalas totalmente diferentes. Podríamos intentar trasladarlas a la misma escala, pero esto habría que hacerlo también cada vez que se intentase pegar dos regiones, lo que conllevaría un gran coste de computación.

Al unir dos regiones se calculará la función de valoración asociada a la nueva región. Este cálculo se hará utilizando la adaptación del `OC1` ya comentada, es decir, se tratará de maximizar el número de aciertos sobre las comparaciones locales de la región en su conjunto. La unión será aceptada si la calidad de la nueva región es mejor que con las regiones por separado.

En la unión de dos regiones, la región resultante tendrá como ámbito de aplicación la suma de las regiones de Voronoi de las dos regiones de partida. Este comportamiento es equivalente a la poda de árboles: si se eliminan las hojas del nodo $\mathcal{N}$ los ejemplos clasificados por las mismas pasarán a ser clasificados por la nueva hoja $\mathcal{H}_{\mathcal{N}}$ resultante de transformar el nodo $\mathcal{N}$ en hoja. En [Quinlan, 1987] se exponen varios métodos de poda. El método utilizado por Quinlan en su sistema `C4.5` es el *Error-Based Pruning* [Quinlan, 1993], que calcula una estimación pesimista de los fallos que puede cometer cada hoja del árbol. Este criterio puede servirnos para medir la calidad de las regiones en función del número de fallos estimado. Así, si la suma del error estimado de las dos regiones que se tratan de unir es mayor o igual que el error estimado de la región resultante de la unión, nos quedaremos con la nueva. En [Esposito et al., 1997] se comparan varios métodos de poda y los resultados denotan que el método utilizado en `C4.5` es el más eficaz. En [Fürnkranz, 1997] se comenta la posibilidad de combinar

Algoritmo 6.3 Esta función parte de una región por regla y busca el crecimiento de las regiones (uniendo aquellas que pueden ser *compatibles*).

Función ObtenerRegiones(*Reglas*, *Comparaciones*, *Ejemplos*):*Regiones*
Regiones = *Reglas*;
CalcularEjemplosDeCadaRegion(*Regiones*, *Comparaciones*, *Ejemplos*);
mientras (No se cumpla el criterio de parada) **hacer**
  (R_1, R_2) = ObtenerRegionesMasCercanas(*Regiones*);
  R = Unir(R_1, R_2, *Comparaciones*, *Ejemplos*);
  $LSIC_R$ = LimiteSuperiorIC($Locales_R$, $Fallos_R$);
  $LSIC_{R_1,R_2}$ = LimiteSuperiorIC($Locales_{R_1} + Locales_{R_2}$, $Fallos_{R_1} + Fallos_{R_2}$);
  si ($LSIC_R \leq LSIC_{R_1,R_2}$) **entonces**
    $Regiones = Regiones \setminus R_1$;
    $Regiones = Regiones \setminus R_2$;
    $Regiones = Regiones \cup R$;
  fin si
fin mientras
retorna *Regiones*;

las estrategias *pre-pruning* (tratar de detectar el ruido durante el crecimiento del árbol) y *post-pruning* (dejar el tratamiento del ruido para la poda). La unión de dos regiones también puede considerarse como la eliminación de antecedentes de una regla en el sentido de que al eliminar un antecedente se está ampliando el ámbito de aplicación de la regla.

6.3. Aplicación del método

En el Algoritmo 6.3 se muestra la manera de llevar a la práctica lo comentado en la sección anterior. Inicialmente se crean tantas regiones como reglas. Se calculan los ejemplos representados por cada región, de manera que se asocian a cada una de ellas las comparaciones locales y cruzadas que clasifica. Las comparaciones cruzadas son aquellas en las que cada objeto de la comparación es clasificado por regiones diferentes. Posteriormente se entra en un proceso iterativo, del que no se sale mientras no se cumpla el criterio de parada, esto es, cuando no se puedan realizar más uniones con calidad suficiente para ser aceptadas.

La unión de dos regiones no se hace como la generalización de `RISE`. En $\mathcal{LACE}$, las reglas tendrán como condición un punto del espacio, cuyo ámbito de aplicación vendrá definido por su región de Voronoi (ver Figura 6.1). La unión de dos regiones se corresponderá con la unión de sus respectivas regiones de Voronoi asociadas a cada región (ver Figura 6.2). Se tomarán las dos regiones más cercanas (R_1, R_2), es decir, aquellas que tienen los vectores de condiciones a mínima distancia, y se intentarán

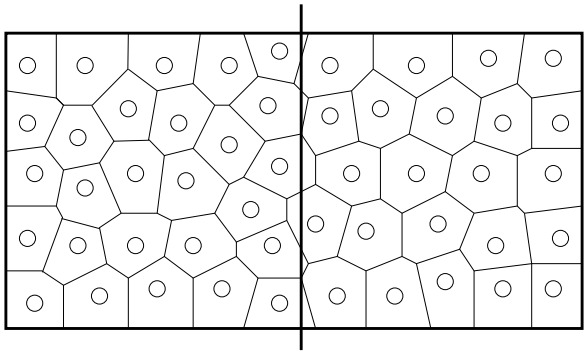
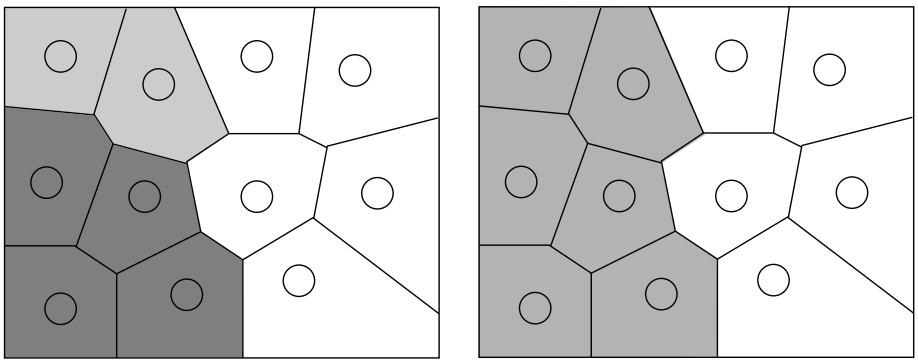


Figura 6.1: En la figura se representa un problema cuya solución viene dada por dos funciones; su ámbito de aplicación se corresponde con cada una de las mitades del rectángulo que representa el espacio del problema. Sobre el rectángulo se encuentran distribuidas las condiciones de las reglas (◯) y sus regiones de Voronoi.



(a) (b)

Figura 6.2: En la figura (a) se muestra el estado anterior al intento de unión de las dos regiones sombreadas, una formada por cuatro reglas (y cuatro regiones de Voronoi) y otra formada por dos reglas (y dos regiones de Voronoi). En la figura (b) se ve como la unión ha creado una nueva región con seis reglas y seis regiones de Voronoi. Las demás regiones no se han visto afectadas por la unión.

Algoritmo 6.4 Dadas dos regiones, trata de unirlas en una región. Si la estimación del error es favorable se aceptará la unión.

Función UNIR($R1$, $R2$, $Comparaciones$, $Ejemplos$):$Region$
$Locales_R = (Locales_{R1} \cup Locales_{R2}) \cup (Cruzadas_{R1} \cap Cruzadas_{R2})$;
$Cruzadas_R = (Cruzadas_{R1} \cup Cruzadas_{R2}) \setminus (Cruzadas_{R1} \cap Cruzadas_{R2})$;

$Diferencias$ = DIFERENCIAS($Locales_R$);
$Función_R$ = OPTIMIZARFUNCIÓN($Diferencias, Función_R, Aciertos_R$);
retorna R;

unir formando una nueva región (R) que tendrá como comparaciones locales

$$Locales_R = (Locales_{R_1} \cup Locales_{R_2}) \cup (Cruzadas_{R_1} \cap Cruzadas_{R_2})$$

y como comparaciones cruzadas

$$Cruzadas_R = (Cruzadas_{R_1} \cup Cruzadas_{R_2}) \setminus (Cruzadas_{R_1} \cap Cruzadas_{R_2})$$

Conociendo las comparaciones locales de la nueva región, es decir, aquellas en las que ambos objetos caen en la nueva región de Voronoi, podemos calcular una función que clasifique correctamente el mayor número de dichas comparaciones, tal como se ve en el Algoritmo 6.4. El cálculo de dicha función se hará utilizando la ya comentada adaptación del `OC1`. Esta nueva región se comparará en calidad con las dos de las que surge. Si resultase que la nueva región tiene mayor calidad, entonces todas las reglas que contiene compartirían la función que acaba de calcularse.

Pero, ¿cómo se evalúa la calidad de las regiones?. El espacio que cubre la región R es la suma de los espacios cubiertos por R_1 y R_2, luego la unión de ambas da lugar a una región de valoración más amplia. Este mismo efecto se consigue cuando se eliminan antecedentes de una regla; al eliminar un antecedente el espacio cubierto por la regla se hace mayor. En [Quinlan, 1993], Quinlan acomete la tarea de eliminar antecedentes irrelevantes después de haber generado las reglas a partir del árbol que construye `C4.5`. Para ello, considera la regla R de la forma

Si A entonces clase C

y una regla R^- más general

Si A^- entonces clase C

donde A^- se obtiene eliminando la condición X de entre las condiciones de A. Cada ejemplo del conjunto de entrenamiento que satisfaga el antecedente A^- puede o no pertenecer a la clase C y puede o no satisfacer la condición X. Los números resultantes pueden organizarse en la *tabla de contingencias* (Tabla 6.1).

Algoritmo 6.5 Esta es la función utilizada por Quinlan para el cálculo del límite superior de la probabilidad de fallo en el sistema `C4.5` Release 8.

```
float Val  [] = {0,0.001,0.005,0.01,0.05,0.10,0.20,0.40,1.00},
      Dev  [] = {4.0,3.09,2.58,2.33,1.65,1.28,0.84,0.25,0.00};

float AddErrs(N, e)
    ItemCount N, e;
{
    static float Coeff=0;
    float Val0, Pr;

    if ( ! Coeff ){
      /* Compute and retain the coefficient value, interpolating from
         the values in Val and Dev */
      int i;
      i = 0;
      while ( CF > Val[i] ) i++;

      Coeff = Dev[i-1] +
              (Dev[i] - Dev[i-1]) * (CF - Val[i-1]) /(Val[i] - Val[i-1]);
      Coeff = Coeff * Coeff;
    }

    if ( e < 1E-6 ){ // Se acierta todo
      return N * (1 - exp(log(CF) / N));
    }
    else if ( e < 0.9999 ){ // Se acierta casi todo
      Val0 = N * (1 - exp(log(CF) / N));
      return Val0 + e * (AddErrs(N, 1.0) - Val0);
    }
    else if ( e + 0.5 >= N ){ // Se falla todo
      return 0.67 * (N - e);
    }
    else{ // Caso general
      Pr = (e + 0.5 + Coeff/2
              + sqrt(Coeff * ((e + 0.5) * (1 - (e + 0.5)/N) + Coeff/4)) )
            / (N + Coeff);
      return (N * Pr - e);
    }
}
```

	Clase C	**Otras clase**
Cumple la condición X	Y_1	E_1
No cumple la condición X	Y_2	E_2

Tabla 6.1: Tabla de contingencias utilizada en `C4.5Rules`.

Observando la tabla, tenemos que la regla R cubre $Y_1 + E_1$ ejemplos, de los que falla E_1; y la regla R^- cubre $Y_1 + Y_2 + E_1 + E_2$ y falla $E_1 + E_2$. Con estos datos, Quinlan calcula una *estimación pesimista* del fallo cometido por las reglas R y R^-. Dicho cálculo se realiza con la función mostrada en el Algoritmo 6.5 que no deja de ser la fórmula

$$IC(r) = \frac{p + \frac{z^2}{2n} \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{6.1}$$

tomada de [Spiegel, 1970], donde n es el número de veces que se aplica la regla r, p es la proporción de fallos y z es, según el nivel de confianza α usado para establecer el intervalo, un valor tomado de la distribución normal. Quinlan propone un nivel de confianza del 25 %. Lo peculiar del Algoritmo 6.5 se encuentra en los niveles de acierto extremos, donde no fallar o fallarlo todo se consideran casos especiales y se tratan individualmente. Una vez efectuado el cálculo de las estimaciones, si la estimación pesimista de R^- no es mayor que la de R entonces se puede eliminar la condición X.

Así pues, aplicamos esta misma idea a nuestro algoritmo. Tenemos dos regiones (R_1 y R_2) que unidas dan lugar a la región R. La estimación pesimista del error de R se calculará con $Locales_R$ como número de cubiertos y $Fallos_R$ como número de errores sobre las comparaciones locales. El valor resultante se comparará con la situación que quedaría si no se efectuase la unión, es decir, el número de comparaciones locales cubiertas sería $Locales_{R_1} + Locales_{R_2}$ y el número de errores sería $Fallos_{R_1} + Fallos_{R_2}$. Si la estimación pesimista de R no es mayor que la de $\{R_1, R_2\}$, entonces se acepta la nueva región en sustitución de las otras dos.

Como consecuencia de la aplicación del mecanismo descrito, cada región estará compuesta por una o varias reglas, teniendo cada regla como antecedente un punto del espacio de ejemplos y como consecuente la función común de la región. La distancia entre dos regiones será la mínima distancia entre los antecedentes de una región respecto a los antecedentes de la otra. La función ObtenerRegionesMasCercanas, busca las regiones más cercanas entre sí que no se hayan intentado unir con anterioridad o que habiéndolo intentado sin éxito, una de las dos ya ha crecido en alguna dirección.

6.4. Eliminación de reglas irrelevantes

Una vez que han sido localizadas las regiones del problema, y por tanto, tenemos un número reducido de funciones, podemos reducir el número de reglas que hay en el interior de cada región. Como se ha dicho, cada región tendrá una única función que comparten todas las reglas pertenecientes a la misma, así que la reducción de reglas puede abordarse como un problema de reducción de instancias, donde en este caso la clase se corresponde con la función asociada a cada región. Las *"instancias"* de la misma clase son las reglas que forman parte de la misma región.

Existen muchos algoritmos de reducción de instancias capaces de reducir considerablemente el conjunto de partida. En [Wilson y Martinez, 2000] aparece una comparativa entre diversos algoritmos de este tipo y se ve cómo la familia de algoritmos `DROP` (*Decremental Reduction Optimization Procedure*) tiene una gran capacidad de eliminación de instancias manteniendo una precisión en la clasificación alta. Esta familia de algoritmos basa su funcionamiento en la siguiente idea:

> Se elimina la instancia P si el número de ejemplos correctamente clasificados con la intervención de P, siguen estando correctamente clasificados si se elimina P.

Cada una de las versiones aplica esta idea con ligeras modificaciones, como el orden de presentación de las instancias o la utilización de distintos filtros de ruido.

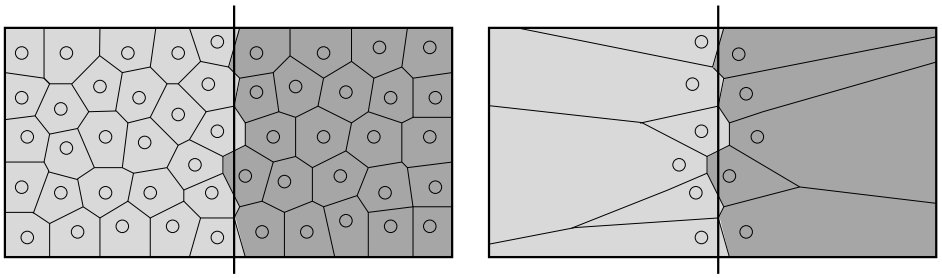
Otro algoritmo desarrollado más reciente es el `ICF` (*Iterative Case Filtering Algorithm*) [Brighton y Mellish, 2002]. Este algoritmo utiliza los conceptos de *accesibilidad* (reachability) y *cobertura* (coverage). La cobertura de una instancia es el conjunto de ejemplos para los que interviene en su evaluación, mientras que la accesibilidad de un ejemplo es el conjunto de ejemplos que se utilizan para su evaluación. La idea de este algoritmo es ir eliminando todas las instancias x tal que $|accesibilidad(x)| > |cobertura(x)|$, es decir, se eliminan las instancias para las que el número de ejemplos que contribuyen a su evaluación es mayor que el número de ejemplos para los que la instancia interviene en su evaluación. En la tabla de resultados de [Brighton y Mellish, 2002], se muestra que este algoritmo se encuentra a la par, en cuanto capacidad de eliminación y precisión, respecto a la familia de algoritmos `DROP`.

Para el problema que nos atañe podría utilizarse cualquiera de estos algoritmos, sin embargo, dado que sabemos cómo es el conocimiento sintetizado por el sistema hasta este punto, puede encontrase una solución más sencilla. Hemos ido formando regiones, cada una de las cuales abarca un conjunto de regiones de Voronoi. La formación de estas regiones se ha realizado mediante un proceso que ha ido evaluando la calidad de las mismas. Por tanto, *confiamos* en las regiones resultantes. Por este motivo no nos interesa modificar su ámbito de aplicación. Lo que queremos es

> borrar, para cada región, aquellas reglas cuya eliminación no modifique su ámbito de aplicación.

Algoritmo 6.6 Este procedimiento elimina aquellas reglas que no son necesarias para delimitar el ámbito de aplicación de las regiones.

```
Procedimiento EliminarReglasIrrelevantes(Regiones, Ejemplos)
si (|Regiones| = 1) entonces
  Eliminar todas las celdas;
  retornar;
fin si
CalcularEjemplosDeCadaRegla(Reglas, Ejemplos);
Ordenar Reglas por "mayor distancia a otra región";
para (cada regla R de Reglas) hacer
  Reglas' = Reglas \ R;
  Eliminar = CIERTO;
  para (cada ejemplo E de Ejemplos_R) hacer
    R_win = ObtenerReglaMásCercana(Reglas', E);
    si (Región(R_win) ≠ Región(R)) entonces
      Eliminar = FALSO;
      break;
    fin si
  fin para
  si (Eliminar) entonces
    Reglas = Reglas';
    Asignar Ejemplos_R a las reglas correspondientes;
  fin si
fin para
```



(a) (b)

Figura 6.3: En ambas figuras se presenta un problema cuyo domino es un rectángulo en el que la mitad de la izquierda se corresponde con una función y la mitad de la derecha con otra. En la figura (a) se muestran las dos regiones resultantes después de haber aplicado el mecanismo de reducción de funciones descrito en este capítulo. El ámbito de aplicación de cada una de las dos regiones se corresponde con la unión de las regiones de Voronoi de las reglas que las componen. En la figura (b) se muestra la situación resultante tras eliminar las reglas irrelevantes. Observamos que el ámbito de aplicación de las dos regiones no se ha visto alterado, mientras que el número de reglas se ha reducido significativamente.

En la Figura 6.3, se ve que partiendo de la situación mostrada en (a) pretendemos quedarnos únicamente con las reglas fronterizas (b) que delimitan el ámbito de aplicación de cada región.

Esta eliminación de reglas irrelevantes para la delimitación del ámbito de aplicación de las regiones puede verse en el Algoritmo 6.6. Puede ocurrir que, al llegar a esta fase, contemos con una única región. En ese caso no necesitamos ninguna regla para delimitar fronteras, así que se eliminan todas las reglas y nos quedamos únicamente con la función asociada a la región. Esto equivale a retornar una *regla por defecto* cuyo ámbito de aplicación será todo el espacio de ejemplos. Esta regla por defecto será la solución final de $\mathcal{LACE}$. Si por el contrario tenemos varias regiones, la situación más habitual, procedemos a aplicar el principio antes expuesto, es decir, borraremos las reglas que no alteran el ámbito de aplicación de su región.

Capítulo 7

Ajuste global del funciones

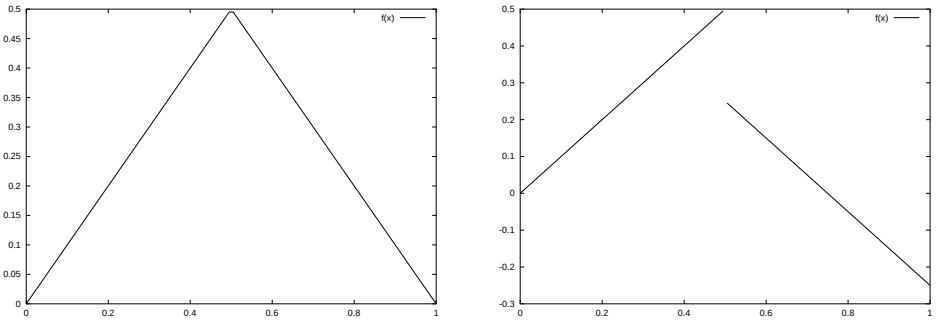
7.1. Introducción

Una vez determinado el número de funciones de un problema y el ámbito de aplicación de cada una de ellas, resta nivelar dichas funciones para dar por finalizado el algoritmo. Las funciones han sido calculadas atendiendo únicamente a las comparaciones locales, porque como ya se ha comentado, estas comparaciones son las que indican la pendiente correcta (el signo y el grado de inclinación) de la función. Sin embargo, las comparaciones cruzadas (cada objeto de la comparación es valorado por una función diferente) pueden no quedar correctamente clasificadas, ya que pueden estar en escalas diferentes. En este capítulo veremos cómo modificar las funciones para que se optimice el número de aciertos sobre las comparaciones cruzadas sin que se altere el número de aciertos sobre las comparaciones locales, puesto que consideramos que nuestras funciones actúan correctamente en su ámbito local.

Obviamente, esta fase del algoritmo no sería necesaria si el proceso comentado en el capítulo anterior concluye que el problema se resuelve con una única función. En este caso, el algoritmo $\mathcal{LACE}$ finalizaría proponiendo como solución la *regla por defecto* cuya función de valoración será la obtenida tras la fase de reducción del número de funciones.

7.2. Método utilizado para la nivelación

Al llegar a este punto del algoritmo ya hemos obtenido un conjunto de regiones, cada una de ellas formada por un grupo de reglas que comparten las misma función. El ámbito de aplicación de la función de una región ($Cobertura(R_i)$) es la unión de las zonas del espacio de ejemplos cubiertas por las regiones de Voronoi de las reglas que



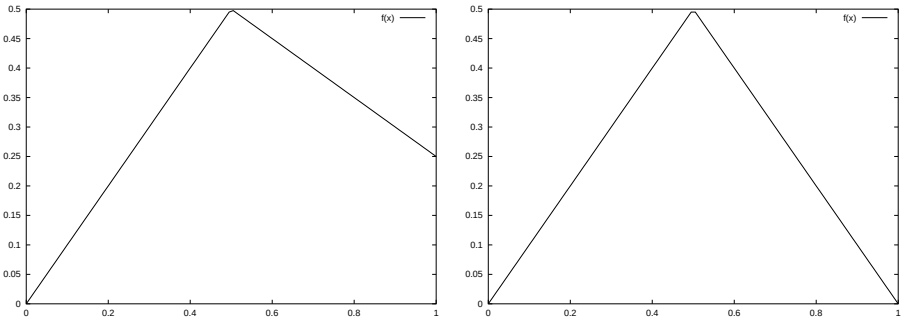
(a) (b)

Figura 7.1: En (a) se representa un problema en el que la función de valoración es $f(x) = \begin{cases} x, & \text{si } x \in [0,0{,}5] \\ 1-x, & \text{si } x \in (0{,}5,1] \end{cases}$. Observando únicamente las comparaciones locales puede llegarse a una solución como la que se muestra en (b). Las comparaciones locales quedarán bien clasificadas, puesto que la pendiente de cada parte del dominio es correcta, sin embargo algunas de las comparaciones cruzadas entre ambas partes del dominio serán clasificadas incorrectamente.

la componen, como ya se ha visto en la Figura 6.2. Así pues, partimos de *reglas* del estilo $w \leftarrow \mathit{Cobertura}(R_i)$, donde para un objeto x situado en la zona de cobertura de la región R_i la valoración propuesta para el mismo será el resultado del producto escalar del vector de coeficientes y el vector descriptivo del objeto $(w \cdot x)$. Si tenemos la comparación $(u < v)$ y ambos objetos se encuentran en la región R_i, es de esperar que la valoración propuesta por el vector de coeficientes w para u sea inferior que la propuesta para v, ya que el vector w fue calculado para que se cumpliese la desigualdad $w \cdot (v - u) > 0$. Sin embargo, la comparación puede resultar equivocada cuando cada uno de los objetos es valorado por la función de distintas regiones.

En la Figura 7.1 vemos un ejemplo de esta situación. En (a) se muestra la solución correcta al problema y en (b) la solución que nuestro algoritmo alcanzaría sin nivelar las diferentes funciones. Si nunca se comparasen objetos de distintas regiones la solución sería razonablemente válida. El problema surge cuando se pretende comparar objetos de regiones diferentes. Los objetos de la parte izquierda del dominio tenderán a recibir mejores valoraciones que los de la parte derecha. La solución a este problema pasa por incluir un término independiente en las funciones de valoración, de manera que las valoraciones de ambas funciones se lleven a una misma escala. De esta manera, la valoración de un objeto x situado en el ámbito de aplicación de la región R_i vendrá dada por la siguiente función:

$$f(x) = (x \cdot w) + b$$



(a) (b)

Figura 7.2: En la figura (a) se representa un problema en el que la función de valoración es $f(x) = \begin{cases} x, & \text{si } x \in [0,0{,}5] \\ \frac{3-2x}{4}, & \text{si } x \in (0{,}5,1] \end{cases}$; se trata de un problema asimétrico. Observando únicamente las comparaciones locales podríamos llegar a una solución como la que se muestra en la Figura 7.1(b). Nivelando las funciones con el término independiente se podría alcanzar la situación que se ve en (b). Sin embargo se necesita modificar el grado de inclinación de las pendientes para alcanzar la mejor solución ya que en este caso la función no tiene pendientes simétricas.

siendo b el término independiente.

El término independiente soluciona el problema de la nivelación, sin embargo, existe otro problema que puede dañar los resultados obtenidos con las comparaciones cruzadas. En la Figura 7.2(a) podemos ver un problema asimétrico. La pendiente de la primera parte del dominio es más pronunciada que la pendiente de la segunda parte. Como el cálculo de las funciones se ha hecho utilizando únicamente las comparaciones locales, no existe una relación calculada entre las pendientes de cada función, así que, después de calcular el término independiente para cada función, se puede llegar a una solución como la que se ve en la parte (b) de la figura. Sin embargo, sigue siendo una solución incorrecta, ya que falta indicar que el grado de inclinación de la función de la derecha debe ser la mitad del de la izquierda.

Veamos cómo se soluciona este problema. Cojamos n objetos u_n tales que $(u_1 < u_2 < \cdots < u_n)$, siendo todos valorados por el vector de coeficientes w, es decir, $(u_1 \cdot w < u_2 \cdot w < \cdots < u_n \cdot w)$. Si multiplicamos la valoración de cada objeto por el mismo valor real positivo a, aunque la valoración de los objetos cambia, el orden de la valoración de los objetos se mantiene, es decir,$(a(u_1 \cdot w) < a(u_2 \cdot w) < \cdots < a(u_n \cdot w))$. Sabiendo que la expresión $a(u_i \cdot w)$ es equivalente a la expresión $(u_i \cdot aw)$, tenemos un *modificador de pendiente* (a) que solucionaría el problema visto en la Figura 7.2. De esta manera, la valoración de un objeto x situado en el ámbito de aplicación de la región R_i será calculada, después de añadir el término independiente y el modificador

de pendiente, mediante la siguiente expresión:

$$f(x) = a(x \cdot w) + b = (x \cdot aw) + b$$

siendo b el término independiente y a el modificador de pendiente.

7.3. Cálculo de los nuevos parámetros

Como se ha dicho, para calcular los nuevos parámetros de las funciones utilizaremos únicamente las comparaciones cruzadas, ya que las comparaciones locales las tenemos correctamente clasificadas y los parámetros que se van a calcular no van a alterar la clasificación de las mismas.

Sea $\mathcal{R}$ el conjunto de regiones y R_i una región perteneciente al mismo. Sea $Cruzadas_{R_i}$ el conjunto de comparaciones cruzadas para las que R_i valora uno de los dos objetos en la comparación. Dada una $C_i \in Cruzadas_{R_i}$, de la forma $u < v$, puede ocurrir que R_i valore a u o a v. Supongamos que valora a v, el objeto ganador en la comparación. En este caso la valoración de u ($Valoración(u)$) nos la dará la función de otra región. Así, tenemos que

$$a(w \cdot v) + b > Valoración(u). \tag{7.1}$$

Despejando a en la desigualdad resulta que

$$\begin{array}{ll} a > \frac{Valoracion(u)-b}{(w \cdot v)} = X & \text{si } (w \cdot v) > 0 \\ a < \frac{Valoracion(u)-b}{(w \cdot v)} = x & \text{si } (w \cdot v) < 0 \end{array} \tag{7.2}$$

luego, para clasificar correctamente esta comparación a deberá ser mayor que X o menor que x, dependiendo del signo del denominador.

Si en lugar de valorar a v, la región R_i valorase a u, el objeto perdedor en la comparación, al despejar a de

$$a(w \cdot u) + b > Valoración(v) \tag{7.3}$$

resultaría

$$\begin{array}{ll} a < \frac{Valoracion(v)-b}{(w \cdot u)} = x & \text{si } (w \cdot u) > 0 \\ a > \frac{Valoracion(v)-b}{(w \cdot u)} = X & \text{si } (w \cdot u) < 0 \end{array} \tag{7.4}$$

Efectuando estos cálculos sobre todas las comparaciones contenidas en el conjunto $Cruzadas_{R_i}$, obtenemos un conjunto de restricciones (X_i y x_i) para el parámetro a. Ordenando dichas restricciones de menor a mayor, y eliminando las menores o iguales a cero, tendremos una situación como la siguiente:

$$X_{132} < X_4 < x_{87} < X_{34} < \cdots < x_{57} < x_{29} < X_{76} < x_{31}$$

El valor óptimo para el parámetro a será aquel que deje más X_i a la izquierda y más x_i a la derecha.

El cálculo del término independiente se realiza de forma similar. Si R_i clasifica al objeto ganador de la comparación v, estaríamos en la situación mostrada en la ecuación (7.1). Despejando la desigualdad resultaría que

$$b > \textit{Valoración}(u) - a(w \cdot v) = Z. \tag{7.5}$$

Despejando b de la ecuación (7.3), para el caso en que R_i valore al objeto perdedor de la comparación u, resulta

$$b < \textit{Valoración}(v) - a(w \cdot u) = z \tag{7.6}$$

Ordenando las restricciones de b de menor a mayor podríamos llegar a la siguiente situación

$$Z_2 < z_{46} < Z_{101} < Z_{62} < \cdots < z_{21} < z_{98} < z_{97} < z_{54}.$$

Al igual que para el cálculo del parámetro a, el mejor valor para el parámetro b será aquel que deje más Z_i a la izquierda y más z_i a la derecha.

En el Algoritmo 7.1 se muestra la integración de este método en $\mathcal{LACE}$. Primero se ordenan las regiones, colocando primero aquellas que mayor número de comparaciones cruzadas tienen. Se toma una región y se busca el modificador de pendiente y el término independiente. Mientras se detecte mejora, tras el cálculo de uno de los dos parámetros, se volverá a tratar de mejorar la función asociada a la región. Se repetirá el mismo proceso con todas las regiones.

La adaptación en su conjunto es un método iterativo, ya que una mejora en una de las funciones puede permitir aumentar el acierto en otras. Esto obliga a que si alguna de las regiones ha visto mejorada su función, se traten de adaptar las funciones del resto de regiones. El proceso finalizará cuando en una iteración completa ninguna de las funciones incremente su acierto. Cuando finalice el proceso, cada regla tomará la función de la región en la que se encuentra contenida. De esta manera, el conjunto de reglas que $\mathcal{LACE}$ presenta como solución final será:

$$\begin{gathered}(x \cdot a_1 w_1) + b_1 \leftarrow u_1 \\ (x \cdot a_2 w_2) + b_2 \leftarrow u_2 \\ \vdots \\ (x \cdot a_n w_n) + b_n \leftarrow u_n\end{gathered}$$

donde u_i representa la condición de la regla que será aplicada por mínima distancia y x la descripción de un posible objeto a valorar.

Algoritmo 7.1 Este procedimiento mejora las funciones obtenidas hasta el momento añadiéndoles un término independiente y modificando ligeramente el grado de inclinación en la pendiente de cada coeficiente.

```
Procedimiento AjustarFuncionesGlobalmente (Regiones, Comparaciones, Ejemplos)
Ordenar Regiones por "mayor número de comparaciones cruzadas";
HayaMejora = CIERTO;
mientras (HayaMejora) hacer
  HayaMejora = FALSO;
  para (cada región R de Regiones) hacer
    HayaIncremento = CIERTO;
    mientras (HayaIncremento) hacer
      (Mayores,Menores) = CalcularMayoresMenores(R);
      MejoraP = OptimizarPendiente(Función_R, Mayores, Menores);
      MejoraTI = OptimizarTérminoIndependiente(Función_R, Mayores, Menores);
      si (MejoraP || MejoraTI) entonces
        HayaMejora = CIERTO;
      si no
        HayaIncremento = FALSO;
      fin si
    fin mientras
  fin para
fin mientras
AsignarFuncionesARegiones();
```

Capítulo 8

Resultados experimentales

8.1. Introducción

En este capítulo se presentan los resultados obtenidos por $\mathcal{LACE}$ en las pruebas experimentales a las que ha sido sometido. Los experimentos se han centrado en evaluar su capacidad de clasificación, así como en analizar otro tipo de factores, como el número de funciones localizadas y la tolerancia del sistema frente al ruido y a la presencia de atributos irrelevantes.

El capítulo comienza presentando los resultados obtenidos en problemas de regresión o aprendizaje con categoría continua adaptados al paradigma del aprendizaje a partir de comparaciones. Compararemos estos resultados con los obtenidos mediante el sistema de aprendizaje para categorías continuas `CUBIST` [Quinlan, 2002], ya que se trata de un sistema reconocido por su alta precisión a la hora de buscar reglas de regresión que pueden ser interpretadas como funciones de calificación.

Posteriormente se mostrará el comportamiento del sistema frente a problemas artificiales diseñados para ejemplificar y probar los distintos aspectos del algoritmo, incluyendo una comparativa entre dos métodos de búsqueda del mejor hiperplano, OC1 y SVM.

La última sección de este capítulo se centrará en la aplicación de $\mathcal{LACE}$ a un problema real.

8.2. Problemas de categoría continua

El sistema $\mathcal{LACE}$ ha sido diseñado para resolver problemas de valoración en los que no está disponible la calificación numérica de los objetos utilizados en el entrenamiento, pero sí la comparación entre pares de objetos. Sin embargo, con el ánimo de poder

Problema	Núm. de atributos	Núm. de ejemplos
bank8FM	8	8192
bodyfat	13	252
bupa	5	345
cpu	6	209
deltaAilerons	5	7129
deltaElevators	6	9517
diabetes	2	43
housing	13	506
kin8NM	8	8192
puma8NH	8	8192
pyrim	27	74
stock	9	950
triazines	60	186
wpbc	32	198

Tabla 8.1: Información relevante sobre los 14 problemas de categoría continua utilizados en las pruebas experimentales. Todos los conjuntos tienen únicamente atributos continuos.

evaluarlo frente a sistemas de predicción numérica como `CUBIST`, se ha hecho una adaptación de conjuntos de categoría continua, ampliamente conocidos por la comunidad científica, reescribiéndolos como problemas de comparación. En la Tabla 8.1 se muestran las características de los problemas utilizados. Los conjuntos *cpu* y *bodyfat* fueron obtenidos de la página web de `CUBIST` [Quinlan, 2002]; *bupa* y *housing* pueden encontrarse en el almacén de la UCI (Universidad de California en Irvine) [Blake y Merz, 1998]; el resto de conjuntos fueron obtenidos del almacén del LIACC (Laboratorio de Inteligencia Artificial y Ciencias de la Computación de la universidad de Oporto) [Torgo, 2002]. Algunos de estos conjuntos ya fueron utilizados en [Díez et al., 2002b] con el mismo propósito que nos ocupa ahora: comparar la solución propuesta por $\mathcal{LACE}$ con la ofrecida por `CUBIST`.

Para producir conjuntos de comparaciones como requiere $\mathcal{LACE}$, cada ejemplo del conjunto de entrenamiento original se compara con otros diez ejemplos elegidos al azar. Esto da lugar a un conjunto de comparaciones de entrenamiento que será la información que se le suministre a nuestro sistema. `CUBIST` aprenderá, como es lógico, a partir del conjunto de entrenamiento original. En este sentido, `CUBIST` parte con ventaja, ya que al disponer de la valoración numérica exacta, no sólo conoce implícitamente el orden que tienen los ejemplos, sino que además dispone de las diferencias exactas entre todos los objetos del conjunto. A nuestro algoritmo le suministramos únicamente la relación de cada ejemplo respecto a otros diez y no respecto a todo el conjunto.

	$\mathcal{LACE}$			CUBIST		$\mathcal{LACE}_1$
Problema	**% Error**	**Funciones**	**Reglas**	**% Error**	**Reglas**	**% Error**
bank8FM	6,57	1,00	1,00	6,38	35,92	6,59
bodyfat	18,92	2,22	3,64	18,04	1,12	17,76
bupa	41,78	13,24	23,08	39,11	1,38	37,21
cpu	14,61	3,10	7,82	14,27	4,92	14,05
deltaAilerons	18,99	1,00	1,00	100,00∗	1,00	18,97
deltaElevators	20,90	1,00	1,00	22,24	3,78	20,88
diabetes	36,58	4,08	6,14	36,18	1,72	30,59
housing	13,42	2,02	3,94	12,05	8,80	13,12
kin8NM	26,96	50,14	98,84	18,51	50,54	26,76
puma8NH	26,71	1,00	1,00	19,22	27,52	26,69
pyrim	25,81	2,22	4,34	28,53	3,34	21,61
stock	10,96	10,94	33,60	4,52	28,00	12,85
triazines	37,18	5,94	9,16	36,51	3,84	36,56
wpbc	43,96	8,32	15,34	40,31	3,56	36,99
Media*	24,95	8,09	16,07	22,76	13,42	23,20

Tabla 8.2: Resultados obtenidos en los problemas continuos adaptados al aprendizaje por comparaciones en pruebas de validación cruzada. $\mathcal{LACE}_1$ es el algoritmo $\mathcal{LACE}$ forzado a generar una única función. La última fila presenta la media (*) de los resultados excluyendo el problema *deltaAilerons*. Se excluye este problema ya que el resultado que obtiene CUBIST no es útil para comparar: falla el 100 % porque genera una única regla con una función de valoración constante. Puede apreciarse que la diferencia en la media del error entre ambos sistemas no es muy grande. Debe tenerse en cuenta que CUBIST parte con cierta ventaja, ya que al disponer de la valoración numérica exacta, no sólo conoce implícitamente el orden que tienen los ejemplos, sino que además dispone de las diferencias exactas entre todos los objetos del conjunto.

Para medir la eficacia de los dos sistemas se empleará el mismo conjunto de test, construido con el método antes descrito: cada ejemplo se compara con otros diez al azar y que no coinciden en ningún caso con los empleados en el entrenamiento. Como el sistema `CUBIST` está diseñado para predecir categorías continuas, para evaluar su calidad sobre el conjunto de test formado por comparaciones, se calcula la valoración que `CUBIST` otorga a los dos objetos que forman cada par, lo que nos permite determinar si la comparación ha sido acertada o no.

Los experimentos realizados son *validaciones cruzadas*, con 10 particiones y repitiendo cada experimento 5 veces, lo que da lugar a 50 pruebas por experimento. Los resultados de $\mathcal{LACE}$, como puede verse en la Tabla 8.2, son similares a los de `CUBIST`. En la parte derecha de la tabla se muestran los resultados de $\mathcal{LACE}$ forzándole a generar una única función (versión que denominamos $\mathcal{LACE}_1$). La única diferencia significativa se establece entre $\mathcal{LACE}$ y $\mathcal{LACE}_1$, donde la hipótesis de que $\mathcal{LACE}_1$ es mejor que $\mathcal{LACE}$ se acepta con una probabilidad del 98 %. En algunos problemas los resultados mejoran considerablemente al utilizar $\mathcal{LACE}_1$, superando la precisión de `CUBIST`. Aunque durante el aprendizaje nuestro sistema comprueba lo que sucedería si se utilizase una única función, parece claro que la comprobación realizada no está produciendo el resultado deseado. La calidad de ambas soluciones (múltiples funciones *vs.* una función) se comprueba mediante el número de aciertos de cada una de ellas en reescritura. En algunos problemas, cuando se generan varias funciones la solución se ajusta mejor a los datos de partida y el error en reescritura es menor que generando una única función. Esto hace que la solución con varias funciones sea la elegida. En este sentido, podría incorporarse algún mecanismo que evaluase la calidad de ambas soluciones de manera que se tenga en cuenta el posible sobreajuste. Una posibilidad podría ser reservar una parte de los datos de entrenamiento para comprobar la calidad de las soluciones. Sin embargo, dependiendo del número de comparaciones que haya en el problema, el hecho de reservar un porcentaje de ellas, puede entorpecer considerablemente el aprendizaje.

8.3. Problemas artificiales

El siguiente experimento utiliza problemas artificiales creados para ilustrar con claridad cómo el sistema resuelve el tipo de problemas para los que ha sido diseñado. Se han preparado 15 problemas, cada uno de ellos con un conjunto de entrenamiento y otro de test, construidos con semillas de generación de aleatorios diferentes. Tanto el conjunto de entrenamiento como el de test está formado por 3000 comparaciones. En todos los problemas la solución óptima se alcanzaría con dos funciones. Los resultados pueden verse en la Tabla 8.3.

DosFun1 Consiste en aprender a valorar objetos descritos por dos atributos (x e y) que varían en el intervalo $[0, 1000]$. Los conjuntos de comparaciones de entrenamiento

Problema	% Error	$\mathcal{LACE}$ Funciones	Reglas
DosFun1	0,47	2	10
DosFun1S	0,00	2	2
DosFun1Mezcla	2,23	4	19
DosFun1MezclaS	0,30	2	2
DosFun2	1,00	3	18
DosFun2S	0,00	2	2
DosFun2Mezcla	8,00	5	17
DosFun2MezclaS	0,10	2	2
Campana	0,00	2	2
Asimétrico	1,47	2	2
Diagonal1	1,20	3	15
Diagonal2	9,60	7	32
Anillos1	11,55	12	94
Anillos2	1,55	4	40
Anillos3	0,00	2	29

Tabla 8.3: Resultados de $\mathcal{LACE}$ en problemas artificiales. Puede verse como el algoritmo, en muchos casos, es capaz de detectar el número correcto de funciones (2). En muchos problemas el error se mantiene por debajo del 1 %.

y test, en este caso, se han construido usando como función de valoración la siguiente:

$$f(x,y) = \begin{cases} x + 10y & x \leq 500 \\ 10x + y & x > 500 \end{cases} \tag{8.1}$$

Las comparaciones se hacen con objetos que pertenezcan a la misma parte del dominio definido. $\mathcal{LACE}$ localiza perfectamente las dos funciones y obtiene un buen resultado: el error es del 0,47 % y obtiene 2 funciones que se aplican en 10 reglas, que son las siguientes:

$$\begin{aligned}
0{,}000039x + 0{,}000392y + 0{,}000058 &\leftarrow (382{,}42, 913{,}38) \\
\text{"} &\leftarrow (407{,}06, 515{,}26) \\
\text{"} &\leftarrow (377{,}83, 92{,}37) \\
\text{"} &\leftarrow (470{,}49, 694{,}43) \\
\text{"} &\leftarrow (445{,}16, 209{,}62)
\end{aligned}$$

$$\begin{aligned}
0{,}000459x + 0{,}000046y + 0{,}0000 &\leftarrow (578{,}06, 130{,}46) \\
\text{"} &\leftarrow (615{,}90, 911{,}22) \\
\text{"} &\leftarrow (689{,}92, 554{,}82) \\
\text{"} &\leftarrow (706{,}98, 765{,}95) \\
\text{"} &\leftarrow (592{,}80, 389{,}52)
\end{aligned}$$

Como puede apreciarse, las dos funciones aprendidas tienen la misma pendiente que las funciones que definen el problema (8.1). Las condiciones representan un punto en el espacio de ejemplos, así que el ámbito de aplicación de cada una de ellas será la región de Voronoi que le corresponda.

DosFun1S El problema *DosFun1S* es como el anterior, con la diferencia de que el rango del atributo x es $[200, 300) \cup [700, 800)$. La separación de los objetos que se valoran por las dos funciones facilita el aprendizaje de $\mathcal{LACE}$. En este problema el sistema no comete ningún error y genera 2 funciones aplicadas por 2 reglas. En concreto, lo aprendido por $\mathcal{LACE}$ es lo siguiente:

$$\begin{array}{rcl} 0{,}000093x + 0{,}000932y - 0{,}9958 & \leftarrow & (279{,}33, 584{,}36) \\ 0{,}004879x + 0{,}000499y - 0{,}0186 & \leftarrow & (721{,}60, 628{,}49) \end{array}$$

DosFun1Mezcla *DosFun1Mezcla* es exactamente igual que el *DosFun1*, pero permitiendo comparaciones entre objetos de cualquier parte del dominio. La dificultad del problema crece y $\mathcal{LACE}$ empeora ligeramente sus resultados: aumenta el error (2,23 %) al generar más funciones de las necesarias (4, aplicadas por 18 reglas).

DosFun1MezclaS El problema *DosFun1MezclaS* es equivalente a *DosFun1S* permitiendo todo tipo de comparaciones. En este caso el error es muy pequeño (0,30 %) ya que $\mathcal{LACE}$ ha sido capaz de aprender el número correcto de funciones (2, que se aplicarán mediante 2 reglas). Este problema junto con el anterior demuestra que tiene más importancia la distancia entre las dos regiones en el eje x que el permitir comparaciones entre objetos que se evalúan con distintas funciones.

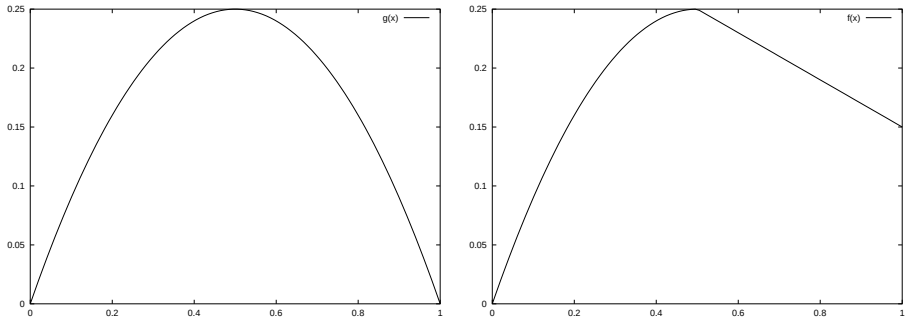
DosFun2x Los problemas *DosFun2*, *DosFun2S*, *DosFun2Mezcla* y *DosFun2MezclaS* tiene las mismas características que los *DosFun1x*, pero en este caso atienden a la función:

$$f(x, y) = \begin{cases} x + y & x \leq 500 \\ x - y & x > 500 \end{cases} \tag{8.2}$$

Este es un problema un poco más complicado, ya que las funciones son planos ortogonales. Se puede ver que el error es, en general, superior al obtenido en los problemas *DosFun1x*.

Campana El problema de la *Campana* es un problema de un atributo que responde a la función $f(x) = -x(x-1)$ en el rango de $x \in [0, 1]$, Figura 8.1(a). Nuestro sistema obtiene el resultado óptimo con sólo dos funciones y un acierto total. La reglas son estas:

$$\begin{array}{rcl} 1{,}0000x + 0{,}0000 & \leftarrow & 0{,}4601 \\ -1{,}0000x + 1{,}0000 & \leftarrow & 0{,}5243 \end{array}$$



(a) (b)

Figura 8.1: A la izquierda la función que define el problema *Campana*; a la derecha el problema *Asimétrico*.

Asimétrico El problema *Asimétrico* (Figura 8.1(b)), diseñado para probar la utilidad del ajuste global de funciones, se define por la expresión:

$$f(x) = \begin{cases} -x(x-1) & x \leq 0{,}5 \\ \frac{-4x+7}{20} & x > 0{,}5 \end{cases} \tag{8.3}$$

Este es un problema complejo, ya que la pendiente de la parte derecha es menos pronunciada que en la izquierda. En este caso no sólo son necesarias dos funciones con distinto signo en su pendiente, sino que además es preciso que la que cubre la parte izquierda tenga una mayor pendiente en valor absoluto. De esta manera, las comparaciones cruzadas entre ambas regiones podrán ser clasificadas correctamente. El resultado obtenido por $\mathcal{LACE}$ es perfecto en el número de funciones y razonablemente aceptable en cuanto al error: 1,47 %. Las reglas que se obtienen se muestran a continuación:

$$\begin{aligned} 1{,}6293x - 0{,}1575 &\leftarrow 0{,}4601 \\ -0{,}9183x + 1{,}0058 &\leftarrow 0{,}5243 \end{aligned}$$

Diagonalx Los problemas *Diagonal1* y *Diagonal2* tiene las mismas funciones que los problemas *DosFun1Mezcla* y *DosFun2Mezcla*. La diferencia está en las condiciones, en este caso la frontera entre las dos funciones viene dada por la diagonal $x = y$. En los resultados se vuelven a apreciar dificultades cuando las funciones son ortogonales.

Anillos En la Figura 8.2, se ve la descripción de los problemas *Anillos*. En estos problemas el dominio se separa en tres zonas. Las zonas interna y externa comparten la misma función. Las líneas en el sentido \ marcan la zona donde se aplica la función $x + y$. Las líneas en el sentido / indican la zona donde se aplica $x - y$. En las tres

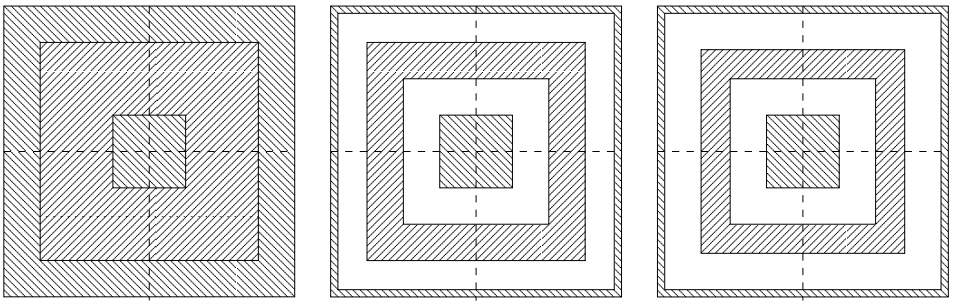


Figura 8.2: Problemas de los Anillos. Las zonas marcadas con \ y / representan el dominio de aplicación de las dos funciones. En las sucesivas versiones del problema se aumenta la separación entre las zonas de aplicación, lo que facilita el aprendizaje.

vesiones del problema las comparaciones son totalmente aleatorias, con lo que se presentan comparaciones en las que los objetos de cada par pueden estar valorados por distintas funciones.

En *Anillos1* no existe ninguna separación entre las diferentes zonas, lo que hace que sea un problema muy complejo. En *Anillos2* no hay ejemplos para valores de x e y comprendidos en $[250, 500]$ ni en $[750, 950]$. En *Anillos3*, las zonas vacías se encuentran en los intervalos $[250, 500]$ y $[700, 950]$.

En la tabla de resultados se ve cómo al aumentar la distancia entre el dominio de las funciones los resultados mejoran, llegando a obtenerse un 100 % de aciertos con 2 funciones en *Anillos3*. Cabe destacar que en este problema $\mathcal{LACE}$ ha asociado las zonas interna y externa asignándolas la misma función de valoración.

8.3.1. Efecto lote

En el Capítulo 1, cuando se citaron los motivos para desarrollar este algoritmo, se comentó la existencia del llamado *efecto lote*:

> ... los expertos calificadores tratan de puntuar los diferentes objetos con un sentido relativo, comparándolos con los restantes del lote. Por eso, un objeto rodeado de otros peores, probablemente tendrá una puntuación más alta que si se presenta rodeado de objetos mejores.

Quiere esto decir que las puntuaciones otorgadas a los objetos de un lote son correctas en sentido relativo. Sin embargo, al contrastar las puntuaciones otorgadas a un mismo objeto en lotes diferentes, podemos encontrarnos con que dicho objeto ha sido calificado con distintas notas.

		$\mathcal{LACE}$		CUBIST	
% Desviación	**Desviación real**	**% Error**	**Reglas**	**% Error**	**Reglas**
0	± 0	0	2	0,48	13
5	± 0,0125	0	2	3,06	11
10	± 0,025	0	2	3,97	9
15	± 0,0375	0	2	7,19	6
20	± 0,05	0	2	12,89	7
25	± 0,0625	0	2	13,57	11

Tabla 8.4: Simulación del efecto lote sobre el problema *Campana*. En cada fila se muestran los resultados correspondientes a un porcentaje de desviación diferente. En la columna *Desviación real* se muestra la desviación correspondiente al porcentaje aplicado. En unos lotes se incrementa la valoración real con la desviación y en otros se decrementa. Se observa que CUBIST, al no estar preparado para ello, sufre en presencia del efecto lote.

Hemos tratado de simular este efecto con el problema de la *Campana*. El conjunto de entrenamiento se formó simulando la existencia de 30 lotes de 100 objetos cada uno. En cada lote la valoración de los ejemplos se desvió en un porcentaje de su verdadera puntuación. En unos lotes la valoración de todos sus ejemplos se incrementó y otros se decrementó, haciéndose la elección de incremento o decremento al azar. Por ejemplo, en el conjunto formado con el 5 % de desviación la valoración de los objetos de unos lotes se incrementó un 0,0125 ($\frac{0,25\times 5}{100}$ ya que la puntuación real de los objetos en la *Campana* varía entre 0 y 0,25) y en otros lotes se decrementó en esa cantidad. En el conjunto de test no se incluyó ninguna desviación.

En la Tabla 8.4 y en la Figura 8.3 pueden verse los resultados obtenidos por $\mathcal{LACE}$ y CUBIST al ir incrementando el porcentaje de desviación de cada lote. $\mathcal{LACE}$ aprende a partir de la posición relativa de los objetos. Esta posición es constante, a pesar del efecto lote, y por tanto no se ve afectado. CUBIST, al considerar las puntuaciones en sentido absoluto, ve decrementada su precisión significativamente a medida que el efecto lote es más acusado. En realidad los resultados de ambos sistemas se adaptan perfectamente para lo que han sido diseñados: CUBIST no está preparado para ese tipo de situaciones, necesita que las puntuaciones absolutas de los expertos sean lo más correctas posible; es decir, necesita expertos muy bien entrenados que no varíen la valoración de un objeto aunque éste les sea presentado en distintas sesiones.

8.3.2. Comportamiento frente al ruido

Se realizó otro grupo de experimentos con el fin de comprobar la estabilidad de $\mathcal{LACE}$ frente al ruido. Se hicieron conjuntos de entrenamiento con distinto porcentaje de ruido. Si tenemos una comparación $u < v$, tal que u es peor que v, la versión ruidosa

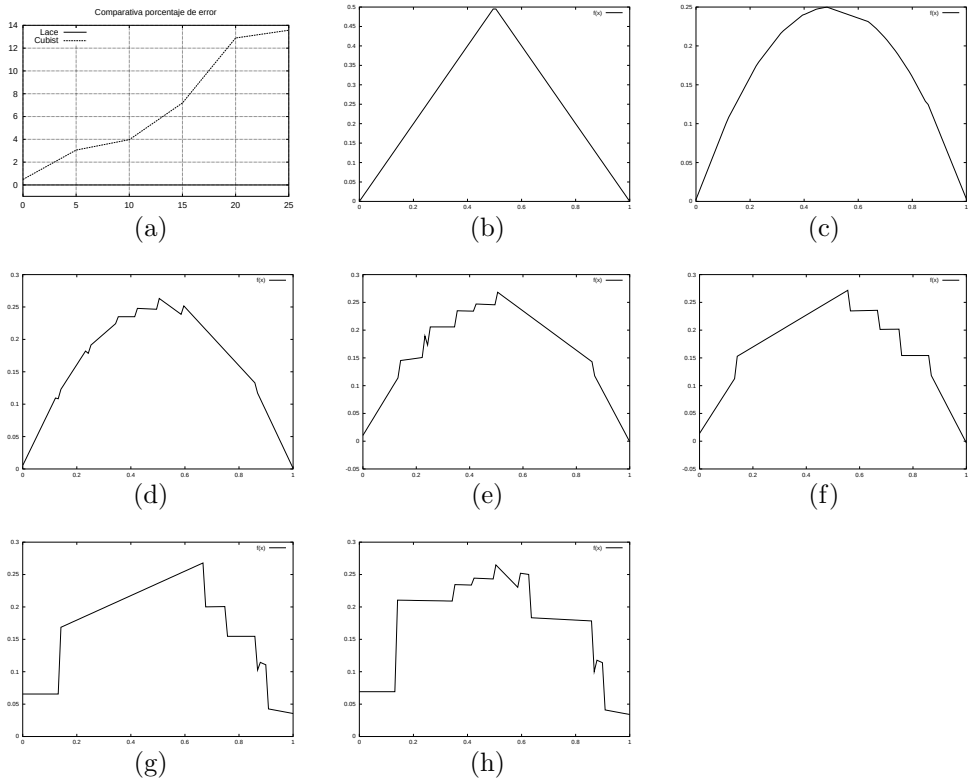


Figura 8.3: En (a) se ve que, a medida que aumenta el *efecto lote*, el porcentaje de error de `CUBIST` se incrementa considerablemente. Los resultados de $\mathcal{LACE}$ no se ven afectados por esta circunstancia, ya que no utiliza las valoraciones otorgadas por los expertos calificadores, sino el orden que proponen dentro de un lote. En (b) se muestra la solución ofrecida por $\mathcal{LACE}$ independientemente del porcentaje de desviación aplicado (ya se ha comentado que nuestro sistema no se ve afectado por este efecto). En (c) puede verse que `CUBIST` es capaz de ajustarse a la forma de la *Campana* casi a la perfección cuando no se aplica ninguna desviación. En el resto de las gráficas (d-h) se pueden apreciar las dificultades de `CUBIST` para aprender cuando el *efecto lote* se hace más acusado en los conjuntos (5 %, 10 %, 15 %, 20 % y 25 %, respectivamente). La función descrita mediante las reglas generadas por `CUBIST` en cada caso va deformándose y perdiendo la apariencia de la *Campana* original a medida que se aumenta el porcentaje de desviación.

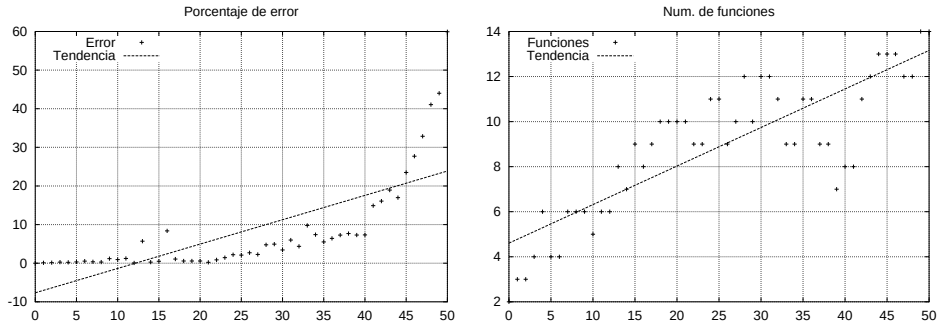
% Ruido	% Error	$\mathcal{LACE}$ Funciones	Reglas
0	0,00	2	2
1	0,10	3	6
2	0,13	3	6
3	0,27	4	6
4	0,23	6	10
5	0,33	4	10
6	0,53	4	7
7	0,37	6	10
8	0,30	6	10
9	1,17	6	10
10	0,93	5	9
15	0,50	9	24
20	0,57	10	19
25	2,10	11	23
30	3,43	12	23
35	5,50	11	26
40	7,33	8	23
45	23,50	13	26
50	59,93	14	28

Tabla 8.5: Evolución de $\mathcal{LACE}$ ante el problema *Campana* al añadir ruido. Se hace patente la robustez del algoritmo en este sentido.

de dicha comparación será $u > v$. El conjunto de test se trata, por supuesto, de un conjunto de comparaciones sin ruido.

En la Tabla 8.5 y en la Figura 8.4, podemos ver el comportamiento de $\mathcal{LACE}$ ante el problema de la *Campana* con distintos porcentajes de ruido. Insertando hasta un 20 % de ruido, $\mathcal{LACE}$ generalmente se encuentra por debajo del 1 % de error. Con el 40 % de ruido, *aún no supera el 10 % de error*. Este buen comportamiento ante el ruido se debe principalmente al mecanismo utilizado para el cálculo de las funciones en su ámbito local. La adaptación del `OC1` empleada, busca un hiperplano que separe la región en la que se encuentran las diferencias normalizadas, de la región en la que no hay. Al ir añadiendo ruido, algunas diferencias irán cambiando de zona, lo que dificulta la búsqueda del hiperplano; sin embargo mientras haya más diferencias normalizadas buenas que ruidosas siempre se obtendrá un hiperplano válido.

En las Figuras 8.5 y 8.6 se ve el comportamiento de $\mathcal{LACE}$ ante los problemas *DosFun1Mezcla* y *DosFun2Mezcla* con ruido. Las líneas de tendencia muestran que el incremento de ruido afecta de forma moderada a la eficacia del sistema. Evidentemente, la tendencia es que al ir añadiendo ruido la precisión empeora, aunque se



(a) (b)

Figura 8.4: Evolución de $\mathcal{LACE}$ ante el problema *Campana* al añadir ruido. En (a) y (b) se muestra la evolución en el porcentaje de error y en el número de funciones a medida que se va incrementando el porcentaje de ruido. Se muestra también, en ambos casos, la línea de tendencia de los resultados obtenidos.

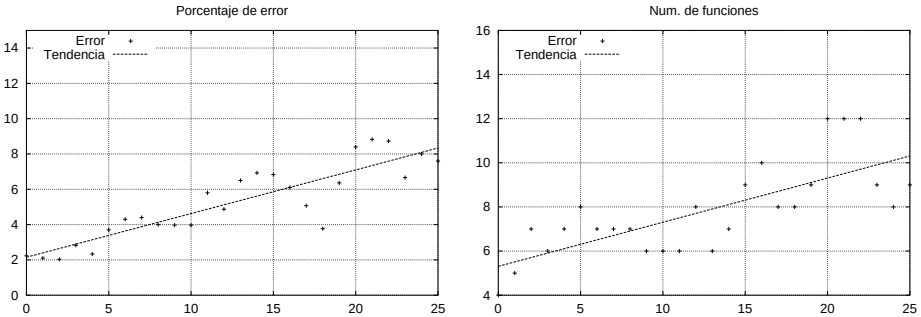
necesitan porcentajes muy altos para desestabilizar al sistema. En cuanto al número de funciones, también se incrementa a medida que se añade ruido, pero nunca llega a límites extremos.

8.3.3. Irrelevantes

Se ha experimentado también el comportamiento del algoritmo ante los atributos irrelevantes. Para ello se han generado problemas a los que se han añadido atributos con valores aleatorios y que no intervienen ni en las condiciones de las regiones de decisión, ni en las propias funciones que definen el problema. En las Tablas 8.6 y 8.7 se muestran los resultados obtenidos ante los problemas de la *Campana* y *DosFun1Mezcla*.

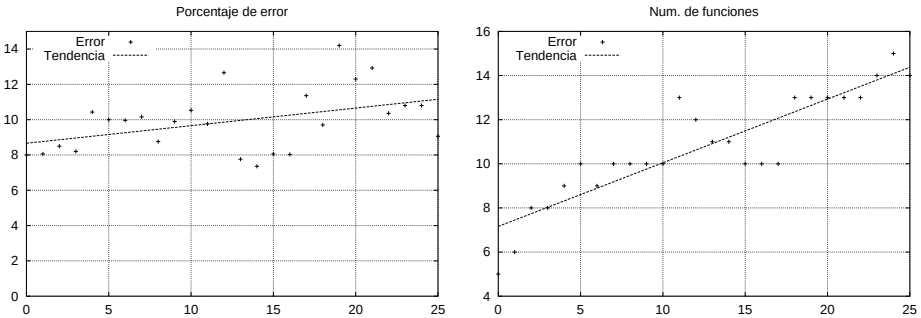
Se puede observar que, mientras el número de atributos relevantes es mayor o igual que el de irrelevantes, el error se mantiene en términos razonables. Sin embargo, al superar este umbral, el error se multiplica. El principal motivo de este comportamiento se debe a que el algoritmo de agrupamiento empleado utiliza la métrica Euclídea, que otorga la misma importancia a todos los atributos. Aunque otras fases importantes, como el cálculo de funciones locales, elimine atributos irrelevantes, éstos siempre permanecerán en las condiciones.

Estos resultados demuestran que para tratar problemas en los que haya un gran número de atributos se necesita incorporar algún mecanismo de cálculo de la relevancia de los mismos. Bien podría tratarse de algún proceso previo o de un mecanismo embebido en el propio sistema.



(a) (b)

Figura 8.5: Evolución frente al ruido de $\mathcal{LACE}$ en el problema *DosFun1Mezcla*. En ambas gráficas los resultados obtenidos se acompañan con la línea de tendencia.



(a) (b)

Figura 8.6: Evolución frente al ruido de $\mathcal{LACE}$ en el problema *DosFun2Mezcla*. Las líneas de tendencia muestran el carácter creciente del error y del número de funciones a medida que se aumenta el ruido.

Número de Irrelevantes	% Error	$\mathcal{LACE}$ Funciones	Reglas
0	0,00	2	2
1	0,87	4	16
2	4,20	5	26
3	15,90	5	29
4	12,00	7	29
5	17,73	9	30

Tabla 8.6: Evolución de $\mathcal{LACE}$ ante el problema de la *Campana* con atributos irrelevantes. En cuanto el número de atributos irrelevantes supera el número de relevantes el error se dispara. Se hace patente la necesidad de incorporar al algoritmo algún mecanismo de relevancia de atributos.

Número de Irrelevantes	% Error	$\mathcal{LACE}$ Funciones	Reglas
0	2,33	4	19
1	8,13	6	28
2	7,67	6	30
3	14,93	7	29
4	18,60	8	29
5	20,37	8	29

Tabla 8.7: Evolución de $\mathcal{LACE}$ ante el problema *DosFun1Mezcla* con atributos irrelevantes. Se puede apreciar que, nuevamente, al superar el número de atributos irrelevantes al de relevantes el error aumenta considerablemente.

Problema	OC1	SVM
$x + y$	0,00	0,43
$x - y$	0,03	0,33
$10x + y$	0,00	0,03
$x + y$		
% ruido		
0	0,00	0,43
5	0,00	0,47
10	0,00	1,23
15	0,00	1,43
20	0,00	1,43
25	0,00	2,80
30	0,00	3,10

Tabla 8.8: OC1 frente a SVM. En la parte superior de la tabla se muestra el porcentaje de error de ambos sistemas cuando pretenden aprender las funciones $x + y$, $x - y$ y $10x + y$. En la parte inferior se ven los resultados obtenidos al incluir ruido en el aprendizaje de la función $x + y$.

8.3.4. OC1 vs. SVM

En el Capítulo 5, vimos que aprender una única función puede traducirse en un problema de separación de clases. La idea es trabajar con la diferencia normalizada entre las características de ambos objetos de cada comparación. Teniendo el conjunto de diferencias normalizadas resultantes, la función que resuelve el problema viene definida por el hiperplano que separa la zona del espacio en la que hay diferencias normalizadas de la zona en la que no las hay. Teniendo en cuenta la traducción del problema a términos de búsqueda de un hiperplano que separa clases, parece inevitable la sugerencia de utilizar las *máquinas de soporte vectorial* como herramienta para la búsqueda del mismo.

En esta sección se mostrarán los resultados obtenidos al evaluar dos técnicas diferentes que se ocupan de buscar hiperplanos: las ya mencionadas máquinas de soporte vectorial y la adaptación del OC1. Para poder evaluar la calidad de ambos sistemas ante el problema que se les proponga, se construyeron conjuntos que pueden ser resueltos por una única función. El conjunto de entrenamiento que se suministró a OC1 estaba formado por las diferencias normalizadas, mientras que a SVM se le dieron las diferencias normalizadas etiquetadas con $(+1)$ y las diferencias normalizadas simétricas con respecto al origen de coordenadas etiquetadas con (-1). También se incluyó ruido, invirtiendo la comparación para OC1 o cambiando la etiqueta para SVM. Los resultados pueden verse en la Tabla 8.8. Se puede observar que, sin añadir ruido, ambos tiene un comportamiento excelente, aunque SVM falla algo más. Al ir incluyendo ruido

en el problema $x + y$, `OC1` se mantiene estable mientras que el error de `SVM` aumenta lentamente al enfrentarse con datos más difícilmente separables. Estos motivos nos llevaron a utilizar en nuestro sistema la adaptación de `OC1` en lugar de `SVM`.

8.4. Aplicación: calidad sensorial de la sidra natural

Tras verificar mediante los problemas artificiales la eficacia del sistema presentado, se hace necesario comprobar su utilidad en problemas de la vida real. Por este motivo hemos enfrentado el sistema $\mathcal{LACE}$ a un problema real: la valoración de la calidad sensorial de la sidra. Agradecemos al Servicio Regional de Investigación y Desarrollo Agroalimentario del Principado de Asturias (SERIDA), especialmente a Juan José Mangas Alonso, la deferencia que ha tenido al proporcionarnos los datos necesarios para este análisis.

El SERIDA junto con la Asociación de Lagareros de Asturias (ALA) ha llevado a cabo el proyecto 1FD97-1229-C02-01 (*Caracterización y tipificación de la sidra natural asturiana*). En su transcurso, se valoraron 91 muestras de sidra natural asturiana con la participación de 14 catadores considerados expertos conocedores del producto. La presentación de las muestras a los catadores se repartió en 18 sesiones diferentes. El número de muestras de cada sesión varió entre tres y siete, habiendo entre tres y diez catadores en cada sesión. Cada catador calificó 11 aspectos (categorías continuas) diferentes de cada una de ellas. En la Tabla 8.9 se describen los 11 aspectos valorados por los catadores, indicando el rango de puntuaciones que puede tomar una muestra en cada una de las categorías. El primer grupo de categorías engloba los aspectos evaluados que definen el comportamiento de cada muestra (apreciado visualmente) en el vaso, es decir, cuando se *escancia* la sidra, o lo que es lo mismo, cuando se vierte la sidra en el vaso. El segundo grupo comprende las sensaciones percibidas por los catadores en los aspectos que se aprecian en la degustación de la sidra. El aspecto que figura en la parte inferior de la tabla (*CalAroma*) refleja la sensación de los catadores ante el olor de la sidra.

Cada muestra de sidra evaluada fue analizada laboratorialmente en el Departamento de Sidras y Derivados del SERIDA, anotándose 64 características físicas y químicas diferentes para cada muestra. El objetivo que se perseguía era encontrar reglas que vinculasen las características (objetivas) físico-químicas con las apreciaciones subjetivas acerca de la calidad sensorial de la sidra [Mangas et al., 1999a] [Mangas et al., 1999b] [Picinelli et al., 2000a] [Picinelli et al., 2000b].

8.4.1. Solución como un problema de regresión

El hecho de que las muestras hayan sido presentadas en sesiones diferentes puede estar dando lugar al ya mencionado *efecto lote*. Por tanto, es posible que las puntuaciones otorgadas por los catadores a las muestras puedan no estar reflejando una valoración

Categorías	Rango de puntuaciones
Vaso	(1-3=Defectos inadmisibles) (4=Límite de aceptabilidad) (5=Correcto) (6=Bien) (7=Notable) (8-9=Excelente)
Espalme	(1=No espalma) (2=Deja cerco) (3=Tarda) (4=Espalme correcto)
Aguante	(1=No) (2=Poco) (3=Si)
Gas	(1=Nada) (2=Poco) (3=Correcto)
Pegue	(1=Espumona) (2=No) (3=Pegue fino y consistente)
CalSabor	(1-3=Defectos inadmisibles) (4=Límite de aceptabilidad) (5=Correcto) (6=Bien) (7=Notable) (8-9=Excelente)
Acidez	(1=Muy débil) (2=Débil) (3=Moderado) (4=Fuerte) (5=Muy fuerte)
Amargor	(1=Muy débil) (2=Débil) (3=Moderado) (4=Fuerte) (5=Muy fuerte)
IntPostG	(1=Muy débil) (2=Débil) (3=Moderado) (4=Fuerte) (5=Muy fuerte)
CalPostG	(1-3=Defectos inadmisibles) (4=Límite de aceptabilidad) (5=Correcto) (6=Bien) (7=Notable) (8-9=Excelente)
CalAroma	(1-3=Defectos inadmisibles) (4=Límite de aceptabilidad) (5=Correcto) (6=Bien) (7=Notable) (8-9=Excelente)

Tabla 8.9: En la tabla pueden verse los 11 aspectos evaluados en cada muestra. Hay tres categorías principales: *Vaso*, *CalSabor* (calidad del sabor) y *CalAroma* (calidad del aroma). *Espalme*, *Aguante*, *Gas* y *Pegue* se consideran subcategorías de *Vaso*. *Acidez*, *Amargor*, *IntPostG* (intensidad posterior del gusto) y *CalPostG* (calidad posterior del gusto) son subcategorías de *CalSabor*. *CalAroma* es una categoría muy relacionada con *CalSabor*. El rango de puntuaciones aplicado en la valoración es bastante diferente dependiendo del aspecto que se esté evaluando; así, hay aspectos valorados de 1 a 9 y otros valorados de 1 a 3.

absoluta de la calidad, sino una valoración relativa respecto a las muestras de cada lote. Este hecho provocaría que la utilización de las calificaciones sensoriales como valores absolutos no fuera útil para establecer relaciones válidas entre evaluaciones sensoriales y laboratoriales.

Para testar esta hipótesis se realizaron análisis estadísticos clásicos sobre el conjunto de datos disponibles. Estos análisis no dieron buenos resultados. La correlación entre las valoraciones de los expertos y las 64 características no superó el 0,6. La utilización de un excesivo número de variables independientes en los análisis estadísticos podría estar introduciendo ruido. Se intentó la reducción del número de atributos a utilizar en una regresión lineal mediante la realización de un análisis de componentes principales, sin embargo la correlación media entre las variables seleccionadas por los diferentes factores no fue sustancialmente superior a la observada anteriormente y el error absoluto medio cometido por la función de regresión seguía siendo excesivamente alto.

Se intentó resolver este problema mediante algoritmos de aprendizaje automático de funciones de regresión y el error absoluto medio alcanzado por `CUBIST` no fue demasiado alto. Sin embargo, prediciendo siempre la media de las valoraciones contenidas en el conjunto de entrenamiento se obtienen resultados similares, e incluso mejores. Los resultados pueden verse en la Tabla 8.10, y son producto de validaciones cruzadas con 10 particiones y 5 repeticiones. Los conjuntos de la izquierda contienen las muestras etiquetadas con la nota media asignada por los catadores. Los conjuntos que se pueden ver a la derecha contienen todas las valoraciones de cada catador para cada muestra. El hecho de que la media obtenga tan buenos resultados indica que la mayoría de las notas propuestas por los catadores se encuentran bastante concentradas en un subrango de valores pequeño. Esta circunstancia puede estar indicando una selección incorrecta de la escala de valores utilizada o que la semántica de las puntuaciones no ha sido asimilada correctamente por el conjunto de los catadores. Como consecuencia, los datos de entrenamiento no son lo suficientemente consistentes como para obtener funciones de regresión adecuadas. Las diferencias a favor o en contra de `CUBIST` son bastante escasas, así que con los datos disponibles el problema no es resoluble con un sistema de regresión no lineal. Viendo los resultados obtenidos tratando de predecir la categoría continua, parece obvio que las puntuaciones otorgadas por los catadores no son puntuaciones absolutas.

8.4.2. Solución como un problema de comparaciones

Como ya se ha comentado, las muestras de sidra fueron presentadas a los catadores en diferentes sesiones, así que podemos interpretar las notas como puntuaciones relativas respecto a las muestras de cada sesión. Por ejemplo, si en una sesión se han presentado 3 muestras, y las puntuaciones para cada muestra han sido 4, 5 y 6, debemos interpretar esas puntuaciones como un orden dentro de la sesión. De esta forma se puede formar un conjunto de entrenamiento de comparaciones. Para llevar esto a cabo

Problema	CUBIST	MEDIA
Acidez	0,3480	0,3420
Aguante	0,4118	0,3372
Amargor	0,3543	0,3209
CalAroma	0,7771	0,6780
CalPostG	0,7696	0,6859
CalSabor	0,7187	0,6708
Espalme	0,9104	0,8646
Gas	0,3296	0,3122
IntPostG	0,2779	0,2414
Pegue	0,4957	0,4521
Vaso	1,1714	1,0937
Media	0,5968	0,5453

Problema	CUBIST	MEDIA
Acidez	0,6464	0,6780
Aguante	0,5182	0,5845
Amargor	0,7356	0,7484
CalAroma	1,0258	1,1227
CalPostG	1,0879	1,1548
CalSabor	1,0167	1,1130
Espalme	0,5261	1,0529
Gas	0,4474	0,5493
IntPostG	0,4592	0,4592
Pegue	0,5158	0,6172
Vaso	1,0770	1,3923
Media	0,7324	0,8611

Tabla 8.10: En la tabla de la izquierda se muestran los resultados obtenidos suministrando como categoría de cada muestra la media de las puntuaciones de todos los catadores. En la derecha se muestran los resultados cuando se le suministran las puntuaciones de cada catador para cada muestra. Los resultados corresponden al error absoluto medio obtenido en validación cruzada; en la primera columna se usa el sistema de aprendizaje de reglas de regresión `CUBIST`; la última columna (etiquetada por `MEDIA`) corresponde a un sistema que incondicionalmente asigna la media de todas las valoraciones a cada muestra. Existe una diferencia muy pequeña entre el error cometido por ambos sistemas. Esto indica que la mayoría de las notas propuestas por los catadores se encuentran bastante concentradas en un pequeño subrango de valores.

	$\mathcal{LACE}$			CUBIST	
Problema	**% Error**	**Funciones**	**Reglas**	**% Error**	**Reglas**
Acidez	12,60	1	1	32,97	1
Aguante	7,14	3	3	15,82	8
Amargor	13,44	1	1	29,67	2
CalAroma	14,59	1	1	28,67	3
CalPostG	7,95	3	3	37,14	1
CalSabor	13,39	3	3	27,03	5
Espalme	9,76	1	1	14,61	10
Gas	11,06	3	3	27,69	4
IntPostG	9,33	2	3	26,92	3
Pegue	7,08	3	3	26,92	1
Vaso	9,13	3	3	32,63	1
Media	10,50	2,18	2,27	27,28	3,55

Tabla 8.11: Resultados tras aplicar reescritura a los conjuntos formados a partir de la media de los catadores. La gran diferencia entre los resultados obtenidos por $\mathcal{LACE}$ y CUBIST viene dada por el escaso número de comparaciones; esta circunstancia provoca un sobreajuste en $\mathcal{LACE}$.

tenemos dos opciones:

- se puede resumir la puntuación de una muestra en una sesión como la media de las puntuaciones que los catadores le han otorgado en esa sesión; o bien
- utilizar individualmente todas las puntuaciones otorgadas por cada catador a cada muestra.

Si utilizamos las medias, el conjunto de comparaciones será más reducido, ya que en cada sesión habrá un único orden. Por contra, considerando el orden propuesto por cada catador en cada sesión habrá un mayor número de comparaciones. Si los calificadores están bien entrenados y sus puntuaciones son coherentes, el conocimiento se verá reforzado y la calidad de la solución será mejor.

En la Tabla 8.11, se muestran los resultados en reescritura obtenidos en los conjuntos formados a partir de las ordenaciones resultantes con las medias de la puntuación de los catadores en cada sesión. $\mathcal{LACE}$ aprende a partir de las comparaciones, mientras que CUBIST aprende a partir de las puntuaciones. El porcentaje de error que se muestra es el error sobre el conjunto de comparaciones, calculándose para CUBIST mediante la valoración que propone para cada muestra de la comparación. Puede verse que $\mathcal{LACE}$ obtiene mejores resultados, tanto en porcentaje de error como en número de reglas. Sin embargo, estos resultados son en reescritura y no representan una medida fiable de lo que sucedería ante conjuntos de comparaciones no vistos. Las

	$\mathcal{LACE}$			CUBIST	
Problema	**% Error**	**Funciones**	**Reglas**	**% Error**	**Reglas**
Acidez	23,68	2,64	2,90	40,22	2,06
Aguante	24,86	2,46	2,56	48,55	3,06
Amargor	24,84	2,86	2,92	47,50	3,72
CalAroma	23,69	2,86	2,98	46,11	2,82
CalPostG	22,07	2,84	3,06	43,69	2,32
CalSabor	23,66	2,80	3,18	43,14	3,98
Espalme	22,89	1,84	1,84	44,57	8,06
Gas	27,57	2,66	2,68	44,02	3,92
IntPostG	26,91	2,50	2,66	45,41	2,92
Pegue	23,13	2,72	2,74	41,82	2,48
Vaso	24,65	2,80	3,04	43,96	2,80
Media	24,36	2,63	2,78	44,45	3,47

Tabla 8.12: Resultados tras aplicar validación cruzada a los conjuntos formados a partir de la media de los catadores. Se observa un incremento notable en el error en comparación con la reescritura. CUBIST casi falla el doble que $\mathcal{LACE}$.

estimaciones mediante validación cruzada pueden verse en la Tabla 8.12, donde puede apreciarse un incremento notable en el porcentaje de error: $\mathcal{LACE}$ se acerca al 25 %, doblando su error en reescritura y CUBIST ronda el 45 %, que supera en un 60 % lo que falla en reescritura.

Este incremento nos está indicando que el conjunto de entrenamiento contiene un número insuficiente de comparaciones. En la elaboración de las funciones locales *es conveniente disponer de un mayor número de comparaciones que de dimensiones.* Si el número de comparaciones es menor que el número de dimensiones, se puede calcular un hiperplano $\mathcal{H}$ que pase por las diferencias normalizadas (resultantes de las comparaciones). Si $\mathcal{H}$ no pasa por el origen de coordenadas, el hiperplano paralelo a $\mathcal{H}$ que pase por el origen de coordenadas será un hiperplano válido para separar la zona del espacio en la que hay diferencias de la zona en la que no hay. Sólo en el caso de existir comparaciones contradictorias, que dan lugar a diferencias normalizadas opuestas, no se podrá encontrar un hiperplano que separe perfectamente; ya que cualquier hiperplano que pase por las dos diferencias opuestas pasará también por el origen de coordenadas. Por tanto, cuando se tienen menos comparaciones que dimensiones, las funciones locales calculadas se ajustan perfectamente a las comparaciones (excepto a las contradictorias). Ésto hace que nuestro sistema, como todos los que aproximan funciones mediante regresión, sea proclive al sobreajuste cuando se dispone de pocos datos.

En el conjunto de problemas de la valoración sensorial de la sidra natural, utilizando las medias de las valoraciones en cada sesión, hay entre 205 y 241 comparaciones.

	$\mathcal{LACE}$			CUBIST	
Problema	**% Error**	**Funciones**	**Reglas**	**% Error**	**Reglas**
Acidez	28,08	1	1	64,06	3
Aguante	25,98	1	1	34,04	11
Amargor	33,90	1	1	49,91	3
CalAroma	33,36	1	1	35,24	7
CalPostG	29,45	1	1	33,80	5
CalSabor	30,85	1	1	40,78	7
Espalme	15,67	1	1	27,69	13
Gas	19,33	4	5	36,43	10
IntPostG	33,16	7	7	100,00∗	1
Pegue	18,25	4	7	29,18	13
Vaso	21,64	7	10	25,43	10
Media*	25,65	2,20	2,90	37,66	8,20

Tabla 8.13: Resultados tras aplicar reescritura a los conjuntos formados a partir de las ordenaciones propuestas por todos los catadores. Comparando estos resultados con los de la Tabla 8.11, se aprecia un notable incremento en el error. En este caso el número de ejemplos es mayor y $\mathcal{LACE}$ no se sobreajusta. (*) No se ha incluido el problema *IntPosG* para el cálculo de la media, ya que CUBIST genera una única función constante que no es útil para comparar objetos.

Para disponer de un mayor número de comparaciones, en lugar de utilizar las medias para establecer un único orden, se crearon conjuntos de entrenamiento en los que se introdujeron las comparaciones resultantes a partir de las ordenaciones de todos los catadores. Como puede verse en las Tablas 8.13 y 8.14, la diferencia de error entre reescritura y validación cruzada ha disminuido, lo que corrobora el razonamiento anteriormente expuesto. En este caso el número de comparaciones de los conjuntos varía entre 808 y 1144, excepto el problema *Gas* que tiene solamente 507 comparaciones y tiene una gran diferencia entre reescritura y validación cruzada en cuanto al porcentaje de error.

Si comparamos los resultados de las Tablas 8.13 y 8.14 vemos que al trabajar con las ordenaciones propuestas por todos los catadores el error es mucho mayor que si se trabaja con la ordenación resultante de las medias.

Cuando trabajamos con las ordenaciones que resultan de la media de los catadores, hay una única ordenación dentro de cada sesión, con lo que no existen incoherencias del tipo $Sidra_1 > Sidra_2$ y $Sidra_2 > Sidra_1$, salvo que una misma muestra sea presentada en varias sesiones. Al trabajar con las ordenaciones de todos los catadores, si no hubiese comparaciones contradictorias, los resultados podrían haber mejorado, sin embargo, el orden propuesto para las muestras de una sesión no suele ser coincidente, dándose en bastantes casos, que *el orden propuesto por dos catadores es totalmente*

		$\mathcal{LACE}$		CUBIST	
Problema	**% Error**	**Funciones**	**Reglas**	**% Error**	**Reglas**
Acidez	32,52	2,84	4,82	58,32	4,32
Aguante	29,56	3,74	7,22	41,27	7,46
Amargor	43,04	4,86	7,94	51,71	2,94
CalAroma	35,42	4,96	8,40	39,88	6,68
CalPostG	33,87	5,48	10,14	48,97	4,70
CalSabor	32,78	5,48	10,08	42,81	6,62
Espalme	17,47	2,98	5,82	29,20	12,66
Gas	28,88	4,58	8,06	43,15	8,94
IntPostG	41,80	5,84	9,56	100,00∗	1,00
Pegue	24,12	4,92	9,44	33,93	11,44
Vaso	27,34	4,84	9,42	30,16	8,88
Media*	30,50	4,47	8,13	41,94	7,46

Tabla 8.14: Resultados tras aplicar validación cruzada a los conjuntos formados a partir de las ordenaciones propuestas por todos los catadores. El porcentaje de error ha aumentado respecto a la reescritura. CUBIST se acerca bastante al 50 % de error. (*) Tal como ocurría en reescritura, no se ha incluido el problema *IntPosG* para el cálculo de la media, ya que CUBIST genera una única función constante.

opuesto. Teniendo esto en cuenta, al utilizar la ordenación que resulta de la media, lo que realmente se está haciendo es *limar discrepancias* entre los catadores, ya que parece que hay una gran diferencia de criterio entre ellos.

Observando los resultados de CUBIST, se reafirma lo comentado. Su error ha aumentado mucho en algunos problemas llegando incluso a fallar un 100 % tanto en reescritura como en validación cruzada en el problema *IntPostG*. Analizando la salida ofrecida por CUBIST para este problema, observamos que éste genera una única regla con una *valoración constante*. Evidentemente, una regla de este estilo le hace fallar todas las comparaciones, ya que todas las muestras obtendrán la misma valoración y por lo tanto ninguna será mejor que otra.

La conclusión es que al tratar todas las ordenaciones de los catadores sin hacer la media, estamos incluyendo ruido en el problema. Para comprobar esta hipótesis, decidimos estudiar el comportamiento de cada catador por separado en tres categorías: *Vaso*, *CalSabor* y *Acidez*. Las dos primeras por ser las dos categorías principales y la última porque los resultados obtenidos con ella se repiten con frecuencia en otras. Se formó entonces un conjunto de entrenamiento por cada catador, de manera que resultaron 14 conjuntos de entrenamiento por cada categoría. En la Tabla 8.15 pueden verse los resultados obtenidos en reescritura por $\mathcal{LACE}$ frente a estos conjuntos. En general, los errores son bastante pequeños, si bien hay que tener en cuenta que, cuanto menor sea el número de ejemplos, más posibilidades hay de encontrar un hiperplano

	Acidez				Vaso				CalSabor			
Catador	**% E**	**F**	**C**	**M**	**% E**	**F**	**C**	**M**	**% E**	**F**	**C**	**M**
1	0,00	1	49	54	0,88	1	114	66	3,54	1	113	65
2	5,04	2	119	65	4,62	2	130	64	9,17	2	120	65
3	0,00	1	32	34	0,00	1	45	34	0,00	1	46	34
4	0,00	1	5	8	0,00	1	20	16	0,00	1	18	16
5	1,33	1	75	45	1,28	1	78	43	0,00	1	76	45
6	1,66	1	123	78	2,06	1	97	59	2,99	2	134	73
7	0,00	1	59	41	0,00	1	54	37	0,00	1	65	39
8	0,00	1	68	56	2,04	1	98	61	5,26	1	76	59
9	3,39	1	59	40	0,00	1	41	30	0,00	1	51	34
10	0,00	1	73	55	4,72	1	106	54	4,55	1	110	55
11	0,00	1	27	20	0,00	1	30	18	0,00	1	33	11
12	0,00	1	71	56	1,08	1	93	52	4,00	1	100	56
13	0,00	1	55	46	7,29	1	96	51	1,45	1	69	50
14	3,28	1	61	53	9,63	2	135	70	8,27	1	133	69

Tabla 8.15: Resultados en reescritura obtenidos para cada catador por separado. Se muestra el porcentaje de error en reescritura (en general bajo), el número de funciones generado (F), el número de comparaciones contenidas en el conjunto de entrenamiento (C) y el número de muestras que ha calificado cada catador (M).

que defina una función de valoración válida.

Si analizamos los resultados de *Acidez*, podemos observar que para nueve de los catadores se encuentra una función que clasifica correctamente todos lo ejemplos; en cuatro de ellos se falla menos de un 4 % con una función, y para el catador número dos se falla un 5 % siendo necesarias dos funciones. Obsérvese que juntando todas las comparaciones de los catadores se falla en reescritura un 28,08 % (ver Tabla 8.13). Si analizamos las otras dos categorías, veremos comportamientos similares. El sistema encuentra dificultades para aprender las funciones de valoración que utilizan los catadores 2 y 14; pero en general $\mathcal{LACE}$ es capaz de encontrar una función de valoración que se ajusta a los criterios de cada catador.

8.4.3. Mapas de catadores

Si se pueden encontrar buenas funciones de valoración que se ajusten a las ordenaciones hechas por cada catador por separado, entonces ¿por qué al juntar todas las comparaciones se incrementa tanto el error?. La respuesta a esta pregunta puede quedar resuelta viendo los mapas auto-organizados de Kohonen [Kohonen, 1995] que se muestran en esta sección.

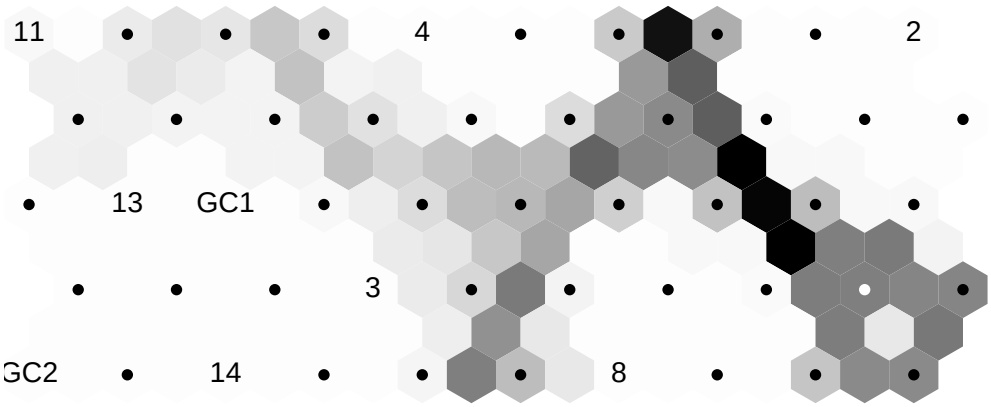


Figura 8.7: Mapa auto-organizado que refleja la similitud entre las funciones aprendidas por $\mathcal{LACE}$ de cada catador en la categoría *Acidez*. Hay dos nodos cuya etiqueta no representa a un solo catador, sino a un grupo: GC1 engloba a los catadores 5 y 12; GC2 incluye a los expertos 1, 6, 7, 9 y 10. Se observan varias regiones en el mapa, que indican el grado de distanciamiento entre las funciones aprendidas de los catadores.

Se tomaron los coeficientes de las funciones que se obtuvieron para cada catador en el problema *Acidez* y se introdujeron en un conjunto de entrenamiento etiquetando cada función con el número del catador al que pertenece. Con este conjunto se entrenó un mapa auto-organizado de Kohonen. El objetivo que se persigue es ver lo parecidas que son las funciones generadas para cada catador de forma que podamos observar si existe mucha diferencia entre la manera de puntuar de cada uno de ellos.

En la Figura 8.7 puede verse el mapa resultante al comprobar la similitud de las funciones aprendidas en el problema *Acidez*. Podemos ver que el catador 2 se encuentra muy distante del resto. Lo mismo sucede con el 8 y el 4, aunque la información de este último no es muy significativa, ya que de él únicamente se tienen cinco comparaciones. El catador 11 también se encuentra algo desplazado. El resto de los catadores se encuentra en una zona común. Así, parece que los catadores 2, 4, 8 y 11 deberían ser descartados, puesto que las funciones de valoración que utilizan son bastante distintas de las empleadas por los demás.

La Figura 8.8 se corresponde con el mapa auto-organizado de las funciones obtenidas para la categoría *Vaso*. Vemos que las funciones aprendidas para los catadores 10 y 14 son totalmente diferentes a las de los demás. En la parte central hay un grupo de catadores muy próximos entre sí; éstos son el 2, 3, 4, 5, 6, 7 y 13. Los demás se encuentran bastante distanciados. El catador número 2 tiene la particularidad de que $\mathcal{LACE}$ ha necesitado dos funciones para explicar su criterio de valoración; una de las funciones se encuentra en la zona central del mapa y la otra más distanciada. Quizás,

Figura 8.8: Mapa auto-organizado que refleja la similitud entre las funciones aprendidas por $\mathcal{LACE}$ de cada catador en la categoría *Vaso*. El nodo etiquetado como GC1 engloba a los catadores 2, 3, 5 y 13; el nodo GC2 comprende a los expertos 8 y 12. Se observa en el mapa a dos catadores muy distanciados del resto, sin embargo, en el centro hay varios bastante próximos.

el mejor grupo de expertos para evaluar la calidad del vaso sea el que aparece en la zona central (excluyendo al 2 por tener una función algo distanciada) ya que, al tener funciones de valoración parecidas, asumimos que siguen un mismo criterio de valoración.

Queda por ver el mapa correspondiente a *CalSabor*, que se muestra en la Figura 8.9. En este caso tenemos al catador número 6 totalmente distanciado del resto y al 10 ligeramente desplazado. El resto se sitúan en una misma zona. En este caso parece que se deben descartar los catadores 6 y 10.

Este análisis de las funciones, junto con las observaciones hechas en los apartados anteriores, nos lleva a formular algunas conclusiones provisionales. Por una parte, cada catador es coherente con su propio criterio de valoración, puesto que sin mezclarlo con otros normalmente se encuentra una única función con un alto porcentaje de aciertos. Sin embargo, no existe un criterio común para todos ellos. Los mapas de Kohonen nos han sugerido algunos descartes para cada categoría; ningún catador sería descartado en las tres categorías consideradas. Esto parece indicar que cada catador puede tener un criterio razonablemente similar al resto cuando califica alguna categoría, pero muy diferente en otras. En otras palabras, la eficacia de los catadores depende de la categoría a juzgar.

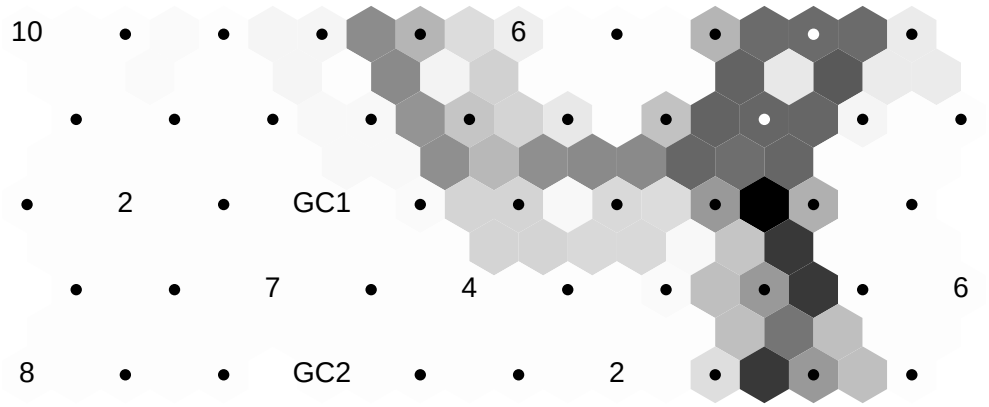


Figura 8.9: Mapa auto-organizado que refleja la similitud entre las funciones aprendidas por $\mathcal{LACE}$ de cada catador en la categoría *CalSabor*. Los catadores 11 y 13 se encuentran etiquetados como GC1. El grupo GC2 está formado por los expertos 1, 3, 5, 9, 12 y 14. Se observa que el 6 está totalmente distanciado de los demás. El catador 10 se encuentra, también, ligeramente apartado.

8.4.4. Agrupación de catadores

Veamos ahora qué sucede si vamos agrupando catadores cuyas funciones de valoración resultan similares según el mapa de Kohonen. Empezaremos por el problema *Acidez* (ver Figura 8.7). Nos situamos en la zona del mapa en la que hay catadores similares; así unimos las comparaciones resultantes de los catadores 5 y 12. En la Tabla 8.16, puede verse que resulta un conjunto con 146 comparaciones (correspondientes a 76 muestras) y $\mathcal{LACE}$ aprende cometiendo un error del 4,11 %. Se van uniendo por proximidad el resto de los catadores de la zona y puede verse como, a medida que aumenta el número de comparaciones, aumenta también el error. Cuando el número de comparaciones aumenta proporcionalmente al número de muestras el incremento de error no es tan notable. Podríamos considerar la figura del *catador perfecto* como aquel que, asistiendo a todas las sesiones y calificando todas las muestras, genera un total de 242 comparaciones coherentes entre sí. Si a este conjunto de comparaciones le seguimos añadiendo comparaciones entre las mismas muestras, éstas pueden ser de dos tipos:

- coherentes con las que ya se tenían, lo que refuerza el conocimiento; o bien,
- contrarias a lo ya dicho, lo que supondría estar añadiendo ruido al problema.

De esto se deduce que existirá un umbral en el número de comparaciones a partir del cual existe un claro riesgo de introducir ruido en los datos de entrenamiento.

Catadores	Muestras	Comparaciones	% Error
$\{5, 12\}$	76	146	4,11
$\{5, 12, 3\}$	82	178	4,49
$\{5, 12, 3, 13\}$	84	233	6,44
$\{5, 12, 3, 13, 14\}$	89	294	10,20
$\{5, 12, 3, 13, 14, 1\}$	91	343	17,20
$\{5, 12, 3, 13, 14, 1, 6\}$	91	466	19,53
$\{5, 12, 3, 13, 14, 1, 6, 7\}$	91	525	21,71
$\{5, 12, 3, 13, 14, 1, 6, 7, 9\}$	91	584	22,60
$\{5, 12, 3, 13, 14, 1, 6, 7, 9, 10\}$	91	657	25,88

Tabla 8.16: Resultados obtenidos en reescritura para el problema *Acidez* al ir uniendo las comparaciones de catadores próximos en el mapa. Al introducir las comparaciones de nuevos catadores aumenta significativamente el error. Con 89 muestras se tienen 294 comparaciones mientras que con 91 muestras (en el último caso) se tienen 657 comparaciones. La mayoría de las 363 comparaciones añadidas ($657 - 294$) son *contradictorias* respecto a lo ya dicho.

Este umbral dependerá del número de comparaciones que se hayan obtenido de cada catador y de su propia coherencia.

En el caso de *Acidez*, podemos ver en la Tabla 8.15 que el catador número 6 es el que dio lugar a más comparaciones (123) calificando 78 muestras, con lo que, en principio, las contradicciones pueden llegar con un número pequeño de comparaciones. En la Tabla 8.16 se puede ver que al introducir nuevos catadores, es decir, añadir sus comparaciones, aumenta el error. Así pues, se confirma nuevamente la diferencia de criterio de cada experto respecto a la valoración de la *Acidez*. Las mismas conclusiones pueden sacarse observando las Tablas 8.17 y 8.18, que son las correspondientes a los

Catadores	Muestras	Comparaciones	% Error
$\{7, 6\}$	76	151	4,64
$\{7, 6, 4\}$	85	171	4,68
$\{7, 6, 4, 13\}$	88	267	10,86
$\{7, 6, 4, 13, 3\}$	88	312	13,43
$\{7, 6, 4, 13, 3, 5\}$	90	390	22,05

Tabla 8.17: Resultados obtenidos en reescritura para el problema *Vaso* al añadir las comparaciones de catadores según sus criterios de valoración. El error se dispara a partir del cuarto catador añadido. Como en *Acidez*, con 171 comparaciones ya se están considerando 85 muestras. El incremento de comparaciones añade ruido al conjunto.

Catadores	Muestras	Comparaciones	% Error
$\{4, 11\}$	35	51	0,00
$\{4, 11, 13\}$	67	120	8,33
$\{4, 11, 13, 2\}$	82	240	12,50
$\{4, 11, 13, 2, 7\}$	82	305	17,38
$\{4, 11, 13, 2, 7, 8\}$	82	381	19,42
$\{4, 11, 13, 2, 7, 8, 1\}$	91	494	20,04
$\{4, 11, 13, 2, 7, 8, 1, 14\}$	91	627	21,05

Tabla 8.18: Resultados obtenidos en reescritura para el problema *CalSabor* al ir uniendo las comparaciones de catadores próximos en el mapa. Se aprecia un aumento notable en el error a medida que aumenta el número de comparaciones utilizado. Se da la misma circunstancia que en los problemas anteriores. En este caso vemos que con 240 comparaciones se están considerando 82 muestras. Añadiendo dos catadores, se están considerando 141 comparaciones nuevas sin que se aumente el número de muestras considerado. Se aprecia un notable incremento en el error, así que las comparaciones añadidas no parece que sean coherentes con las que ya se tenían.

problemas *Vaso* y *CalSabor* respectivamente.

En las Tablas 8.16, 8.17 y 8.18, correspondientes a la evolución del error a medida que se añaden comparaciones de catadores para cada categoría analizada, se puede apreciar que a partir de un cierto número de comparaciones el error aumenta significativamente. Por este motivo, decidimos seleccionar un grupo de catadores para cada categoría. El criterio seguido fue formar un conjunto cuyo error en reescritura esté próximo al que se tenía empleando la media de valoraciones de todos ellos. Así, los catadores seleccionados para cada problema fueron:

- En *Acidez* nos quedamos con los catadores $\{3, 5, 12, 13, 14\}$. Éstos suman 294 comparaciones considerando 89 muestras con un 10,20 % de error en reescritura. Este error es inferior al obtenido con la media de los catadores (12,60 %) y se encuentra muy distanciado del obtenido con las comparaciones de todos ellos (28,08 %).

- En el problema *Vaso*, se seleccionaron los catadores $\{4, 6, 7, 13\}$, que dan lugar a 267 comparaciones considerando 88 muestras con un error del 10,86 %, ligeramente superior al 9,13 % obtenido con la media de los catadores y bastante inferior al obtenido con el conjunto de todos ellos, 21,64 %.

- En *CalSabor* el conjunto de catadores seleccionado es $\{2, 4, 11, 13\}$, que dan lugar a 240 comparaciones considerando 82 muestras con un error del 12,50 %, inferior al obtenido con la media de los catadores (13,39 %) y con el conjunto de expertos al completo (30,85 %).

Problema	% Error	Funciones	Reglas	Media	Todos
Acidez	23,18	2,82	3,62	23,68	32,52
Vaso	23,48	2,96	3,62	24,85	27,34
CalSabor	32,08	2,60	2,84	23,66	32,78

Tabla 8.19: Resultados obtenidos en validación cruzada tras seleccionar un grupo de catadores con funciones de valoración similares. En la parte derecha de la tabla se muestra un resumen de los resultados que se tenían para estos problemas en validación cruzada (Tablas 8.12 y 8.14). La columna *Media* muestra los resultados obtenidos al utilizar como entrenamiento la media de todos los catadores. Bajo *Todos* se ven los resultados que se tienen cuando se utilizan las ordenaciones propuestas por todos los expertos. Al seleccionar un conjunto de catadores el error baja considerablemente en los dos primeros casos.

Problema	% Error	Funciones	Reglas
Acidez	23,13	3,01	4,43
Vaso	23,60	3,21	4,52
CalSabor	30,42	2,78	3,17

Tabla 8.20: Resultados obtenidos aplicando un *leave one out* a los mismos conjuntos de la Tabla 8.19. Los resultados se mantienen, mejorándose ligeramente en *CalSabor*.

Si observamos los resultados obtenidos en validación cruzada por el conjunto de catadores propuesto para cada problema (Tabla 8.19), vemos que en *Acidez* se falla un 23,18 %, bastante inferior al 32,52 % que se tenía con todos los catadores (véase Tabla 8.14). Fijándonos en *Vaso*, observamos que el error baja a un 23,48 %, cuando con todos los catadores se tenía un 27,34 %. Se observan más dificultades en el problema *CalSabor*, donde se pasa de un 32,78 % con todos los catadores a un 32,08 % con los seleccionados. Estos resultados se repiten efectuando una estimación del error mediante un *leaving one out* con cada uno de los conjuntos (Tabla 8.20). Vemos que en los problemas *Acidez* y *Vaso* los resultados son muy similares, mientras que en *CalSabor* el porcentaje de fallo baja a un 30,42 %.

8.4.5. Conclusiones

Tratando el problema de la calidad sensorial de la sidra natural como un problema de regresión, nos encontramos con que los resultados obtenidos por `CUBIST` y `MEDIA` no son nada explicativos. La razón de esto es que las puntuaciones otorgadas por los expertos se concentran en un rango muy reducido de valores y no cabe interpretarlas

con un carácter absoluto, como hacen los métodos de regresión lineales o no lineales.

El hecho de que las muestras fuesen presentadas en diferentes sesiones ha podido provocar el denominado *efecto lote*. Esto nos lleva a sugerir la lectura de las puntuaciones de los expertos como indicaciones relativas de la calidad sensorial de las muestras de sidra de cada sesión. Para justificar experimentalmente esta hipótesis utilizamos el algoritmo $\mathcal{LACE}$. Para ello se transforman en conjuntos de comparaciones de preferencias las tablas que vinculan las descripciones analíticas de las muestras con las calificaciones de cada categoría sensorial. Abordamos esta transformación de dos maneras:

- Considerando la puntuación de una muestra en una sesión como la media de las puntuaciones que los catadores le han otorgado en esa sesión. En este caso los resultados obtenidos fueron bastante buenos: en reescritura se acierta un 89,5 % de las comparaciones con solamente 2,18 funciones; mientras que en validación cruzada se acierta un 75,64 % con 2,63 funciones. En este planteamiento, `CUBIST` tiene serias dificultades para aprender reglas de regresión que estén de acuerdo con las comparaciones utilizadas como test.

- Considerando individualmente todas las puntuaciones otorgadas por cada catador a cada muestra se obtienen peores resultados: en reescritura se acierta un 74,35 % con 2,2 funciones; mientras que en validación cruzada el aciertos baja hasta un 69,5 % con 4,47 funciones. La diversidad de criterios y el efecto lote dañan aún más el aprendizaje de `CUBIST`.

Los resultados obtenidos considerando individualmente las valoraciones o preferencias de los catadores reflejan la presencia de ruido. Es decir, criterios de preferencia opuestos entre los distintos catadores e incluso entre lo manifestado por un catador en distintas sesiones. Un método efectivo para localizar estas discrepancias podría servir como herramienta para la selección y entrenamiento de los expertos. El uso de estas técnicas potenciaría enormemente los estudios sobre la calidad sensorial o sobre cualquier otra forma de relacionar preferencias subjetivas con propiedades medibles de un tipo de objeto.

$\mathcal{LACE}$ resume los criterios de preferencia mediante vectores: los coeficientes de las funciones lineales que induce. Por tanto, pueden cuantificarse las diferencias entre los criterios de valoración de un grupo de expertos. Para ilustrar esta aplicación de nuestro sistema, hemos utilizado mapas auto-organizados de Kohonen que representan visualmente estas diferencias. En el caso de la calidad sensorial de la sidra natural, utilizando este planteamiento, pueden formarse grupos de catadores coherentes entre sí; su criterio de preferencia común puede obtenerse utilizando de nuevo $\mathcal{LACE}$ y los porcentajes de acierto son similares a los alcanzados utilizando las valoraciones medias de todos los catadores de cada sesión. Esto respalda las posibilidades de $\mathcal{LACE}$ como herramienta para la selección y entrenamiento de los catadores.

Capítulo 9

Conclusiones

En esta memoria se presenta un nuevo sistema de aprendizaje, que llamamos $\mathcal{LACE}$, un acrónimo de $\mathcal{L}$earning to $\mathcal{A}$ssess by $\mathcal{C}$omparisons $\mathcal{E}$xamples. A partir de un conjunto de pares de objetos ordenados por algún criterio de preferencia, el sistema obtiene una función lineal a trozos que asigna valores más altos a los mejores objetos. Aunque el resultado final de $\mathcal{LACE}$ es una función, no se trata de un sistema de regresión, ya que no parte de un conjunto de entradas y sus correspondientes salidas en la función objetivo. En nuestro caso no se conoce la clase de los ejemplos de entrenamiento; en este sentido $\mathcal{LACE}$ es un algoritmo de aprendizaje sólo parcialmente supervisado: únicamente se sabe que algunos objetos han de tener mejores valoraciones que otros, pero sin puntos de referencia.

La principal motivación que nos ha llevado a desarrollar este sistema ha sido nuestra experiencia en algunos problemas reales en los que se debía aprender a valorar objetos a partir de las calificaciones otorgadas por un grupo de expertos. Si estas calificaciones se consideran en sentido absoluto, los sistemas de regresión resultan muy poco explicativos sobre el criterio de los expertos. La razón es que los expertos, en lugar de calificar los objetos, manifiestan numéricamente sus preferencias relativas al lote de productos que están observando, lo que llamamos el *efecto lote*. En las sesiones de calificación los expertos tienden a valorar los objetos respecto a los demás del lote, y no en un sentido absoluto como se espera cuando se asigna una calificación numérica. Esta situación es particularmente habitual en los objetos de tipo biológico y especialmente en la industria alimenticia, en la cual las reglas para decidir el grado de la calidad de un producto normalmente no están bien definidas, pero el orden de los productos, en cuanto a calidad, es bastante constante y aceptado por parte de los consumidores y expertos.

El algoritmo propuesto parte de una interpretación geométrica para plantear la búsqueda de una función de valoración como una función de separación. Cada par de

objetos comparados da lugar a dos diferencias opuestas; una función capaz de separar estas dos colecciones de vectores puede interpretarse como una solución, es decir, una función de valoración coherente con las preferencias indicadas en las comparaciones. Estas funciones de separación, localmente, pueden ser lineales con lo que se pueden aplicar algoritmos ya utilizados en aprendizaje automático. Lo que debe resolver entonces $\mathcal{LACE}$ es cómo encontrar las regiones donde las soluciones parciales pueden ser lineales, y cómo combinar estas aproximaciones locales para formar una función global que sea lo más coherente posible con el conjunto de preferencias. Siempre que fue posible se adaptaron soluciones ya ensayadas en otras herramientas de aprendizaje automático. Así, se utilizaron:

- el algoritmo de agrupamiento *Growing Neural Gas* (`GNG`) [Fritzke, 1995] como herramienta para la búsqueda de regiones con soluciones parciales,
- el mecanismo de generalización de instancias utilizado en el sistema de aprendizaje de categorías simbólicas `RISE` (*Rule Induction from a Set of Exemplars*) [Domingos, 1996] para agrupar regiones que pudiesen compartir la misma función de valoración,
- la filosofía aplicada en los sistemas de la familia `DROP` (*Decremental Reduction Optimization Procedure*) [Wilson y Martinez, 2000] para la reducción del número de prototipos necesarios para delimitar el ámbito de aplicación de cada función, y finalmente
- el mecanismo de búsqueda de hiperplanos presentado en el sistema `CART` (*Classification and Regression Trees*) [Breiman et al., 1984] y ya utilizado en el sistema `OC1` (*Oblique Classifier 1*) [Murthy et al., 1994] que empleamos para la búsqueda de funciones de valoración locales en cada región.

El algoritmo final tiene una complejidad temporal aceptable, aún en el peor de los casos, del orden de $\mathcal{O}(r^2 \cdot a \cdot n \log n)$, donde n es el número de comparaciones, a el número de atributos que describen cada objeto y r es el número máximo de regiones con posibles soluciones locales, que nunca supera las 100.

El algoritmo construido se ha probado con problemas artificiales y reales para mostrar las habilidades del método propuesto. Los resultados reflejan un grado de precisión muy satisfactorio. En los problemas con ruido el algoritmo tiene un comportamiento excelente, mostrando más dificultades en conjuntos con atributos irrelevantes cuando éstos superan en número a los relevantes.

Para poder establecer algún tipo de comparación de los resultados de $\mathcal{LACE}$ con algún otro sistema de aprendizaje hemos utilizado conjuntos de entrenamiento de regresión disponibles en los almacenes más conocidos en la red. Estos conjuntos fueron transformados en pares de preferencias según el valor conocido de su clase continua. De esta forma $\mathcal{LACE}$ fue comparado con `CUBIST`, un sistema comercial de regresión no

lineal de Quinlan [Quinlan, 2002]. Las comparaciones fueron muy satisfactorias para $\mathcal{LACE}$ teniendo en cuenta que no fue diseñado para resolver este tipo de situaciones.

Se testó el funcionamiento del sistema en un problema real: la valoración de la calidad sensorial de la sidra natural. Aprendiendo a partir del orden generado en cada sesión por la media de las valoraciones de los catadores, $\mathcal{LACE}$ obtuvo buenos resultados, mientras que `CUBIST` presentó serias dificultades en su aprendizaje. Aprendiendo a partir de la unión de todos los órdenes de cada sesión para cada catador, los resultados obtenidos por nuestro sistema empeoraron ligeramente. En este caso, la diversidad de criterios y el efecto lote dañaron aún más el aprendizaje de `CUBIST`.

Utilizando los coeficientes de las funciones de valoración aprendidas para cada uno de los expertos calificadores, hemos podido cuantificar las diferencias entre sus criterios de valoración. Con este propósito, hemos utilizado mapas auto-organizados de Kohonen [Kohonen, 1995]. Conociendo estas diferencias, hemos podido formar grupos de catadores coherentes entre sí; aprendiendo nuevamente a partir de sus comparaciones se alcanzan porcentajes de acierto similares a los obtenidos con la media de todos los catadores en cada sesión. Esto indica la posibilidad de utilizar nuestro sistema, $\mathcal{LACE}$, como una herramienta para la selección y entrenamiento de calificadores.

Apéndice A

Complejidad temporal

En este apéndice estudiaremos la complejidad temporal de $\mathcal{LACE}$ y veremos que se trata de un algoritmo eficiente incluso en el peor caso. Consideraremos para este análisis que el número de comparaciones es mayor que el número de atributos del problema ($n > a$). Este algoritmo aplica secuencialmente (ver Algoritmo 3.1) los siguientes pasos:

- ReescribirEjemplos. Se toman las n comparaciones y se calcula la diferencia normalizada de cada una. Esto da una complejidad de $\mathcal{O}(a \cdot n)$.

- Gng. Con cada comparación se adapta la red. La adaptación de la red consiste en adaptar la celda más cercana a cada ejemplo y las celdas vecinas. En el cálculo de la distancia entre una celda y un ejemplo interviene el número de atributos a. Así que la complejidad temporal de esta fase es $\mathcal{O}(a \cdot n)$.

- CalcularFuncionesLocales. Realiza la búsqueda del mejor hiperplano (función de valoración) para cada regla. El método utilizado es el mismo que en `OC1`. Como los autores indican en [Murthy et al., 1994], la complejidad de la búsqueda de un hiperplano es $\mathcal{O}(a \cdot n \log n)$. Como mucho se calculan r hiperplanos (número de reglas máximo), así que la complejidad de esta fase es $\mathcal{O}(r \cdot a \cdot n \log n)$.

- ObtenerRegiones. En cada intento de unir regiones calcula la función local usando la metodología empleada en `OC1`. La complejidad de esta fase es $\mathcal{O}(r^2 \cdot a \cdot n \log n)$, ya que el máximo número de intentos de unión posibles es r^2.

- EliminarReglasIrrelevantes. Elimina las reglas que no ayudan a delimitar fronteras. Comprueba que al quitar una regla, sus ejemplos asociados no pasen a ser clasificados por otra región. Tiene una complejidad $\mathcal{O}(r \cdot a \cdot n)$.

- AjustarFuncionesGlobalmente. Nivela las regiones. En el peor de los casos una región tendrá n comparaciones cruzadas, luego su complejidad es $\mathcal{O}(r \cdot n \log n)$.

- CalcularFunciónÚnica. Busca la mejor función que clasifique todos los ejemplos. Su complejidad temporal es $\mathcal{O}(a \cdot n \log n)$.

La complejidad de $\mathcal{LACE}$ se corresponde con la más alta de los pasos anteriores. Es decir, $\mathcal{LACE}$ tiene la complejidad $\mathcal{O}(r^2 \cdot a \cdot n \log n)$, lo que supone un coste computacional aceptable. El único parámetro de $\mathcal{LACE}$ que afecta a la complejidad y que es ajeno a los datos de entrenamiento es r, el número de reglas que, en el peor de los casos alcanzará el valor máximo de 100.

Bibliografía

[Aha, 1990] Aha, D. W. (1990). *A Study of Instance-based Algorithms for Supervised Learning Tasks: Mathematical, Empirical, and Psychological Evaluations*. PhD thesis, University of California at Irvine.

[Aha et al., 1991] Aha, D. W., Kibler, D. y Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–67.

[Aho et al., 1974] Aho, A., Hopcroft, J. y Ullman, J. (1974). *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co.

[Albertí et al., 2002] Albertí, P., Sañudo, C., Bahamonde, A., Olleta, J. L., Panea, B., Goyache, F., Alonso, J., Díez, J. y Fernández, I. (2002). Spectrophotometric characterisation of colour classification system of beef meat. En *48th International Congress of Meat Science and Technology*, páginas 454–455.

[Bahamonde et al., 2001] Bahamonde, A., Goyache, F., del Coz, J. J., Quevedo, J. R., López, S., Alonso, J., Ranilla, J., Luaces, O. y Díez, J. (2001). La inteligencia artificial en la clasificación de canales. En *I Congreso Nacional de la Carne*, páginas 95–110.

[Blake y Merz, 1998] Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. University of California, Irvine, Dept. of Information and Computer Sciences.

[Bollen et al., 2002] Bollen, A. F., Kusabs, N. J., Holmes, G. y Hall, M. A. (2002). Comparison of consumer and producer perceptions of mushroom quality. En W. J. Florkowski, S. E. P. y Shewfelt, R. L., editores, *Integrated View of Fruit and Vegetable Quality Internatonal Multidisciplinary Conference*, páginas 303–311. Georgia.

[Branting, 1999] Branting, L. K. (1999). Active exploration in instance-based preference modeling. *Lecture Notes in Computer Science*, 1650:29–43.

[Branting y Broos, 1997] Branting, L. K. y Broos, P. S. (1997). Automated acquisition of user preferences. *International Journal of Human-Computer Studies*, 46:55–77.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A. y Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth Publishers.

[Brighton y Mellish, 2002] Brighton, H. y Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(6):153–172.

[Broos y Branting, 1994] Broos, P. y Branting, K. (1994). Compositional instance-based learning. En *National Conference on Artificial Intelligence*, páginas 651–656.

[Cohen et al., 1999] Cohen, W. W., Schapire, R. E. y Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270.

[Cristianini y Shawe-Taylor, 2000] Cristianini, M. y Shawe-Taylor, J. (2000). *An introduction to support vector machine*. Cambridge University Press.

[del Coz, 2000] del Coz, J. J. (2000). *BETS: Sistema de aprendizaje basado en la selección de ejemplos paradigmáticos*. PhD thesis, Departamento de Informática, Universidad de Oviedo en Gijón.

[del Coz y Bahamonde, 1999] del Coz, J. J. y Bahamonde, A. (1999). Mapas autoorgnizados con atributos discretos. En *VIII Conferencia de la Asociación Española para la Inteligencia Artificial - III Jornadas de Transferencia Tecnológica de Inteligencia Artificial*, páginas 117–124.

[del Coz et al., 1999] del Coz, J. J., Luaces, O., Quevedo, J. R., Alonso, J., Ranilla, J. y Bahamonde, A. (1999). Self-organizing cases to find paradigms. *Lecture notes in computer sciences*, 1606:527–536.

[Díez et al., 2001a] Díez, J., Alonso, J., López, S., Bahamonde, A. y Goyache, F. (2001a). Una aplicación informática para la representación informática de la conformación de canales bovinas. En *IX Jornadas de Producción Animal de la AIDA. ITEA (Información Técnica Económica Agraria. Revista de la asociación Interprofesional para el desarrollo agrario)*, páginas 550–552.

[Díez et al., 2002a] Díez, J., Bahamonde, A., Alonso, J., López, S., del Coz, J. J., Quevedo, J. R., Ranilla, J., Luaces, O., Álvarez, I., Royo, L. J. y Goyache, F. (2002a). Artificial intelligence techniques point out differences in classification performance between light and standard bovine carcasses. *Meat Science*, 63(4) en prensa.

[Díez et al., 2002b] Díez, J., del Coz, J. J., Luaces, O., Goyache, F., Peña, A. M. y Bahamonde, A. (2002b). Learning to assess from pair-wise comparisons. *VIII Conferencia Iberoamericana de Inteligencia Artificial (IBERAMIA 2002). Lecture Notes in Computer Science*, 2527:481–490.

[Díez et al., 2002c] Díez, J., Goyache, F., Alonso, J., del Coz, J. J., Quevedo, J. R., López, S., Fernández, I., Luaces, O. y Bahamonde, A. (2002c). Técnicas de inteligencia artificial en la clasificación de canales bovinas. *Nuestra cabaña*, I(316):12–18.

[Díez et al., 2001b] Díez, J., Goyache, F., Alonso, J., del Coz, J. J., Quevedo, J. R., López, S., Ranilla, J., Luaces, O. y Bahamonde, A. (2001b). Utilización de técnicas de inteligencia artificial en la clasificación de canales bovinas. En *IX Conferencia de la Asociación Española para la Inteligencia Artificial - IV Jornadas de Transferencia Tecnológica de Inteligencia Artificial*, páginas 1229–1238.

[Díez et al., 2002d] Díez, J., Luaces, O. y Ranilla, J. (2002d). Aplicación de un proceso de selección de reglas ajeno al nivel de impureza. *Inteligencia Artificial, Revista Iberoamericana de I. A.*, I(15):32–39.

[Domingos, 1994] Domingos, P. (1994). The rise system: Conquering without separating. En *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, páginas 704–707, New Orleans, US. LA: IEEE Computer Society Press.

[Domingos, 1995] Domingos, P. (1995). Rule induction and instance-based learning: A unified approach. En *International Join Conferences on Artificial Intelligences*, páginas 1226–1232.

[Domingos, 1996] Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168.

[Esposito et al., 1997] Esposito, F., Malerba, D. y Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

[Fritzke, 1994a] Fritzke, B. (1994a). Fast learning with incremental rbf networks. *Neural Procesing Letters*, 1(1):2–5.

[Fritzke, 1994b] Fritzke, B. (1994b). Growing cell structures — a self-organizing for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460.

[Fritzke, 1995] Fritzke, B. (1995). A growing neural gas network learns topologies. En Tesauro, G., Touretzky, D. S. y Leen, T. K., editores, *Advances in Neural Information Processing Systems 7*, páginas 625–632. MIT Press, Cambridge MA.

[Fritzke, 1997] Fritzke, B. (1997). Some competitive learning methods. Technical report, Institute for Neural Computation, Ruhr-Iniversität Bochum.

[Fritzke, 2002] Fritzke, B. (2002). DemoGNG Release 1.5, [http://www.neuroinformatik.ruhr-uni-bochum.de/ini/vdm/research/gsn/ /demogng/gng.html].

[Fürnkranz, 1997] Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–171.

[Fürnkranz, 2002] Fürnkranz, J. (2002). Round robin rule learning. *Journal of Machine Learning Research*, 2:721–747.

[Goyache et al., 2001a] Goyache, F., Bahamonde, A., Alonso, J., López, S., del Coz, J. J., Quevedo, J. R., Ranilla, J., Luaces, O., Alvarez, I., Royo, L. J. y Díez, J. (2001a). The usefulness of artificial intelligence techniques to assess subjective quality of products in the food industry. *Trends in Food Science and Technology*, 12:370–381.

[Goyache et al., 2001b] Goyache, F., del Coz, J. J., Quevedo, J. R., López, S., Alonso, J., Ranilla, J., Luaces, O., Alvarez, I. y Bahamonde, A. (2001b). Using artificial intelligence to design and implement a morphological assessment system in beef cattle. *Animal Science*, 73:49–60.

[Hastie et al., 2002] Hastie, T., Tibshirani, R. y Friedman, J. H. (2002). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Corrected printing.

[Joachims, 2001a] Joachims, T. (2001a). *The maximum-margin approach to learning text classifiers: methods, theory y algorithms*. PhD thesis, Universitat Dortmund.

[Joachims, 2001b] Joachims, T. (2001b). A statistical learning model of text classification with support vector machines. En Croft, W. B., Harper, D. J., Kraft, D. H. y Zobel, J., editors, *Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval*, páginas 128–136, New Orleans, US. ACM Press, New York, US.

[Kohonen, 1995] Kohonen, T. (1995). Self-organizing maps. *Springer Series of Information Science*.

[Kusabs et al., 1998] Kusabs, N., Bollen, F., Trigg, L., Holmes, G. y Inglis, S. (1998). Objective measurement of mushroom quality. En *New Zealand Institute of Agricultural Science and the New Zealand Society for Horticultural Science Annual Convention*, página 51. New Zealand.

[Luaces, 1999] Luaces, O. (1999). *Un sistema de aprendizaje de reglas explícitas mediante la generalización de instancias*. PhD thesis, Departamento de Informática, Universidad de Oviedo en Gijón.

[Luaces et al., 1999] Luaces, O., del Coz, J. J., Quevedo, J. R., Alonso, J., Ranilla, J. y Bahamonde, A. (1999). Autonomous clustering for machine learning. *Lecture notes in computer sciences*, 1606:497–506.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. En *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, páginas 281–297. Berkeley, University of California Press.

[Mangas et al., 1999a] Mangas, J. J., Moreno, J., Rodríguez, R., Picinelli, A. y Suárez, B. (1999a). Analysis of polysaccharides in cider: their effect on sensory foaming properties. *Journal of Agricultural and Food Chemistry*, 47:152–156.

[Mangas et al., 1999b] Mangas, J. J., Rodríguez, R., Suárez, B., Picinelli, A. y Dapena, E. (1999b). Study of the phenolic profile of cider apple cultivars at maturity by multivariate techniques. *Journal of Agricultural and Food Chemistry*, 47:4047–4052.

[Martinetz, 1993] Martinetz, T. M. (1993). Competitive hebbian learning rule forms perfectly topology preserving maps. En *ICANN'93: International Conference on Artificial Neural Networks*, páginas 427–434. Amsterdam, Springer.

[Martinetz y Schulten, 1991] Martinetz, T. M. y Schulten, K. J. (1991). A "neural-gas"network learns topologies. *Artificial Neural Networks*, páginas 397–402.

[Martinetz y Schulten, 1994] Martinetz, T. M. y Schulten, K. J. (1994). Topology representing networks. *Neural Networks*, 7(3):507–522.

[Matsumoto y Nishimura, 1998] Matsumoto, M. y Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1):3–30.

[Meilgaard et al., 1991] Meilgaard, D., Civille, G. V. y Carr, B. T. (1991). Sensory evaluation techniques. *Boca Raton CRC Press*.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

[Murthy et al., 1994] Murthy, S. K., Kasif, S. y Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32.

[Picinelli et al., 2000a] Picinelli, A., Suárez, B., García, L. y Mangas, J. J. (2000a). Changes in phenolic contents during sparkling apple winemaking. *American Journal of Enology and Viticulture*, 51:144–149.

[Picinelli et al., 2000b] Picinelli, A., Suárez, B., Moreno, J., Rodríguez, R., Caso, L. M. y Mangas, J. J. (2000b). Chemical characterization of Asturian cider. *Journal of Agricultural and Food Chemistry*, 48:3997–4002.

[Quevedo y Bahamonde, 1999] Quevedo, J. R. y Bahamonde, A. (1999). Aprendizaje de funciones usando inducción sobre clasificaciones discretas. En *VIII Conferencia de la Asociación Española para la Inteligencia Artificial - III Jornadas de Transferencia Tecnológica de Inteligencia Artificial*, páginas 64–71.

[Quevedo et al., 2001] Quevedo, J. R., del Coz, J. J. y Díez, J. (2001). Filtrando atributos para mejorar procesos de aprendizaje. En *IX Conferencia de la Asociación Española para la Inteligencia Artificial - IV Jornadas de Transferencia Tecnológica de Inteligencia Artificial*, páginas 123–132.

[Quinlan, 1987] Quinlan, J. R. (1987). Simplifaying decision trees. *International Journal of of Man-Machine Studies*, 27:221–234.

[Quinlan, 1992] Quinlan, J. R. (1992). Learning with continuous classes. En *Procedings fifth Australian Joint Conference on Artificial Intelligence.*, páginas 343–348. World Scientific, Singapore.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, California.

[Quinlan, 2002] Quinlan, J. R. (2002). Cubist release 1.09, [http://www.rulequest.com/cubist-info.html].

[Spiegel, 1970] Spiegel, M. R. (1970). *Estadística.* McGraw-Hill, Atlacomulco, México.

[Tesauro, 1989] Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. *Advances in Neural Information Processing*, páginas 99–106.

[Torgo, 2002] Torgo, L. (2002). Regression data sets repository at LIACC (laboratorio de inteligencia artificial y ciencias de la computación). [http://www.liacc.up.pt/ ltorgo/Regression/DataSets.html]. University of Porto, Portugal.

[Utgoff y Clouse, 1991] Utgoff, P. E. y Clouse, J. A. (1991). Two kinds of training information for evaluation function learning. En *Ninth National Conference on Artificial Intelligence*, páginas 596–600.

[Utgoff y Saxena, 1987] Utgoff, P. E. y Saxena, S. (1987). Learning a preference predicate. En *Fourth International Workshop on Machine Learning*, páginas 115–121.

[Vapnik, 1998] Vapnik, V. (1998). *Statistical Learning Theory.* John Wiley. New York, p.732.

[Wang y Witten, 1997] Wang, Y. y Witten, I. H. (1997). Induction of model trees for predicting continuous classes. En *European Conference on Machine Learning*, páginas 128–137.

[Wilson y Martinez, 2000] Wilson, D. R. y Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286.