ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Combinación del aprendizaje multitarea y del algoritmo EM en problemas de clasificación con datos incompletos

AUTOR: Juan José Alcaraz Jiménez
DIRECTOR: José Luis Sancho Gómez
CODIRECTOR: Pedro José García Laencina

Septiembre 2006

Autor	Juan José Alcaraz Jiménez.	**E-mail**	JuanjoAlcaraz@gmail.com
Director	José Luis Sancho Gómez.	**E-mail**	josel.sancho@upct.es
Codirector	Pedro José García Laencina	**E-mail**	pedroj.garcia@upct.es
Título del PFC	Combinación del aprendizaje multitarea y del algoritmo EM en problemas de clasificación con datos incompletos.		
Descriptores	Redes neuronales, MTL, EM, datos incompletos.		

Resumen

El propósito de este proyecto de fin de carrera consiste en combinar la capacidad de modelado de funciones de densidad del algoritmo EM con la potencia de las arquitecturas de redes neuronales MTL para poder resolver problemas de clasificación con conjuntos de datos incompletos.

Titulación	Ingeniero de Telecomunicación
Intensificación	Planificación y Gestión de Telecomunicaciones
Departamento	Tecnologías de la Información y Comunicaciones
Fecha de Presentación	Septiembre - 2006

A mis padres y a mi hermana

Agradecimientos

Quiero comenzar agradeciendo a José Luis, director de este proyecto, el haber despertado en mí el interés por el campo de la inteligencia artificial, así como el haberme brindado la posibilidad de realizar este proyecto.

Por otro lado, quiero señalar la gran ayuda que me ha prestado Pedro, codirector de este proyecto. Sin sus explicaciones sobre el funcionamiento de las redes neuronales ni su labor de supervisión y guía, hubiera sido imposible llevar a cabo este trabajo.

También me gustaría destacar a mis compañeros de carrera, quienes con sus bromas y amistad han hecho muy placentero el transcurso de estos últimos cinco años. Y a mis amigos de Cartagena y Mazarrón, por todos los inolvidables momentos de diversión que me han hecho vivir.

Finalmente, quiero dedicar un agradecimiento especial a mi familia; por el cariño, apoyo incondicional, alegrías y comprensión que me ofrecen cada día. Cualquier meta que logre en mi vida será gracias a vosotros.

ÍNDICE

Capítulo 1

Introducción

El problema fundamental al que nos enfrentamos con este trabajo consiste en la clasificación de una muestra dentro de un conjunto de clases diferentes predeterminadas. Cada muestra o patrón está representada por un conjunto de propiedades medidas, que se conoce por vector de características. Uno de los métodos más potentes, y por ello ampliamente utilizado, para solucionar problemas de clasificación es el uso de redes neuronales artificiales (RNA). Dado un conjunto de entrenamiento, que no es más que una serie de ejemplos, las redes neuronales son capaces de descubrir patrones complejos ocultos en los datos, y basarse en ellos a la hora de asignar una clase a una nueva muestra.

Por ejemplo, un patrón puede ser una señal sonora y su vector de características el conjunto de coeficientes espectrales extraídos de ella. Si se pretende resolver un problema de reconocimiento del hablante, podemos entrenar una red neuronal artificial para que, a partir de una cierta cantidad de patrones representativos de cada individuo, sea capaz de identificar al hablante ante una nueva señal sonora que no ha sido utilizada durante el aprendizaje de la máquina. Otro claro ejemplo es la detección en una persona de la existencia o no de una enfermedad a partir de los resultados de un conjunto de pruebas médicas.

Cuando tratamos con problemas reales, el conjunto de entrenamiento con frecuencia no es de la calidad que a nosotros nos gustaría. Un problema típico que puede sufrir el conjunto de entrenamiento es la pérdida de algunos de sus valores. Por ejemplo, si cada muestra nos aporta información acerca de seis características, puede que no sepamos el valor de dos de ellas. La ausencia de estos valores puede estar provocada por causas de origen tecnológico (corte del suministro eléctrico) o porque simplemente son imposibles de medirse (un paciente no se puede someter a una prueba determinada pero si al resto).

Ante la presencia de valores perdidos podemos actuar de diferentes maneras. La más simple de ellas consistiría en desechar todas las muestras a las que les falte alguna característica. Siendo este procedimiento el más usado, hay gran cantidad de problemas para los que sería impracticable. Imaginemos por ejemplo, que cada muestra tiene dieciséis características, y que al 99% de las muestras les falta por lo menos alguno de estos dieciséis valores. ¡Nos quedaríamos prácticamente sin datos!

Existen otros métodos que proponen soluciones más o menos acertadas para el tratamiento de los datos perdidos dependiendo del problema. Sin embargo, la filosofía del método que describiremos en este trabajo consiste en realizar una estimación de los datos incompletos orientada a la clasificación de los mismos e intentar extraer el máximo de información posible del conjunto de entrenamiento.

Dentro del campo de las redes neuronales, las diferentes estructuras de la red (forma de interconectar neuronas) son ilimitadas, así como la elección para el comportamiento de sus neuronas. Nosotros optaremos por un esquema de perceptrones multicapa (*Multilayer perceptron*, MLP).

Por otro lado, la configuración que hemos elegido para la red neuronal explota un concepto interesante: el aprendizaje multitarea (*MultiTask Learning*, MTL). En MTL, el aprendizaje de una determinada tarea (tarea principal) puede mejorarse si lo realizamos de forma simultánea al aprendizaje de otras tareas relacionadas (tareas secundarias). Por ejemplo, un buen entrenador de baloncesto no se limitaría a poner a sus jugadores a

disputar partidos entre ellos todo el tiempo; sino que, aparte de los partidillos, también realizaría ejercicios específicos para el tiro a canasta, la resistencia física, la técnica de entrar a canasta, la musculación, etc. Del mismo modo, las redes MTL además de aprender la tarea principal, aprenden otras tareas secundarias relacionadas con la misma.

En nuestro caso, la tarea principal es la tarea de clasificación, y las tareas secundarias consistirán en la estimación de los valores de cada una de sus características que presentan valores perdidos, a partir del resto de entradas observadas. El aprendizaje de estas estimaciones se empleará posteriormente para rellenar los valores incompletos de las muestras y conseguir así un problema de clasificación con datos completos. De este modo, se combina la imputación y clasificación en una misma RNA, estando la imputación de datos guiada y orientada por el aprendizaje de la tarea principal de clasificación.

El método que siguen las redes neuronales para aprender se basa en la minimización de una función de error. Para cada muestra de entrenamiento la red proporciona un resultado, posteriormente se evalúa la diferencia entre ese resultado y el valor real que deberíamos haber obtenido y dependiendo de el análisis del valor de esta diferencia modificaremos los parámetros de nuestra red. Sin embargo, en la aplicación de una red MTL del tipo que utilizamos, surgen problemas derivados de la minimización del error de estimación. Cuando, para una determinada característica, las muestras de un conjunto de entrenamiento se distribuyen en dos grupos separados una cierta distancia, la red tiende a imputar en el espacio vacío que hay entre los grupos, ya que de este modo minimiza su función de error. Por tanto, en este caso, los valores imputados son muy poco probables, o incluso improbables, teniendo en cuenta la distribución del conjunto de datos. Estas dificulatades motivaron la realización de este proyecto fin de carrera (PFC).

El objetivo de este PFC consiste en implementar un algoritmo de modelado de funciones de densidad de probabilidad (fdp) a partir del conjunto de entrenamiento, emplear esta información para guiar el aprendizaje de las tareas secundarias de la red MTL y, de este modo, intentar conseguir imputaciones más probables en el espacio de entrada.

Para realizar el modelado de las densidades de probabilidad es preciso contar con un método potente y capaz de trabajar con valores perdidos. Por esta razón se ha escogido el modelado de mezclas de gaussianas (*Gaussian Mixture Models*) entrenado mediante el algoritmo EM (*Expectation-Maximization*).

Tras analizar los motivos que nos llevan al desarrollo de este PFC, a continuación se exponen los objetivos fundamentales del mismo y se introducen los conceptos de aprendizaje máquina y datos incompletos. Finalmente, se muestran las líneas de trabajo desarrolladas.

1.1. Objetivos del proyecto

Con este trabajo hemos perseguido dos objetivos de una naturaleza bien distinta. El primero de ellos radica en el estudio del estado del arte del algoritmo EM, de sus aplicaciones y de sus limitaciones. Mientras que el segundo consiste combinar la estimación de fdp mediante el algoritmo EM con una red MTL que realiza una imputación orientada a clasificación para mejorar el rendimiento de ésta. El primero de los objetivos trata de sintetizar las teorías desarrolladas por diferentes autores y aplicarla a problemas concretos. En cambio el segundo propone un nuevo método para solucionar un problema de gran interés científico.

En la primera etapa nos centramos en el modelado de mezclas de gaussianas mediante el algoritmo EM, y hemos tratado de:

- Implementar una versión robusta del algoritmo, con alta tasa de convergencia.
- Desarrollar un conjunto de funciones que flexibilicen el uso del algoritmo, haciendo posible su aplicación en varios tipos de problemas.
- Crear un entorno visual que nos permita estudiar el comportamiento del algoritmo frente a los diferentes problemas.

En la segunda etapa, abordamos la clasificación de conjuntos incompletos mediante MTL, siendo nuestros objetivos:

- Estudiar el problema de la calidad de imputación, analizando sus posibles causas.

- Proponer diferentes soluciones al mismo, basándonos en los resultados obtenidos en la primera etapa.

1.2. El aprendizaje máquina

La capacidad de aprender es probablemente la más útil y menos entendida que los humanos poseemos. El proceso de aprendizaje humano parece a menudo fácil. Sin embargo, cuando intentamos crear máquinas capaces de aprender, basándonos en el aprendizaje humano, surge la verdadera complejidad de la tarea. El aprendizaje máquina es una subárea de la inteligencia artificial que trata del diseño y desarrollo de sistemas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos, es decir, crear sistemas que aprendan automáticamente una determinada tarea a partir de la experiencia dada por un conjunto de datos.. El interés en este campo ha sido creciente a lo largo de las últimas décadas, hasta el punto de desbordar el campo de la ciencia y saltar a las pantallas de cine con películas como *Matrix*, *Terminator* o *Artificial Intelligence*. Existen muchos métodos con un grado variable de éxito en diferentes tipos de problemas. Sin embargo, ningún método consigue acercarse a la capacidad del cerebro humano de aprender nuevas tareas.

El método de aprendizaje máquina que trataremos en este trabajo se conoce como red neuronal artificial (RNA). Estas redes se diseñan para simular el aprendizaje del sistema nervioso humano. Consisten en un gran número de nodos, o neuronas, interconectados unos con otros para resolver tareas. Las redes neuronales se han revelado como un medio eficiente para mejorar el funcionamiento basado en la experiencia, pero hay aún mucho trabajo por hacer en este campo. En el capítulo 3 estudiaremos detenidamente el funcionamiento de las redes neuronales.

No existe una definición universalmente aceptada de red neuronal artificial, aunque la mayoría de personas que trabajan con ellas estarían de acuerdo en definir una RNA como un "*sistema de procesamiento de información formado por un conjunto de procesadores simples organizados en paralelo, cada uno de los cuales tiene una pequeña cantidad de*

memoria”. Las unidades operan sólo con la información disponible localmente, que reciben a través de las conexiones con otras unidades mediante canales de comunicación por los que fluye información de tipo numérico (frente a simbólico). Las redes neuronales artificiales tienen una regla de aprendizaje que les permite aprender a partir de los datos que reciben del exterior. Por aprendizaje se entiende el proceso de adaptación de las conexiones entre neuronas, ya que es en dichas conexiones, y no en las unidades de procesamiento, donde reside el conocimiento de una RNA.

Uno de los campos donde está más extendido el uso de las redes neuronales es en reconocimiento de patrones. La mayoría de los problemas en reconocimiento de patrones son resueltos mediante el aprendizaje de una única tarea. Este aprendizaje, conocido como *aprendizaje monotarea* o STL (Single Task Learning), obvia las ventajas y el funcionamiento real del aprendizaje humano. En éste, el aprendizaje de una tarea se ve mejorado y completado por el aprendizaje de otras tareas relacionadas con la primera. Así por ejemplo, una persona que durante su formación ha aprendido simultáneamente recursos humanos y administración de empresas realizará más eficientemente la selección del personal de una empresa que otra que únicamente tiene conocimientos de administración de empresas o recursos humanos. Para ello durante el aprendizaje de una tarea principal se añaden tareas extra relacionadas. Este tipo de aprendizaje se conoce como *Aprendizaje Multitarea* o MTL (Multitask Learning) [2]. Recientemente, muchos trabajos han aplicado estas ventajas al aprendizaje máquina, entre ellos los trabajos llevados a cabo por *García Laencina* en relación a los conjuntos de datos incompletos [3].

1.2.1 Aplicaciones de las redes neuronales

Las características de las redes neuronales artificiales las hacen bastante apropiadas para aplicaciones en las que no se dispone a priori de un modelo identificable que pueda ser programado, pero si se dispone de un conjunto básico de ejemplos de entrada (previamente clasificados o no). Esto es particularmente útil en aplicaciones donde la complejidad de los datos o tareas convierte el diseño manual de dicha función impracticable. Asimismo, son altamente robustas al ruido y son fácilmente paralelizables.

Las redes neuronales artificiales han sido aplicadas con éxito en:

- Aproximación de funciones, o análisis de regresión, incluyendo predicción de series temporales y modelado.
- Clasificación, incluyendo reconocimiento de patrones y secuencias, detección de fallos y toma de decisiones.
- Procesado de datos, incluyendo filtrado, agrupamiento o clustering y compresión.

Dentro de estas áreas podemos destacar las siguientes aplicaciones:

- Sistemas de identificación y control (control de vehículos, de procesos...).
- Reconocimiento de secuencias (gestos, habla, reconocimiento de textos manuscritos,...).
- Diagnósticos médicos, predicción de reacciones adversas de medicamentos, entendimiento de causa de ataques epilépticos...
- Optimización del uso de recursos escasos
- Aplicaciones financieras (detección de patrones de fraude, predicciones en el mercado financiero, evaluación del riesgo de créditos...).
- Juego de partidas y toma de decisiones (backgammon, ajedrez, carreras...), reconocimiento de formas (sistemas de radar, identificación de rostros, reconocimiento de objetos,...).
- Minería de datos (o descubrimiento de conocimiento en bases de datos, “KDD”).
- Visualización y filtro de *spam* en correos electrónicos.
- Predicciones de tiempo atmosférico...

También se pueden utilizar en problemas para los que no existen modelos matemáticos precisos o algoritmos con complejidad razonable; por ejemplo la red de Kohonen ha sido aplicada con un éxito más que razonable al clásico problema del viajante (un problema para el que no se conoce solución algorítmica de complejidad polinómica)[4].

Otro tipo especial de redes neuronales artificiales se ha aplicado en conjunción con los algoritmos genéticos (AG) para crear controladores para robots. En este tipo de

aplicación el genoma del AG lo constituyen los parámetros de la red (topología, algoritmo de aprendizaje, funciones de activación, etc.) y la adecuación de la red viene dada por la adecuación del comportamiento exhibido por el robot controlado (normalmente una simulación de dicho comportamiento) [5].

Por tanto, podemos decir que las redes neuronales artificiales son actualmente aplicadas con éxito en infinidad de campos y con unas expectativas de crecimiento muy elevadas.

1.3. Reconocimiento de patrones

En problemas de clasificación, la tarea es asignar una clase a un nuevo vector de entradas entre un número determinado de clases o categorías. De cualquier modo, hay muchas otras tareas de reconocimiento de patrones, a las que nos referiremos como problemas de regresión, en los que las salidas representan los valores de variables continuas. Ejemplos de este tipo de problemas incluyen la determinación de la fracción de aceite en una tubería a partir de medidas de la atenuación de los rayos gamma que pasan a través de la tubería, y la predicción de la tasa de cambio de divisas en algún momento del futuro, dados sus valores en varios instantes de tiempo recientes. En realidad, el término regresión se refiere a una clase específica de función definida en términos de promediados sobre una cantidad aleatoria. Tanto los problemas de regresión como los de clasificación pueden verse como casos particulares del problema de la aproximación de funciones. En el caso de los problemas de regresión, se pretende aproximar una función continua de regresión, mientras que para los problemas de clasificación, lo que intentaremos aproximar son las probabilidades de pertenencia a las diferentes clases expresadas como funciones de los valores de entrada. La mayoría de los asuntos que deben ser controlados para la resolución de problemas de reconocimiento de patrones, son comunes tanto a clasificación como a regresión.

En este trabajo nos centraremos principalmente en problemas de clasificación, ya que como veremos más adelante, ésta es la tarea que realmente se desea resolver. Por otro lado, si el conjunto de datos de entrada presenta valores perdidos en algunas de sus

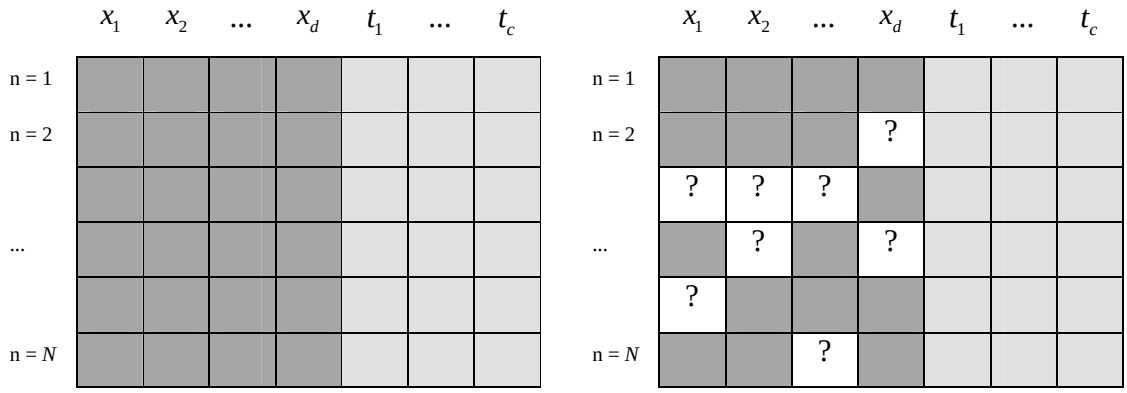
características, podemos considerar adicionalmente las tareas de estimación de valores perdidos asociadas a cada característica incompleta como un problema de regresión. Por tanto, en general, abordaremos la resolución conjunta de una tarea discreta de clasificación y tantas tareas continuas de estimación de datos perdidos como características incompletas. A continuación presentamos un problema genérico de clasificación con entradas incompletas, mostrando la notación que emplearemos en este trabajo.

Supongamos un problema de clasificación, que consiste en el cálculo de una función discriminante *f* que realice un mapeado del espacio de entrada *X* al espacio de salida *C*, donde se asigne cada vector de entrada a una de las *c* posibles clases que componen el espacio de salida *C*.

$$f : X \longrightarrow C \tag{1.1}$$

En tareas convencionales de clasificación, todos los valores de los atributos de cada muestra son conocidos, es decir, el conjunto de entrada *X* es completamente observable. Considérese un conjunto de *N* vectores de entrada, donde el vector *n*-ésimo $\mathbf{x^{(n)}}$ está definido por *d* características o atributos reales, $\mathbf{x^{(n)}} = \left(x_1^{(n)}, x_2^{(n)}, ..., x_1^{n}\right)$. Cada caso está asociado a una determinada clase, conocida como salida deseada $t^{(n)}$**.** Por otro lado, es posible codificar $t^{(n)}$ en un vector que use una codificación *1-de-c*. Por ejemplo, si hay cinco posibles clases y la muestra n-ésima pertenece a la tercera, su vector objetivo será $\mathbf{t^{(n)}}$=(0,0,1,0,0).

En las tareas de clasificación analizadas en este texto, los valores de algunas características de entrada pueden estar incompletas (presentar valores perdidos). La figura 1-1(a) muestra un problema de clasificación clásico con un conjunto completo de datos. En cambio, en la figura 1-1(b) podemos ver un problema de clasificación donde algunos vectores son incompletos. En este trabajo, consideraremos que los valores perdidos no están siempre en el mismo atributo de nuestras muestras, asumiendo la hipótesis MCAR. Por tanto, es posible que el atributo i-ésimo de una muestra esté perdido mientras que el mismo atributo para otra sea conocido. Nos referiremos a un valor perdido con el símbolo ?; de este modo, la muestra $x^{(4)}$=(0.1,-0.2,?,0.3,?) presenta valores perdidos en sus atributos tercero y quinto.

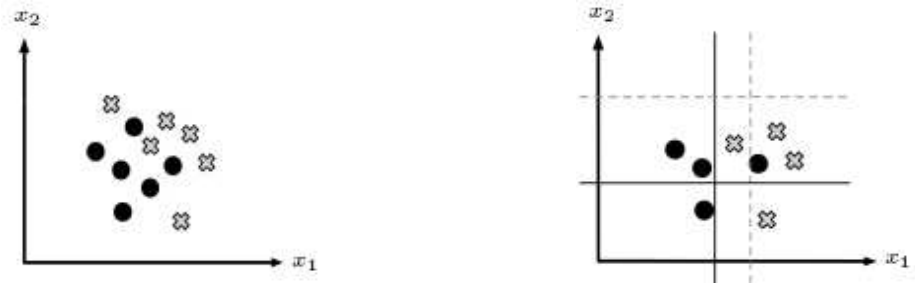


a) Problema de clasificación típico con datos completos

b) Problema de clasificación con muestras incompletas, denotadas por el símbolo ?.

Figura 1-1 Dos hipotéticos problemas de clasificación, el primero (a) con datos completos y el segundo (b con muestras que presentan valores incompletos .

Para explicar mejor el problema de los datos perdidos en clasificación, haremos uso de un ejemplo sencillo. Supongamos un problema de clasificación de *N* muestras bidimensionales $x^{(n)} = (x_1^{(n)}, x_2^{(n)})$, donde cada muestra puede pertenecer a dos posibles clases etiquetadas con +1 o -1. Además, cualquier atributo de cualquier vector puede ser un valor perdido.



a) Conjunto de datos completo

b) Conjunto de datos incompleto

Figura 1-2 Problema bidimensional de clasificación entre dos clases.

La figura 1-2 representa muestras completas e incompletas usando diagramas de dispersión. La figura 1-2(a) representa un conjunto de datos completo, mientras que la figura 1-2(b) representa la versión incompleta del conjunto, donde un 33% de los atributos

son desconocidos. Representamos una muestra incompleta $x^{(n)} = (x_1^{(n)}, ?)$ como una línea vertical con componente horizontal $x_1^{(n)}$; y $x^{(n)} = (?, x_2^{(n)})$ como una línea horizontal con componente vertical $x_2^{(n)}$. Para dimensiones mayores, una muestra completa puede ser representada con una línea, plano, hiperplano, etc., dependiendo del número de valores perdidos que presente.

1.4. Valores perdidos

En el aprendizaje biológico y computacional, el entorno a menudo no proporciona información completa al sistema de aprendizaje. Por ejemplo, un sistema de visión puede encontrar muchos ejemplos de un objeto en los que no vemos alguna parte del mismo, y sin embargo tiene que encontrar un modelo para el objeto completo. Similarmente, podemos necesitar un controlador adaptativo para aprender a mapear las lecturas de un sensor con las acciones correspondientes, incluso si los sensores son poco fiables y en algunas ocasiones no consiguen proporcionar lecturas. Ejemplos de conjuntos de datos con características incompletas (*missing*) son muy frecuentes en el aprendizaje máquina.

1.4.1 Motivos de la pérdida de datos

Los datos perdidos son relativamente frecuentes en muchos problemas con datos reales, algunas de las razones por las que pueden producirse son las siguientes:

- Fallos (cortes) durante la transmisión.
- Enmascaramiento de señales debido al ruido (reconocimiento de voz).
- Pruebas médicas imposibles de realizar.
- Fallos durante el proceso de adquisición de datos de un sensor.

La ausencia de datos, ya sea en unidades completas o en características de algunos patrones, crea sesgos potenciales en las estimaciones de los parámetros de interés. Respecto a los datos incompletos en ítems hay diversas maneras de tratarlos. En los años setenta, la regla general era olvidarlos, por lo que su tratamiento consistía en la eliminación de los casos con información incompleta. En los años ochenta se generalizó el tratamiento de los

datos incompletos a través de la búsqueda de un valor que posteriormente sería asignado al dato faltante. En la década de los noventa se produjo un cambio en la filosofía del tratamiento de los datos incompletos: ya no importa buscar un valor, sino modelar la incertidumbre alrededor de él, y se comienzan a realizar las primeras imputaciones múltiples.

Si la proporción de datos faltantes en cada variable es baja y se asume que los valores perdidos son aleatorios, éstos no representarán un problema importante en un análisis estadístico de tipo univariado. Sin embargo, en un análisis multivariado la situación anterior puede ser problemática. Así, si por ejemplo se registran diez variables en cien casos, con un 5% de valores faltantes en cada una de ellas, con valores predichos totalmente aleatorios, y con un pequeño solapamiento de éstos entre los casos (es decir, un caso no contiene muchos valores ausentes, sino que éstos se reparten en diferentes casos), es muy probable que, si se aplica el tradicional método de eliminación de registros, casi la mitad de la muestra se pierda en un análisis estadístico multivariado que incluya las diez variables, mientras que en cada análisis univariado sólo se perdería un 5% de casos.

1.4.2 Tipos de datos perdidos

A continuación trataremos de definir de un modo más riguroso el concepto de datos perdidos. En este trabajo revisaremos el problema del aprendizaje a partir de datos incompletos desde una perspectiva estadística. El marco estadístico que adoptaremos [7] realiza una distinción entre el medio, que asumiremos que genera datos completos, y el mecanismo de pérdida de datos, que oculta algunas de las salidas del medio de forma que no sean observables por la máquina. El problema del aprendizaje supervisado consiste en formar un mapeo de las entradas a las salidas deseadas. El proceso de aprendizaje no supervisado consiste generalmente en extraer alguna descripción estadística compacta de las entradas. En ambos casos, la máquina se puede beneficiar del conocimiento de restricciones tanto en el proceso de generación de datos (por ejemplo, si se corresponde con una determinada función de densidad) como del mecanismo que causó el patrón de datos incompletos (que es independiente del proceso de generación de los datos). El uso de la teoría estadística nos permite formalizar las consecuencias de estas restricciones y nos

proporciona un marco para derivar algoritmos de aprendizaje que hacen uso de estas consecuencias.

Asumiremos que el conjunto de datos $\mathbf{X}=\{\mathbf{x}^{(n)}\}$ $n=1..N$ puede dividirse en una componente observada $\mathbf{X}^{\mathbf{o}}$ y una componente perdida $\mathbf{X}^{\mathbf{m}}$. Cada vector de datos $\mathbf{x}^{(n)}$ puede tener diferentes patrones de características perdidas. Nosotros no distinguiremos a partir de ahora entre entradas y componentes objetivo del vector de datos. Formalizaremos la noción de mecanismo de perdida de datos definiendo una matriz indicadora de datos perdidos R, tal que

$$R_i^{(n)} = \begin{cases} 1, & x_i^{(n)} \ observado \\ 0, & x_i^{(n)} \ perdido \end{cases} . \tag{1.2}$$

Tanto el proceso de generación de datos como el mecanismo de pérdida de datos son considerados como un proceso aleatorio, con distribución de probabilidad conjunta dada por

$$P(\mathbf{X},\mathbf{R}\,|\,\boldsymbol{\theta},\boldsymbol{\Phi}) \;=\; P(\mathbf{X}\,|\,\boldsymbol{\theta})P(\mathbf{R}\,|\,\mathbf{X},\boldsymbol{\Phi})\,. \tag{1.3}$$

Usamos los parámetros $\boldsymbol{\theta}$ para el proceso de generación de datos y un conjunto de parámetros distintos, $\boldsymbol{\Phi}$, para el mecanismo de pérdida de datos. Una vez que el modelo de probabilidad de los datos ha sido descompuesto como en (1.3), podemos distinguir tres tipos anidados de mecanismos de pérdida de datos. Los datos pueden ser:

- *Perdidos de forma completamente aleatoria* (MCAR, Missing Completely At Random):

  $$P(\mathbf{R}\,|\,\mathbf{X},\boldsymbol{\Phi}) \;=\; P(\mathbf{R}\,|\,\boldsymbol{\Phi}) \tag{1.4}$$

  Es decir, la probabilidad de que $\mathbf{x}^{(n)}$ se pierda es independiente de los valores en el vector de datos.

- *Perdidos de forma aleatoria* (MAR, Missing At Random):

  $$P(\mathbf{R}\,|\,\mathbf{X}^{\mathbf{o}},\mathbf{X}^{\mathbf{m}},\boldsymbol{\Phi}) = (\mathbf{R}\,|\,\mathbf{X}^{\mathbf{o}},\boldsymbol{\Phi}) \tag{1.5}$$

  Es decir, que la probabilidad de que $x_d^{(n)}$ (atributo *d*-ésimo del vector $\mathbf{x}^{(n)}$) esté perdido es independiente de los valores de las componentes perdidas de los datos,

pero puede depender de los valores de las componentes observadas. Por ejemplo, $x_d^{(n)}$ puede estar perdido para ciertos valores de $x_e^{(n)}$ ($d \neq e$) suponiendo que $x_e^{(n)}$ es siempre observado. El experimento en 5.1.4 ilustra este caso.

- *No perdidos de forma aleatoria* (NMAR, Not Missing At Random). El mecanismo de pérdida puede depender de los valores de las componentes perdidas ($P(\mathbf{R} \mid \mathbf{X^o}, \mathbf{X^m}, \mathbf{\Phi})$). Si $P(R_d^{(n)} \mid x_d^{(n)}, \mathbf{\Phi})$ es una función de $x_d^{(n)}$ se dice que los datos están censurados. Por ejemplo, si un sensor falla cuando sus entradas exceden algún rango sus salidas estarán censuradas[1].

El tipo de mecanismo de pérdida de datos es crítico en la evaluación de los algoritmos de aprendizaje para manejar datos incompletos. Las estrategias de máxima verosimilitud y Bayesiana pueden manejar conjuntos de datos MAR o MCAR. Algunas aplicaciones de aprendizaje más simples pueden manejar datos MCAR pero fallan con datos MAR en general. No existen estrategias generales para datos NMAR.

Tanto para las técnicas Bayesianas como para las de máxima verosimilitud, las estimaciones de los parámetros $\mathbf{\Phi}$ y $\boldsymbol{\theta}$ están relacionados con los datos observados, $\mathbf{X^o}$, y con $\mathbf{R}$ vía $P(\mathbf{X^o}, \mathbf{R} \mid \boldsymbol{\theta}, \mathbf{\Phi})$. Para métodos de máxima verosimilitud, la verosimilitud es

$$L(\boldsymbol{\theta}, \mathbf{\Phi} \mid \mathbf{X^o}, \mathbf{R}) \propto P(\mathbf{X^o}, \mathbf{R} \mid \boldsymbol{\theta}, \mathbf{\Phi}) \tag{1.6}$$

y para métodos Bayesianos, la probabilidad a posteriori es

$$P(\boldsymbol{\theta}, \mathbf{\Phi} \mid \mathbf{X^o}, \mathbf{R}) \propto P(\mathbf{X^o}, \mathbf{R} \mid \boldsymbol{\theta}, \mathbf{\Phi}) P(\boldsymbol{\theta}, \mathbf{\Phi}) . \tag{1.7}$$

Nos gustaría cerciorarnos de las condiciones bajo las cuales los parámetros del proceso de generación de datos pueden ser estimados independientemente de los parámetros del mecanismo de pérdida de datos. Dado que

[1] En la literatura sobre el tema, en ocasiones se emplea el término censura para los tipos de pérdidas en una dimensión ocasionadas por el valor de otra dimensión. En este caso se trataría de un mecanismo MAR, donde los algoritmos de máxima verosimilitud se muestran especialmente eficientes. Nosotros denominaremos a este tipo de mecanismo de pérdida dependiente de otras dimensiones como "censura cruzada" para diferenciarla de la censura clásica.

$$P(\mathbf{X^o}, \mathbf{R} \mid \boldsymbol{\theta}, \boldsymbol{\Phi}) = \int P(\mathbf{X^o}, \mathbf{X^m} \mid \boldsymbol{\theta}) P(\mathbf{R} \mid \mathbf{X^o}, \mathbf{X^m}, \boldsymbol{\Phi}) d\mathbf{X^m} \quad (1.8)$$

podemos observar que si

$$P(\mathbf{R} \mid \mathbf{X^o}, \mathbf{X^m}, \boldsymbol{\Phi}) = P(\mathbf{R} \mid \mathbf{X^o}, \boldsymbol{\Phi}) \quad (1.9)$$

es decir, si el mecanismo de pérdida no es NMAR, entonces

$$P(\mathbf{X^o}, \mathbf{R} \mid \boldsymbol{\theta}, \boldsymbol{\Phi}) = P(\mathbf{R} \mid \mathbf{X^o}, \boldsymbol{\Phi}) \int P(\mathbf{X^o}, \mathbf{X^m} \mid \boldsymbol{\theta}) d\mathbf{X^m} = P(\mathbf{R} \mid \mathbf{X^o}, \boldsymbol{\Phi}) P(\mathbf{X^o} \mid \boldsymbol{\theta}) \quad (1.10)$$

La ecuación (1.10) pone de manifiesto que si los datos son MAR, entonces la verosimilitud puede ser factorizada. Para métodos de máxima verosimilitud esto implica directamente que maximizar $L(\boldsymbol{\theta} \mid \mathbf{X^o}) \propto P(\mathbf{X^o} \mid \boldsymbol{\theta})$ como una función de $\boldsymbol{\theta}$ es equivalente a maximizar $L(\boldsymbol{\theta}, \boldsymbol{\Phi} \mid \mathbf{X^o}, \mathbf{R})$. Por consiguiente, los parámetros del mecanismo de pérdida de datos pueden ser ignorados para el propósito de estimar $\boldsymbol{\theta}$ [7].

Para los métodos Bayesianos, el mecanismo de pérdida de datos no puede ser ignorado a menos que añadamos el requisito adicional de que lo anterior es factorizable: $P(\boldsymbol{\theta}, \boldsymbol{\Phi}) = P(\boldsymbol{\theta})P(\boldsymbol{\Phi})$. Estos resultados implican que los conjuntos de datos que son NMAR, como los datos censurados, no pueden ser manejados mediante métodos Bayesianos o basados en verosimilitud a menos que un modelo del mecanismo de pérdida de datos sea aprendido también. En el lado positivo, los resultados también implican que la condición MAR, que es más débil que la condición MCAR, es suficiente para el aprendizaje Bayesiano o basado en verosimilitud.

1.5. Alternativas para el tratamiento de valores perdidos

Ante el problema de la existencia de valores perdidos la comunidad científica ha propuesto gran cantidad de soluciones. Como no podía ser de otra forma, la respuesta natural que se dio en un comienzo a la ausencia de algunos valores fue la eliminación de las muestras que contenían dichos valores. Este tipo de prácticas se conocen como *procedimientos basados en registros completos.* Esta opción sólo da resultados aceptables cuando el número de datos faltantes es pequeño y tienen un origen aleatorio. Cuando no se cumplen dichas condiciones estos procedimientos pueden conducir a importantes sesgos, motivo por el cual no suelen ser la opción recomendada [7].

Cuando por desgracia no se puede emplear la anterior técnica porque no se cumplen las condiciones necesarias, nos vemos obligados a recurrir a los *procedimientos basados en la imputación del valor faltante*. Se entiende por imputación el proceso de estimación de datos incompletos y posterior "relleno" de los mismos. Consisten en imputar un valor a cada dato perdido para, posteriormente, aplicar un análisis convencional de datos completos. Uno de los procedimientos más habituales es la *imputación a la media*.

La imputación a la media consiste como su nombre indica en imputar a la media de los datos de los que disponemos, ahora bien, podemos distinguir dos versiones: la imputación a la media incondicional y la imputación a la media condicional. En la imputación a la media incondicional simplemente imputamos el valor de un atributo a la media de los valores observados para esa dimensión. En la media condicional, sin embargo, tendremos en cuenta el valor de las otras dimensiones para imputar el valor. En problemas de clasificación, donde cada vector de entrada está asociado a una determinada clase, podemos realizar una imputación a la media condicional e incondicional, pero a su vez condicionada a la clase a la que pertenece cada vector de entrada.

De cualquier modo, los métodos más potentes están basados en el aprendizaje máquina. En particular, nos centraremos en los métodos para el manejo de datos perdidos por medio de las redes neuronales artificiales en los problemas de clasificación. El problema cuando usamos redes neuronales artificiales con datos perdidos es encontrar la salida de una unidad cuando algunos de los valores están ausentes a la entrada del nodo, ya que, generalmente, cada neurona de la capa oculta de la RNA calcula la suma ponderada de todas sus entradas.

La mayoría de los procedimientos han sido propuestos usando el siguiente método: primero, preprocesamos los datos de entrada para obtener un conjunto de datos completo y rectangular, y entonces resolvemos el problema de clasificación usando una RNA. *Sharpe* y *Solly* [10] usan modelos basados en el perceptrón multicapa (MLP) para estimar los valores perdidos en conjuntos de datos incompletos. Con este método, llamado redes neuronales reducidas (Reduced NNs), se crea un conjunto de MLPs, y cada MLP aprende cada una de

las posibles combinaciones de atributos ocultos, usando solamente los atributos completos como entradas y los atributos ocultos como salidas. Estos MLPs son entrenados con vectores de entrada completos, y posteriormente estas redes estiman los valores perdidos del conjunto de datos. Los valores estimados son sustituidos en los datos para obtener un conjunto completo de entradas. Finalmente, el conjunto completo se introduce en una RNA clasificadora. El principal inconveniente de este método es que requiere un número muy elevado de neuronas, además de que la estimación de datos perdidos propuesta no está orientada a clasificación. *Yoon* y *Lee* [11] sugieren un algoritmo que está compuesto por 3 pasos, el primero de ellos, entrenar un MLP únicamente con la porción observada del conjunto de datos para aprender la tarea de clasificación, el segundo paso, la estimación de los atributos perdidos con el MLP entrenado y la porción incompleta de datos usando el algoritmo de retropropagación en las entradas, y posteriormente entrenamiento de un MLP con el conjunto completo de datos para aprender la tarea de clasificación. Este método de imputación está orientado a resolver la tarea de clasificación, pero no puede manejar valores perdidos durante la etapa de operación, además cuando todos los casos presentan por lo menos un valor perdido, no puede calcular los primeros valores para los pesos durante la etapa de entrenamiento. *Batista* y *Monard* [12] realizan un estudio del método de los k-vecinos más cercanos (K-NN) como procedimiento de imputación, y tras eso, un algoritmo de aprendizaje máquina clasifica el conjunto completo de datos obtenido. Por otro lado, *Markey*, *Tourassy*, *Margolis* y *DeLong* [13] analizan el efecto de los datos perdidos en RNA entrenadas en tres casos: sin datos incompletos, reemplazando los valores perdidos sumando imputación incondicional a la media y procedimiento de imputación múltiple.

Otra rama de métodos emplea modelos de mezclas gaussianas (GMM) incorporadas al algoritmo EM para realizar las estimaciones [8] como veremos en el apartado 2.5. El principal inconveniente es que este tipo de algoritmos son menos potentes en clasificación que los basados en RNA, y por lo tanto en problemas complejos ofrecen resultados más pobres. De forma similar, *Tresp, Neuneir* y *Ahmad* [14] usan las ventanas de de Parzen (aunque ellos si incluyeron RNA) para estimar la distribución y densidad condicional del conjunto de datos.

Otros métodos propuestos adaptan sus procesos de aprendizaje para ser capaces de tratar con entradas con valores perdidos. *Ishibuchi, Miyazaki, Kwon* y *Tanaka* [15] usan una representación a intervalos de datos incompletos con las entradas perdidas. Siendo el espacio de entrada *d*-dimensional, es posible normalizarlo en un hipercubo *d*-dimensional, estando cada dimensión comprendida entre cero y uno. Si cada entrada incompleta es representada por un intervalo que incluye todos sus valores posibles, un valor perdido es representado por el intervalo [0,...,1]. El aprendizaje de la red propuesta es adaptado para considerar la representación de entradas por intervalos. *Viharos, Novaki* y *Vincze* [16] desarrollan un método para manejar datos perdidos basado en el uso de una variable indicadora o "bandera" de validación para cada muestra de entrada. La bandera de validación indica si un valor del vector de entrada es desconocido o no, y los pesos de las entradas se actualizan teniendo en cuenta esta información.

En los últimos años, se han desarrollado algunas soluciones al problema de los datos perdidos con objeto de resolver al mismo tiempo el problema de la clasificación y la estimación de valores perdidos. *Bengio y Gingras* [17] proponen una Red Neuronal Recurrente (*Recurrent, Neural Network,* RNN) con retroalimentación a las unidades de entrada para manejar los valores perdidos durante el proceso de entrenamiento. Las conexiones recurrentes conectan la capa oculta con las entradas que presentan datos incompletos. Estas RNN tienen la característica de minimizar el error de clasificación y estimar valores perdidos al mismo tiempo durante el aprendizaje, pero no aprenden las diferentes tareas asociadas a las características incompletas, es decir, no trabaja como un esquema MTL, obviando las ventajas que aportal. *Parveen, Green* y *Abdel-Haleem* [18] extienden este procedimiento al reconocimiento automático del habla, añadiendo salidas extras para aprender las características incompletas. Este modelo puede ser considerado como una variante limitada de MTL porque no explota todas las ventajas de MTL, como relación entre tareas, subredes privadas, entradas extra y consistencia, que si son empleadas en el modelo que nosotros emplearemos y que será tratado con detenimiento en el capítulo 3.

1.6. Métodos de estimación de la función de densidad de probabilidad basados en la función de verosimilitud

El método que proponemos para mejorar la capacidad de clasificación de la RNA con estructura MTL que hemos empleado implica la estimación de la fdp de los datos, para de este modo guiar a la red neuronal durante el proceso de estimación de valores perdidos. En estas circunstancias (necesidad de estimar fdp y presencia de valores perdidos) la estimación basada en la maximización de una función de verosimilitud mediante el algoritmo EM se presenta como la solución. A continuación ofrecemos una breve introducción al concepto que será ampliado en el capítulo 2.

El modelado de funciones de densidad de probabilidad (fdp) basado en la función de verosimilitud, a diferencia del modelado paramétrico, no considera fijos los parámetros que definen la distribución, sino que se basa en calcular la verosimilitud de diferentes parámetros dados los datos observados en la muestra [19].

Si denominamos $\boldsymbol{\theta}$ al parámetro o parámetros que definen la distribución poblacional con función de densidad $p(\mathbf{X}|\boldsymbol{\theta})$, e $\mathbf{X^o}$ al conjunto de datos observados, la función de verosimilitud $L(\boldsymbol{\theta}|\mathbf{X^o})$ es una función de $\boldsymbol{\theta}$ proporcional a $p(\mathbf{X}|\boldsymbol{\theta})$ que determina la verosimilitud de los posibles valores de $\boldsymbol{\theta}$.

La función de verosimilitud aparece al invertir el papel de la función de densidad (o función de probabilidad si la variable es discreta). En lugar de suponer que conocemos $\boldsymbol{\theta}$ y queremos calcular las probabilidades de diferentes $\mathbf{X^o}$, suponemos que hemos observado una muestra $\mathbf{X^o}$ y evaluamos la verosimilitud de diferentes valores de $\boldsymbol{\theta}$ [20].

La función de verosimilitud se define partiendo de la función de probabilidad de la distribución conjunta de la muestra, que se obtiene como el producto de las probabilidades individuales de cada valor muestral. Así, a partir de la función de distribución de una variable *x,* se puede obtener fácilmente la probabilidad de cada muestra y su producto.

Puesto que la función de verosimilitud refleja la probabilidad que cada parámetro tiene de ser el generador de los datos muestrales observados, el mejor estimador será el resultado de maximizar dicha función de verosimilitud. De cara a simplificar el cálculo matemático, los estimadores máximo-verosímiles (EMV) se suelen obtener maximizando el logaritmo natural de la función de verosimilitud en lugar de la propia función de verosimilitud. En principio, el método de estimación máximo-verosímil puede ser empleado sin ninguna modificación cuando la probabilidad de un valor faltante no depende de los propios valores de la variable que lo contiene ni de los de otras variables que intervienen en el análisis, es decir, los datos perdidos son *MCAR*. En este texto emplearemos un refinado método de modelado de fdp basado en la estimación de máxima verosimilitud que se denomina algoritmo EM (*Expectation-Maximization*), con el que seremos capaces de tratar con eficacia datos perdidos con mecanismos tanto *MAR* como *MCAR*.

1.7. Estructura del proyecto

La estructura de este PFC viene condicionada por la necesidad de contextualizar al lector antes de presentar los problemas específicos planteados. De este modo, en el capítulo 1 hemos presentado la motivación y objetivos de este trabajo y hemos presentado los conceptos fundamentales de aprendizaje máquina y el problema de los datos incompletos.

En el capítulo 2 estudiaremos el problema de la estimación de la función de densidad de probabilidad de un conjunto de datos. Comenzaremos repasando algunos métodos sencillos para realizar esta estimación, pasando posteriormente a estudiar de forma detallada el potente método que emplearemos para este trabajo, el algoritmo EM. Revisaremos tanto los fundamentos teóricos como los detalles de implementación del mismo, así como posibles variantes.

Posteriormente, en el capítulo 3 analizaremos el funcionamiento de las redes neuronales. Empezaremos revisando los fundamentos de estas redes para pasar a analizar una estructura novedosa más compleja, la arquitectura de red neuronal para aprendizaje

multitarea. Seguiremos profundizando más en MTL en el capítulo 4, con el estudio de las redes multitarea para conjuntos de datos incompletos. Estas redes extienden la aplicación del aprendizaje multitarea a problemas de clasificación con valores de entrada incompletos, pero presentan algunos problemas para ciertos tipos de distribuciones de los datos. Pues bien, terminaremos este capítulo proponiendo una solución a estos problemas mediante la inclusión en el esquema MTL de la información proporcionada por un algoritmo EM.

En el capítulo 5 explicaremos los experimentos realizados en dos direcciones. Por un lado, veremos las posibles aplicaciones y limitaciones del algoritmo EM de forma general. Por otro lado, analizaremos la respuesta de nuestro modelo híbrido de EM y MTL a los problemas en los que habíamos detectado dificultades con la aplicación de ambos por separado. Finalizaremos en el capítulo 6 reflexionando sobre el proyecto y sus resultados, y exponiendo las líneas de trabajo futuras.

Capítulo 2

Funciones de densidad de probabilidad y algoritmo EM

En este capítulo consideraremos el problema del modelado de la función de densidad de probabilidad $p(\mathbf{x})$, dado un número finito de puntos de datos $\mathbf{x}^{(n)}, n = 1, ..., N$ obtenidos a partir de esa función de densidad. Los métodos que describiremos pueden ser usados para construir sistemas clasificadores considerando cada una de las clases C_k a su vez, y estimando las densidades condicionales para la clase correspondientes $p(\mathbf{x}/C_k)$ haciendo uso del hecho de que cada punto de los datos esta etiquetado de acuerdo con su clase. Estas densidades pueden entonces ser usadas en el teorema de de Bayes para obtener las probabilidades a posteriori correspondientes a una nueva medida de **x**, que puede ser usada a su vez para realizar una clasificación de **x**. La estimación de densidad también puede ser aplicada a datos no etiquetados (es decir, datos sin ninguna etiqueta de clase) donde tiene varias aplicaciones. En este trabajo, proponemos su aplicación para obtener la distribución de datos en el espacio de entrada como una parte del proceso de entrenamiento para una RNA basada en MTL.

Consideraremos tres enfoques alternativos para estimar la densidad de probabilidad. El primero de estos involucra *métodos paramétricos* en los que se asume una forma

funcional específica para el modelo de densidad. Este modelo contiene un cierto número de parámetros, que son optimizados durante el entrenamiento de éste, usando el conjunto de datos de entrada. El inconveniente de este enfoque es que la forma particular de la función paramétrica elegida podría ser incapaz de proveer una buena representación de la "verdadera" densidad. En contraste con este método, la técnica de *estimación no-paramétrica* no asume ninguna forma funcional particular, sino que permite que la forma de la función de densidad sea determinada enteramente por los datos. Tales métodos típicamente sufren el problema de que el número de parámetros en el modelo crece con el tamaño del conjunto de datos, de forma que el modelo puede volverse rápidamente inmanejable. El tercer enfoque, generalmente conocido como *estimación semiparamétrica*, intenta combinar las ventajas de los dos enfoques ya que el número de parámetros adaptativos puede ser incrementado de una manera sistemática para construir modelos cada vez más flexibles, pero donde el número total de parámetros en el modelo puede variar independientemente del tamaño del conjunto de datos. En este trabajo emplearemos un método semiparamétricos basados en modelos de mezclas de gaussianas (GMM). Queremos resaltar que el modelado preciso de densidades de probabilidad a partir de conjuntos finitos de datos en espacios de alta dimensionalidad (donde alta puede ser tan bajo como 10) es, en general, extremadamente difícil.

2.1. Probabilidad a priori, densidad de probabilidad y probabilidad a posteriori.

2.1.1 Probabilidad a priori

La probabilidad clásica, es también conocida como probabilidad a priori, debido a que si utilizamos ejemplos previsibles como monedas no alteradas, dados no cargados y mazos de barajas normales, entonces podemos establecer la respuesta de antemano, sin necesidad de lanzar una moneda, un dado o tomar una carta. No tenemos que efectuar experimentos para poder llegar a conclusiones.

Este planteamiento de la probabilidad presenta serios problemas cuando intentamos aplicarlo a problemas de toma de decisiones menos previsibles. El planteamiento clásico es

erróneo, ya que asume que no existen situaciones que son bastante improbables pero que podemos concebir como reales.

Dadas *c* clases, denotaremos la probabilidad a priori por $P(C_k)$, y expresaremos con ella la fracción de observaciones que pertenecerán a la clase C_k.

2.1.2 Densidad de probabilidad

Una función de densidad de probabilidad *p(x)* especifica que la probabilidad de que la variable *x* tome valores dentro del intervalo entre dos puntos $x = a$ y $x = b$ es dada por:

$$P(x \in [a,b]) = \int_a^b p(x)dx \tag{2.1}$$

La función *p(x)* es normalizada de manera que $P(x \in [a,b]) = 1$ si el intervalo $[a,b]$ corresponde al espacio de definición de *x* entero. Nótese que usaremos letras mayúsculas para probabilidades y letras minúsculas para densidades de probabilidad.

2.1.3 Probabilidad a posteriori

Supongamos conocidas $P(C_k)$ y $p(x|C_k)$ y que además disponemos de una medida de *x*. La pregunta clave es: ¿cómo influye este conocimiento sobre la decisión acerca de la clase a asignar?

Probabilidad a priori y densidad de probabilidad (verosimilitud) se combinan en la ***Regla de Bayes***, que indica cómo el valor observado, *x*, modifica la decisión basada en C_k, a través de $p(x|C_k)$ introduciendo la ***probabilidad a posteriori***, $P(C_k|x)$ que se interpreta como *la probabilidad de que la clase cierta sea* C_k *dado que el valor observado es x.*

$$P(C_k|x) = \frac{p(x|C_k)P(C_k)}{p(x)} \tag{2.2}$$

donde la densidad incondicional *p(x)* es dada por

$$p(x) = \sum_{k=1}^{c} p(x|C_k)P(C_k) \tag{2.3}$$

lo que asegura que las probabilidades a posteriori suman la unidad

$$\sum_{k=1}^{c} p(x \mid C_k) = 1 \tag{2.4}$$

En la práctica, es usual modelar las densidades condicionales de clase $p(x|C_k)$ mediante formas funcionales parametrizadas. Cuando éstas son funciones de los parámetros nos referiremos a ellas como funciones de *verosimilitud*, para el valor observado de *x*. El teorema de Bayes puede entonces expresarse de la forma

$$posterior = \frac{verosimilitud \times prior}{factor\ de\ normalización} \tag{2.5}$$

2.2. Métodos paramétricos

2.2.1 La distribución normal o gaussiana

La función de densidad normal o gaussiana es ampliamente usada en multitud de áreas científicas, y será la base sobre la que construiremos nuestro modelo de representación de las funciones de densidad de probabilidad que estimaremos. Para el caso de una única variable la podemos escribir de la forma

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{2.6}$$

donde μ y σ^2 son llamados media y varianza respectivamente. Si generalizamos a un espacio *d*-dimensional, la densidad normal queda

$$p(x) = \frac{1}{(2\pi)^{d/2}\left|\Sigma\right|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\} \tag{2.7}$$

donde μ es ahora un vector *d*-dimensional, Σ es una matriz de covarianza *d* x *d*, y $\left|\Sigma\right|$ es el determinante de Σ.

El amplio uso de la función normal o gaussiana, se debe a sus numerosas propiedades importantes para la estimación de densidad paramétrica o semiparamétrica, algunas de las cuales son:

1. Tiene propiedades analíticas relativamente simples que permiten que muchos resultados útiles puedan ser obtenidos explícitamente. Por ejemplo, cualquier momento de la distribución puede ser expresado como función de μ y de Σ.
2. El *teorema central del límite* afirma que, bajo circunstancias bastantes generales, la media de *M* variables aleatorias tiende a distribuirse normalmente, en el límite cuando *M* tiende a infinito. La condición principal es que la varianza de ninguna variable domine sobre el resto. Una aplicación común es la suma de un conjunto de variables obtenidas independientemente de la misma distribución. En la práctica, la convergencia tiende a ser bastante rápida, de modo que para valores de *M* tan pequeños como 10 la aproximación a una distribución normal puede ser muy buena. Podríamos esperar que las medidas de los fenómenos que ocurren naturalmente y están constituidos por varias componentes, sigan una distribución que se parezca a la normal.
3. Las densidades marginales (obtenidas integrando algunas de las variables) y las condicionales (obtenidas fijando alguna de las variables) de cualquier distribución normal también siguen una distribución normal.
4. Bajo cualquier transformación lineal no singular del sistema de coordenadas, la distancia de Mahalanobis mantiene su forma cuadrática y permanece definida positiva. De este modo, tras la transformación la distribución seguirá siendo normal pero con distinta media y varianza.

En la práctica, la razón principal para elegir una distribución normal es usualmente su simplicidad analítica.

2.2.2 Máxima verosimilitud

Una vez escogida una forma paramétrica para la función de densidad $p(x)$, la siguiente etapa es calcular los parámetros que definen la función que aproxima *p(x)* a partri del conjunto de datos de entrada. Existen dos métodos principales para resolver este problema, conocidos respectivamente como *máxima verosimilitud* (“maximum likelihood”) e *inferencia Bayesiana*. Aunque ambos métodos conducen a menudo a resultados similares, su base conceptual es bastante diferente. El método de máxima verosimilitud trata de

encontrar los valores óptimos de los parámetros maximizando una función de verosimilitud derivada del conjunto de entrenamiento. En cambio, en el método Bayesiano los parámetros son descritos por una distribución de probabilidad. Ésta es inicialmente fijada a una distribución a priori, que es entonces convertida a una distribución posteriori, a través del teorema de Bayes, una vez que los datos han sido observados. La expresión final para la densidad de probabilidad deseada de las variables de entrada es dada por una integral sobre todos los posibles valores de los parámetros, ponderados por sus distribuciones a posteriori. Nótese que el enfoque Bayesiano no implica fijar los parámetros a valores específicos, a diferencia del método de máxima verosimilitud.

A continuación, repasaremos más profundamente el método de máxima verosimilitud, ya que nuestro trabajo se fundamenta en él. Igualmente, restringiremos el análisis al caso de la función de densidad normal.

Supongamos una función de densidad $p(\mathbf{x})$ que depende de un conjunto de parámetros $\boldsymbol{\theta} = (\theta_1, ..., \theta_M)$. Para conseguir una dependencia con los parámetros explicita, escribiremos la función de densidad de la forma $p(\mathbf{x} \mid \boldsymbol{\theta})$. También tenemos un conjunto de N vectores $\mathbf{X} = \left\{ \mathbf{x^{(1)}}, \mathbf{x^{(2)}}, ..., \mathbf{x^{(n)}}, ..., \mathbf{x^{(N)}} \right\}$. Si estos vectores son obtenidos independientemente de la distribución $p(\mathbf{x} \mid \boldsymbol{\theta})$, entonces la densidad de probabilidad conjunta del conjunto completo de datos es dado por

$$p(\mathbf{X} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{x^{(n)}} \mid \boldsymbol{\theta}) \equiv L(\boldsymbol{\theta}) \tag{2.8}$$

donde $L(\boldsymbol{\theta})$ puede verse como una función de $\boldsymbol{\theta}$ para un $\mathbf{X}$ fijado, en cuyo caso se le denomina como verosimilitud de $\boldsymbol{\theta}$ para el conjunto $\mathbf{X}$. La técnica de máxima verosimilitud calcula el valor de $\boldsymbol{\theta}$ maximizando $L(\boldsymbol{\theta})$. Esto se corresponde con la intuitiva y razonable idea de elegir el $\boldsymbol{\theta}$ que es más probable que haya hecho surgir los datos observados. En la práctica suele ser más conveniente considerar el logaritmo negativo de la verosimilitud

$$E = -\ln L(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \ln p(\mathbf{x^{(n)}} \mid \boldsymbol{\theta}) \tag{2.9}$$

y encontrar un mínimo de E. Esto es equivalente a maximizar L ya que el logaritmo negativo es una función monótonamente decreciente. El logaritmo de la verosimilitud negativo puede ser visto como una función de error, como veremos posteriormente. Llegados a la expresión (2.9), bastará con seguir un proceso numérico iterativo para obtener el $\boldsymbol{\theta}$ óptimo. Además, para el caso especial de la densidad normal multivariante, podemos encontrar la solución de máxima verosimilitud por diferenciación analítica de (2.9), con $p(\mathbf{x}|\boldsymbol{\theta})$ dado por (2.7).

2.3. Métodos no paramétricos

En esta sección consideraremos algunas de las técnicas no paramétricas más importantes para la estimación de densidad de probabilidad. El término "no paramétrico" se usa para describir funciones de densidad de probabilidad cuya forma no es conocida de antemano, sino que depende de los propios datos.

2.3.1 Histogramas

El problema básico de la estimación de densidad no paramétrica es muy simple. Dado un conjunto de datos, se pretende modelar la distribución de probabilidad que generó los datos, sin asumir previamente ninguna forma determinada para la función de distribución. Una solución muy sencilla es el uso de histogramas. El histograma se obtiene dividiendo un espacio de entrada en una serie de intervalos, y aproximando la densidad en cada valor del eje mediante la fracción de muestras que caen dentro del intervalo al que pertenece. Este proceso representa una forma simple de estimación de densidad no paramétrica.

En la figura 2-1 mostramos un ejemplo simple de estimación de densidad usando el método del histograma. En rojo pintamos la función de densidad original que generó los datos, y en azul el histograma. Como se puede apreciar, es tarea nuestra elegir el punto de inicio en el eje y el número de intervalos. Los resultados normalmente no son demasiado sensibles a la posición de inicio, pero el parámetro M juega un papel crucial. La figura 2-1 muestra los histogramas con resultados a partir de valores de M de 5, 10 y 25. Observamos que el número de intervalos (o de manera más precisa, la amplitud del intervalo) actúa

como parámetro de suavizamiento. Si la amplitud del intervalo es demasiado pequeña entonces la densidad estimada es muy picuda, mientras que si es demasiado grande estaremos perdiendo la verdadera estructura de densidad.

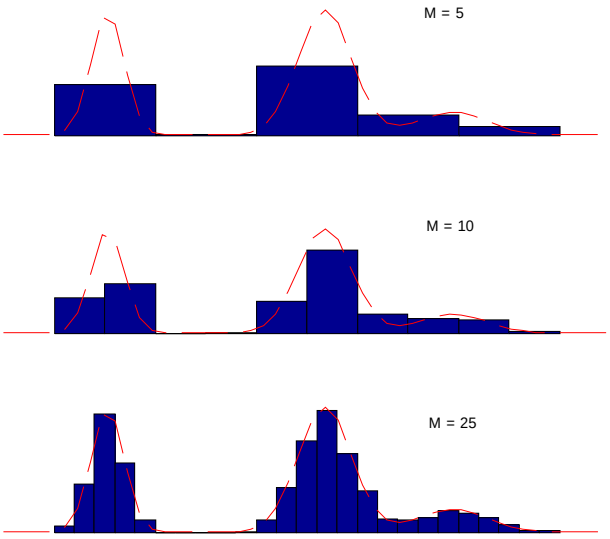


Figura 2-1 Estimación de fdp mediante histogramas, e influencia del parámetro *M*.

El histograma debe tomarse como una herramienta rápida que nos permita obtener una primera visualización de los datos en una o dos dimensiones. Cuando la dimensionalidad de los datos aumenta, el uso de histogramas no es eficiente. Si para una dimensión necesitamos M intervalos, para un espacio de características d-dimensional necesitaremos M^d intervalos, y este crecimiento exponencial lo convierte en inviable.

2.3.2 Métodos tipo Kernel

La probabilidad de que un nuevo vector **x**, obtenido a partir de una función de densidad desconocida $p(\mathbf{x})$ caiga dentro de alguna región R del espacio-**x** es, por definición, dada por

$$P = \int_R p(\mathbf{x}')d\mathbf{x}' \tag{2.10}$$

Si tenemos N muestras obtenidas independientemente a partir de $p(\mathbf{x})$ entonces la probabilidad de que K de ellas caigan en la región R es dada por la ley binomial

$$P(K) = \frac{N!}{K!(N-K)!} P^K (1-P)^{N-K} \tag{2.11}$$

De este modo, esperamos que se pueda obtener una buena estimación de la probabilidad *P* a partir de la fracción media de muestras que caen en *R*, de forma que

$$P \approx K / N \tag{2.12}$$

Si asumimos que ***p(x)*** es continua y que no varía apreciablemente sobre la región *R*, entonces podemos aproximar (2.10) por

$$P = \int_R p(\mathbf{x}')d\mathbf{x}' \approx p(\mathbf{x})V \tag{2.13}$$

donde V es el volumen de *R*, y ***x*** es algún punto que caiga dentro de *R*. De (2.12) y (2.13) obtenemos el resultado intuitivo

$$p(\mathbf{x}) \approx \frac{K}{NV} \tag{2.14}$$

Atendiendo a la estimación de densidad obtenida en (2.14) podemos adoptar dos métodos básicos. El primero consiste en elegir un valor fijo para *K* y determinar el volumen correspondiente *V* de los datos. Alternativamente podemos fijar el volumen *V* y determinar *K* a partir de los datos. Esto nos lleva a las *técnicas de estimación de densidad tipo Kernel*, que describimos a continuación.

Supongamos una región *R* que es un hipercubo con lados de longitud *h* centrados en el punto *x*. Entonces su volumen es proporcionado por

$$V = h^d \tag{2.15}$$

Podemos encontrar una expresión para *K*, el numero de muestras que caen en esta región, definiendo una función Kernel *H(***u***)*, también conocida como *Ventana de Parzen* dada por

$$H(\mathbf{u}) = \begin{cases} 1 & \left|u_j < 1/2\right| \quad j = 1,...,d \\ 0 & resto \end{cases}. \tag{2.16}$$

De este modo, *H(***u***)* se corresponde con un cubo unidad centrado en el origen. Por lo tanto, para cada muestra $\mathbf{x}^{(n)}$, la cantidad $H((\mathbf{x}-\mathbf{x}^{(n)})/h)$ es igual a la unidad si el punto $\mathbf{x}^n$ cae dentro del hipercubo de lado *h* centrado en ***x***, y es cero si no es así. El número total de muestras que caen dentro del hipercubo es simplemente

$$K = \sum_{n=1}^{N} H\left(\frac{\mathbf{x}-\mathbf{x}^{(\mathbf{n})}}{h}\right). \tag{2.17}$$

Si sustituimos (2.17) y (2.15) en (2.14) obtenemos la siguiente estimación para la densidad en el punto **x**:

$$\tilde{p}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^d} H\left(\frac{\mathbf{x}-\mathbf{x}^{(\mathbf{n})}}{h}\right)\frac{1}{2} \tag{2.18}$$

donde $\tilde{p}(\mathbf{x})$ denota la densidad estimada mediante ventanas de Parzen. Esta estimación de densidad puede verse como la superposición de *N* cubos de lado *h*, con un hipercubo centrado en cada una de las muestras. Este método es similar al del histograma excepto porque en vez de intervalos tenemos hipercubos cuya posición está determinada por los datos. Sin embargo, todavía tenemos el problema de las discontinuidades. Para solucionarlo, una opción común es emplear una función kernel gaussiana, que queda:

$$\tilde{p}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{d/2}} \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}^{(\mathbf{n})}\right\|^2}{2h^2}\right) \tag{2.19}$$

En general, si las funciones de kernel satisfacen

$$H(\mathbf{u}) \geq 0 \tag{2.20}$$

y

$$\int H(\mathbf{u})d\mathbf{u} = 1 \tag{2.21}$$

entonces la estimación de $\tilde{p}(\mathbf{x})$ satisfará $\tilde{p}(\mathbf{x}) \geq 0$ y $\int \tilde{p}(\mathbf{x})d\mathbf{x} = 1$

2.3.3 Los K vecinos más próximos

Como comentábamos en el apartado anterior, para estimar la densidad de probabilidad a partir de (2.14) podemos tomar dos estrategias, fijar *V* e ir variando *K* o hacerlo a la inversa. Mientras que el método de las ventanas de Parzen hace uso de la primera estrategia, el método de los vecinos más próximos lo que se propone es, fijado un valor de *K*, ir aumentando el volumen de una hiperesfera centrada en cada muestra hasta que esta abarque a sus *K* vecinos más próximos.

Una desventaja de este método es que la estimación resultante no es una verdadera densidad de probabilidad, ya que la integral a lo largo de todo el espacio $\boldsymbol{x}$ diverge. Sin embargo, como ventaja cabe destacar la posibilidad de emplear el algoritmo de los *K*-vecinos más próximos para clasificar, asignando a una muestran la clase a la cual pertenezca un número mayor de sus vecinos.

Una desventaja común al método de kernel y al de los vecinos más próximos es que es necesario almacenar todas las muestras del conjunto de entrenamiento, y esto podría provocar problemas de almacenamiento así como lentitud a la hora de evaluar la densidad de nuevas muestras.

2.4. Mezcla de gaussianas

Hasta ahora hemos considerado dos enfoques diferentes a la estimación de densidad, paramétrico y no paramétrico, cada uno de los cuales tiene sus meritos y limitaciones. En particular el enfoque paramétrico asume una forma específica para la función de densidad, que podría ser muy diferente de la verdadera densidad. Sin embargo, los modelos paramétricos permiten que la función de densidad sea evaluada muy rápidamente para nuevos valores del vector de entrada. Los métodos paramétricos, en cambio, permiten formas muy generales de función de densidad, pero sufre el hecho de que el número de variables en el modelo crece directamente con el número de puntos de entrenamiento. Esto nos lleva a modelos que pueden ser muy lentos evaluando los nuevos vectores.

Para combinar las ventajas tanto de los métodos paramétricos como no paramétricos necesitamos encontrar técnicas que no estén restringidas a una forma funcional específica, y en las que a su vez, el tamaño del modelo crezca solamente con la complejidad del problema a resolver, y no simplemente con el tamaño del conjunto de datos. Esto nos conduce a una clase de modelos que llamaremos *semiparamétricos*. La desventaja que presenta es el entrenamiento del modelo, ya que es computacionalmente intensivo comparado con los procedimientos simples empleados para los métodos paramétricos y no paramétricos.

Nosotros nos centraremos en una forma particular de forma de función de densidad, llamada *modelo de mezclas*. Además de proporcional potentes técnicas para la estimación de densidad, los modelos de mezclas encuentran importantes aplicaciones en el contexto de las redes neuronales y en técnicas para la estimación de densidades condicionales como veremos más adelante. Como veremos posteriormente, existen varios métodos para estimar los parámetros de un modelo de mezclas. El método que usaremos para este trabajo es el algoritmo EM, que está basado en la estimación de máxima verosimilitud. Su elección se debe a su gran eficacia para modelar funciones de densidad, así como a su capacidad de asimilar los valores perdidos.

En el método no paramétrico basado en kernel para estimar la densidad, la función de densidad está representada como una superposición lineal de funciones de kernel, con una función de kernel centrada en cada muestra. Ahora, consideraremos modelos en los que la función de densidad también está formada por una combinación lineal de funciones base, pero en la que el número *M* de funciones base es tratado como un parámetro del modelo y es típicamente mucho menor que el número *N* de muestras. Por consiguiente escribiremos el modelo de mezclas como una combinación lineal de las densidades componentes $p(\mathbf{x} \mid w_j)$ de la forma

$$p(\mathbf{x}) = \sum_{j=1}^{M} P(\mathbf{x} \mid w_j) P(w_j) \tag{2.22}$$

Tal representación es denominada una distribución de mezcla y los coeficientes $P(w_j)$ son los parámetros de mezcla. Llamaremos a $P(w_j)$ la probabilidad a priori de que una muestra haya sido generada por la componente *j*-ésima de la mezcla. Estas probabilidades a priori han sido seleccionadas para cumplir las restricciones

$$\sum_{j=1}^{M} P(w_j) = 1 \tag{2.23}$$

$$0 \le P(w_j) \le 1 \tag{2.24}$$

Además, las funciones de densidad de las componentes $p(\mathbf{x} \mid w_j)$ son normalizadas de forma que

$$\int p(\mathbf{x} \mid w_j) d\mathbf{x} = 1 \tag{2.25}$$

y por lo tanto puede ser vistas como densidades condicionales de clase. Para generar una muestra a partir de la distribución de probabilidad (2.22), seleccionamos aleatoriamente primero una de las componentes w_j con probabilidad $P(w_j)$, y entonces generamos una muestra con la correspondiente densidad de componente $p(\mathbf{x}|w_j)$. Una propiedad importante de los modelos de mezclas es que, para muchos tipos de funciones de densidad escogidas para las componentes, pueden aproximar cualquier densidad continua con una precisión determinada, siempre que el modelo tenga un número suficientemente grande de componentes y que los parámetros del modelo sean elegidos correctamente.

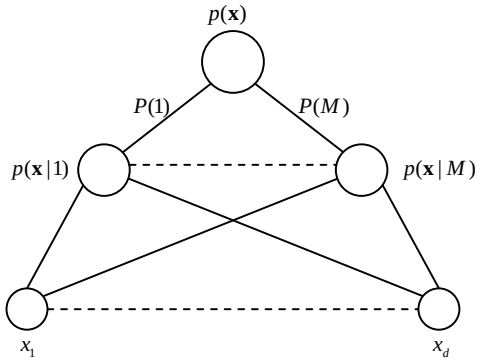


Figura 2-2 Representación del modelo de mezclas (2.22) en forma de diagrama de red. Para las densidades de las componentes gaussianas, $p(\mathbf{x}|w_j)$ dadas por (2.26), las líneas que conectan las entradas x_i con las componentes $p(\mathbf{x}|w_j)$ representan los elementos μ_{ji} de los correspondientes vectores de medias μ_j.

En un problema real de clasificación, la técnica del modelado de mezclas se puede aplicar separadamente a cada clase C_k. De esta forma, cada densidad condicional de clase $p(\mathbf{x}|w_j)$ se representa por un modelo independiente de mezclas de la forma (2.22)

Establecido el enlace con las probabilidades a priori y las densidades condicionales, podemos introducir las correspondientes probabilidades a posteriori, que podemos expresar usando el teorema de Bayes de la forma

$$p(w_j|\mathbf{x}) = \frac{p(\mathbf{x}|w_j)P(w_j)}{p(\mathbf{x})} \tag{2.26}$$

donde *p(**x**)* es dado por (2.22). Estas probabilidades a posteriori satisfacen

$$\sum_{j=1}^{M} P(w_j \mid \mathbf{x}) = 1. \tag{2.27}$$

El valor de $p(\mathbf{x} \mid w_j)$ representa la probabilidad de que una componente particular w_j sea responsable de generar una muestra **x**.

Particularizando para los modelos de mezclas en los que las densidades de cada componente vienen dadas por funciones de distribución gaussianas, nos podemos encontrar con dos formas comunes de expresar la mezcla. La primera de estas representaciones se emplea para problemas poco complejos y en los que se pretende obtener un algoritmo más rápido, y consiste en asumir que las matrices de covarianza son un múltiplo escalar de la matriz identidad de modo que $\Sigma_j = \sigma_j^2 I$ (donde I es la matriz identidad) y por lo tanto

$$p(\mathbf{x}) = \sum_{j=1}^{M} \frac{1}{(2\pi)^{d/2}\sigma_j} \exp\left\{-\frac{\left\|\mathbf{x}-\boldsymbol{\mu}_\mathbf{j}\right\|^2}{2\sigma_j^2}\right\} P(w_j). \tag{2.28}$$

La segunda de las formas de expresar la mezcla de gaussianas es su expresión natural, en la que no se fuerza la matriz de covarianza a una estructura diagonal. Esta expresión es más general y permite la representación de funciones más complicadas. Como contrapartida su coste computacional es mucho mayor. Ya que nuestro objetivo es obtener una estimación precisa de la densidad, emplearemos

$$p(\mathbf{x}) = \sum_{j=1}^{M} \frac{1}{(2\pi)^{d/2}\left|\boldsymbol{\Sigma}_\mathbf{j}\right|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_\mathbf{j})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_\mathbf{j})\right\} P(w_j) \tag{2.29}$$

donde $\left|\boldsymbol{\Sigma}_\mathbf{j}\right|$ y $\boldsymbol{\Sigma}^{-1}$ son respectivamente es el determinante y la inversa de la matriz de covarianza de la componente w_j de la mezcla.

2.4.1 Datos incompletos

El enfoque del modelado de mezclas nos permite el aprendizaje a partir de conjuntos de datos con entradas incompletas. Aprender según este enfoque es un problema clásico de estimación que requiere un modelo probabilístico explícito y un algoritmo para estimar los parámetros del modelo.

Los modelos de mezclas han sido usados recientemente para problemas de aprendizaje supervisado en la forma de arquitectura de "mezclas de expertos" [22]. Esta arquitectura es un modelo de regresión paramétrica con una estructura modular similar al árbol de decisión no-paramétrico y a los modelos de tira adaptativa [23]. El enfoque que presentaremos en este texto difiere de estos enfoques basados en regresión en que el objetivo del aprendizaje es estimar la densidad de los datos. No realizaremos distinción entre variables de entrada y de salida; la densidad conjunta es estimada y esta estimación es usada para formar una correspondencia entrada/salida. Enfoques similares de estimación de densidad han sido tratados por Tresp [25], entre otros, para modelos de mezclas de gaussianas.

La característica más llamativa de los modelos de mezclas en el contexto de este PFC es que pueden tratar con datos incompletos. De hecho, el mismo problema de estimar densidades de mezclas puede se visto como un problema de datos con pérdidas (las "etiquetas" para las densidades de las componentes están perdidas, como veremos en el apartado 2.5.6) y podemos desarrollar un algoritmo de Esperanza-Maximización (*Expectation-Maximization*) [26] para manejar ambas clases de datos con pérdidas.

2.4.2 Estimación de parámetros

En el apartado 2.2.2 veíamos la forma de obtener los parámetros de una distribución gaussiana de múltiples variables la estimación de máxima verosimilitud. En el caso de las mezclas de gaussianas, si tuviésemos una etiqueta que indicase que componente generó cada dato, o en su defecto, la proporción en la que se encuentran las componentes en la mezcla, nos encontraríamos igualmente ante un caso fácil de estimación de parámetros.

Desgraciadamente, no es habitual disponer de esta información para estimar los parámetros de la mezcla. A esto se le conoce como el problema de la estimación de densidad de la mezcla.

Para estimar los parámetros de la mezcla, nos centraremos en el método de máxima verosimilitud, siendo, de lejos, la técnica más popular para estimar parámetros. Anteriormente, para la función gaussiana multivariante, definíamos la verosimilitud sobre un conjunto de datos de la siguiente forma:

$$L(\boldsymbol{\theta}) = p(\mathbf{X} \mid \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{x^{(n)}} \mid \boldsymbol{\theta}) \tag{2.30}$$

donde $\boldsymbol{\theta}$ era el conjunto de parámetros que, en el caso de las mezclas de gaussianas multidimensionales son el vector de media $\boldsymbol{\mu}$ y a la matriz de covarianza $\boldsymbol{\Sigma}$. Ahora debemos redefinir el significado de $\boldsymbol{\theta}$ para adaptarlo a la mezcla de gaussianas. De este modo queda $\boldsymbol{\theta} = \{\boldsymbol{\mu_1}, ..., \boldsymbol{\mu_M}, \boldsymbol{\Sigma_1}, ..., \boldsymbol{\Sigma_M}, P(w_1), ..., P(w_M)\}$ donde recordemos que $P(w_j)$ son las proporciones de cada componente dentro de la mezcla.

Asumamos que $\hat{\boldsymbol{\theta}}$ es el conjunto de parámetros para el cual $L(\boldsymbol{\theta})$ alcanza su máximo como función de $\boldsymbol{\theta}$ para un conjunto de datos $\mathbf{X}$. Entonces la estimación de máxima verosimilitud (*Maximum Likelihood Estimation*, MLE) del conjunto de parámetros es la elección del conjunto de parámetros que maximiza la función de densidad de probabilidad de los datos, llamada función de verosimilitud. Intuitivamente, la MLE es una elección razonable como estimador, ya que nos lleva a un conjunto de parámetros para el que la muestra observada es más probable.

Como comentábamos anteriormente, usualmente es conveniente tratar con el logaritmo de la función de verosimilitud más que con la propia función. De este modo

$$\ln L(\boldsymbol{\theta}) = \ln p(\mathbf{X} \mid \boldsymbol{\theta}) = \ln \prod_{n=1}^{N} p(\mathbf{x^{(n)}} \mid \boldsymbol{\theta}) = \sum_{n=1}^{N} \ln p(\mathbf{x^{(n)}} \mid \boldsymbol{\theta}) \tag{2.31}$$

Hay dos inconvenientes asociados con la estimación de máxima verosimilitud. El primer problema es el de encontrar el máximo global y verificar que realmente se ha encontrado un máximo global. Lo que significa que el conjunto de parámetros óptimo es aquel que, dentro

del dominio en que están definidos los parámetros (recordemos que el dominio está restringido por (2.23) y (2.24)), proporciona un máximo a la función de verosimilitud. Al definir un estimador de máxima verosimilitud, deben tenerse en cuenta dos dificultades prácticas asociadas con la estimación de máxima verosimilitud para densidades de mezclas.

La primera dificultad es que no siempre se puede considerar que el dominio de definición de los parámetros de la mezcla es un conjunto para el cual la función logaritmo de verosimilitud está acotada por arriba, y por lo tanto no siempre hay puntos del dominio en los que dicha función alcance un máximo. La segunda dificultad es que podemos tomar multitud de opciones para el dominio en las cuales la función logaritmo de verosimilitud alcanza sus máximos locales más grandes. Si tenemos $\boldsymbol{\theta}_i$ y $\boldsymbol{\theta}_j$ que pertenecen a la misma familia paramétrica para ciertas componentes *i* y *j*, entonces el valor de la función logaritmo de verosimilitud no cambiará si los parámetros $\boldsymbol{\theta}_i$ y $\boldsymbol{\theta}_j$ se intercambian. La preocupación principal es que la función logaritmo de verosimilitud puede (y lo hace a menudo) tener máximos locales. Desde el punto de vista computacional no se puede hacer mucho acerca de este problema, solo que se debe decidir si se acepta el máximo como global o si se prosigue la búsqueda de otros.

El segundo problema es el de la sensibilidad numérica. Es decir, como de sensible es la estimación a pequeños cambios en los datos. Algunas veces unas muestras sutilmente diferentes producen un conjunto completamente diferente de estimaciones de máxima verosimilitud.

El método tradicional para estimar la máxima verosimilitud consiste en preparar un sistema de ecuaciones que satisfagan la estimación de máxima verosimilitud e intentar resolverlo. Para encontrar las ecuaciones de verosimilitud consideraremos las derivadas parciales de la función logaritmo de verosimilitud con respecto de los componentes de $\boldsymbol{\theta}$.

Si $\hat{\boldsymbol{\theta}}$ es un estimador de máxima verosimilitud, entonces las ecuaciones de verosimilitud están determinadas por:

$$\nabla_{\boldsymbol{\theta}_j} L = 0, \quad j = 1, ..., M \tag{2.32}$$

Para obtener las ecuaciones de verosimilitud determinadas por las proporciones, que están restringidas para ser no negativas y de suma unidad, se sigue el método de Redner y Walker (1984), que nos lleva al siguiente resultado

$$\hat{P}(w_j) = \frac{1}{N} \sum_{n=1}^{N} P(w_j \mid \mathbf{x}^{(\mathbf{n})}) \tag{2.33}$$

Como vemos, el método de la estimación de máxima verosimilitud que hemos analizado es complicado y sufre inconvenientes de importancia. Además, aunque proporciona información sobre la naturaleza de la solución no proporciona un método directo para calcular los parámetros [21]. En realidad nos conduce a ecuaciones acopladas de alta no linealidad, ya que los parámetros a estimar se encuentran implícitamente en la parte derecha de la ecuación. Todo parece indicar que tendremos que buscar un método iterativo para encontrar los parámetros de máxima verosimilitud, lo que nos lleva al empleo del algoritmo EM.

2.5. El algoritmo EM

2.5.1 Introducción

El algoritmo de Esperanza-Maximización (EM) es un método ampliamente aplicado en la computación iterativa de la estimación de máxima verosimilitud (ML), útil en problemas de datos incompletos. En cada iteración del algoritmo EM, hay dos pasos llamados el *paso de esperanza (o expectación)* o paso-E y el *paso de maximización* o paso-M. Este algoritmo fue desarrollado, en 1977, por *Dempster, Laird* y *Rubin* en su artículo fundamental [26]. La situación en la que el algoritmo EM muestra toda su potencia es en los problemas de datos incompletos, donde la estimación de máxima verosimilitud resulta difícil debido a la ausencia de alguna parte de los datos dentro de una estructura de datos simple y familiar. El algoritmo EM está estrechamente relacionado con el método *ad-hoc* para la estimación con datos perdidos, donde los parámetros son estimados después de haber imputado los datos perdidos con valores determinados al inicio. Estos valores de imputados son actualizados por sus valores predichos usando estos estimadores iniciales de los parámetros. Los

parámetros son entonces reestimados, y así sucesivamente, procediendo iterativamente hasta la convergencia.

Las situaciones donde el algoritmo EM puede ser aplicado no incluyen solamente situaciones con datos incompletos, sino también una amplia variedad de situaciones donde la presencia de datos incompletos no es del todo natural o evidente. Éstas incluyen modelos estadísticos como efectos aleatorios, mezclas, convoluciones, modelos linearles logarítmicos, etc. Problemas de estimación hasta ahora intratables para estas situaciones han sido resueltos o se han simplificado procedimientos complicados de estimación de la verosimilitud usando el algoritmo EM. El algoritmo EM tiene, por lo tanto, aplicaciones en casi todos los contextos estadísticos y en casi todos los campos donde se hayan aplicado técnicas estadísticas: tratamiento de imágenes médicas, corrección de censos, epidemiología del SIDA, y entrenamiento de redes neuronales artificiales, entre otros.

La idea básica del algoritmo EM es asociar con un problema de datos incompletos dado, un problema de datos completos para el cual la estimación de máxima verosimilitud es computacionalmente más fácilmente abordable; por ejemplo, el problema de datos completos elegido puede producir una solución en forma cerrada de la MLE o puede ser tratable con computación MLE con un paquete de estándar de cálculo. La metodología del algoritmo EM consiste por lo tanto en reformular el problema en términos de un problema de datos completos más fácilmente resoluble, estableciendo relaciones entre la verosimilitud de estos dos problemas, y explotando el cálculo más simple de la MLE del problema de datos completos en el paso-M del algoritmo iterativo de cálculo.

Como ya hemos comentado anteriormente, el algoritmo EM consta de dos pasos: paso-E y paso-M. El paso-E consiste en generar datos para conseguir un problema de datos completos, usando el conjunto de datos observados del problema de datos incompletos y el valor actual de los parámetros, de modo que el cálculo del paso-M sea más simple al poder ser aplicado a este conjunto de datos completo y rectangular. Más concretamente, es logaritmo de la verosimilitud de este problema de datos completos lo que es fabricado en el paso-E. Como está basado parcialmente en datos no observados, es reemplazado pos sus

esperanzas condicionales dados los datos observados, donde este paso-E es efectuado usando el ajuste actual para los parámetros desconocidos. Comenzando desde unos valores iniciales adecuados para los parámetros, los pasos-E y los pasos-M son repetidos hasta la convergencia. A pesar de que el algoritmo EM ha sido aplicado con éxito en diversos contextos, puede ser tormentosamente lento en converger en ciertas situaciones. Esta debilidad ha fomentado el desarrollo de versiones modificadas del algoritmo [24].

2.5.2 Propiedades

A fin de dar al lector una rápida idea del potencial del algoritmo como una herramienta útil en problemas de estimación estadística, resumiremos a continuación las razones de su atractivo. También mencionaremos los inconvenientes que presenta. Algunas de las ventajas del algoritmo EM son las siguientes:

- Es numéricamente estable con cada iteración EM, es decir, en cada iteración aumenta la verosimilitud (excepto llegados al punto de convergencia).
- Bajo condiciones bastante generales, podemos confiar en que el algoritmo EM alcance la convergencia. Es decir, empezando desde un punto arbitrario $\boldsymbol{\theta}^{\mathbf{0}}$ en el espacio de los parámetros, acabaremos en la mayoría de los casos en un máximo local, exceptuando las elecciones de $\boldsymbol{\theta}^{\mathbf{0}}$ muy inadecuadas o alguna patología local de la función de verosimilitud.
- Es típicamente fácil de implementar, ya que se basa en cálculos de datos completos: el paso-E de cada iteración solamente implica tomar esperanzas sobre distribuciones condicionales de datos completos, y el paso-M de cada iteración simplemente requiere estimaciones de verosimilitud de datos completos, lo que a menudo es una forma cerrada.
- No entraña grandes dificultades en el proceso de programación, ya que no está implicada ninguna evaluación de la verosimilitud ni de sus derivadas.
- El algoritmo EM requiere poco espacio de almacenamiento y puede generalmente llevarse a cabo en un ordenador sencillo. Por ejemplo, no tiene que almacenar la matriz de información ni su inversa en ninguna iteración.

- Puesto que el problema de los datos completos es más o menos estándar, el paso-M puede llevarse a cabo con frecuencia usando paquetes estadísticos estándar en situaciones donde la estimación de máxima verosimilitud de datos completos no existe en una forma cerrada. En otras situaciones de este tipo, extensiones del algoritmo EM como GEM (*Greedy EM*) y los algoritmos de maximización de esperanza condicional (*Expectation Conditional Mean*, ECM) suelen permitir que el paso-M sea implementado de una manera relativamente simple. Además, estas extensiones comparten la convergencia monotónica estable del algoritmo EM.
- El trabajo analítico requerido es mucho más simple que con otros métodos, puesto que solamente necesita ser maximizada la esperanza condicional del logaritmo de la función de verosimilitud para el problema de datos completo. Aunque se puede necesitar una cierta cantidad de trabajo analítico para desarrollar el paso-E, no es complicado en muchas aplicaciones.
- El coste por iteración es generalmente bajo, lo que puede compensar el mayor número de iteraciones necesitadas por el algoritmo EM comparado con otros procedimientos competitivos.
- Observando el incremento monotónico de la verosimilitud (si se puede evaluar fácilmente) a lo largo de las iteraciones, es fácil monitorear la convergencia y los errores de programación.
- El algoritmo EM puede usarse para proporcionar estimaciones de los valores de los datos perdidos.

Algunas de las desventajas que sufre el algoritmo EM son las siguientes:

- No tiene un procedimiento incluido para proporcionar una estimación de la matriz de covarianza de las estimaciones de los parámetros. De cualquier modo, esta desventaja puede ser eliminada fácilmente empleando una metodología adecuada asociada con el algoritmo EM.
- Puede converger de forma desesperadamente lenta en problemas donde hay demasiada información incompleta e incluso en problemas aparentemente inocuos.

- Como los métodos tipo Newton, no garantiza convergencia al máximo global cuando hay múltiples máximos. Además, en este caso, la estimación obtenida depende esencialmente del valor inicial. Sin embargo, en general, ningún algoritmo de optimización garantiza la convergencia a un máximo global. Existen otros procedimientos como el temple simulado (*simulated annealing*) para afrontar tales situaciones. De cualquier modo, son complicados de aplicar.
- En algunos problemas, el paso-E puede ser analíticamente intratable, aunque en estas situaciones aún nos queda la posibilidad de efectuarlo mediante el método Monte Carlo, que no trataremos en este texto.

2.5.3 El algoritmo

Supongamos un conjunto incompleto (algunas características de algunos patrones se han perdido) de datos $\mathbf{X}$, asumiremos que se puede descomponer en dos componentes, la componente observada $\mathbf{X^o}$ y la componente perdida $\mathbf{X^m}$. Especificaremos con $p(\mathbf{X}|\boldsymbol{\theta})$ la función de densidad de probabilidad que generó los datos. De este modo podemos escribir:

$$p(\mathbf{X}|\boldsymbol{\theta}) = p(\mathbf{X^o}, \mathbf{X^m}|\boldsymbol{\theta}) = p(\mathbf{X^m}|\mathbf{X^o}, \boldsymbol{\theta})\, p(\mathbf{X^o}|\boldsymbol{\theta}) \tag{2.34}$$

si tomamos logaritmos para calcular la verosimilitud quedará como sigue:

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \ln p(\mathbf{X^m}|\mathbf{X^o}, \boldsymbol{\theta}) + \ln p(\mathbf{X^o}|\boldsymbol{\theta}) \tag{2.35}$$

nos interesa optimizar los parámetros respecto a la parte observada, por lo que la despejaremos

$$\ln p(\mathbf{X^o}|\boldsymbol{\theta}) = \ln p(\mathbf{X}|\boldsymbol{\theta}) - \ln p(\mathbf{X^m}|\mathbf{X^o}, \boldsymbol{\theta}) \tag{2.36}$$

teniendo en cuenta que

$$L(\boldsymbol{\theta}|\mathbf{X}) = p(\mathbf{X}|\boldsymbol{\theta}) \tag{2.37}$$

y si sustituimos $\ln L(\boldsymbol{\theta}|\mathbf{X})$ por $l(\boldsymbol{\theta}|\mathbf{X})$ nos queda

$$l(\boldsymbol{\theta}|\mathbf{X^o}) = \ln p(\mathbf{X}|\boldsymbol{\theta}) - \ln p(\mathbf{X^m}|\mathbf{X^m}, \boldsymbol{\theta}) \tag{2.38}$$

El algoritmo EM pretende encontrar un valor de $\boldsymbol{\theta}$ que maximice $p(\mathbf{X^o}|\boldsymbol{\theta})$ dada una observación $\mathbf{X^o}$. La función $p(\mathbf{X^m}|\mathbf{X^o}, \boldsymbol{\theta})$ juega un papel importante, ya que relaciona la función de probabilidad del conjunto total con la de la componente observada. Además, fijados los parámetros permite estimar un valor de $\mathbf{X^m}$, y fijado $\mathbf{X^m}$ nos permite verificar como de bien se ajustan los parámetros. Puesto que no conocemos los valores de la

componente perdida, será preciso calcular esperanzas para nuestras funciones, y esto se logra mediante un promediado:

$$l(\boldsymbol{\theta} \mid \mathbf{X^o}) = \int \ln p(\mathbf{X^o}, \mathbf{X^m} \mid \boldsymbol{\theta})\, p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta}) d\mathbf{X^m} - \int \ln p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta})\, p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta}) d\mathbf{X^m} \quad (2.39)$$

para una mayor claridad definiremos las funciones $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}})$ y $H(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}})$

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) = E\left\{\ln p(\mathbf{X} \mid \boldsymbol{\theta}) \mid \mathbf{X^o}, \boldsymbol{\theta}^{\mathbf{k}}\right\} = \int \ln p(\mathbf{X} \mid \boldsymbol{\theta})\, p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta}^{\mathbf{k}}) d\mathbf{X^m} \quad (2.40)$$

$$H(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) = E\left\{\ln p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta}) \mid \mathbf{X^o}, \boldsymbol{\theta}^{\mathbf{k}}\right\} = \int \ln p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta})\, p(\mathbf{X^m} \mid \mathbf{X^o}, \boldsymbol{\theta}) d\mathbf{X^m} \quad (2.41)$$

entonces podemos expresar

$$l(\boldsymbol{\theta} \mid \mathbf{X^o}) = Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) - H(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) \quad (2.42)$$

En la notación empleada, cuando escribimos $\boldsymbol{\theta}$ nos referimos a una variable aleatoria, mientras que si añadimos el subíndice superior $\boldsymbol{\theta}^{\mathbf{k}}$ a lo que nos referimos es a la estimación *k*-ésima de esos parámetros.

Los cuatro pasos del algoritmo EM son los siguientes:

1. Fijar *k=0* e inicializar $\boldsymbol{\theta}$ con un $\boldsymbol{\theta}^{\mathbf{0}}$ arbitrario.
2. Paso-E: calcular $Q(\boldsymbol{\theta}^{\mathbf{k+1}} \mid \boldsymbol{\theta}^{\mathbf{k}})$.
3. Paso-M: encontrar $\boldsymbol{\theta}^{k+1}$ tal que $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}})$ quede maximizado.
4. Si $\boldsymbol{\theta}^{k+1} \neq \boldsymbol{\theta}^{k}$ incrementar *k* y volver al paso 2.

Basándonos en el artículo Dempster, Laird y Rubin [26] podemos afirmar que para maximizar la verosimilitud bastará con encontrar un conjunto de parámetros $\boldsymbol{\theta}^{\mathbf{k+1}}$ que maximicen $Q(\boldsymbol{\theta} \mid \theta^{k})$, ya que está demostrado que si encontramos $\boldsymbol{\theta}^{\mathbf{k+1}}$ que consigan

$$Q(\boldsymbol{\theta}^{\mathbf{k+1}} \mid \boldsymbol{\theta}^{\mathbf{k}}) \geq Q(\boldsymbol{\theta}^{\mathbf{k}} \mid \boldsymbol{\theta}^{\mathbf{k}}) \quad (2.43)$$

entonces también quedará minimizado $H(\theta \mid \theta^{k})$:

$$H(\boldsymbol{\theta}^{\mathbf{k+1}} \mid \boldsymbol{\theta}^{\mathbf{k}}) \leq H(\boldsymbol{\theta}^{\mathbf{k}} \mid \boldsymbol{\theta}^{\mathbf{k}}) \quad (2.44)$$

y por lo tanto, tendremos un aumento de la verosimilitud:

$$l(\boldsymbol{\theta}^{\mathbf{k+1}} \mid \mathbf{X^o}) \geq l(\boldsymbol{\theta}^{\mathbf{k}} \mid \mathbf{X^o}) \quad (2.45)$$

Teniendo en cuenta que la función $H(\boldsymbol{\theta}|\boldsymbol{\theta}^{\mathbf{k}})$ representa la función logaritmo de verosimilitud del conjunto total $l(\boldsymbol{\theta}|\mathbf{x})$ (datos incompletos incluidos), podemos afirmar que

$$l(\boldsymbol{\theta}^{\mathbf{k+1}}|\mathbf{X}) \geq l(\boldsymbol{\theta}^{\mathbf{k}}|\mathbf{X}) \longrightarrow l(\boldsymbol{\theta}^{\mathbf{k+1}}|\mathbf{X}^{\mathbf{o}}) \geq l(\boldsymbol{\theta}^{\mathbf{k}}|\mathbf{X}^{\mathbf{o}}) \tag{2.46}$$

Es decir, si maximizamos el logaritmo de la verosimilitud del conjunto total de datos, estaremos maximizando el logaritmo de la verosimilitud del conjunto observado de datos.

2.5.4 EM para modelos de mezclas

A continuación aplicaremos el algoritmo EM a una mezcla de funciones de distribución. Comenzaremos recordando la expresión de la probabilidad de una muestra dada por una mezcla

$$p(\mathbf{x}^{(\mathbf{n})}) = \sum_{j=1}^{M} P(\mathbf{x}^{(\mathbf{n})}|w_j;\boldsymbol{\theta}_{\mathbf{j}})P(w_j) \tag{2.47}$$

donde denotábamos cada componente de la mezcla como w_j y estaba parametrizada por $\boldsymbol{\theta}_{\mathbf{j}}$.

Comenzaremos asumiendo datos completos. A partir de la ecuación (2.47) y de la suposición de independencia vemos que el logaritmo de la verosimilitud de los parámetros dado el conjunto de datos es

$$l(\boldsymbol{\theta}|\mathbf{X}) = \sum_{n=1}^{N} \ln \sum_{j=1}^{M} P(\mathbf{x}^{(\mathbf{n})}|w_j;\boldsymbol{\theta}_{\mathbf{j}})P(w_j) \tag{2.48}$$

Por el principio de máxima verosimilitud el mejor modelo de los datos tiene parámetros que maximizan $l(\boldsymbol{\theta}|\mathbf{X})$. Esta función, de cualquier modo, no es fácil de maximizar numéricamente porque incluye el logaritmo de una suma.

Intuitivamente, no está claro que componente de la mezcla generó una muestra dada y por lo tanto, que parámetros modificar para ajustarnos a esa muestra. El algoritmo EM para modelos de mezclas es un método iterativo para resolver este. La intuición indica que si tuviésemos acceso a una variable aleatoria "oculta" **z** que indicase que datos fueron generados por cada componente, entonces el problema de maximización total podría descomponerse en un conjunto de maximizaciones simples.

Usando las variables indicadoras binarias $\mathbf{Z} = \{\mathbf{z^{(n)}}\}$ *n=1...N*, definidas de manera que $\mathbf{z^{(n)}} = \left(z_1^{(n)}, ..., z_M^{(n)}\right)$ y z_j^n si $\mathbf{x^n}$ es generada por la gaussiana *j-ésima*, podemos escribir una función de logaritmo de verosimilitud de datos completos

$$l(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{j=1}^{M} z_j^{(n)} \log\left[P(\mathbf{x^{(n)}} \mid \mathbf{z^{(n)}}; \boldsymbol{\theta}) P(\mathbf{z^{(n)}}; \boldsymbol{\theta}) \right], \tag{2.49}$$

que no implica el logaritmo de una suma. Así solucionamos algunos de los problemas presentes en el apartado 2.4.2.

Puesto que *z* es desconocido, $l(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{Z})$ no puede ser utilizada directamente, por lo tanto en su lugar trabajaremos con su esperanza, denotada por $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{\mathbf{k}})$. Como muestra (Dempster y otros., 1977), $l(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{Z})$ puede ser maximizado iterando los pasos:

$$\begin{array}{ll} Paso-E & Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) = E\left\{ l(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{Z}) \mid \mathbf{X}, \boldsymbol{\theta}^{\mathbf{k}} \right\} \\ Paso-M & \boldsymbol{\theta}^{\mathbf{k+1}} = \arg_{\boldsymbol{\theta}} \max Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) \end{array} \tag{2.50}$$

La esperanza o paso-E computa el logaritmo de la verosimilitud del conjunto de datos completo, y la maximización o paso-M encuentra los parámetros que maximizan esta verosimilitud. En la práctica, para densidades de la familia exponencial, como es nuestro caso, el paso-E se reduce a calcular la esperanza sobre los datos perdidos de los estadísticos suficientes requeridos para el paso-M. Estos dos pasos forman la base del algoritmo EM para modelado de mezclas.

2.5.5 Incorporando valores perdidos en EM

Otra importante aplicación de EM es aprender a partir de conjuntos de datos con valores perdidos [7]. Esta aplicación ha sido seguida en la literatura estadística principalmente en problemas de estimación de densidad sin mezclas[2]. Nosotros mostraremos ahora como combinar las aplicaciones para datos perdidos de EM con las de aprendizaje de parámetros de mezclas, resultando en un conjunto de algoritmos para datos incompletos de aplicación en clustering, clasificación y aproximación de funciones y modelado de densidades de probabilidad. Como comentábamos en la introducción, fueron la capacidad de combinar el

[2] Entre las excepciones destacamos el uso de densidades de mezclas en el contexto de modelos normales contaminados para una estimación más robusta desarrollado por Little y Rubin en 1987 [7].

aprendizaje de parámetros de mezclas junto con la asimilación de datos perdidos las características del algoritmo EM que nos llevaron a elegirlo como método para modelar funciones de densidad.

Usando la notación definida anteriormente, $\mathbf{x^n}$ es dividida en $(\mathbf{x^{(o,n)}}, \mathbf{x^{(m,n)}})$ donde cada vector de entrada puede tener diferentes patrones de componentes perdidas. Para manejar datos perdidos rescribimos el algoritmo EM incorporando las variables indicadoras del algoritmo (2.50) y las entradas perdidas, $\mathbf{X^m}$.

$$\begin{aligned} &Paso-E \qquad Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) = E\left\{ l(\boldsymbol{\theta} \mid \mathbf{X^o}, \mathbf{X^m}, \mathbf{Z}) \mid \mathbf{X^o}, \boldsymbol{\theta}^{\mathbf{k}} \right\} \\ &Paso-M \qquad \boldsymbol{\theta}^{\mathbf{k+1}} = \arg_{\boldsymbol{\theta}} \max Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}}) \end{aligned} \tag{2.51}$$

El valor esperado en el paso-E es tomado con respecto a ambos conjuntos de variables perdidas. Procederemos a ilustrar este algoritmo para dos clases de modelos, mezclas de gaussianas y mezclas multinomiales, que usaremos más tarde como bloques de construcción para clasificación y aproximación de funciones.

2.5.6 Datos con valores reales: mezcla de gaussianas

Los datos con valores reales pueden ser modelados como una mezcla de *M* Gausianas. Comenzaremos con el algoritmo de *estimación de datos completos*. Para este modelo, el paso-E se simplifica, reduciéndose a calcular $E[z_j^{(n)} \mid \mathbf{x^{(n)}}, \boldsymbol{\theta}^{\mathbf{k}}]$. Dada la naturaleza binaria de $z_j^{(n)}$, $E[z_j^{(n)} \mid \mathbf{x^{(n)}}, \boldsymbol{\theta^{(k)}}]$, que denotaremos por $h_j^{(n)}$, es la probabilidad de que la gaussiana *j* generase el dato *n*.

$$h_j^n = \frac{|\hat{\boldsymbol{\Sigma}}_{\mathbf{j}}|^{-1/2} \exp\{-0.5(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{j}})^T \hat{\Sigma}_j^{-1}(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{j}})\}}{\sum_{l=1}^{M} |\hat{\boldsymbol{\Sigma}}_{\mathbf{l}}|^{-1/2} \exp\{-0.5(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{l}})^T \hat{\Sigma}_l^{-1}(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{l}})\}} \tag{2.52}$$

El paso-M vuelve a estimar las medias y covarianzas de las gaussianas[3] usando el conjunto de datos ponderado por $h_j^{(n)}$:

$$\hat{\boldsymbol{\mu}}_{\mathbf{j}}^{\mathbf{k+1}} = \frac{\sum_{n=1}^{N} h_j^n \mathbf{x}^{(\mathbf{n})}}{\sum_{n=1}^{N} h_j^n} \tag{2.53}$$

$$\hat{\boldsymbol{\Sigma}}_{\mathbf{j}}^{\mathbf{k+1}} = \frac{\sum_{n=1}^{N} h_j^{(n)} (\mathbf{x}^{(\mathbf{n})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{k+1}})(\mathbf{x}^{(\mathbf{n})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{k+1}})^T}{\sum_{n=1}^{N} h_j^{(n)}}. \tag{2.54}$$

Para incorporar datos perdidos empezamos retomando el logaritmo de la verosimilitud de los datos completos (2.49) y separando los factores del interior del logaritmo,

$$l(\boldsymbol{\theta} \mid \mathbf{X}^{\mathbf{o}}, \mathbf{X}^{\mathbf{m}}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{j=1}^{M} z_j^{(n)} \ln P(\mathbf{x}^{(\mathbf{n})} \mid \mathbf{z}^{(\mathbf{n})}; \boldsymbol{\theta}) + \sum_{n=1}^{N} \sum_{j=1}^{M} \mathbf{z}_{\mathbf{j}}^{(\mathbf{n})} \ln P(\mathbf{z}^{(\mathbf{n})}; \boldsymbol{\theta}). \tag{2.55}$$

Podemos ignorar el segundo término puesto que no nos hace falta para maximizar $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{\mathbf{k}})$. Particularizando la ecuación (2.55) para la mezcla de gaussianas, es evidente que, si lo único que está perdido son las variables $z_j^{(n)}$, el paso-E puede reducirse a estimar $E\left[z_j^{(n)} \mid \mathbf{x}^{(\mathbf{n})}, \boldsymbol{\theta}^{\mathbf{k}} \right]$ como antes. Para el caso en el que estamos interesados, con tanto $z^{(n)}$ como $\mathbf{x}^{(\mathbf{m,n})}$ perdidas, expandimos la ecuación (2.55) usando los superíndices *m* y *o* para denotar subvectores y submatrices de los parámetros que corresponden a las componentes perdidas y observadas de los datos, para obtener

$$\begin{aligned} l(\boldsymbol{\theta} \mid \mathbf{X}^{\mathbf{o}}, \mathbf{X}^{\mathbf{m}}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{j=1}^{M} z_j^{(n)} \Bigg[& \frac{d}{2} \ln 2\pi + \frac{1}{2} \ln \left| \boldsymbol{\Sigma}_{\mathbf{j}} \right| \\ & - \frac{1}{2} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right)^T \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{-1,oo}} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right) \\ & - \frac{1}{2} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right)^T \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{-1,om}} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right) \\ & - \frac{1}{2} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right)^T \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{-1,mm}} \left(\mathbf{x}^{(\mathbf{n,o})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}} \right) \Bigg] \end{aligned} \tag{2.56}$$

[3] Si bien este proceso asume probabilidades iguales a priori para las gaussianas, si las probabilidades a priori se interpretan como parámetros de la mezcla ellas también pueden ser aprendidas en el paso de maximización.

Obsérvese que después de tomar la esperanza, los estadísticos suficientes para los parámetros incluyen tres términos desconocidos $z_j^{(n)}$, $z_j^{(n)}\mathbf{x}^{(\mathbf{m,n})}$ y $z_j^{(n)}\mathbf{x}^{(\mathbf{m,n})}\mathbf{x}^{(\mathbf{m,n})^T}$. De este modo, debemos calcular:

$$E[z_j^{(n)} \mid \mathbf{x}^{(\mathbf{o,n})}, \boldsymbol{\theta}^{\mathbf{k}}],\ E[z_j^{(n)}, \mathbf{x}^{(\mathbf{m,n})} \mid \mathbf{x}^{(\mathbf{o,n})}, \boldsymbol{\theta}^{\mathbf{k}}]\ y\ E[z_j^{(n)}, \mathbf{x}^{(\mathbf{m,n})}, \mathbf{x}^{(\mathbf{m,n})^T} \mid \mathbf{x}^{(\mathbf{o,n})}, \boldsymbol{\theta}^{\mathbf{k}}] \qquad (2.57)$$

Un método intuitivo para tratar con datos perdidos es usar las estimaciones actuales de la densidad de datos para calcular la esperanza de los datos perdidos en un paso-E, completar los datos con estas esperanza, y entonces usar estos datos completos para re-estimar los parámetros en el paso M. Sin embargo, como veremos en el experimento 5.1.4, este método no nos proporciona resultados adecuados ni siquiera cuando tratamos con una única gaussiana bidimensional; la esperanza de los datos perdidos siempre yace a lo largo de una línea, lo que sesga la estimación de la covarianza. Por otro lado, el enfoque que surge de la aplicación del algoritmo EM especifica que debemos usar la estimación actual de densidad para calcular la esperanza de cualquier término incompleto que aparezca en la maximización de la verosimilitud. Para la mezcla de gaussianas estos términos incompletos son las interacciones entre la variable indicadora $z_j^{(n)}$ y el primer y segundo momento de $\mathbf{x}^{(\mathbf{m,n})}$. Por lo tanto, el cálculo de la esperanza de los datos perdidos $z_j^{(n)}$ y $\mathbf{x}^{(\mathbf{m,n})}$ a partir del modelo y la sustitución de estos valores en el paso M no resultan suficiente para garantizar un incremento en la verosimilitud de los parámetros.

La primera de las esperanzas de a resolver, $E[z_j^{(n)} \mid \mathbf{x}^{(\mathbf{o,n})}, \boldsymbol{\theta}^{\mathbf{k}}]$ ya la habíamos calculado previamente en (2.52), y llamándola $h_j^{(n)}$. No debemos dejarnos engañar por la diferencia que presentaba la esperanza que dio lugar a $h_j^{(n)}$ en (2.52), y que denotábamos $E[z_j^{(n)} \mid \mathbf{x}^{(\mathbf{n})}, \boldsymbol{\theta}^{\mathbf{k}}]$ con respecto a la esperanza que estamos calculando actualmente. Ambas expresiones solo emplean los datos que son observados, pero anteriormente no lo señalábamos ya que al no haber datos perdidos, el conjunto completo era observado.

Para calcular el resto de las esperanzas de (2.57), definiremos

$$\hat{\mathbf{x}}_{\mathbf{j}}^{(\mathbf{m,n})} = E\left[\mathbf{x}^{(\mathbf{m,n})} \mid z_j^{(n)} = 1, \mathbf{x}^{(\mathbf{o,n})}, \boldsymbol{\theta}^{\mathbf{k}}\right] = \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{m}} + \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{mo}}\boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{oo}^{-1}}(\mathbf{x}^{(\mathbf{o,n})} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{o}}) \qquad (2.58)$$

como la regresión lineal de mínimos cuadrados entre $\mathbf{x}^{(m,n)}$ y $\mathbf{x}^{(o,n)}$ predicha por la gaussiana *j-ésima*. Similarmente tenemos

$$E\left[z_j^{(n)} \mathbf{x}^{(m,n)} \mid \mathbf{x}^{(o,n)}, \boldsymbol{\theta}^{k} \right] = h_j^{(n)} \hat{\mathbf{x}}_j^{(m,n)} \tag{2.59}$$

y

$$E\left[z_j^{(n)} \mathbf{x}^{(m,n)} \mathbf{x}^{(m,n)^T} \mid \mathbf{x}^{(o,n)}, \boldsymbol{\theta}^{k} \right] = h_j^{n} \left(\boldsymbol{\Sigma}_j^{mm} - \boldsymbol{\Sigma}_j^{mo} \boldsymbol{\Sigma}_j^{oo^{-1}} \boldsymbol{\Sigma}_j^{mo^T} + \hat{\mathbf{x}}_j^{(n,m)} \hat{\mathbf{x}}_j^{(n,m)^T} \right) \tag{2.60}$$

El paso M usa estas esperanzas sustituidas en las ecuaciones (2.53) y (2.54) para re-estimar las medias y covarianzas. Sin embargo este proceso no es directo. Para ello habrá que separar el conjunto de datos $\mathbf{X}^{(n)}$ en sus partes observada y perdida, descomponiendo las ecuaciones hasta que todos los términos sean conocidos y después sustituir. El problema viene a la hora de descomponer, ya que las características de las muestras quedarán desordenadas, y tras los cálculos habrá que volver a ordenarlas. La descomposición que realizamos para operar con cada vector de entrada $\mathbf{x}^{(n)}$ sigue este esquema:

$$\mathbf{x}^{(n)} = \begin{pmatrix} \mathbf{x}^{(o,n)} \\ \mathbf{x}^{(m,n)} \end{pmatrix}, \quad \boldsymbol{\mu}_j = \begin{pmatrix} \boldsymbol{\mu}_j^{o} \\ \boldsymbol{\mu}_j^{m} \end{pmatrix}, \quad \boldsymbol{\Sigma}_j = \begin{pmatrix} \boldsymbol{\Sigma}_j^{oo} & \boldsymbol{\Sigma}_j^{om} \\ \boldsymbol{\Sigma}_j^{mo} & \boldsymbol{\Sigma}_j^{mm} \end{pmatrix} \tag{2.61}$$

donde las componentes observadas y perdidas de $\boldsymbol{\mu}_j$ y $\boldsymbol{\Sigma}_j$ se corresponden con las características observadas y perdidas para el vector $\mathbf{x}^{(n)}$. Puesto que el patrón de pérdidas no es uniforme (las características que presentan valores perdidos son diferentes para cada muestra), tendremos que realizar esta descomposición "muestra a muestra", realizar los cálculos para esa muestra y reordenar ese resultado para poder realizar las operaciones globales sobre todo el conjunto.

Estos detalles complican ligeramente el algoritmo que hasta ahora resultaba bastante sencillo. Nótese también que a la hora de desarrollar nuestras ecuaciones, hemos empleado el superíndice *(-1,oo)* para denotar la inversa seguida por operaciones de submatrices, mientras que con *(oo^{-1})* denotábamos el orden inverso.

2.5.7 Datos de valores categóricos: mezcla de multinomiales

Supongamos que nuestros datos en lugar de estar definidos en un dominio continuo, lo están en un dominio discreto, es decir, que cada dimensión solo puede tomar un valor entre

varios valores discretos. Por ejemplo, imaginemos los datos de un censo incluyen el país de nacimiento y el tipo de actividad que realiza un individuo. La variable 1, que será el país de nacimiento tomará, por ejemplo los valores España, Francia e Inglaterra. Por otro lado, la variable 2 que será el tipo de actividad que desarrolla podrá incluir estudiante, trabajador, parado o jubilado. En este caso la mezcla de gaussianas no nos es de ninguna utilidad, y por lo tanto nos vemos obligados a buscar otro modelo de mezclas para modelar este tipo de datos.

Un tipo particular de datos de este tipo son los datos binarios, en los que el dominio se restringe a los valores 0 y 1. Para problemas de este tipo se puede emplear una función llamada mezcla de Bernouillis, sin embargo en nuestro trabajo implementaremos otro tipo de función que es más general, ya que acepta cualquier dominio de datos categórico y no solo el binario. El modelo que emplearemos será el de la mezcla de funciones de densidad multinomiales. Pero para poder implementarlo el primer paso es codificar las variables de nuestro problema.

La codificación que emplearemos para cada variable será del tipo *one-hot*, es decir, que solo uno de los dígitos estará activo (+1) para cada variable. Necesitaremos tantos dígitos por variable como valores pueda tomar ésta. De este modo, nuestro problema pasaría a tener siete dimensiones, tres para expresar el país y cuatro para el tipo de actividad desempeñada. Por ejemplo, un francés jubilado tendría el código 0100001.

Prosiguiendo con nuestro estudio, la función de densidad de probabilidad multinomial tiene esta forma:

$$p(\mathbf{x}) = \prod_{i=1}^{d} \mu_i^{x_i} \tag{2.62}$$

donde $\boldsymbol{\mu}$ sería el único parámetro de nuestra función de densidad y el subíndice *i* hace referencia a la dimensión *i*-ésima de nuestro conjunto de datos. Para obtener el valor del vector $\boldsymbol{\mu}$ calculamos

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{n=1}^{N} \mathbf{x^{(n)}}}{N}. \tag{2.63}$$

Si extendemos a la mezcla de multinomiales según (2.22) quedará

$$p(\mathbf{x}) = \sum_{j=1}^{M} P(w_j) \prod_{i=1}^{d} \mu_{ji}^{x_i} \tag{2.64}$$

Para este modelo, el paso-E con un conjunto de datos completo quedaría

$$h_j^{(n)} = \frac{\prod_{i=1}^{d} \hat{\mu}_{ji}^{x_i}}{\sum_{j=1}^{M} \prod_{i=1}^{d} \hat{\mu}_{li}^{x_i}} \tag{2.65}$$

y el paso-M reestimaría el parámetro $\boldsymbol{\mu}$ mediante

$$\hat{\boldsymbol{\mu}}_{\mathbf{j}}^{\mathbf{k+1}} = \frac{\sum_{n=1}^{N} h_j^{(n)} \mathbf{x^{(n)}}}{\sum_{n=1}^{N} h_j^{(n)}}. \tag{2.66}$$

Por último, si decidimos emplear una mezcla multinomial y gaussiana, es decir, una mezcla en la que cada componente estima los valores continuos con una función gaussiana y los categóricos con una función multinomial, el paso-M no quedaría modificado. Y para el paso-E, la única estimación que tendríamos que cambiar sería

$$h_j^{(n)} = \frac{(\prod_{i=1}^{d} \mu_{ji}^{x_i^{(n)}}) \, |\hat{\boldsymbol{\Sigma}}_{\mathbf{j}}|^{-1/2} \exp\{-0.5(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{j}})^T \hat{\boldsymbol{\Sigma}}_{\mathbf{j}}^{-1} (\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{j}})\}}{\sum_{l=1}^{M} \left[(\prod_{i=1}^{d} \mu_{li}^{x_i^{(n)}}) \, |\hat{\boldsymbol{\Sigma}}_{\mathbf{l}}|^{-1/2} \exp\{-0.5(\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{l}})^T \hat{\Sigma}_l^{-1} (\mathbf{x^{(n)}} - \hat{\boldsymbol{\mu}}_{\mathbf{l}})\} \right]}. \tag{2.67}$$

2.5.8 Clustering

La estimación de un modelo de mezclas gaussianas es una forma suave de *agrupamiento* o *"clustering"* [22]. Además, si se emplea un modelo de matriz de covarianza completa, los ejes principales de la gaussiana se alinearán con los componentes principales de los datos dentro de cada cluster. Para datos binarios o categóricos también se pueden desarrollar algoritmos de clustering basados en los modelos de mezclas de multinomiales o de Bernuillis. Ilustraremos la extensión de estos algoritmos de clustering a los problemas de datos perdidos con un ejemplo simple de reconocimiento de caracteres.

Figura 2-3 Aprendiendo patrones de dígitos con ruido gaussiano y datos incompletos

En este ejemplo, el algoritmo de mezcla de gaussianas fue usado sobre un conjunto de entrenamiento de 100 dígitos 35-dimensionales en escala de grises con un 50% de píxeles perdidos. La primera fila de la figura 2-3 muestra las plantillas de 5x7 usadas para generar el conjunto de datos. En la segunda fila podemos ver las plantillas con ruido Gaussiano añadido. En la tercera, además del ruido hemos quitado un 50% de los píxeles. El conjunto de entrenamiento consistía en 10 muestras de este tipo, con ruido y píxeles incompletos para cada dígito. En la cuarta fila podemos ver las medias de las doce gaussianas en un punto asintótico (30 épocas[4] sobre un conjunto de 100 patrones) usando la imputación a la media heurística. En la quinta fila podemos observar las medias de las doce gaussianas tras 60 épocas del algoritmo EM sobre el mismo conjunto de datos.

El algoritmo EM aproxima las medias de los clusters a partir de este conjunto de datos altamente deficiente bastante bien. Los resultados muestran que EM mejora la imputación a la media tanto respecto a la distancia entre las medias de las gaussianas y las plantillas, como respecto a las verosimilitudes.

[4] Emplearemos el término época para referirnos a una iteración sobre todo el conjunto de entrenamiento.

2.5.9 Aproximación de funciones

Hasta aquí hemos hablado de vectores de datos sin hacer referencia a "entradas" y "objetivos". En el aprendizaje supervisado, sin embargo, generalmente queremos predecir variables objetivo a partir de algún conjunto de variables de entradas (esto es, queremos aproximar una función que relacione estos dos conjuntos de variables). Generalmente, en la notación clásica de aprendizaje supervisado, tenemos un vector de entrada **x** y un vector objetivo o salidas deseadas **t**. En cambio, para explicar el funcionamiento del algoritmo EM para aproximación de funciones, asumiremos que cada vector de datos $\mathbf{x}^{(n)}$ se puede descomponer en un subvector "entrada", $\mathbf{x}^{(i,n)}$, y un subvector "objetivo" o de salida, $\mathbf{x}^{(t,n)}$, entonces la relación entre variables de entrada y objetivo puede ser expresada a través de la densidad condicional $P(\mathbf{x}^{(t,n)}|\mathbf{x}^{(i,n)})$. Esta densidad condicional puede ser rápidamente obtenida a partir de la densidad conjunta entrada/salida, que es la densidad que todos los modelos de mezclas de arriba intentan estimar. De este modo, según este enfoque, la distinción entre aprendizaje supervisado, es decir, aproximación de funciones y aprendizaje no supervisado, es decir, estimación de densidad, es semántica, dependiendo de si los datos se consideran compuestos por entradas y salidas o no.

Centrándonos en el modelo de mezcla de gaussianas, observamos que la densidad condicional $P(\mathbf{x}^{(t,n)}|\mathbf{x}^{(i,n)})$ es también una mezcla de gaussianas. Dada una entrada particular la salida estimada debería resumir esta densidad. Si requerimos una única estimación de la salida, un candidato natural es el estimador de mínimos cuadrados (LSE), que toma la forma $\hat{\mathbf{x}}^{\mathbf{t}}(\mathbf{x}^{(i,n)}) = E(\mathbf{x}^{(t,n)} \mid \mathbf{x}^{(i,n)})$. Expandiendo la esperanza obtenemos

$$\hat{\mathbf{x}}^{\mathbf{t}}(\mathbf{x}^{(i,n)}) = \sum_{n=1}^{N} h_j^{(n)} \left[\boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{t}} + \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{ti}} \boldsymbol{\Sigma}_{\mathbf{j}}^{\mathbf{ii}^{-1}} (\mathbf{x}^{(i,n)} - \boldsymbol{\mu}_{\mathbf{j}}^{\mathbf{i}}) \right] \tag{2.68}$$

que es una suma convexa de las aproximaciones lineales de mínimos cuadrados dada por cada gaussiana. Los pesos en la suma, $h_j^{(n)}$, varían de forma no lineal sobre el espacio de entrada y se pueden ver como correspondientes a la salida de un clasificador que asigna a cada muestra en el espacio de entrada una probabilidad de pertenencia a cada gaussiana. El estimador de mínimos cuadrados tiene interesantes relaciones con modelos como CART [27], MARS [27] y mezcla de expertos [22], ya que la mezcla de gaussianas particiona

competitivamente el espacio de entrada y aprende una superficie de regresión linear en cada partición. Esta similitud también fue remarcada por *Tresp*, *Ahmad* y *Neuneier* [25].

Si las matrices de covarianza de las gaussianas son consideradas diagonales, el estimador de mínimos cuadrados se simplifica hasta quedar

$$\hat{\mathbf{x}}(\mathbf{x}^{(\mathbf{i,n})}) = \sum_{j=1}^{M} h_j^{(n)} \boldsymbol{\mu}_\mathbf{j}^\mathbf{t} \tag{2.69}$$

el promediado de las medias de salida, ponderados por la proximidad de $x_i^{(n)}$ a las medias de entrada de las gaussianas. Esta expresión tiene una forma idéntica a las redes de funciones de base radial (*Radial Basis Functions*, RBF) normalizadas (Moody y Darken, 1989; Poggio y Girosi, 1989), aunque los dos algoritmos provienen de enfoque diferentes. En el límite, cuando las matrices de covarianza de las gaussianas se aproximan a cero, la aproximación se convierte en un mapeo de vecino más próximo.

No todos los problemas de aprendizaje nos llevan a estimadores de mínimos cuadrados. Muchos problemas implican aprender una correspondencia uno-a-varios entre las variables de entrada. Las densidades condicionales resultantes son multimodales y un único valor de la salida dado no podrá reflejar apropiadamente este hecho [28]. Para tales problemas se prefiere un estimador estocástico, donde la salida es muestreada de acuerdo a

$$\hat{\mathbf{x}}(\mathbf{x}^{(\mathbf{i,n})}) \sim P(\mathbf{x}^{(\mathbf{t,n})} \mid \mathbf{x}^{(\mathbf{i,n})}), \tag{2.70}$$

a un estimador de mínimos cuadrados. Para problemas de aprendizaje que impliquen variables discretas los estimadores LSE y estocástico tienen una interpretación diferente. Si quisiéramos obtener la probabilidad a posteriori de la salida dada la entrada usaríamos el estimador LSE. Por otro lado, si quisiéramos obtener estimadores de salidas que caigan en nuestro espacio discreto de salida usaríamos el estimador estocástico.

2.5.10 Clasificación

La clasificación es un caso especial de aproximación de funciones. Y al igual que ocurría para aquellas se basa en la estimación de una relación entre “entradas” y “objetivos”. Del mismo modo, el hecho de que se trate de un problema de clasificación es una cuestión semántica, ya que nuestro algoritmo será exactamente el mismo, solo que esta vez entre los

datos de entrenamiento, incluiremos algunos que son considerados como “objetivos o salidas deseadas”. Además, el problema de clasificación exige que nuestra mezcla cuente entre sus funciones de estimación de densidad con funciones multinomiales, ya que la clase a la que pertenecen las muestras está representada por variables categóricas.

Los problemas de clasificación implican el aprendizaje de un mapeo del espacio de entrada de atributos a un conjunto discreto de etiquetas de clase. El enfoque del modelado de mezclas presentado aquí se ajusta rápidamente a problemas de clasificación mediante un modelado de las etiquetas de clase como variables multinomiales. Por ejemplo, si los atributos son valores reales y hay C etiquetas de clases, el modelo de mezclas con componentes gaussianas y multinomiales siguiente

$$P(\mathbf{x^{(o,n)}}, t^{(n)} = c \mid \boldsymbol{\theta}) = \sum_{j=1}^{M} P(w_j) \frac{\mu_j^c}{(2\pi)^{d/2} \left|\boldsymbol{\Sigma}_\mathbf{j}\right|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x^{(n)}} - \boldsymbol{\mu}_\mathbf{j})^T \Sigma^{-1} (\mathbf{x^{(n)}} - \boldsymbol{\mu}_\mathbf{j})\right\} \quad (2.71)$$

denota la probabilidad conjunta de que los datos tengan atributos **x,** cuya clase es denotada por la variable discreta $t^{(n)}$, pertenezcan a la clase *c*, donde las μ_j^c son los parámetros de las variables de clase multinomiales. Es decir, $\mu_j^c = P(t^{(n)} = c \mid w_j, \boldsymbol{\theta})$, y $\sum_{c=1}^{C} \mu_j^c = 1$.

Los atributos perdidos y las etiquetas perdidas de clase (es decir, muestras no etiquetadas) son rápidamente manejados con el algoritmo EM. En el paso-E, los atributos perdidos son completados usando las mismas ecuaciones que para la mezcla de gaussianas excepto por

$$h_j^{(n)} = P(\mathbf{x^{(o,n)}}, t^{(n)} = c \mid w_j, \boldsymbol{\theta}) = \frac{\mu_j^c P(\mathbf{x^{(o,n)}} \mid w_j, \boldsymbol{\theta})}{\sum_{l=1}^{M} \mu_l^c P(\mathbf{x^{(o,n)}} \mid w_l, \boldsymbol{\theta})}. \quad (2.72)$$

Por otro lado, si una etiqueta de clase está perdida, $h_j^{(n)}$ se queda como

$$h_j^{(n)} = \frac{P(\mathbf{x^{(o,n)}} \mid w_j, \boldsymbol{\theta})}{\sum_{l=1}^{M} P(\mathbf{x^{(o,n)}} \mid w_l, \boldsymbol{\theta})} \quad (2.73)$$

exactamente como en la mezcla de gaussianas. La etiqueta de clase es entonces completada con un vector de probabilidad cuya *c*-ésima componente es $\sum_{j=1}^{M} h_j^{(n)} \mu_j^c$.

Una vez que el modelo de clasificación ha sido estimado, la etiqueta más probable para una entrada particular *x* puede ser obtenida computando $P(t^{(n)} = c \mid \mathbf{x^{(n)}}, \boldsymbol{\theta})$. Similarmente, las densidades condicionales pueden ser calculadas evaluando $P(\mathbf{x^{(n)}} \mid t^{(n)} = c, \boldsymbol{\theta})$. Condicionando sobre clases de este modo obtenemos densidades condicionales de clases que son a su vez mezclas de gaussianas.

2.5.11 Detalles de implementación

Hasta ahora desarrollado teóricamente el algoritmo EM, revelando una gran capacidad en cuanto a flexibilidad y potencia. Sin embargo, cuando hemos aplicado el algoritmo a algunos experimentos, han surgido contratiempos que no estaban previstos en la teoría, y en esta sección describiremos como los hemos resuelto, o disminuido su efecto.

2.5.11.1 Evitar soluciones singulares

En algunas ocasiones, las secuencias de iteraciones producidas por el algoritmo EM para mezclas de densidades normales de múltiples variables convergen a soluciones singulares, es decir, puntos de la frontera del dominio de definición de los parámetros de la mezcla que están asociados a matrices singulares. Hosmer observó que sí se incluyen suficientes observaciones etiquetadas en una mezcla de densidades normales entonces, con probabilidad uno, la función de logaritmo de verosimilitud alcanza su máximo valor en un punto en el que las matrices de covarianza son definidas positivas (requisito de nuestra implementación)[5]. Esta consideración, en cierta manera, alivia el problema de que una secuencia de iteraciones EM tenga una solución singular.

[5] Para implementar el paso-E del algoritmo EM es necesario calcular la raíz cuadrada del determinante de la componente observada de la matriz de covarianza. Si esta no está definida positiva, el resultado de dicha operación sería imaginiario.

El problema de las soluciones singulares provoca que las matrices de covarianza, que deberían ser reales y semidefinidas positivas, tomen valores no apropiados. Es decir, aparecen valores imaginarios y las matrices dejan de estar semidefinidas positivas. Este hecho puede ser debido a la imprecisión de los cálculos realizados, ya que al representar internamente los valores con codificación binaria incurrimos en un error de cuantificación, que en condiciones normales no suele afectar, pero sí cuando tratamos con valores infinitesimales.

Para tratar de evitar el problema de las soluciones singulares, nos asegurararemos de que la matriz de covarianza se mantiene siempre semidefinida positiva. Para ello, en primer lugar, se comprueba mediante la factorización de Cholesky que la matriz es semidefinida positiva, y posteriormente, en caso de que no lo sea, se incrementa en un uno por ciento de su valor todos los elementos de la diagonal. Aumentando la diagonal de la matriz de covarianza nos deshacemos de las matrices singulares. El algoritmo, se supone que será capaz de enmendar los trastornos de la matriz en las siguientes iteraciones, y salir así de la zona de peligro. Sin embargo, en ocasiones el algoritmo nos vuelve a llevar a una matriz de covarianza singular. En estos casos, el algoritmo comienza a oscilar y puede paralizarse el aprendizaje.

2.5.11.2 Preservando la generalidad del aprendizaje

Al igual que las redes neuronales, si el conjunto de entrenamiento no es muy grande y entrenamos durante un número elevado de iteraciones, el algoritmo EM puede *sobreentrenar* (*overfitting*), lo que nos llevará a una pérdida en la capacidad de generalización.

Por ejemplo, imaginemos un conjunto de entrenamiento que conste de pocos casos y que ha sido generado a partir de una distribución gaussiana. En el caso de que las muestras presenten mucha desviación típica. Si, por efectos de la aleatoriedad, varias muestras están suficientemente separadas del resto, puede que el algoritmo EM interprete que se trata de un grupo de muestras generado por una función de distribución diferente, y les asigne una

componente de la mezcla de gaussianas. Esta situación no sería deseable, ya que estaríamos alterando el modelado del total del conjunto.

Para evitar este problema, hemos optado por la solución de incluir un conjunto de validación. Es decir, tomar una parte del conjunto de entrenamiento y emplearla para evitar el sobreentrenamiento de la mezcla de gaussianas. Durante una primera parte del aprendizaje, tanto la función de verosimilitud del conjunto de entrenamiento como la del conjunto de validación irán aumentando. Sin embargo, en una determinada época del entrenamiento, la función de verosimilitud del conjunto de entrenamiento seguirá aumentando, mientras que la del conjunto de validación comenzará a decrecer. Emplearemos el conjunto de validación para detectar cuando la mezcla empieza a sobreentrenarse, como veremos en la siguiente sección.

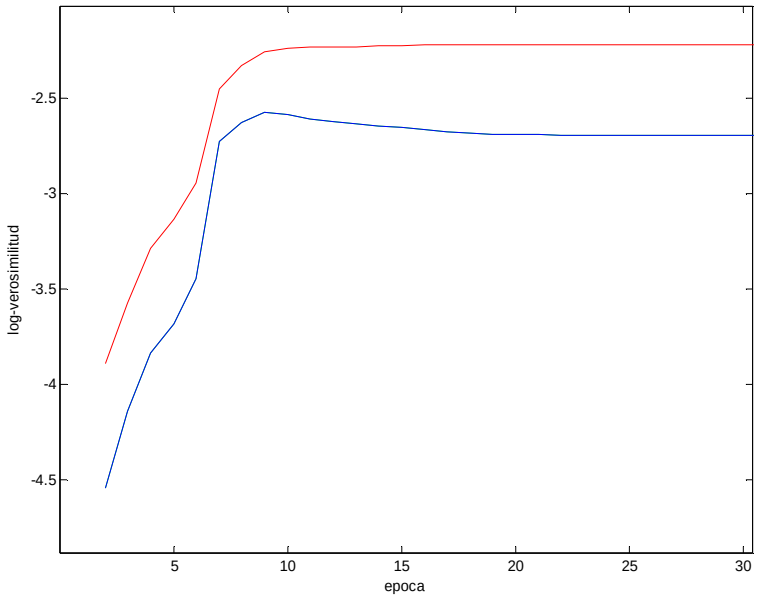


Figura 2-4 Efecto del sobreentrenamiento. Evolución de la función de verosimilitud para conjuntos de entrenamiento y validación.

En la figura 2-4 podemos observar en rojo la evolución de la función de verosimilitud sobre el conjunto de entrenamiento, y en azul, sobre el conjunto de validación. Como podemos observar, a partir de la décima época no tiene sentido seguir

entrenando, ya que en realidad estamos sobreentrenando. A la hora de decidir que porción de los datos de los que disponemos vamos a usar para validar tenemos que llegar a un compromiso. Por un lado, cuantos más datos destinemos a validación mejor vamos a poder estimar la generalización de nuestro entrenamiento. Pero por otro lado, al quitar muestras del conjunto de entrenamiento estamos empobreciendo el proceso de aprendizaje, ya que el algoritmo tiene menos "ejemplos" para aprender. En general, un criterio considerado aceptable consiste en destinar a validación tres veces menos datos que a entrenamiento.

Siguiendo por esta línea, en problemas artificiales, para comprobar el funcionamiento de nuestro algoritmo, necesitamos disponer de un conjunto de test. Es decir, en un principio tenemos el proceso de aprendizaje, que se basa en el conjunto de entrenamiento y cuya calidad está vigilada por el conjunto de validación; y posteriormente, pasamos a la fase de operación, en la que se aplica lo aprendido sobre un conjunto nuevo de muestras (conjunto de test), y medimos la precisión del aprendizaje. Por lo tanto, si disponemos de un conjunto de datos correspondientes a un problema, y queremos saber como se comporta nuestro algoritmo frente a él, tendremos que dividir el conjunto total en tres subconjuntos: uno de entrenamiento, otro de validación y otro de test. Un buen criterio para elegir las proporciones sería tomar la mitad, la sexta parte y la tercera parte, respectivamente, para estos subconjuntos.

2.5.11.3 Criterio de parada

Según los desarrollos teóricos vistos en las secciones previas, una de las principales ventajas del algoritmo EM es que, salvo casos singulares, con cada iteración la función de verosimilitud solo puede aumentar o permanecer constante. Lo ideal sería esperar a que la función de verosimilitud permaneciese constante, ya que eso significaría que nos encontraríamos en un máximo de la misma. Sin embargo, en general nos interesará detener la simulación antes de este momento por dos motivos. El primer motivo es que, como hemos observado en el apartado anterior, llegado el momento del sobreentrenamiento puede resultar ineficaz seguir con al aprendizaje. El segundo motivo se basa en que hay que llegar a un compromiso entre la precisión de los resultados y el tiempo de cálculo. Atendiendo a estos dos motivos, hemos implementado dos criterios de parada. El primero

de ellos consiste en detener el entrenamiento cuando la función de verosimilitud sobre el conjunto de entrenamiento haya descendido *n* veces seguidas. Con ello evitamos el sobreentrenamiento.

Para satisfacer el segundo de los motivos, nos basaremos en un cálculo realizado sobre una ventana de los últimos valores de la evolución de la función de verosimilitud y su comparación con un parámetro α_{fin} definido arbitrariamente a priori.

$$\alpha = 1 - \frac{valor\ máximo\ en\ la\ ventana}{valor\ medio\ en\ la\ ventana} \tag{2.74}$$

Es decir, tomamos los valores de la función de verosimilitud correspondientes a los *n* últimos valores estimados para los parámetros de la mezcla, y definimos la ventana de cálculo como el conjunto de esos n valores. Posteriormente calculamos α en esa ventana y si obtenemos un valor inferior al parámetro α_{fin} definido con anterioridad, damos por finalizado el proceso de aprendizaje. Con la aplicación de este método conseguimos detectar cuando el proceso se ha estancado, y así detener el aprendizaje. Fijando el parámetro α_{fin} a un valor bajo (pero superior a cero) forzaremos al algoritmo a continuar un mayor número de épocas.

2.5.11.4 Selección del número de componentes de la mezcla

Una cuestión importante en cuanto a la aplicación del algoritmo EM se refiere es la de su dimensionado, que consiste en determinar el número de componentes de la mezcla. Cuando representamos gráficamente un problema sencillo (de una o dos dimensiones) podemos ser capaces de distinguir a simple vista en cuantos grupos de similares características se podría dividir nuestro conjunto de datos. Sin embargo, conforme aumenta el número de dimensiones, la inspección visual no es un instrumento válido, y es preciso diseñar una alternativa automática para la selección del número de componentes de la mezcla.

En general, podemos afirmar que el empleo de un número excesivo de componentes solamente va a repercutir negativamente en el tiempo de procesado, que aumentará de manera lineal con los mismos, y no en la calidad del modelado de la mezcla. Esto se debe a que si hay más componentes de las necesarias, cuando el algoritmo estima sus parámetros,

tiende a agrupar algunas de ellas. Por ejemplo, si tenemos un conjunto de datos que se podría dividir en dos grupos y empleamos cinco componentes, simplemente se distribuirán las cinco entre los dos grupos, y la mezcla resultante modelará bien el problema. Estos es debido a que algunas de estas componentes tienden a las soluciones singulares comentadas anteriormente, y debido a la modificación artificial del algoritmo que hemos incluido, las componentes tienden a "desaparecer" y "reaparecer" en otra parte del espacio del problema. Es decir, las hiperesferas gaussianas asociadas a estas componentes comienzan a hacerse cada vez más pequeñas hasta que la "impureza" del algoritmo desvirtúa su forma, y captan otro grupo de datos, que aumenta la verosimilitud respecto a la nueva forma, al que comienzan a envolver. Tras este cambio, probablemente acabe desembocando en otra situación de este tipo la misma componente, o alguna que se encuentre en situación similar.

De cualquier modo, lo ideal sería elegir el menor número posible de componentes que modelen de manera relativamente eficaz los datos, es decir, con buena capacidad de generalización. Esto es lo que nos llevó a basar nuestro criterio de selección automático en un barrido del número de componentes empezando por un mínimo y terminando bien en un máximo, o bien cuando un criterio de parada nos indique que es inútil seguir aumentando el número de componentes. El criterio de finalización del barrido que hemos empleado es el mismo que hemos descrito en la sección anterior para detectar el estancamiento. Sin embargo es menos eficaz debido a la presencia de aumentos y decrementos en la función de verosimilitud respecto del número de componentes. Esto es debido a que cada vez que aumentamos el número de componentes estas se inicializan de nuevo, y el proceso de inicialización incluye una semilla aleatoria.

De igual modo que en el apartado anterior, una vez finalizado el barrido seleccionamos el número de componentes que proporcione el máximo valor para la función de verosimilitud sobre el conjunto de validación. Siendo éste el número de componentes definitivo. De manera adicional, una vez escogido el número de componentes, podemos profundizar en el aprendizaje.

Capítulo 3

Aprendizaje Multitarea

En este apartado trataremos de comprender el funcionamiento de una red neuronal basada en aprendizaje multitarea (MTL). Antes de entrar en los detalles de funcionamiento del aprendizaje multitarea repasaremos en que consiste una red neuronal y veremos algunos conceptos claves para su comprensión.

En inteligencia artificial, las **redes de neuronas artificiales** (**RNA**), referidas habitualmente de forma más sencilla como *redes de neuronas* o *redes neuronales,* son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz el cerebro que se caracterizan por su generalización y su robustez.

Las Redes neuronales artificiales son sistemas implementados de manera tal que funcionen como las neuronas biológicas de los seres vivos. El cerebro humano continuamente recibe señales de entrada de muchas fuentes y las procesa a manera de crear una apropiada respuesta de salida. Nuestros cerebros cuentan con millones de neuronas que se interconectan para elaborar “Redes Neuronales”. Las RNAs fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de

unidades llamadas “neuronas” o “nodos” conectadas unas con otras. Estas conexiones tienen una gran semejanza con las dendritas y los axones en los sistemas nerviosos biológicos.

3.1. Aprendizaje monotarea (Single Task Learning)

El aprendizaje monotarea (STL) es la configuración más habitual que presentan las redes neuronales artificiales, y consiste en dedicar toda la red al aprendizaje exclusivo de una única tarea, como puede ser la clasificación. Antes de analizar el funcionamiento de las redes neuronales basadas en STL, vamos a introducir sus orígenes y fundamentos.

3.1.1 Breve reseña histórica

La investigación en RNAs ha experimentado tres periodos de extensa actividad. El primer pico en los años 1940 fue debido al trabajo iniciado por McCulloch y Pitts [33]. McCulloch era un psiquiatra y neuroanatomista de formación; pasó unos 20 años pensando acerca de la representación de un evento en el sistema nervioso. Pitts era un prodigio matemático, quien se unió a McCulloch en 1942. En su artículo clásico [34], McCulloch y Pitts describen un cálculo lógico de las redes neuronales que reunían los estudios de neurofisiología y lógica matemática. Con un número suficiente de tales unidades simples y conexiones sinápticas ajustadas apropiadamente y operando sincronizadamente, McCulloch y Pitts mostraron que una red neuronal constituida así, sería capaz de calcular en principio cualquier función computable. Este fue un resultado muy significativo y a partir de su descubrimiento se ha acordado en general el nacimiento de las disciplinas en redes neuronales e inteligencia artificial [35].

El segundo pico ocurrió en los años ‘60 con el Teorema de Convergencia del Perceptrón de Rosenblatt [36] y el trabajo de Minsky y Papert mostrando las limitaciones de un perceptrón simple [37]. Los resultados de Minsky y Papert desalentaron el entusiasmo de la mayoría de los investigadores.

El resultante estancamiento en la investigación en redes neuronales duró casi veinte años, aunque existió una actividad importante que emergió en los ‘70 llamada *mapas autoorganizativos* usando aprendizaje competitivo.

Desde comienzos de 1980 hasta nuestros días las redes neuronales artificiales han recibido un interés considerable. Los mayores desarrollos tras este resurgimiento incluyeron el enfoque de energía de Hopfield en 1982 [38] y el algoritmo de aprendizaje de retropropagación para perceptrones multicapa propuesto inicialmente por Werbos [39], reinventado varias veces, y después popularizado por Rumelhart y otros en 1986 [40].

3.1.2 La neurona biológica

Una neurona (o célula nerviosa) es una célula biológica especial que procesa información (figura 3-1). Está compuesta de un cuerpo celular, o *soma*, y dos tipos de ramas que llegan al exterior: el *axón* y las *dendritas*. El cuerpo celular tiene un núcleo que contiene información acerca de rasgos hereditarios y un plasma que sostiene el equipo molecular para producir material necesario para la neurona. Una neurona recibe señales (impulsos) de otras neuronas a través de sus dendritas (receptores) y transmite señales generadas por su cuerpo celular a lo largo del axón (transmisor), el cual eventualmente se divide en ramales y subramales. En los extremos de estos ramales están las *sinapsis*. Una sinapsis es una estructura elemental y una unidad funcional entre dos neuronas (un ramal axón de una neurona y una dendrita de otra). Cuando el impulso alcanza el extremo de la sinapsis, ciertos el elementos químicos llamados neurotransmisores son liberados. Los neurotransmisores se difunden a través del espacio sináptico, para aumentar o inhibir, dependiendo del tipo de la sinapsis, la tendencia de la neurona receptora a emitir impulsos eléctricos. La efectividad de la sinapsis puede ser ajustada por las señales que pasan a través de ella de tal manera que las sinapsis pueden *aprender* de las actividades en las que participan. Esta dependencia de la historia actúa como una memoria, la cual es posiblemente responsable de la memoria humana [33].

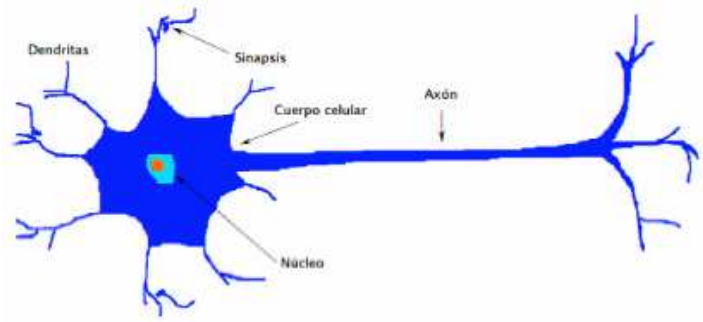


Figura 3-1: Diagrama de una neurona biológica

La corteza cerebral contiene cerca de 10^{11} neuronas, masivamente conectadas. Cada neurona está conectada con otras 10^3 a 10^4 neuronas. En total, el cerebro humano contiene aproximadamente 10^{14} a 10^{15} interconexiones. Las neuronas se comunican a través de un corto tren de pulsos, normalmente de milisegundos de duración, a una velocidad de conmutación mucho más lenta que en circuitos electrónicos. Las neuronas artificiales usadas para construir RNAs son mucho menos complejas que las encontradas en el cerebro.

3.1.3 La neurona artificial

La unidad básica de una RNA es la neurona. Aunque hay varios tipos de neuronas diferentes, la más común es la de tipo McCulloch-Pitts (1943). En la siguiente figura puede verse una representación de la misma

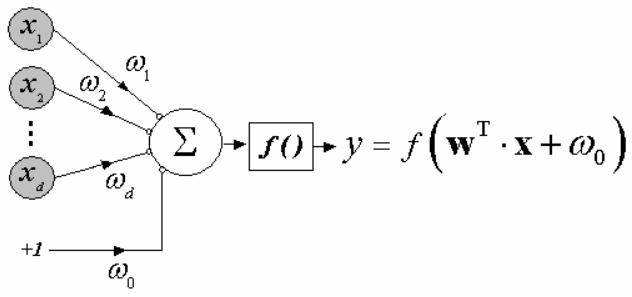


Figura 3-2: Diagrama del perceptrón simple

Este modelo se conoce como ***perceptrón simple*** de McCulloch-Pitts, y es la base de la mayor parte de la arquitectura de las RNA. Una neurona artificial es un procesador

elemental, en el sentido de que procesa un vector de entrada $\mathbf{x} = \{x_1, x_2, ..., x_d\}$ y produce una respuesta o salida única y. Los elementos clave de una neurona artificial los podemos ver en la figura anterior y son los siguientes:

- Las ***entradas*** $\mathbf{x}$ que reciben los datos de otras neuronas a las que esta conectada

- Los ***pesos sinápticos*** ω_i. En una neurona artificial a las entradas que vienen de otras neuronas se les asigna un peso. El parámetro ω_0 se suele conocer como *bias* o *sesgo*, y puede ser considerado como otro peso de una entrada extra fijada permanentemente a +1.

- Una regla de propagación. Con dichas entradas y los pesos sinápticos, se realiza una suma ponderada.

- Una función de activación. El valor obtenido con la regla de propagación, se filtra a través de una función conocida como función de activación y es la que nos da la salida de la neurona. Según para lo que se desee entrenar, se suele escoger una función de activación u otra. En la tabla 3-1 se muestran las funciones de activación mas usuales

El perceptrón es capaz tan sólo de resolver funciones definidas por un *hiperplano* (objeto de dimensión N-1 contenida en un espacio de dimensión N) que corte un espacio de dimensión N. Un ejemplo de una función que no puede ser resuelta es el operador lógico XOR. Ejemplos sencillos de hiperplanos para una, dos o tres dimensiones serían un punto, una recta o un plano, respectivamente. Pues bien, un perceptrón sólo puede resolver una función, si todos los posibles resultados del problema pueden separarse en dos secciones mediante un hiperplano.

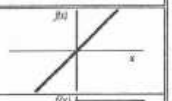
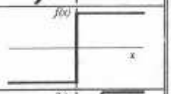
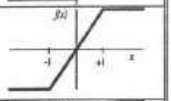
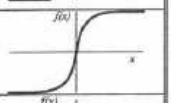
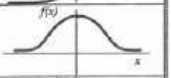
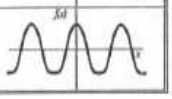
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	$[-1,+1]$	

Tabla 3-1: Funciones de activación habituales

3.1.4 Arquitectura de las redes neuronales artificiales

Desde un punto de vista matemático, se puede ver una red neuronal como un grafo *dirigido y ponderado* donde cada uno de los nodos son neuronas artificiales y los arcos que unen los nodos son las conexiones sinápticas. Al ser *dirigido*, los arcos son unidireccionales, es decir la información se propaga en un único sentido, desde una neurona origen a neurona destino. Por otra parte es ponderado, lo que significa que las conexiones tienen asociado un número real, un *peso*, que indica la importancia de esa conexión con respecto al resto de las conexiones.

Lo usual es que las neuronas se agrupen en capas de manera que una RNA esta formada por varias capas de neuronas. Aunque todas las capas son conjuntos de neuronas, según la función que desempeñan, suelen recibir un nombre especifico. Las más comunes son las siguientes:

- *Capa de entrada*: Las neuronas de la capa de entrada, reciben los datos que se proporcionan a la RNA para que los procese.

- *Capas ocultas*: Estas capas introducen grados de libertad adicionales en la RNA. El número de ellas puede depender del tipo de red que estemos considerando. Este tipo de capas realiza gran parte del procesamiento.

- *Capa de salida*: Esta capa proporciona la respuesta de la red neuronal. Normalmente también realiza parte del procesamiento.

El número de capas en una red multicapa se define como el número de capas de pesos, en vez de capas de neuronas. Es decir, una RNA con una capa de entrada, dos capas ocultas y una capa de salida, se dice que es una red de 3 capas, pues presenta 3 capas de pesos[6].

Si la arquitectura de la red no presenta ciclos, es decir, no se puede trazar un camino de una neurona a sí misma, la red se llama unidireccional (*feedforward*). A partir de ahora siempre nos referiremos a redes neuronales unidireccionales (***FFNN***, *FeedForward Neural Network*).

3.1.5 Propiedades

Las **Redes de Neuronas Artificiales** tienen muchas ventajas debido a que están basadas en la estructura del sistema nervioso, principalmente el cerebro.

- Aprendizaje: Las RNA tienen la habilidad de aprender mediante una etapa que se llama *etapa de aprendizaje*. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.
- Auto organización: Una RNA crea su propia representación de la información en su interior, descargando al usuario de esto.
- Tolerancia a fallos. Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo aceptablemente aún si se daña parcialmente.

[6] Algunos autores cuando hablan de capas de una red neuronal se refieren a capas de neuronas, y no de pesos. El número de capas de neuronas es una unidad mayor que el número de capas de pesos.

- Flexibilidad: Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (ej. si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente)
- Tiempo real: La estructura de una RNA es paralela, por lo cuál si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

3.1.6 El perceptrón multicapa

La forma más habitual de organizar las neuronas y capas de una RNA es la que se denomina ***perceptrón multicapa*** (***MLP***, *MultiLayer Perceptron*). El **MLP** (Rosenblatt, 1969) es el modelo más típico de las redes neuronales artificiales con aprendizaje supervisado. El entrenamiento de estas redes, se basa en la presentación sucesiva y de forma reiterada, de pares de vectores en las capas de entrada y salida (vectores entrada y salida deseada). La red crea un modelo a base de ajustar sus pesos en función de los vectores de entrenamiento, de forma que a medida que se pasan estos patrones, para cada vector de entrada la red producirá un valor de salida más similar al vector de salida esperado. Estas redes también se llaman de ***retropropagación*** (***backpropagation***), nombre que viene dado por el tipo de aprendizaje que utilizan.

En general, una red estándar de *L*-capas consiste de una capa de entrada, *L-1* capas ocultas, y una capa de salida de neuronas todas sucesivamente conectadas (completamente o parcialmente) con conexiones de alimentación hacia adelante, pero sin enlaces entre unidades de la misma capa o enlaces retroalimentados entre capas.

La figura 3-3 muestra un diagrama con su topología típica. La información se propaga de capa en capa (de izquierda a derecha), por medio de las conexiones entre las neuronas de cada capa. En los perceptrones multicapa cada neurona emplea generalmente la función de activación de tipo sigmoide.

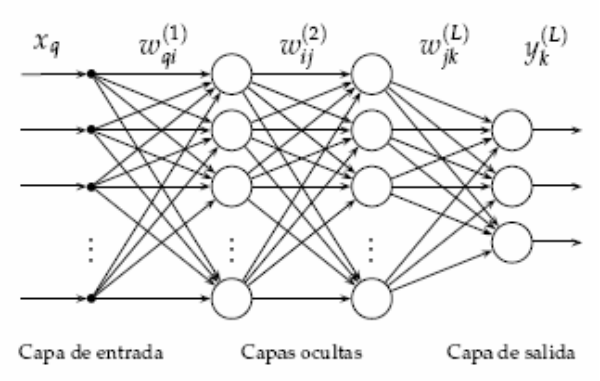


Figura 3-3: Topología genérica de un MLP

Un MLP con una capa oculta puede formar cualquier región convexa en este espacio. Las regiones convexas se forman mediante la combinación de las regiones formadas por cada neurona. Un MLP con dos capas ocultas puede generar regiones de decisión arbitrariamente complejas. El proceso de separación en clases que se lleva a cabo consiste en la partición de la región deseada en pequeños hipercubos. Cada hipercubo requiere *2d* neuronas en la primera capa oculta (siendo *d* el número de entradas a la red), una por cada lado del hipercubo, y otra en la segunda capa oculta, que lleva a cabo la operación lógica AND de la salida de los nodos del nivel anterior. La salida de los nodos de este segundo nivel se activarán solo para las entradas de cada hipercubo. Los hipercubos se asignan a la región de decisión adecuada mediante la conexión de la salida de cada nodo del segundo nivel solo con la neurona de salida (tercera capa) correspondiente a la región de decisión en la que este comprendido el hipercubo llevándose a cabo una operación lógica OR en cada nodo de salida. Este procedimiento se puede generalizar de manera que la forma de las regiones convexas sea arbitraria, en lugar de hipercubos.

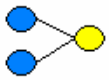
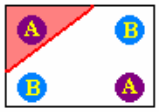
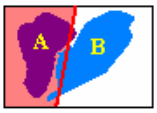
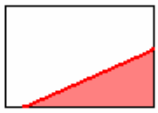
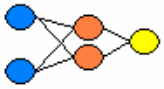
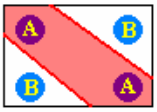
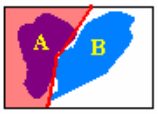
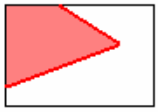
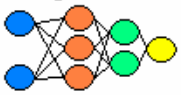
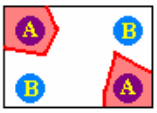
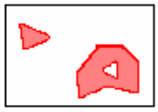
ARQUITECTURA	REGIÓN DE DECISIÓN	EJEMPLO 1: XOR	EJEMPLO 2: CLASIFICACIÓN	REGIONES MÁS GENERALES
Sin capa oculta 	Hiperplano (2 regiones)			
Una capa oculta 	Regiones polinómicas convexas			
Dos capas ocultas 	Regiones arbitrarias			

Tabla 3-2: Relación de arquitecturas típicas del MLP y sus "habilidades"

3.1.7 Método de entrenamiento: Algoritmo Backpropagation

3.1.7.1 Fundamentos

En 1986, tras 20 años de parón en el desarrollo de RNA, surge el ***algoritmo de retropropagación*** de los errores (***BP***, ***BackPropagation algorithm***) que se ha convertido sin duda en el algoritmo más empleado desde que Rumelhart y McClelland popularizaran este método. La idea básica, sencilla en su filosofía, requirió un desarrollo matemático fuerte para conseguir su convergencia. Se consideran los errores en las capas ocultas, que al propagarse hasta la capa de salida, producen los errores que realmente se miden. En definitiva, con este algoritmo se retropropagan los errores de la capa de salida hacia las capas anteriores, y emplear estos errores retropropagados para ajustar los pesos de las entradas de la correspondiente capa.

El algoritmo **BP** para **MLPs** es una generalización del algoritmo **LMS** (Least Mean Squares), pues ambos algoritmos realizan la actualización de los pesos en base al error cuadrático medio. Este algoritmo trabaja en modo supervisado, por tanto es necesario tener un conjunto de N patrones de entrenamiento junto con sus N salidas deseadas o *targets*. En la figura 3-4 se describe la estructura de un perceptrón de dos capas sobre la que se analizará el algoritmo de retropropagación. Las entradas (procedentes del patrón n-ésimo, con $n = 1 \ldots N$) de la neurona j-ésima de la primera capa oculta se denotan como $y_i(n)$,

siendo $\omega_{ij}(n)$ los pesos asociados a la conexión de cada uno de los pesos a la neurona *j*-ésima. La combinación lineal de las entradas en la neurona *j*-ésima ponderadas con los respectivos pesos se denota $v_j(n)$ y $\varphi(\cdot)$ es la función de activación empleada, generalmente la función sigmoide.

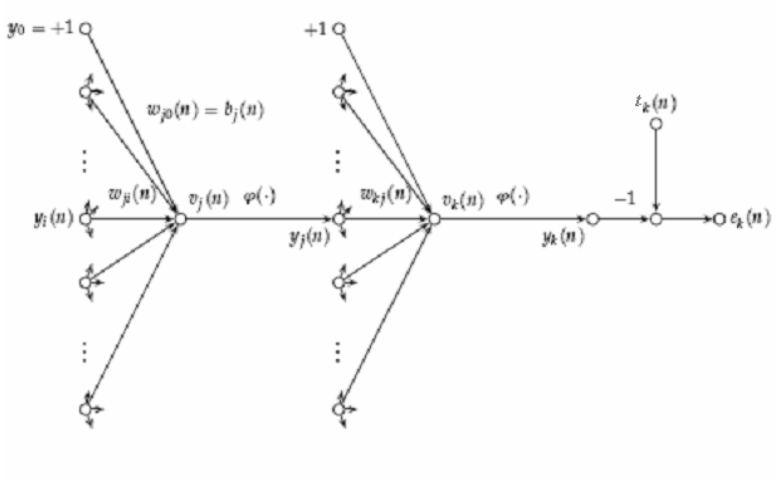


Figura 3-4 Diagrama de conexiones y pesos de un MLP con dos capas ocultas

Se define la señal de error en la salida de la neurona *k* (perteneciente a la capa de salida) en la iteración *n* (esto es, la presentación del *n*-ésimo patrón de entrenamiento) por:

$$e_k(n) = t_k(n) - y_k(n) \tag{3.1}$$

El valor instantáneo del error para la neurona *k* es $e_k^2(n)/2$. Así mismo, el valor instantáneo del error total *E(n)* se obtiene sumando todos los errores correspondientes a cada una de las neuronas de la capa de salida, esto es

$$E(n) = \frac{1}{2}\sum_k e_k^2(n) \tag{3.2}$$

donde *k* va desde uno hasta el número de neuronas de la capa de salida de la red. El error cuadrático promedio, teniendo en cuenta los *N* patrones (*n* = 1…N), es obtenida sumando todos los *E(n)* sobre todos los *n* y normalizando con respecto al tamaño del conjunto, de forma que

$$E_{prom} = \frac{1}{N}\sum_n E(n) \tag{3.3}$$

Tanto *E(n)* como E_{prom} están en función de los parámetros libres (pesos sinápticos y sesgos) de la red. Para un conjunto dado de entrenamiento, E_{prom} representa la función de coste como medida de rendimiento del aprendizaje. El objetivo del proceso de aprendizaje es ajustar los parámetros libres de la red para minimizar E_{prom}. Para esto, se considera un método simple de entrenamiento en el que los pesos se actualizan presentando uno a uno los patrones hasta que se termine una *época*, esto es, hasta que se presente completamente el conjunto de entrenamiento. Los ajustes a los pesos se hacen de acuerdo a los errores respectivos calculados para cada patrón presentado a la red.

Considérese una neurona *k* que está recibiendo un conjunto de señales producidas por una capa de neuronas a su izquierda (ver figura 3-4). El algoritmo de retropropagación aplica la corrección $\Delta\omega_{kj}(n)$ al peso $\omega_{kj}(n)$, que es proporcional a la derivada parcial $\partial E(n)/\partial \omega_{kj}(n)$. Esta derivada parcial determina la dirección de búsqueda en el espacio de los pesos para el peso $\omega_{kj}(n)$.

La corrección $\Delta\omega_{kj}(n)$ aplicada a $\omega_{kj}(n)$ está definida por la regla delta:

$$\Delta\omega_{kj}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{kj}(n)} \tag{3.4}$$

donde η es la tasa de aprendizaje del algoritmo. El signo menos describe el descenso del gradiente en el espacio de pesos (es decir, busca una dirección que el cambio de peso reduzca el valor de *E(n)*). Finalmente se tiene que

$$\Delta\omega_{kj}(n) = \eta \cdot \delta_k(n) \cdot y_j(n) \tag{3.5}$$

donde el gradiente local $\delta_k(n)$ está definido por

$$\delta_k(n) = e_k(n)\varphi_k'(\upsilon_k(n)) \tag{3.6}$$

donde la prima en $\varphi_k'(\upsilon_k(n))$ denota diferenciación.

De acuerdo con las dos anteriores ecuaciones existe un factor involucrado que es la señal de error $e_k(n)$. En este contexto, se pueden identificar dos casos distintos, dependiendo donde este localizada la neurona. En el caso descrito, la neurona *k* pertenece a

la capa de salida, por lo tanto se puede determinar el gradiente local de una manera simple, ya que se tiene una respuesta deseada $d_k(n)$ y $e_k(n)$, que se puede calcular fácilmente.

El segundo caso se presenta cuando una neurona *j* está localizada en una capa oculta de la red, cuando no se ha especificado una respuesta deseada para esa neurona. De acuerdo a esto, la señal de error para una neurona oculta se determina recursivamente en términos de las señales de error de todas las neuronas con las cuales esa neurona oculta está directamente conectada. Considérese la situación descrita en la figura 3-5. Se redefine el gradiente local $\delta_j(n)$ para la neurona oculta como

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \varphi_j'(\upsilon_j(n)). \tag{3.7}$$

Finalmente se obtiene la formula de retropropagación para el gradiente local $\delta_j(n)$:

$$\delta_j(n) = \varphi_j'(\upsilon_j(n)) \sum_k \delta_k(n)\omega_{kj}(n). \tag{3.8}$$

De esta forma la corrección aplicada $\Delta\omega_{ji}(n)$ a l peso que conecta la neurona *i* con la neurona *j* está definido por la regla delta:

$$\Delta\omega_{ji}(n) = -\eta \cdot \delta_j(n) \cdot y_i(n) \tag{3.9}$$

En la aplicación del algoritmo de retropropagación se distinguen dos pasos distintos. El primer paso es referido como el paso hacia adelante, y el segundo como paso hacia atrás. En el *paso hacia adelante* los pesos se mantienen inalterables y las salidas de los nodos de la red se calculan neurona por neurona. Esta fase de cómputo comienza en la primera capa oculta presentándole el vector de entrada, calculando las respectivas salidas y pasando esta información hacia las demás capas ocultas (si existen); la fase termina cuando se calcula la señal de error (comparar la salida de la neurona con su respuesta deseada) para cada neurona en la capa de salida.

El *paso hacia atrás*, por otra parte, comienza en la capa de salida pasando las señales de error hacia la izquierda a través de la red, capa por capa, y recursivamente calculando el δ (gradiente local) para cada neurona. Este proceso recursivo permite que los pesos sinápticos de la red sufran cambios de acuerdo con la regla delta (3.9). Para una

neurona en la capa de salida su gradiente local se calcula con (3.6). Por consiguiente, esta se usa para calcular los cambios en los pesos que alimentan la capa de salida. Dados estos δs se usa la ecuación (3.8) para calcular los gradientes de todas las neuronas de la penúltima capa y así aplicar las correcciones a los respectivos pesos. Este cómputo recursivo se realiza capa por capa hasta ajustar los pesos de toda la red. La figura siguiente muestra la representación como grafo de la ecuación (3.8), asumiendo que la capa de salida consta de m_L neuronas.

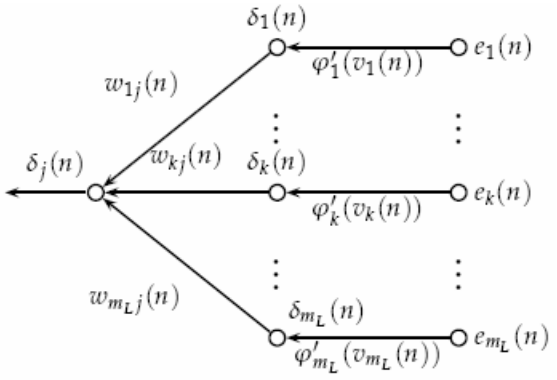


Figura 3-5 Representación de la propagación hacia atrás de los errores en una neurona

3.1.7.2 Aplicación del algoritmo de retropropagación

En el modo de entrenamiento secuencial o estocástico [35], la actualización de pesos se efectúa después de la presentación de cada patrón de entrenamiento. Para ser específicos, considérese una época consistente en *N* patrones de entrenamiento organizados en el orden $\{(\mathbf{x}_1,\mathbf{t}_1),(\mathbf{x}_2,\mathbf{t}_2),...,(\mathbf{x}_N,\mathbf{t}_N)\}$ (pares entrada-respuesta deseada). El primer patrón se presenta a la red, y se realiza la secuencia de operaciones hacia adelante y hacia atrás, descrita anteriormente, y como resultado se tiene un cierto ajuste de pesos. Luego se presenta el segundo patrón de igual forma y así sucesivamente hasta el último patrón, momento en el cual se concluye una época.

Para este modo de operación secuencial, el algoritmo realiza ciclos a través de la muestra de entrenamiento $(\mathbf{x}_n,\mathbf{t}_n)_{n=1}^N$ de la siguiente manera [35]:

1. *Inicialización*. Obtener los pesos sinápticos de una distribución de probabilidad uniforme, cuya varianza y desviación estándar de los campos locales inducidos de las neuronas se encuentren en la transición entre las partes lineales y saturadas de la función de activación sigmoide.

2. *Presentaciones de los ejemplos de entrenamiento*. Presentar una época de ejemplos de entrenamiento a la red. Realizar los pasos hacia adelante y hacia atrás, descritos en los puntos 3 y 4, respectivamente.

3. *Propagación hacia adelante*. Para un ejemplo de entrenamiento $(\mathbf{x}_n, \mathbf{t}_n)$, calcular las sumas ponderadas. La suma ponderada $\upsilon_j(n)$ para la neurona *j* en la capa *l* es:

$$\upsilon_j^{(l)}(n) = \sum_{i=0}^{m} \omega_{ji}^{(l)}(n) \cdot y_i^{(l-1)}(n) \tag{3.10}$$

donde $y_i^{(l-1)}(n)$ es la salida de neurona *i* en la capa anterior *l-1* en la iteración *n*, $\omega_{ji}^{(l)}(n)$ es el peso de la neurona *j* en la capa *l* que es alimentada por la neurona *i* en la capa *l-1*, y *m* es la cantidad de neuronas de la capa anterior. En este caso se tienen en cuenta las entradas para los sesgos. Asumiendo el uso de una función sigmoide, la salida de la neurona *j* en la capa *l* es

$$y_j^{(l)}(n) = \varphi_j\left(\upsilon_j^{(l)}(n)\right) \tag{3.11}$$

Si la neurona *j* está en la capa de entrada (esto es, *l = 0*), entonces

$$y_j^{(0)}(n) = x_j^n \tag{3.12}$$

donde x_j^n es el *j*-ésimo elemento del vector de entrada $\mathbf{x}_n$. Si la neurona *j* está en la capa de salida (esto es, *l = L*, donde *L* es el índice de la última capa de la red), entonces el error se calcula así:

$$e_j(n) = t_j(n) - y_j^{(L)}(n) \tag{3.13}$$

donde $d_j(n)$ es el *j*-ésimo elemento del vector de respuestas deseadas $\mathbf{d}_n$

4. *Propagación hacia atrás.* Calcular los gradientes locales de la red, definidos por

$$\delta_j^{(l)}(n) = \begin{cases} \text{Para la neurona } j \text{ en la capa de salida } L: \\ e_j^{(L)}(n) \cdot \varphi_j'\left(v_j^{(L)}(n)\right) \\ \text{Para la neurona } j \text{ en la capa oculta } l: \\ \varphi_j'\left(v_j^{(l)}(n)\right) \sum_k \delta_k^{(l+1)}(n) \cdot \omega_{kj}^{(l+1)}(n) \end{cases} \quad (3.14)$$

El ajuste en los pesos de la red de la capa l se realiza de acuerdo a la regla delta generalizada

$$\omega_{ji}^{(l)}(n+1) = \omega_{ji}^{(l)}(n) - \eta \cdot \delta_j^{(l)}(n) \cdot y_i^{(l-1)}(n) \quad (3.15)$$

donde η es la tasa de aprendizaje. Escoger un valor adecuado influye en la convergencia del algoritmo, por lo que en general se toman valores de η pequeños, pero esto significa que tendrá que hacer un gran número de iteraciones. Si la tasa de de aprendizaje es grande, el resultado son cambios drásticos en los pesos, avanzando muy rápidamente por la superficie de error, con el riesgo de estar oscilando alrededor del mínimo. Este efecto se puede contrarrestar con tasa adaptativa o con la adición del término de momento.

5. *Iteración.* Iterar los cómputos hacia adelante y hacia atrás de los puntos 3 y 4 presentando nuevas épocas de ejemplos de entrenamiento a la red hasta que el criterio de terminación se cumpla. El orden de presentación de los ejemplos de entrenamiento debería ser aleatorio de una época a otra. Los parámetros de momento y de tasa de aprendizaje son habitualmente ajustados (usualmente decrementados) cuando el número de iteraciones se incrementa.

3.1.7.3 Dificultades

Como puede verse en la literatura relativa a RNAs, se ha escrito mucho acerca de ventajas, pero también de los problemas de los perceptrones multicapa. A la hora de aplicar las redes neuronales en problemas reales pueden surgir numerosos contratiempos, por ejemplo, parámetros cuyo ajuste es arbitrario. A continuación enumeramos algunos de los más importantes:

- *Numero de capas y de neuronas*. No hay procedimientos claros y definitivos para decidir el mejor número de capas y/o neuronas en el MLP. Hay publicaciones

sobre cotas mínimas y máximas, generalmente para problemas de clasificación, en función del número de patrones de entrenamiento, pero en la práctica no se utilizan. Generalmente se determina de forma intuitiva y/o experimental, pero sin duda la experiencia del investigador sirve de gran ayuda.

■ *Conjunto de entrenamiento*. Se puede decir casi lo mismo que en el apartado anterior. Lo que está claro, es que si se pretende un buen aprendizaje del dominio del problema, o una buena generalización de los patrones, estos deben cubrir todo el espectro del dominio. El número de patrones a emplear dependen del problema y la experiencia del investigador ayuda. Se suele determinar casi siempre (a veces solo se dispone de patrones concretos) de forma experimental.

■ *Preparación de los datos*. En general, los datos no se usan directamente en la forma en que se obtienen o los proporciona el problema. Casi siempre se adaptan los datos reales al modelo usado. Por ejemplo, las salidas hay que normalizarlas en [0,1] si se usa activación sigmoide, porque sino, el error nunca podrá reducirse de forma satisfactoria. También es habitual normalizar las entradas, aún cuando no sea necesario. Obviamente habrá un post-procesamiento de resultados, para hacerlos válidos en la realidad del problema.

■ *Velocidad de convergencia*. El algoritmo de retropropagación es lento y se han propuesto muchas modificaciones y variantes para mejorar la velocidad del entrenamiento.

■ *Mínimos locales*. Algo importante que se debe tener en cuenta es la posibilidad de existencia de mínimos locales en la superficie de error (ver figura 3-6).El algoritmo de retropropagación con descenso de gradiente no garantiza que se alcance un mínimo global, una vez que se alcance un mínimo, el algoritmo se detiene, quedando a veces “atrapado” en un mínimo local, del que no puede salir, y por tanto el aprendizaje no se hace bien. Una forma de obviar esto es realizar el aprendizaje varias veces, partiendo de pesos diferentes cada vez, y seleccionar la ejecución que mejor resuelve el problema. Algunas de las variantes buscan usar

otros procesos de optimización no lineal más seguros que el de gradiente descendente.

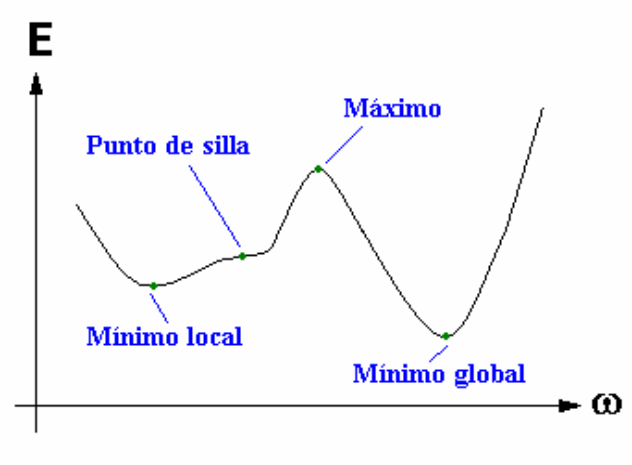


Figura 3-6 Función de error E en función de los pesos *w*

■ *Sobreentrenamiento*. Es el problema de que se aprende muy bien los ejemplos de entrenamiento, pero cuando se usa el MLP entrenado, con muestras que no ha aprendido, no es capaz de dar buenas respuestas. Se dice entonces que la red no generaliza bien. Se puede evitar esto seleccionando muy bien el conjunto de entrenamiento y deteniendo el entrenamiento cuando el error sea suficientemente pero no excesivamente pequeño. Para detectar este momento idóneo una técnica común consiste en emplear un conjunto de validación, como explicábamos en 2.5.11.3 para el algoritmo EM.

■ *Saturación*. Se puede producir cuando las salidas esperadas en cada neurona de salida son 0 o 1. Al pretender acercar las salidas de la red a estos valores, nos ponemos en las zonas donde la función de activación tiene tangente de pendiente casi cero y entonces no se produce apenas modificación de los pesos y por tanto del error. La forma de resolverlo es cambiar los valores de salidas esperadas a 0.1 y 0.9. Esto puede requerir adaptación de los datos reales. También se produce si los pesos se hacen muy grandes, por que los resultados de la red ya no varían.

- *Estabilidad y robustez*. Un tema interesante, sobre todo para las implementaciones en hardware de la RNA entrenada, es la estabilidad del aprendizaje sobre variaciones en los pesos (y/o las entradas). Hay medidas de estabilidad estadística, que se pueden usar para seleccionar el MLP entrenada más estable de entre varias ejecuciones. También hay métodos que tienen en cuenta estas medidas en el entrenamiento de la red, aunque son más lentos.

3.2. Aprendizaje multitarea (MultiTask Learning)

3.2.1 Introducción

El mundo en que vivimos requiere que aprendamos y resolvamos muchas tareas y conceptos. Estas tareas y conceptos que obedecen las mismas leyes físicas y están derivadas de la misma cultura humana, son preprocesadas por el mismo *hardware* sensitivo. Quizás es la similaridad de las múltiples tareas que aprendemos la que nos permite aprender tanto con tan poca experiencia.

Se aprende a jugar al tenis en un mundo que te pide aprender muchas otras cosas. También se aprende a andar, correr, saltar, ejercitarse, agarrar, lanzar, balancearse, reconocer objetos, predecir trayectorias, descansar, hablar, leer, estudiar, practicar, etc. Estas tareas no son la misma (correr en el tenis es diferente de correr en una pista) y sin embargo están relacionadas. Quizás, las similitudes entre las miles de tareas que se aprenden son lo que nos permite aprender una de ellas, incluyendo el tenis, con sólo un poco de entrenamiento.

Una red neuronal artificial entrenada a partir de cero para una única, difícil y aislada tarea es poco probable que la aprenda bien. Por ejemplo, una red con una retina de 1000x1000 píxeles de entrada es poco probable que aprenda a reconocer objetos complejos en escenas del mundo real dado un cierto número de muestras de entrenamiento disponibles. ¿No sería mejor forzar al sistema aprendiz a aprender varias cosas simultáneamente? Si las tareas pueden compartir lo que aprenden, el aprendiz puede encontrar más fácil aprenderlas en conjunto que de manera aislada. De este modo, si

entrenamos simultáneamente una red para que reconozca bordes de objetos, formas, regiones, subregiones, texturas, reflexiones, realces, sombras, textos, orientación, tamaño, distancia, etc... Puede que aprenda mejor a reconocer objetos complejos del mundo real. Se denomina a este enfoque como *Aprendizaje Multitarea* ("Multitask Learning", MTL).

El término Aprendizaje Multitarea fue acuñado por *Rich Caruana* en su tesis [41]. Para desarrollar su teoría se basó en el concepto de *sesgo inductivo,* introducido por *Baxter* [42], que es todo aquello que induce a una máquina a preferir unas hipótesis sobre otras. *Caruana* analiza MTL en perceptrones multicapa (MLP, Multilayer Perceptron). La manera más sencilla de implementarlo consiste en añadir salidas extra para aprender las distintas tareas secundarias, junto con la principal, compartiendo todas las tareas la única capa oculta de la red. El hecho de que un conjunto de neuronas ocultas estén conectadas a las salidas asociadas a las distintas tareas permite que lo que se aprenda en una salida contribuya al aprendizaje del resto, mejorando así la capacidad de generalización de la red neuronal. La tarea que se debe aprender mejor se conoce como la *tarea principal* ("main task"), mientras que las tareas que ayudan a mejorar la capacidad de generalización de la tarea principal se denominan tareas extra o secundarias ("extra tasks"). Se entiende por dominio a la unión de la tarea principal y las tareas extra.

3.2.2 Arquitecturas neuronales usadas en MTL

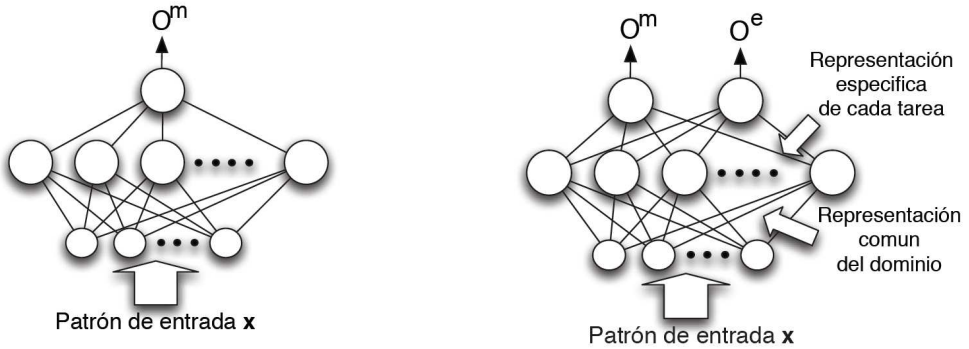
3.2.2.1 Arquitectura estándar para STL y MTL

Antes de analizar el aprendizaje multitarea, repasaremos el esquema STL. Considérese un conjunto $\mathfrak{M}$, asociado a una única tarea, formado por un conjunto de patrones de entrada $\mathbf{X}^{\mathbf{m}}$ y un conjunto de salidas deseadas $\mathbf{T}^{\mathbf{m}}$. Una arquitectura neuronal para STL consta de una capa oculta de neuronas completamente interconectadas y una salida asociada a la tarea que se desea aprender (ver figura 3-7(a)). Esta red puede ser entrenada para minimizar una función de error entre las salidas de esta red y las salidas deseadas. Por lo tanto, aprende únicamente las salidas deseadas $\mathbf{T}^{\mathbf{m}}$ usando $\mathbf{X}^{\mathbf{m}}$.

$\mathbf{X} = \left\{ \mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}, ..., \mathbf{x}^{(N)} \right\}$	Conjunto de vectores de entrada
$\mathbf{T} = \left\{ \mathbf{t}^{(1)}, ..., \mathbf{t}^{(n)}, ..., \mathbf{t}^{(N)} \right\}$	Conjunto de salidas deseadas
$\mathfrak{M} = \left\{ \mathbf{X}^{\mathbf{m}}, \mathbf{T}^{\mathbf{m}} \right\}$	Conjunto asociado a la tarea principal
$\boldsymbol{\varepsilon} = \left\{ \mathbf{X}^{\mathbf{e}}, \mathbf{T}^{\mathbf{e}} \right\}$	Conjunto para aprender las tareas extra
$\mathbf{x}^{(\mathbf{n})} = \left\{ x_1^{(n)}, ..., x_d^{(n)}, ..., x_D^{(n)} \right\}$	Vector de entrada
$\mathbf{t}^{(\mathbf{n})} = \left\{ t_1^n, ..., t_j^n, ..., t_p^n \right\}$	Vector de salidas deseadas
$\mathbf{o}^{\mathbf{m}}, \mathbf{o}^{\mathbf{e}}$	Salidas asociadas a las tareas

Tabla 3-3: Nomenclatura

En una primera aproximación al MTL, la tarea principal y una o varias tareas extra son aprendidas conjuntamente mediante una misma red con una capa oculta neuronas completamente interconectadas y tantas salidas como tareas a aprender. Considérese un conjunto $\mathfrak{M}$, asociado a una tarea principal, formado por un conjunto de patrones de entrada $\mathbf{X}^{\mathbf{m}}$ y un conjunto de salidas deseadas $\mathbf{T}^{\mathbf{m}}$, y un conjunto $\boldsymbol{\varepsilon}$, asociado a las tareas secundarias, formado por $\mathbf{X}^{\mathbf{e}}$ y $\mathbf{T}^{\mathbf{e}}$. En este caso, la red será entrenada para minimizar una función de error entre las salidas de la red y las salidas deseadas para cada tarea. El objetivo es que la red obtenga con exactitud nuevos valores para la tarea principal ante futuras entradas (o conjuntos de test). En general, nos interesa aprender con precisión la tarea principal, mientras que el aprendizaje de las tareas extra nos interesa sólo como guía para la tarea principal, mejorando así la capacidad de generalización de está última.



a) Arquitectura STL estándar

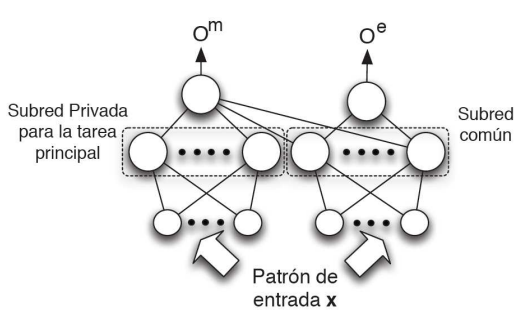
b) Arquitectura MTL estándar

Figura 3-7 Arquitecturas neuronales estándar para STL y MTL

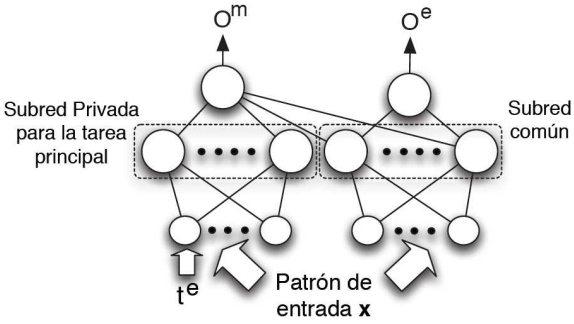
La figura 3-7(b) muestra la arquitectura MTL estándar con una capa oculta de neuronas y una salida asociada a cada tarea. Se observa como la capa oculta de neuronas está compartida por todas las tareas. Esta es la idea base del aprendizaje multitarea: compartir la información aprendida mientras las tareas se aprenden en paralelo. En cuanto a los pesos de la red, los pesos de la primera capa son actualizados según el error total de todas las tareas, mientras que los pesos de segunda capa son actualizados en función del error de cada tarea a la que se encuentran asociados.

3.2.2.2 Arquitectura con subredes privadas

Es posible mejorar el rendimiento y las prestaciones del MTL usando arquitecturas neuronales más complicadas que el esquema estándar MTL. Para ello, Caruana propuso en su tesis añadir una subred específica o privada que aprendiese únicamente la tarea principal. La figura 3-8(a) muestra este esquema donde se tiene dos capas separadas de neuronas ocultas o dos subredes separadas. Una de ellas es una subred privada empleada solamente por la tarea principal, mientras que la otra es la subred común compartida por la tarea principal y las tareas secundarias. Esta subred común soporta el MTL. Esta arquitectura es asimétrica ya que la tarea principal afecta al aprendizaje de la tarea extra mediante la subred común, mientras que la tarea extra no puede afectar a la subred privada reservada para la tarea principal.



a) Arquitectura MTL con una subred privada para la tarea principal



b) Arquitectura MTL con una subred privada para cada tarea principal y entradas extra.

Figura 3-8 Arquitecturas neuronales MTL con una subred privada

Hasta ahora se ha supuesto que, durante el aprendizaje de la tarea principal y las tareas secundarias, los vectores de entrada *x* son los mismos para todas las neuronas ocultas, según nuestra notación, $\mathbf{X^m} = \mathbf{X^e} = \mathbf{X}$. Usando entradas extra es posible aumentar las prestaciones de la arquitectura. Para ello, las salidas deseadas $\mathbf{t^e}$ junto con los vectores de entrada **x** se usan como entradas de la subred privada para aprender la tarea principal, $\mathbf{X^m} = \{\mathbf{X}, \mathbf{T^e}\}$. La figura 3-8(b) muestra esta arquitectura. De esta forma, se añade información a priori acerca del dominio en la subred privada, siendo mejor la generalización de la tarea principal.

Capítulo 4

Imputación Multitarea

Hasta este momento, como comentábamos en la introducción, la mayoría de los métodos para tratar datos perdidos en problemas de clasificación están enfocados de acuerdo a tres estrategias. La estrategia más usada consiste en separar el problema en la resolución de dos tipos de tareas: la tarea de clasificación y la tarea de imputación. En primer lugar, un modelo aprende los valores de las características incompletas (tareas de imputación) para estimar los valores perdidos usando el conjunto completo de datos de entrada. Posteriormente, otro modelo, como por ejemplo una red neuronal, aprende la tarea de clasificación usando el conjunto de datos modificado con los valores perdidos estimados.

Por otro lado, la segunda estrategia consiste en aprender la tarea de clasificación usando todos los datos de entrenamiento. Para ello, se modifica el mecanismo de aprendizaje de una RNA para que sea capaz de manejar valores perdidos. En este caso, la imputación no es usada para resolver la tarea de clasificación.

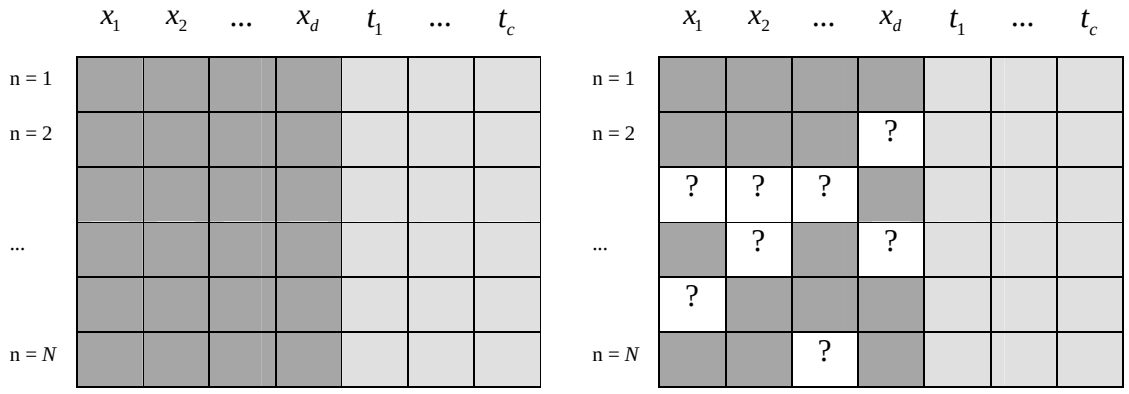
La tercera solución es aprender simultáneamente las tareas de clasificación y de imputación en una misma red [44]. En este trabajo seguiremos esta estrategia, de la que describiremos tres variantes. La ventaja de este método es que la estimación de los valores perdidos está orientada por el aprendizaje de la tarea de clasificación gracias al

aprovechamiento de todas las ventajas de MTL. A continuación, explicaremos como estas redes MTL aprenden y trabajan en problemas generales de clasificación, donde cada vector de entrada con *d* atributos reales puede pertenecer a una de *c* posibles clases. Tras esta explicación estudiaremos las fases de entrenamiento y operación.

4.1. Topologías

Supongamos un problema de clasificación de *c* clases descrito por *N* vectores de entrada compuestos por *d* atributos reales. Considerese que *m* de las *d* dimensiones ($m \leq d$) están incompletas (tienen algún valor perdido), y que cualquier vector de entrada puede tener un valor perdido en una de estas dimensiones, como se muestra en la figura 4-1(b). Además, definiremos el vector $a=[a_1, a_2, ...a_k, ..., a_m]$ cuyas componentes son los *m* atributos incompletos en el conjunto de datos. Por consiguiente, este problema estará compuesto por dos tipos de tareas diferentes:

- *Tarea principal*: tarea de clasificación entre *c* clases.
- *Tareas secundarias*: *m* tareas de imputación asociadas con cada característica incompleta



a) Problema de clasificación típico con datos completos

b) Problema de clasificación con muestras incompletas, denotadas por el símbolo ?.

Figura 4-1 Problema típico de clasificación con datos completos (a) o incompletos (b).

A continuación estudiaremos tres esquemas diferentes de redes neuronales para aprender en paralelo la tarea de clasificación y las características incompletas usando MTL. La figura 4-2 muestra una red neuronal MTL con una capa común que aprende en paralelo todas las tareas, las de clasificación y las de imputación. Obsérvese que la tarea de clasificación está compuesta por *c* unidades de salida, y que cada tarea de imputación tiene asociada una única unidad de salida. La capa de entrada está compuesta por *d* unidades para cada atributo, y además, hay *c* unidades de entrada extra que indican a que clase pertenece el vector de entradas, es decir, las salidas deseadas de clasificación. Todas las entradas están conectadas con las neuronas de la capa oculta común mediante un peso. La capa oculta puede estar compuesta por un número diferente de neuronas dependiendo del problema de clasificación. Usaremos entradas lineales y tangentes hiperbólicas como función de activación para todas las neuronas ocultas y salidas lineales para esta red MTL.

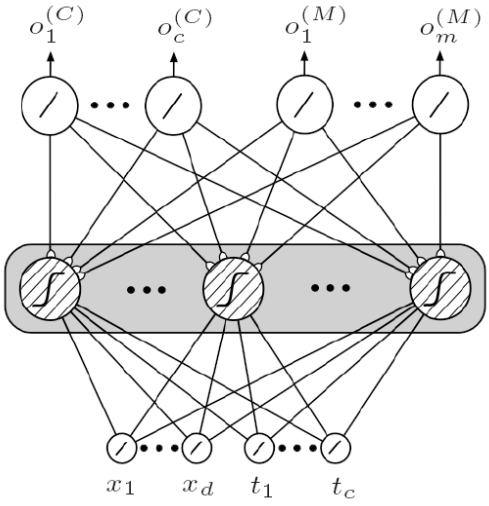
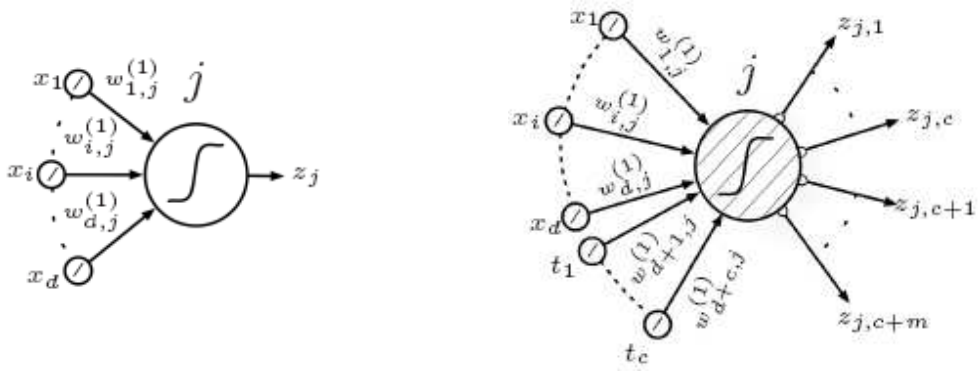


Figura 4-2 Red MTL con una capa oculta común que aprende las tareas de clasificación e imputación al mismo tiempo. Las neuronas ocultas han sido implementadas para evitar mapeados directos entrada-salida. Además, se usan los objetivos de clasificación como entradas extra para ayudar al aprendizaje de las tareas secundarias.

En nuestra notación, $w_{i,j}^{(1)}$ denota un peso de la primera capa, yendo de la unidad de entrada *i* a la unidad oculta *j*, y $w_{0,j}^{(1)}$ denota el sesgo para la unidad oculta *j*. La notación es similar para los pesos en la segunda capa, $w_{j,k}^{(2)}$ denota un peso de la segunda capa, yendo de

la neurona oculta *i* a la unidad de salida *k*, y $w_{0,k}^{(1)}$ denota el sesgo para la unidad de salida *k*. En todas las topologías de redes mostradas en este trabajo, los sesgos estarán implicitos con objeto de simplificar las figuras. Distinguiremos cada tipo de tareas usando un superíndice: $o^{(C)}$ es una salida de clasificación, y $o^{(M)}$ es una salida para aprender una característica con datos perdidos.



a) Neurona clásica b) Neurona para imputación MTL

Figura 4-3 Tipos de neuronas. Primero, en (a), se muestra la clásica neurona que calcula la suma de productos de sus pesos y sus entradas. En (b), está representada la neurona oculta que calcula una suma producto diferente dependiendo de la neurona de salida a la que esté conectada.

A continuación, explicaremos como las neuronas ocultas procesan la información dependiendo de las tareas aprendidas y sus conexiones ponderadas. Emplearemos una neurona oculta diferente a la neurona clásica. Ésta aprende en paralelo varias tareas y calcula diferentes salidas dependiendo de la unidad de salida asociada, en contraste con la clásica neurona oculta que aprende una o varias tareas y calcula la misma salida para todas las unidades de la capa de salida. En la figura 4-3(a) se muestra una neurona clásica cuya salida se puede escribir como

$$z_j = g\left(\sum_{i=1}^{d} w_{i,j}^{(1)} x_i + w_{0,j}^{(1)}\right) \tag{4.1}$$

Estas neuronas trabajan de una forma clásica porque calculan la suma del producto de sus pesos y sus señales de entrada Por otro lado, la figura 4-3(b) muestra la neurona que emplearemos que aprende al mismo tiempo varias tareas, la tarea principal de clasificación y la tarea secundaria de imputación asociada. Este tipo de neuronas están conectadas a

todas sus unidades de entrada y de salida, pero calculan una suma del producto de sus pesos y entradas diferente para cada unidad de salida. Estas neuronas no incluyen la suma producto de la señal de entrada que tienen que aprender como unidad de salida. Esto se hace para evitar conexiones directas entre la entrada a aprender y la salida de imputación asociada a la misma. Para explicarlo, supongamos que estas neuronas tienen que aprender el atributo incompleto a_s del patrón $\mathbf{x}$ y calculan la suma del producto de todas sus entradas y pesos, incluyendo la entrada x_{a_s}. En ese caso, habría una conexión directa, y por lo tanto, la salida de imputación $o_s^{(M)}$ sería directamente dependiente de x_{a_s}. Por esta razón se evitan las conexiones directas entrada-salida durante el cálculo de la suma producto. Además, los objetivos de clasificación son usados como entradas extra para las tareas secundarias, es decir, se usan salidas deseadas de clasificación como guía para el aprendizaje de las tareas de imputación. Siguiendo esto, las salidas de este tipo de neuronas calculan las siguientes expresiones,

Para k=1,...,c

$$z_{j,k} = g\left(\sum_{i=1}^{d} w_{i,j}^{(1)} x_i + w_{0,j}^{(1)}\right) \tag{4.2}$$

Para k=c+1,...,c+m

$$z_{j,k}^{(s)} = g\left(\sum_{\substack{i=1 \\ i+c\neq k}}^{d} w_{i,j}^{(1)} x_i + \sum_{i=1}^{c} w_{d+i,j}^{(1)} t_i + w_{0,j}^{(1)}\right) \tag{4.3}$$

donde *g()* es la función de activación. En la figura 4-3 se muestra una representación de este tipo de neurona. Las dos salidas diferentes $z_{j,k}^{(s)}$ (correspondientes a (4.2) y (4.3)) están marcadas con una flecha que comienza con un círculo para distinguirlas de las salidas típicas de una neurona oculta clásica. También hemos usado esta representación en la Figura 4-2. Finalmente, las salidas de la red para imputación MTL se obtienen mediante una combinación lineal de las salidas de las neuronas ocultas usando una segunda capa de unidades de procesamiento.

Después de presentar el esquema básico MTL para manejar datos incompletos, proponemos otras soluciones siguiendo el esquema MTL básico. Es posible implementar

esquemas MTL más complicados, pero más eficientes, usando subredes privadas o específicas. Un primer método consiste en añadir una subred privada para aprender la tarea principal de clasificación, como podemos ver en figura 4-4.

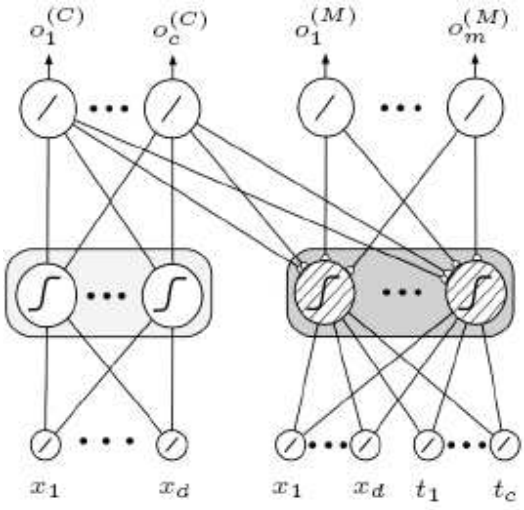


Figura 4-4 Red MTL con una capa oculta común que aprende las tareas de clasificación e imputación al mismo tiempo, y una subred privada usada solamente para la tarea principal.

Esta subred privada está conectada a todas sus entradas pero no a sus entradas extra (los objetivos o “tarjets” de clasificación), y aprende específicamente la tarea principal; por consiguiente, el uso de una subred privada ayuda a obtener una mejor generalización de la tarea de clasificación. Las neuronas ocultas en esta subred trabajan como la neurona clásica mostrada en figura 4-3(a). El tamaño de la subred privada tiene que ser optimizado para conseguir un buen rendimiento para la tarea principal. Por otro lado, el número de neuronas ocultas en la subred común depende de todas las tareas, porque esta subred soporta la transferencia MTL.

Tanto la solución comentada en la figura 4-2 como la que acabamos de ver en la figura 4-4 tienen una subred común que es compartida por todas las tareas, en particular, los pesos comunes de la primera capa. MTL supone que las tareas aprendidas están relacionadas, y estas relaciones son lo que contribuye al aprendizaje. Esta suposición por defecto provoca que las tareas no relacionadas disminuyan el rendimiento de la generalización de todas las tareas en la red MTL causando una pérdida de conocimiento

para algunas tareas. La pérdida de conocimiento no es aceptable y debería ser evitada. Por esta razón, es esencial considerar la relación entre las tareas y como cada tarea es aprendida durante el entrenamiento de las redes MTL, como veremos en la próxima sección, donde explicaremos el entrenamiento y el aprendizaje. Una alternativa a los métodos previos es usar una subred común para cada tarea de imputación, que aprende la tarea principal y la tarea secundaria asociada.

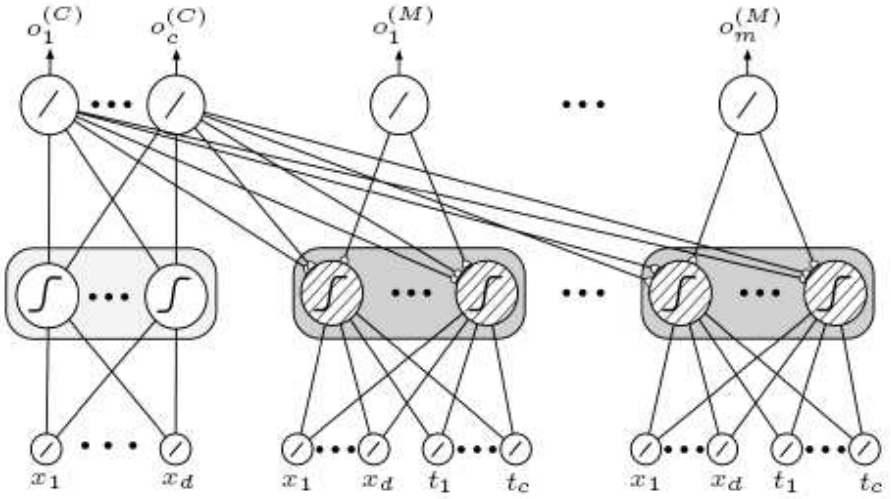


Figura 4-5 Red MTL con una subred privada usada solamente para la tarea principal, y *m* subredes comunes que aprenden las tareas de imputación y la tarea de clasificación al mismo tiempo.

La Figura 4-5 muestra este método con una subred privada dedicada solamente a la tarea principal y *m* subredes usadas por la tarea de clasificación y las tareas de imputación asociadas. De este modo se evita el aprendizaje entre tareas secundarias no relacionadas, y cada una es guiada por el aprendizaje de la tarea de clasificación, es decir, los pesos comunes a cada subred privada están influenciados por el aprendizaje de la tarea principal y la tarea de imputación asociada. La desventaja obvia de este esquema es el incremento del número total de nodos ocultos, y con ello, dimensionar adecuadamente cada subred.

En estos esquemas neuronales, el vector de objetivos total $t^{(n)}$ esta compuesto por el vector objetivo de la tarea de clasificación y las componentes correspondientes a cada tarea de imputación, es decir, por los atributos con algún valor perdido. De este modo, por

ejemplo, el vector objetivo total para un problema de dos dimensiones con valores perdidos en ambos atributos se puede escribir como $\mathbf{t^{(n)}} = \left(\mathbf{t^{(n,C)}}, x_{a_1}^{(n)}, x_{a_2}^{(n)}\right)$. Nótese como las características de entrada con valores perdidos representan los objetivos de las tareas secundarias. Por ejemplo, supongamos un vetor de entrada $x^{(3)}$=(0.1,0.25) cuya salida deseada $t_3^{(C)}$ es igual a -1, entonces, en los esquemas de imputación mediante MTL, sus vector objetivo total es $t^{(3)} = \left(-1, 0.1, 0.25\right)$.

4.2. Inicialización, aprendizaje y operación

Con el objeto de explicar como funcionan estas redes MTL, dividimos la implementación de los esquemas en tres fases:

- *Inicialización.* En esta fase se inicializan los pesos y se normaliza el conjunto de datos de entrada.
- *Aprendizaje*. Se actualizan los pesos y se estiman los datos perdidos.
- *Operación*. Una vez que la red MTL ha sido entrenada, se encarga de clasificar las nuevas muestras.

Todos los esquemas MTL mencionados se pueden implementar usando las tres etapas siguientes de una forma similar.

4.2.1 Inicialización

En la fase de inicialización, antes de comenzar el aprendizaje inicializaremos aleatoriamente todos los pesos con valores del intervalo

$$\left[-\frac{1}{2}\sqrt{\frac{3}{nentradas}}, +\frac{1}{2}\sqrt{\frac{3}{nentradas}}\right] \tag{4.4}$$

donde *nentradas* es el número total de entradas *d*+*c*. Además, en esta fase se normaliza el conjunto de entrenamiento para que tenga media cero y varianza unidad, y después de eso, se fijan los valores perdidos a cero. Con esta inicialización previa a cero pretendemos alterar lo menos posible el proceso de aprendizaje sin entrar en preimputaciones complejas. El pretendido valor neutral del cero se debe a que todos los conjunto de datos con los que

trabajan nuestras redes neuronales han sido previamente normalizados a media cero y varianza uno.

4.2.2 Aprendizaje

Tras la inicialización, pasamos a la *fase de aprendizaje*. El aprendizaje está basado en la definición de una función de error, que es minimizada con respecto a los pesos (y sesgos) de la red. En esta red, usamos la función de error suma de cuadrados, definida como:

$$E = \frac{1}{2}\sum_{n=1}^{N}\left(\left\| \mathbf{o}^{(\mathbf{n,C})} - \mathbf{t}^{(\mathbf{n,C})} \right\|^2 + \sum_{k=1}^{m}\left(o_k^{(n,M)} - x_{a_k}^{(n)} \right)^2 \right) \tag{4.5}$$

donde $\mathbf{o}^{(\mathbf{n,C})}$ y $o_k^{(n,M)}$ son, respectivamente, la salida de clasificación y la salida de imputación *k*-ésima obtenidas por el vector de entrada $\mathbf{x}^{(\mathbf{n})}$. Podemos reescribir (4.5),

$$E = E^{(C)} + E^{(M)} = E^{(C)} + \sum_{k=1}^{m} E_k^{(M)} \tag{4.6}$$

donde $E^{(C)}$ es el error de clasificación y $E^{(M)}$ el error de imputación del atributo a_k. Estas funciones de error dependen de las salidas obtenidas y de las salidas deseadas. Si $x^{(n)}$ presenta valores perdidos, sus vector de objetivos ("tarjets") total estará incompleto. En estos casos no es posible calcular las diferencias para los objetivos de imputación incompletos porque son desconocidos. Por esta razón, las diferencias asociadas a cada objetivo de imputación desconocido se establecen a cero.

Después de obtener las diferencias, se pueden evaluar las derivadas del error *E* con respecto a los pesos, y usar estas derivadas para encontrar los valores de los pesos que minimizan la función de error mediante el método de optimización por gradiente. En particular, usamos el método de gradiente descendente de modo secuencial con tasa de aprendizaje adaptativa y término de momento.

Para explicar el aprendizaje, primero es necesario distinguir entre pesos de la capa de salida $w_{i,j}^{(2)}$, que solamente están influenciado por la tarea a la que están conectados, y pesos de la capa de entrada, $w_{i,j}^{(1)}$. Estos pesos de la primera capa se pueden dividir en dos grupos: pesos asociados a una subred privada y pesos asociados a la subred común. Los

pesos de la subred privada solamente están influenciados por la función de error $E^{(c)}$, mientras que los pesos de cada subred común están influenciados por los errores en las tareas a las que contribuyen a aprender. Por ejemplo, en la figura 4-2 suponemos que todos los atributos están incompletos ($m=d$), en este caso, los pesos de la primera capa asociados con la entrada x_1 son actualizados solamente por los errores de todas las unidades de la salida $o_1^{(M)}$, porque x_1 no contribuye directamente al aprendizaje de estas tareas de imputación. Por este motivo, podemos dividir los pesos en diferentes grupos, dependiendo del error de la tarea que provoca su actualización, y de paso, usamos una tasa adpatativa de aprendizaje η_g diferente para cada uno de estos grupos de pesos.

Durante el aprendizaje, los pesos de la capa *l*-ésima son actualizados iterativamente como sigue,

$$w_{i,j}^{(l)}(\tau+1) = w_{i,j}^{(l)}(\tau) + \Delta w_{i,j}^{(l)}(\tau) \tag{4.7}$$

donde τ indica el paso de iteración. Cuando se usa el método de gradiente descendente con término de momento, la actualización de los pesos viene dada por

$$\Delta w_{i,j}^{(l)}(\tau) = -\eta_g \nabla E\,|_{w_{i,j}^{(l)}(\tau)} + \mu \Delta w_{i,j}^{(l)}(\tau-1) \tag{4.8}$$

donde η_g es la tasa de aprendizaje asociada con el grupo de pesos al que pertenece $w_{i,j}^{(l)}$, y μ es el parámetro de momento. Con $\Delta E\,|_{w_{i,j}^{(l)}(\tau)}$ denotamos el gradiente de *E* evaluado en $w_{i,j}^{(l)}$.

Otra cuestión importante es que los valores incompletos son estimados usando las salidas de imputación durante la etapa de entrenamiento. El aprendizaje de la tarea de clasificación afecta a estos valores imputados, y por lo tanto, la imputación está orientada a resolver la tarea de clasificación. La imputación se realiza cuando el aprendizaje de las tareas de imputación se empieza a detener.

4.2.3 Operación

Por último nos queda la *fase de operación.* La operación de este método depende de la presencia de valores perdidos en $\mathbf{x}^{(n)}$. Si $\mathbf{x}^{(n)}$ es completamente conocido, no es necesaria imputación, y la red MTL clasifica directamente la muestra de entrada usando la salida de clasificación $\mathbf{o}^{(n,C)}$. Si por el contrario $\mathbf{x}^{(n)}$ tiene datos incompletos, se emplean las salidas de imputación $\mathbf{o}_k^{(n,M)}$ para estimar los datos perdidos. Estas funciones de imputación son función de $\mathbf{t}^{(n,C)}$ como parte de la entrada. Sin embargo, esta información no está disponible en modo de operación. Con el propósito de resolver este problema, chequeamos todos los posibles valores $\mathbf{t}^{(n,C)}$ y seleccionamos el más consistente. La consistencia de $\mathbf{t}^{(n,C)}$ es una medida de la diferencia entre $\mathbf{t}^{(n,C)}$ y $\mathbf{o}^{(n,C)}$ producida por la red después de que se haya realizado la imputación de los valores perdidos usando el correspondiente $o_k^{(M)}$.

4.3. El problema de la calidad de imputación

El método descrito en los apartados anteriores supone un avance en la resolución de problemas de clasificación con conjuntos de datos incompletos. Para lograrlo se basa en el empleo de neuronas comunes para aprender las tareas de imputación y de clasificación. De este modo las imputaciones que realiza la red (tareas secundarias) estarán orientadas a la resolución del problema de clasificación (tarea principal). Su aplicación se ha llevado a cabo con éxito en varios tipos de problemas, pero sin embargo parece que con otros problemas su funcionamiento se podría mejorar. Veámoslo con un ejemplo.

Imaginemos el problema de clasificación de dos clases ilustrado en la figura 4-6. Una de las clases, que hemos coloreado de rojo, consta de un conjunto de muestras agrupado, mientras que para la clase coloreada de azul, podemos observar que las muestras de esa clase se dividen en dos subgrupos, uno centrado en (0.5,0) y el otro en (1,0). Si este conjunto de entrenamiento está completo, y empleamos una red neuronal clásica (como puede ser el perceptrón multicapa) para aprender la tarea de clasificación, obtendremos un resultado satisfactorio. Sin embargo el problema que pretendemos resolver tiene algunos

datos perdidos en la dimensión *x*, y por lo tanto, no podemos aplicar un MLP de forma tradicional.

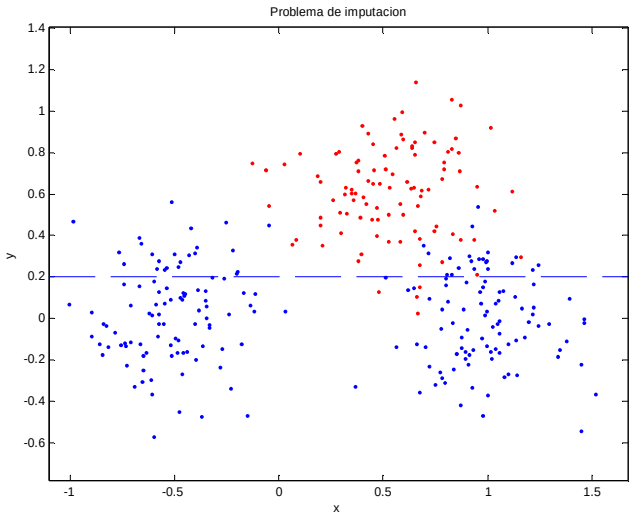


Figura 4-6 En esta figura se muestra el caso en el que una clase está dividida en varios grupos, y debido a la existencia de datos perdidos, la red no tiene información suficiente para decidir a que grupo pertenece. La recta horizontal indica la posición en la que puede estar el dato (?, 0,2).

Intentaremos, por tanto, aplicar una de las redes de imputación MTL explicadas en 4.1. Independientemente de la configuración de la capa oculta, tendremos una salida dedicada a la tarea principal y otra salida dedicada a la estimación de la única característica con valores perdidos, la dimensión *x*. Centrémonos en la tarea de imputación. Como hemos comentado anteriormente, este aprendizaje de la estimación de la dimensión *x*, solo se llevará a cabo para aquellas muestras del conjunto de entrenamiento de las que conozcamos el valor de *x*, ya que si no, no podríamos saber si estamos realizando bien la tarea de imputación. Durante la fase de entrenamiento, las únicas entradas conectadas (previo paso por la capa oculta) a esta salida serán la etiqueta de clase *t* (+1 para la clase azul y -1 para la clase roja) y los valores de la dimensión *y*. A partir de estas dos entradas, la red debe ser capaz de aprender a estimar un valor para *x*. Y aquí llega el problema: ¿qué ocurre cuando nos llega un dato de la clase azul? Supongamos que se trata del dato (0.5, 0.2). Puesto que el valor de la dimensión *x* se empleará para calcular la función de error, el problema al que

se enfrentará nuestra red consistirá en estimar un valor para la dimensión *x* sabiendo que pertenece a la clase azul y que la dimensión *y* vale 0.2. Ilustramos la situación en la figura 4-6.

Como se puede observar, la red no dispone de ningún valor de utilidad para decidir a qué grupo de la clase azul destinar la imputación. Si nos fijamos en la función de error que se emplea para guiar el aprendizaje (4.5), vemos que el error depende del cuadrado de la distancia del valor imputado al valor real. De acuerdo con esta fórmula, si nuestra red imputase todos los valores de *x* al grupo de la izquierda, para aproximadamente la mitad de las muestras esta diferencia sería pequeña, aproximémosla a 0, mientras que para las muestras pertenecientes al grupo de la derecha la contribución a la función de error sería del orden de la distancia entre ambos grupos elevada al cuadrado, la denotaremos por D^2. Una situación análoga ocurre para las muestras que pertenezcan al grupo de la derecha. Por lo tanto la suma de las contribuciones a la función de error de todas las muestras sería aproximadamente $(0+D^2)N/2+(D^2+0)N/2=D^2N$. Si por el contrario, la red imputa entre los dos grupos, la contribución a la función de error sería aproximadamente: $N\left(D/2\right)^2=ND^2/4$. Puesto que el valor que toma la función de error es menor en la zona que separa ambos grupos que dentro de cualquiera de ellos, la minimización de la función de error guiará a la red hacia una imputación entre las dos nubes y, teniendo en cuenta la distribución de los datos, no hubiese podido alejarse más del resultado pretendido.

Fue este problema el que motivó el nacimiento de este trabajo. Se pensó que si fuésemos capaces de modelar la función de densidad de probabilidad del problema, podríamos incorporarla a la función de error para, evitar imputaciones a zonas claramente improbables, como la que hemos descrito. Inmediatamente se pensó en el modelado de mezclas de gaussianas algoritmo EM, debido a su capacidad para estimar densidades de probabilidad de conjuntos datos que presentan valores perdidos. Además, ofrece una fdp continua y derivable, a diferencia de otros métodos (como le histograma), lo que favorece su incorporación a la función de error que se minimiza mediante la red MTL.

4.4. Incorporación de EM a la imputación MTL

Una vez identificado el problema y decidida la dirección hacia la solución, falta trazar el camino que nos lleve a ella. En principio, esta solución consistiría en estimar la fdp de los datos e implicarla en el cálculo de la función de error para dirigir las imputaciones a zonas pobladas. Con esto pretendemos evitar que la red MTL realice imputaciones no consistentes (improbables) teniendo en cuenta la fdp obtenida mediante la mezcla de gaussianas entrenada con EM.

El objetivo es ponderar las diferencias de las funciones de error asociadas a las tareas de imputación. Si la función de error asociada a la tarea de imputación de la característica incompleta a_k es

$$E_k^{(M)} = \frac{1}{2}\sum_{n=1}^{N}\left(o_k^{(n,M)} - x_{a_k}^{(n)}\right)^2 \tag{4.9}$$

se pretende ponderar las diferencias calculadas para cada patrón, de tal manera que se eviten imputaciones improbables según la densidad de probabilidad estimada. Para ello, es necesario incluir un factor que modifique convenientemente las diferencias en función de como probable sea el valor estimado por la red MTL. A continuación se explica la solución que se ha propuesto en este PFC.

4.4.1 Solución propuesta

El primer paso a seguir consiste en emplear el algoritmo EM para, a partir de todo el conjunto de entrenamiento (muestras con valores perdidos incluidos), modelar la función de densidad de probabilidad de los datos por medio de una mezcla de gaussianas. A la hora de realizar el modelado, se empleará un conjunto de validación que nos permitirá seleccionar el número de componentes de la mezcla, así como también detener el aprendizaje para evitar el sobreentrenamiento.

Posteriormente, con la mezcla ya estimada, se calcula la función de densidad de probabilidad de cada muestra, pero no sería éste el valor que añadiremos a la función de error (4.9). El factor que queremos incluir en la función de error debe variar de manera

contraria a la función de probabilidad, ya que pretendemos que cuando la probabilidad sea grande, el factor sea próximo a cero (lo que hará disminuir el error), y que cuando la probabilidad sea pequeña, este factor crezca. También hay que tener en cuenta que, para ciertas muestras, puede que la función de densidad de probabilidad máxima entre cualquier estimación posible sea muy pequeña, mientras que para otras muestras, la función de densidad de la estimación alcance un valor muy grande. Tendremos que incluir un factor de normalización, que será el máximo valor que pueda tomar la función de densidad de $\mathbf{x^{(n)}}$, fijadas todas las dimensiones excepto aquella cuya tarea de estimación estemos aprendiendo (dimensión a_k). Finalmente, el factor que decidimos incluir en la función de error es:

$$1-\frac{p(o_k^{(n,M)} \mid \mathbf{x}=\mathbf{x^{(n)}})}{\max\, p_{a_k}(x_{a_k} \mid \mathbf{x}=\mathbf{x^{(n)}})} \tag{4.10}$$

Cuando la característica a_k toma el valor estimado $o_k^{(n,M)}$ condicionado a que el resto de atributos tomen los valores del vector de entrada *n*-ésimo, es decir, $x_1 = x_1^{(n)}, ..., x_{k-1} = x_{k-1}^{(n)}, x_{k+1} = x_{k+1}^{(n)}, ..., x_d = x_d^{(n)}$

y $p(o_k^{(n,M)} \mid \mathbf{x}=\mathbf{x^{(n)}})$ es la el valor de la densidad de probabilidad estimada con la mezcla de gaussianas. Lo incorporaremos a la función de error mostrada en (4.5) de la siguiente forma

$$E=\frac{1}{2}\sum_{n=1}^{N}\left(\left\|o^{(n,C)}-t^{(n,C)}\right\|^2+\sum_{k=1}^{m}\left(\left(o_k^{(n,M)}-x_{a_k}^{(n)}\right)\left(1-\frac{p(o_k^{(n,M)} \mid \mathbf{x}=\mathbf{x^{(n)}})}{\max\, p_{a_k}(x_{a_k} \mid \mathbf{x}=\mathbf{x^{(n)}})}\right)\right)^2\right) \tag{4.11}$$

Con esta modificación, conseguimos que la función de error disminuya considerablemente si nos acercamos a zonas donde sea muy probable que exista un dato. Sin embargo, aún podemos afinar más en la estimación de la función de densidad de probabilidad, ya que el algoritmo EM que hemos implementado se puede entrenar con datos etiquetados. La estimación con datos etiquetados implica el uso de una mezcla de funciones gaussianas (para la situación de las muestras) y multinomiales (para la información de clase). De este modo, podemos hacer una estimación de la función de densidad de probabilidad que no solo dependa de todas las dimensiones de la muestra excepto de aquella cuya tarea de imputación queramos aprender, sino también de la clase a la que pertenezca esa muestra. Introduciendo esta nueva información en (4.11), la función de error quedaría

$$E = \frac{1}{2}\sum_{n=1}^{N}\left(\left\|o^{(n,C)} - t^{(n,C)}\right\|^2 + \sum_{k=1}^{m}\left(\left(o_k^{(n,M)} - x_{a_k}^{(n)}\right)\left(1 - \frac{p(o_k^{(n,M)} \mid \mathbf{x} = \mathbf{x^{(n)}}, \mathbf{t} = \mathbf{t^{(n)}})}{\max\, p_{a_k}(x_{a_k} \mid \mathbf{x} = \mathbf{x^{(n)}}, \mathbf{t} = \mathbf{t^{(n)}})}\right)\right)^2\right) \quad (4.12)$$

En el capítulo siguiente se mostrarán las ventajas que aporta esta modificación y sus características principales.

Capítulo 5

Resultados

En este capítulo se mostrarán los resultados obtenidos en distintos experimentos. Por un lado analizaremos el comportamiento del algoritmo EM sobre ciertos problemas para entender su funcionamiento y posibles aplicaciones. Por otro lado, veremos como la aplicación del algoritmo EM, junto con una red MTL, como la explicada en capítulo 4, servirá para mejorar las prestaciones de ésta.

5.1. Aplicaciones de EM

5.1.1 Experimento 1. Nubes gaussianas

Este primer experimento nos servirá para comprobar la capacidad del algoritmo EM para modelar la densidad de probabilidad de un conjunto de datos por medio de una mezcla de gaussianas.

El conjunto de datos bidimensional, que podemos observar en la figura 5-1, ha sido generado por una mezcla de gaussianas original cuyos parámetros de medias, covarianzas y proporciones de mezcla hemos elegido arbitrariamente. Puesto que el algoritmo EM también emplea una mezcla de gaussianas para modelar la función de densidad de

probabilidad de los datos, el resultado deseado sería que esta mezcla generada por EM exhibiese los mismos parámetros que la mezcla original, con la que generamos los datos.

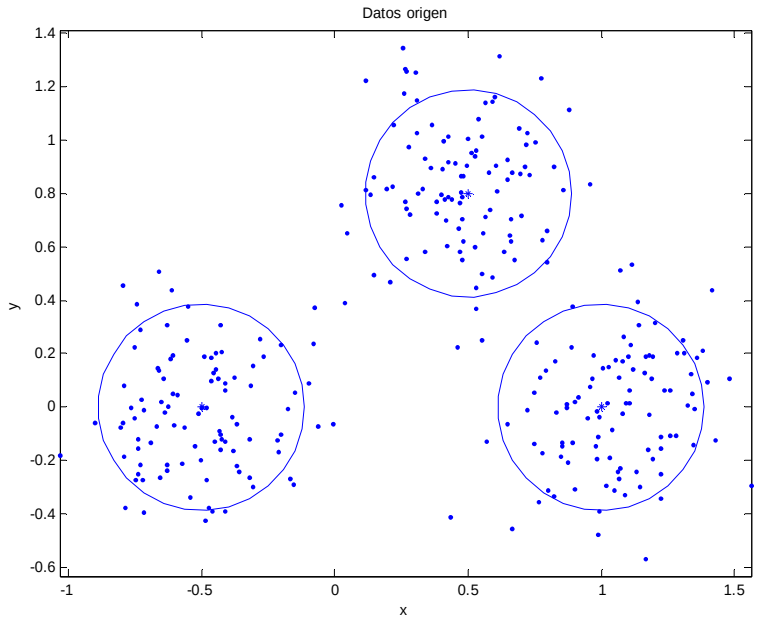


Figura 5-1 Conjunto de datos de entrenamiento generado a partir de una mezcla de gaussianas original. La media de las componentes está marcada con un "*" y las circunferencias indican la forma que tiene la matriz de covarianza.

Para facilitar la comparación entre ambas mezclas, fijamos el número de componentes a emplear por EM a tres, al igual que en la mezcla original. Inicializaremos las matrices de covarianza a matrices identidad, y para fijar un valor inicial de la media de las componentes, emplearemos el algoritmo k-medias [45]. El algoritmo EM irá modificando las medias, covarianzas y proporciones de mezcla de dichas componentes para maximizar la verosimilitud sobre el conjunto de datos. En la tabla 5-1 podemos ver los parámetros de la mezcla original, mientras que en la tabla 5-2 vemos los estimados por EM. Sin embargo, la mejor forma de comparar las similitudes de la estimación con la realidad es observando la figura 5-2.

Puesto que los parámetros de ambas mezclas son similares, podríamos generar un nuevo conjunto de datos a partir de la mezcla de gaussianas estimada. Esto es importante, porque quiere decir, que hemos resumido con 21 valores (3 para las proporciones, 6 para las medias y 12 para las matrices de covarianza) un conjunto de 300 muestras de dos dimensiones, es decir, 600 valores.

Mezcla original:

Proporción	μ_x	μ_y	Σ_{xx}	Σ_{xy}	Σ_{yx}	Σ_{yy}
0.33	-0.5	0	0.05	0	0	0.05
0.33	1	0	0.05	0	0	0.05
0.33	0.5	0.8	0.05	0	0	0.05

Tabla 5-1 Parámetros de la mezcla de gaussianas que hemos usado para generar los datos.

Mezcla estimada:

Proporción	μ_x	μ_y	Σ_{xx}	Σ_{xy}	Σ_{yx}	Σ_{yy}
0.34	-0.5	-0.02	0.04	0	0	0.05
0.33	1.05	-0.01	0.04	0	0	0.04
0.33	0.48	0.82	0.04	0.01	0.01	0.05

Tabla 5-2 Parámetros de la mezcla de gaussianas que el algoritmo EM ha estimado.

Posteriormente, y como ilustración del concepto de sobreentrenamiento del que hablábamos en 2.5.11.2, podemos ver en la figura 5-3 la evolución de la función de verosimilitud para el conjunto de entrenamiento y de validación. En ella se puede observar el pico de sobreentrenamiento en la época[7] 9.

[7] Recordemos que denotamos con época una iteración del algoritmo sobre todo el conjunto.

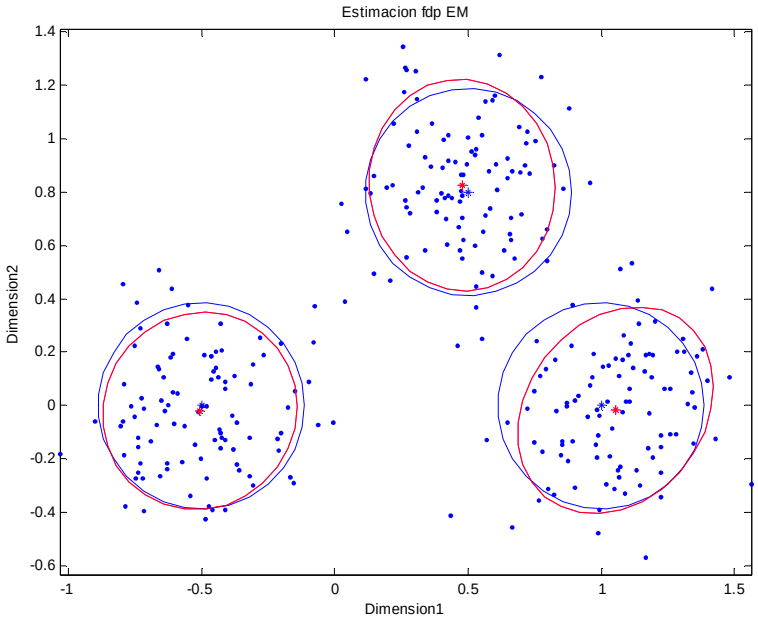


Figura 5-2 Sobre el conjunto de entrenamiento hemos pintado las dos mezclas de gaussianas: la original en azul y la estimada en rojo.

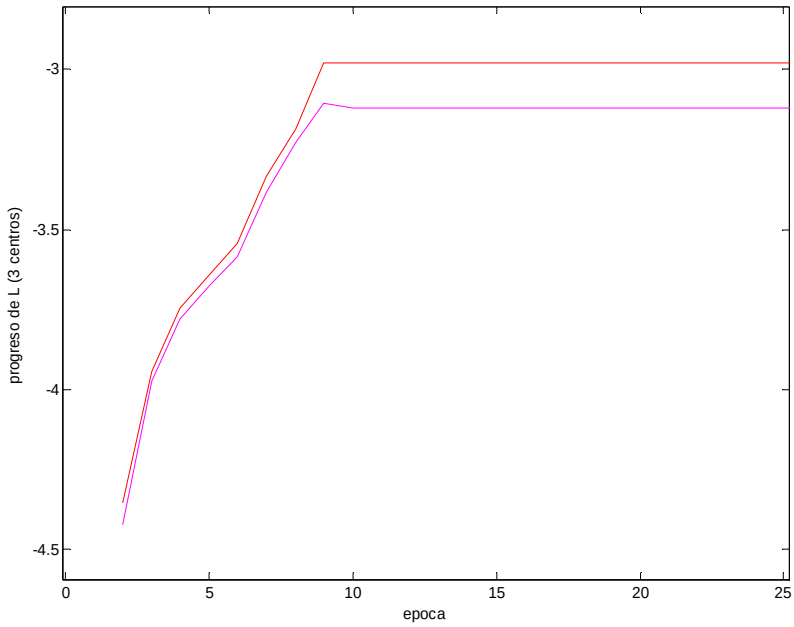


Figura 5-3 Evolución de la media del logaritmo de la verosimilitud para todas las muestras. En rojo, para el conjunto de entrenamiento, y en magenta para el de validación.

5.1.2 Experimento 2. Anillo

En el experimento anterior hemos creado los datos a partir de una mezcla de gaussianas, lo que ha supuesto que la estimación sea muy sencilla para el algoritmo EM. Con este nuevo problema pretendemos analizar el comportamiento del algoritmo frente a funciones de densidad más complejas. Además, veremos la influencia que puede tener el número de componentes en la estimación.

En definitiva, en este problema, comprobaremos la limitación del uso de mezclas para modelar funciones de densidad. Como sabemos, los métodos semiparamétricos como la mezcla de gaussianas nos permiten una flexibilidad intermedia entre los métodos paramétricos y no paramétricos. Pues bien, esta flexibilidad intermedia puede no ser suficiente como demostrará la representación en 3D de la función de densidad de probabilidad estimada.

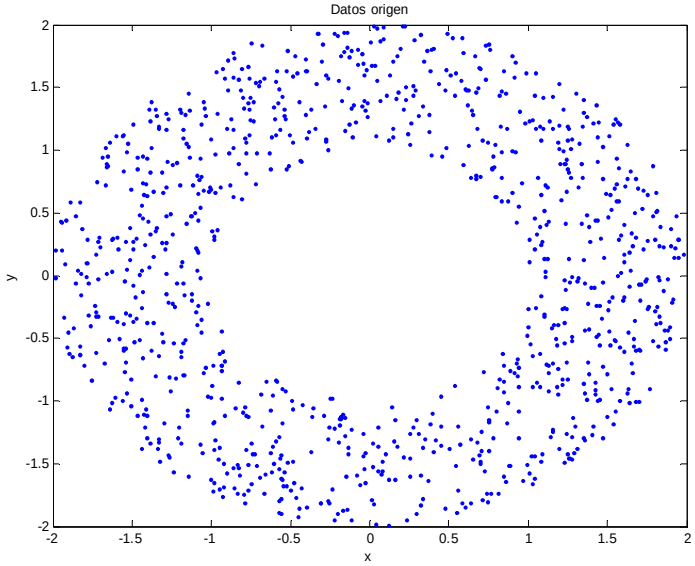


Figura 5-4 Conjunto de entrenamiento en forma de anillo

El conjunto de datos de entrenamiento se distribuye de forma uniforme a lo largo de un anillo centrado en 0, y cuyos radios interior y exterior valen respectivamente 1 y 2. En la figura 5-4 podemos observar este conjunto, y en la figura 5-5 podemos ver su función de

densidad de probabilidad correspondiente. La particularidad de esta forma es que mientras la media de los datos cae en el centro del anillo, la fdp en ese punto deberá ser cero, así como en su entorno más próximo, hasta llegar al radio interior del anillo. Además, tenemos zonas de fuertes discontinuidades seguidas de zonas donde la función de densidad de probabilidad debería ser plana.

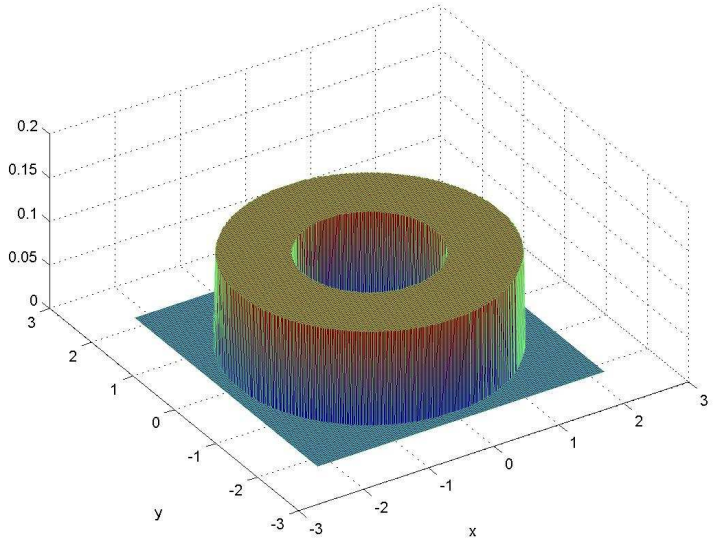


Figura 5-5 FDP real del conjunto de entrenamiento

Cuando recibimos un conjunto de datos no sabemos cual es el número de componentes óptimas a emplear para modelar la densidad de probabilidad que se nos plantea, surge entonces el problema de elegir un número determinado de componentes. Intuitivamente sabemos que cuanto más complicada sea la forma de la densidad de probabilidad a estimar, será necesario un mayor número de componentes, sin embargo, a menudo no somos conscientes de la complejidad de la función de densidad, ya que podemos tratar con conjuntos de tamaño considerable tanto en número de casos como en número de variables. Como comentábamos en 2.5.11.4, solucionaremos este problema realizando un barrido de componentes, es decir, probaremos con un rango de valores dado un número inicial y un número máximo de componentes (queda a discreción del usuario la elección de estos valores en función de su criterio personal).

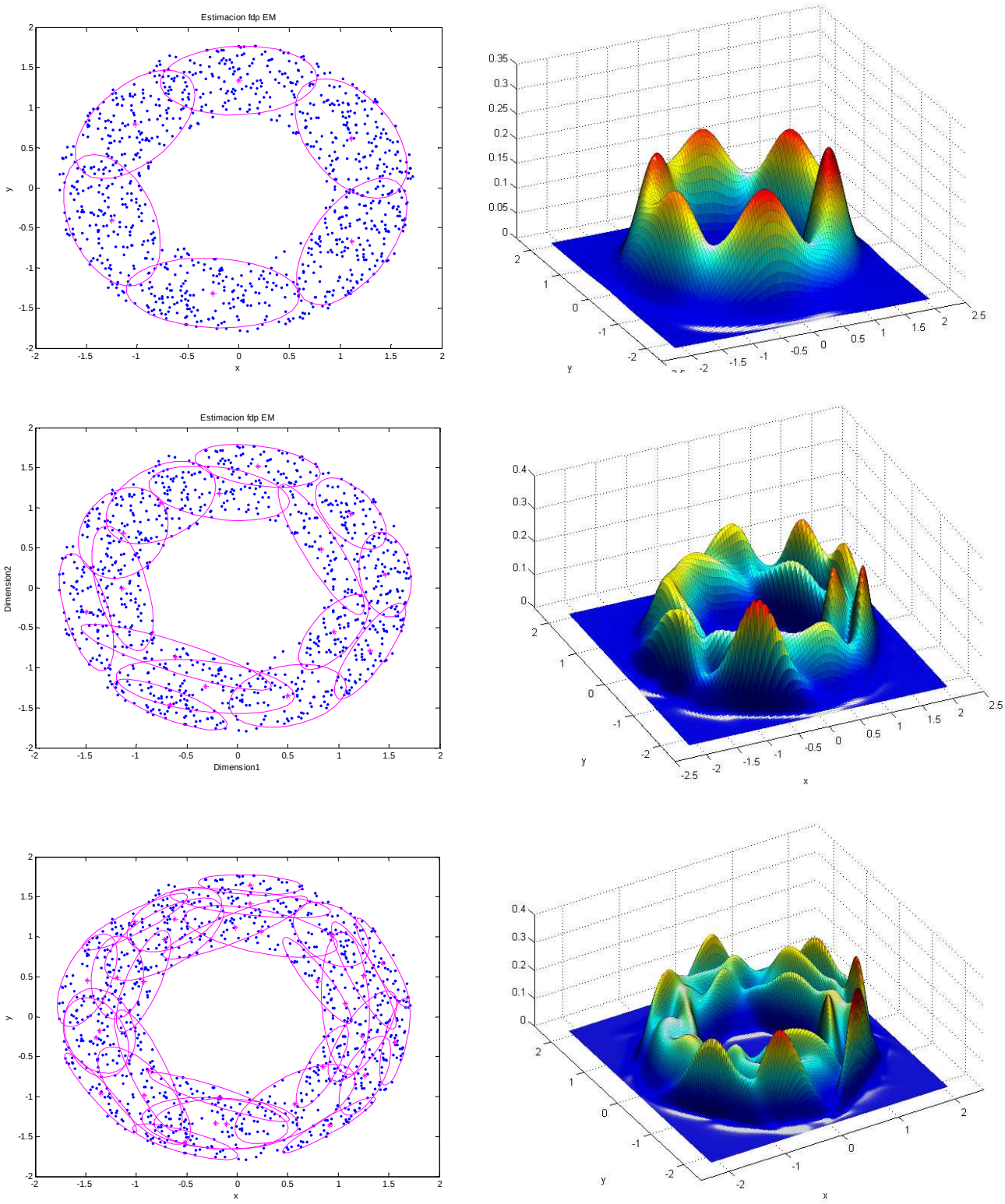


Figura 5-6 En estos gráficos podemos apreciar la configuración de las mezclas gaussianas para (de arriba a abajo) 6, 15 y 25 componentes.

figura 5-6 En la figura 5-6 hemos comprobado que las mezclas de gaussianas tienen dificultades para modelar la superficie plana elevada del anillo, y que el aumento del número de componentes no mejora claramente este modelado, y además provoca un incremento considerable del tiempo de procesado. Por tanto, el uso de núcleos gaussianos no es adecuado para modelar esta distribución. Esto podría solucionarse escogiendo adecuadamente el tipo de núcleo o "kernel" que se usa en la estimación de la *fdp*.

Además. añadir un número excesivo de componentes cuando el conjunto de entrenamiento no es suficientemente grande, puede provocar un sobreajuste. Esto se puede apreciar en la figura 5-7(b), donde añadir más componentes (hasta 25) conlleva una pérdida de generalización y un sobreajuste del conjunto de datos, con respecto a una mezcla con menos componentes figura 5-7(a).

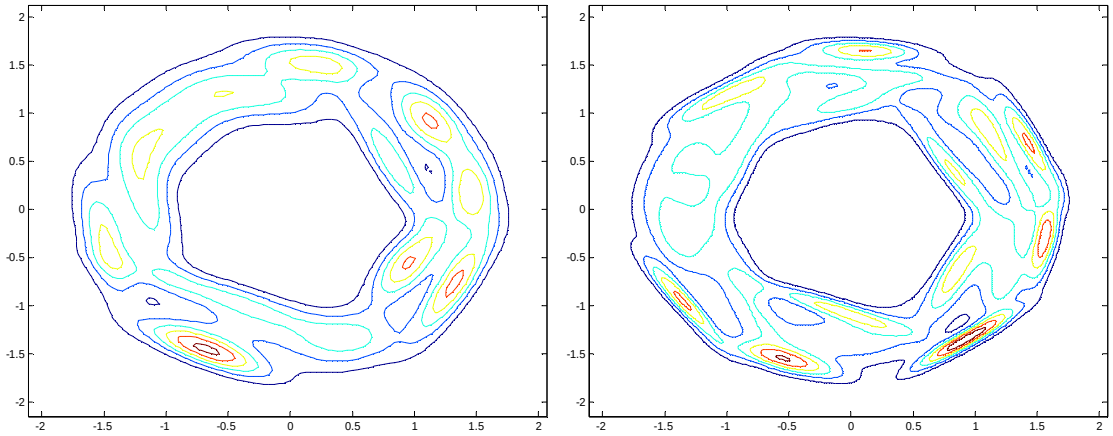


Figura 5-7 Curvas de contorno para las mezclas de 15 (izquierda) y 25 (derecha) componentes.

Sin embargo, aunque el modelado de la función de densidad de probabilidad no sea muy preciso, si nos puede ayudar para, por ejemplo, el propósito comentado en el apartado 4.3. Lo que pretendíamos con la estimación de la función de densidad de probabilidad era guiar las imputaciones de una red MTL hacia zonas probables, y alejarlas de las zonas poco pobladas. En un problema como el del anillo, la red neuronal de aprendizaje MTL combinado de clasificación e imputación (capítulo 4), no proporcionaría un resultado

satisfactorio, ya que imputaría los datos perdidos al centro del anillo. En este caso, los resultados del algoritmo EM, por pobres que sean, pueden ayudar a la red MTL.

5.1.3 Experimento 3. Aproximación de funciones

Como describíamos en el apartado 2.5.9, una de las posibles aplicaciones del algoritmo EM es la aproximación de funciones. Mediante este experimento sencillo, ilustraremos esa capacidad.

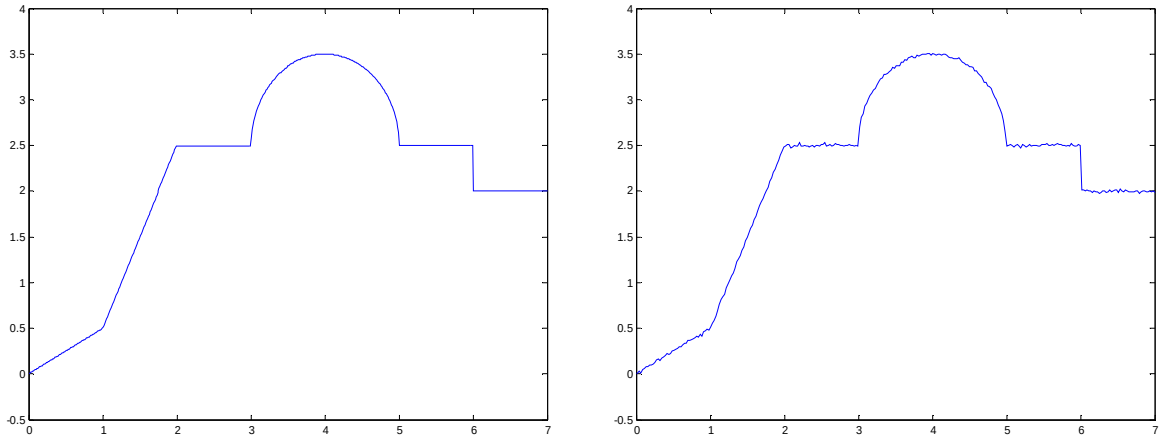


Figura 5-8 Función que queremos aproximar. A la izquierda la función original, a la derecha la función más un pequeño factor de ruido para evitar alineaciones perfectas.

El conjunto de datos estará generado a partir de una función arbitrariamente elegida para este experimento. En la función se alternan tramos rectos, curvos y discontinuidades. La mezcla de gaussianas es un elemento adecuado para tratar con estas discontinuidades, ya que se puede asignar una o varias componentes a cada tramo. En la figura 5-8, a la izquierda, vemos la función que hemos creado. Sin embargo, y he aquí un inconveniente del algoritmo EM, no podemos estimar tramos perfectamente rectos (ni puntos, ni planos perfectos). Cuando una componente intenta modelar un tramo recto, se sitúa a lo largo del mismo, y su matriz de covarianza empieza a decrecer de forma que los elementos de su diagonal tienden a cero. Y es aquí donde surge el problema, podemos tratar con datos muy pequeños, pero cuando comienzan a hacerse infinitesimales, los programas de cálculo ven desbordada su capacidad, y la respuesta es imprevisible. Para solucionar este problema, de una forma práctica y sencilla, añadiremos una componente muy pequeña de ruido a los

datos, para que su alineación no sea perfecta. El resultado lo podemos ver en la figura 5-8, a la derecha. Como se puede ver, la componente de ruido es muy reducida.

El conjunto de entrenamiento consistirá en un barrido de 300 muestras a lo largo del eje *x* con sus correspondientes valores de *y=f(x)*. Para estimar los tramos rectos hubiese sido suficiente un número de muestras bastante menor, pero puesto que la función consta de una semicircunferencia, es preferible emplear un buen número de muestras para que la dibujen con precisión. Como no estamos seguros de cuantas componentes serán necesarias, vamos a hacer un barrido del número de componentes de las mezclas, de 1 a 25.

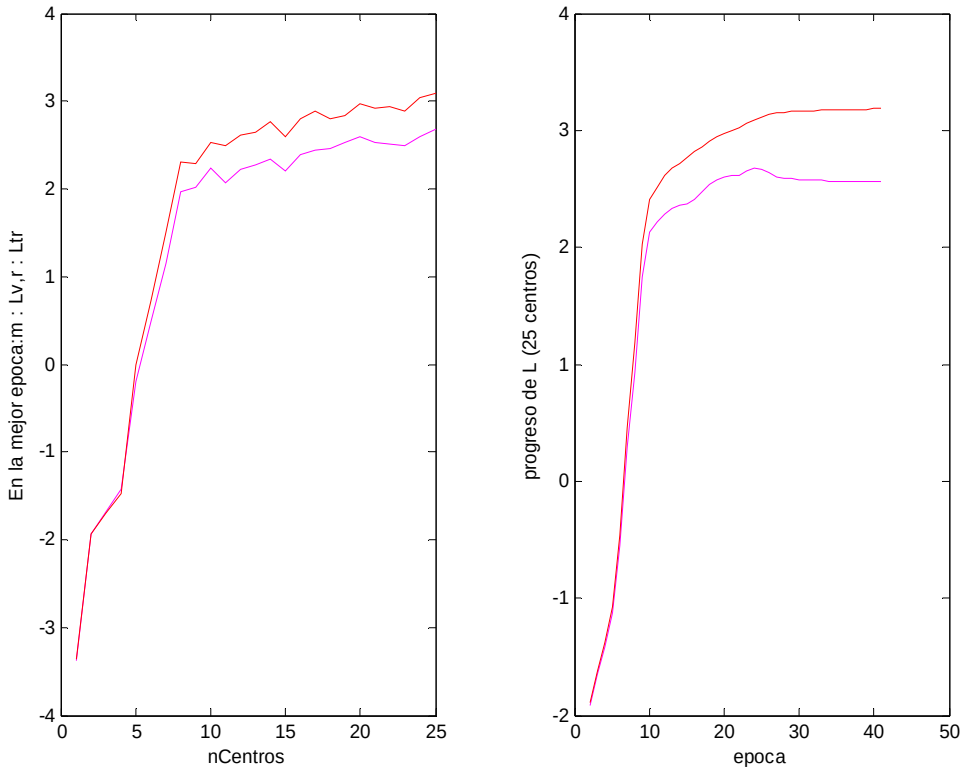


Figura 5-9 Evolución de la verosimilitud en función del número de componentes (izquierda) y, para el número de componentes que haya ofrecido el mejor resultado, evolución de la verosimilitud con respecto a las épocas (derecha) . En rojo conjunto de entrenamiento, y en magenta conjunto de validación.

Por último, una vez estimada la mezcla, que será la mejor, en términos del conjunto de validación, entre todos los números de componentes probados, la usaremos para estimar la función *f(x)* de un conjunto de muestras cuya dimensión *x* está distribuida uniformemente a lo largo del eje, es decir, intentaremos aproximar la función.

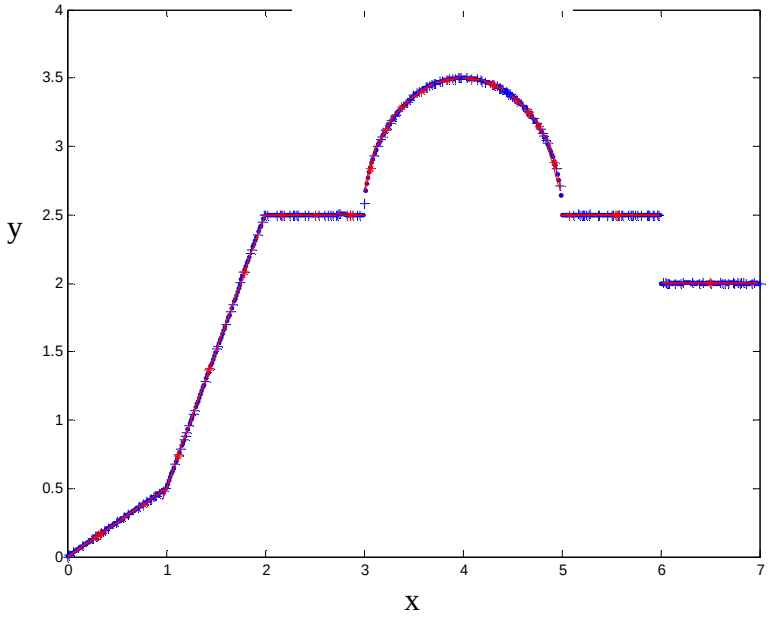


Figura 5-10 Estimación de la función. En azul los puntos de los conjuntos de entrenamiento (puntos) y de validación (cruces), y en rojo la distribución de la mezcla de gaussianas tras los cálculos de EM. Las elipses rojas representan las componentes de la mezcla de gaussianas.

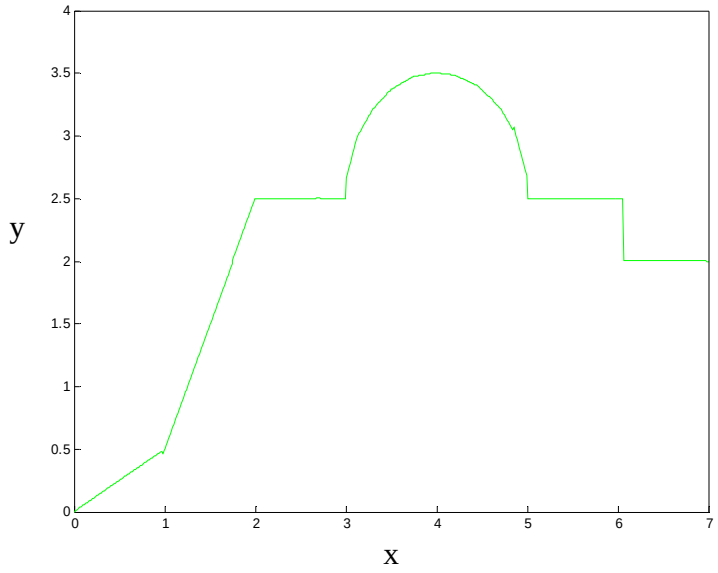


Figura 5-11 Aproximación de la función original llevada a cabo por EM.

5.1.4 Experimento 4. Datos censurados

En el capítulo 2 discutíamos la capacidad de los algoritmos basados en MLE frente a otras estrategias de aprendizaje más simples. Más en concreto, señalábamos la existencia de un conjunto de problemas en los que los algoritmos Bayesianos y de máxima verosimilitud ofrecían respuestas bastante más acertadas que los mecanismos más simples. Nos referíamos al conjunto de problemas MAR (missing at random). En estos problemas, la probabilidad de que $x_i^{(n)}$ esté perdido es independiente de los valores de las dimensiones perdidas de los datos, pero puede depender de los valores de las dimensiones observadas. En estos casos decimos que los datos sufren una censura cruzada, aunque para este ejercicio hablaremos simplemente de censura. Veamos un ejemplo de este tipo en la figura 5-12.

Los problemas censurados son aquellos en los que a todas las muestras que cumplen un determinado requisito presentan información incompleta. En nuestro caso hemos eliminado la información de la dimensión *y* a todos aquellos datos cuyo valor en la dimensión *x* sobrepasaba un umbral determinado (x^{umbral} = 4.5). Los datos completos fueron generados a partir de una gaussiana con media

$$\mu = \begin{pmatrix} 5 \\ 5 \end{pmatrix} \tag{4.13}$$

y matriz de covarianza

$$\Sigma = \begin{pmatrix} 5/4 & 9/4 \\ 9/4 & 17/4 \end{pmatrix} \tag{4.14}$$

Las muestras con valores *y* perdidos son representados por círculos huecos en la recta *y*=0.

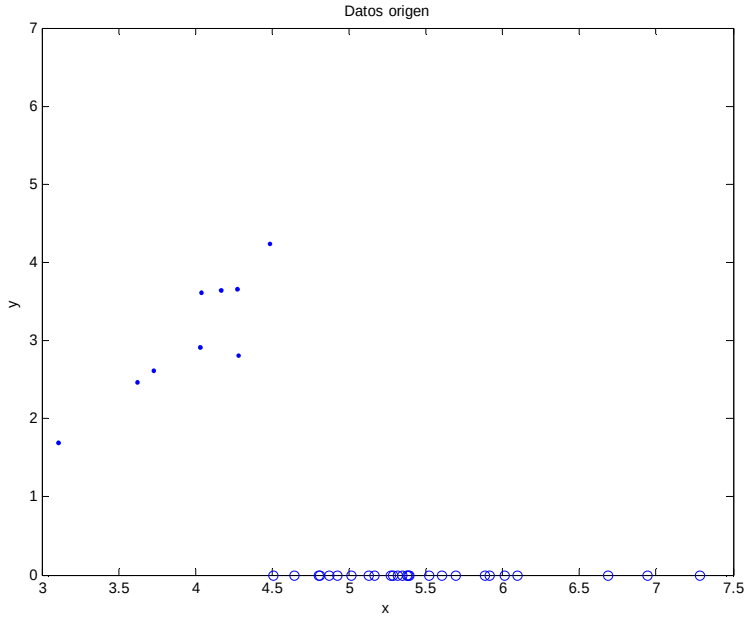


Figura 5-12 Problema de datos censurados. Los puntos indican muestras observadas, y los círculos muestras con valores perdidos en la dimensión *y*. Mecanismo de pérdida: todas las muestras cuyo valor en *x* supere 4.5 pierden su valor en *y*.

Imaginemos que queremos estimar la media (μ_x, μ_y) y la matriz de covarianza Σ de una distribución normal de dos dimensiones, a partir de un conjunto de datos $\mathbf{X} = \{(x^n, y^n)\}$ $n=1...N$ donde algunas de los valores observados de y^n están perdidos, como se puede ver en figura 5-12. Si estimamos μ_x a partir de la media de x^n y μ_y mediante la media de los valores observados de y^n, estimaremos a la baja μ_y ya que hemos ignorado la estructura de covarianza en los datos observados. Un método heurístico más inteligente consistiría en usar la estructura de covarianza para rellenar los valores de las características perdidas de y^n

mediante la regresión de ellas basada en x^n. De cualquier modo, incluso este método heurístico nos llevaría a una estimación sesgada de la matriz de covarianza, ya que los datos rellenados caerían a lo largo de la línea de regresión. Ambas técnicas de rellenado, conocidas como imputación a la media e imputación por regresión, nos ofrecen resultados insatisfactorios incluso en este problema simple de estimación de parámetros. Veamos ahora como actúa EM ante este problema.

El algoritmo EM tiene en cuenta los valores perdidos, pero no usa sus estimaciones para calcular la matriz de covarianza, lo que le permite a la vez captar la estructura de covarianza de las muestras y producir una estimación de la matriz de covarianza no sesgada.

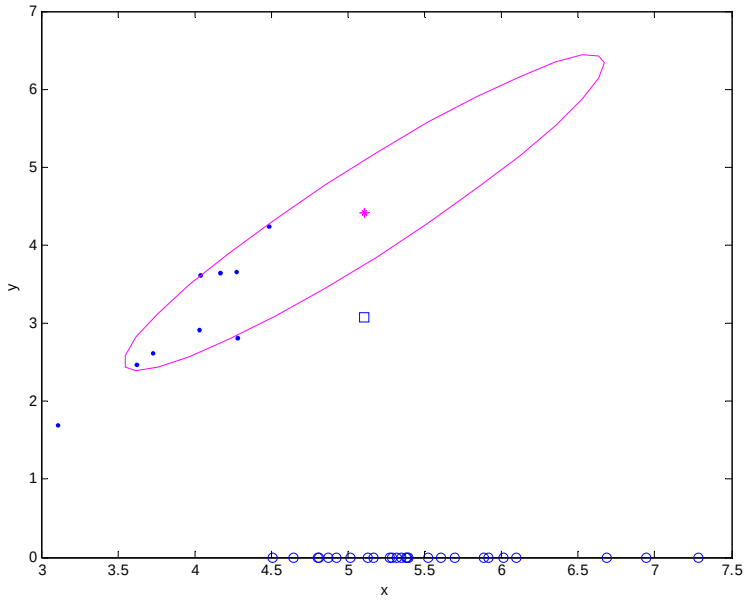


Figura 5-13 Problema de datos censurados. Estimación EM frente a imputación a la media.

En la figura 5-13, el cuadrado indica la media calculada sobre los datos observados. El asterisco y la elipse indican respectivamente la media y la desviación estándar calculada a partir de los datos incompletos usando el algoritmo EM ¡Nótese que la estimación de máxima verosimilitud de μ_y es más alta que cualquiera de los valores observados de y!

5.1.5 Experimento 5. Problema IRIS

Con este problema analizaremos el aprendizaje del algoritmo EM con información etiquetada, es decir, asociada a clase. Para poder llevar a cabo el aprendizaje, el algoritmo deberá recurrir a la mezcla de gaussianas y multinomiales que describíamos en 2.5.7. También comprobaremos la capacidad de clasificación, tanto en el caso de tener conjuntos de entrada completos como con conjuntos de información incompleta.

El problema IRIS está basado en datos reales. Consiste en un conjunto de 150 muestras de flores de tres variedades distintas. Cada una de estas muestras consta de cuatro valores. Dos de los valores son la longitud y la anchura de pétalo, y los otros dos la longitud y la anchura de sépalo. El total de 150 muestras se divide en tres clases de 50 muestras cada una. Estas tres clases se corresponden con tres variedades de lirios: *iris-setosa, iris-versicolor* e *iris-virgínica*. El problema de clasificación consistirá por tanto en, dada una muestra, asignarle una variedad de lirio correspondiente.

Como hemos comentado, el conjunto de muestras consta de cuatro dimensiones. Por lo tanto el primer problema al que nos enfrentamos es cómo dibujar los datos, ya que solo contamos con dos dimensiones físicas para dibujar. Optaremos por una solución sencilla, dibujar las dos dimensiones más importantes, que sabemos que son la tercera (longitud de pétalo) y la cuarta (anchura de pétalo). En la figura 5-14 se muestra el problema Iris proyectado sobre sus dos componentes principales.

A continuación aplicaremos al conjunto de entrenamiento el Algoritmo EM, para que de este modo aprenda a modelar la distribución de forma etiquetada, y posteriormente sea capaz de clasificar. Para poder evaluar la generalidad del aprendizaje, solo entrenaremos el algoritmo EM con 100 muestras (de las cuales la tercera parte dedicada a validación), mientras que dejaremos otras 50 para evaluar la capacidad de clasificación (conjunto de test). Seguimos así el criterio expresado en 2.5.11.2.

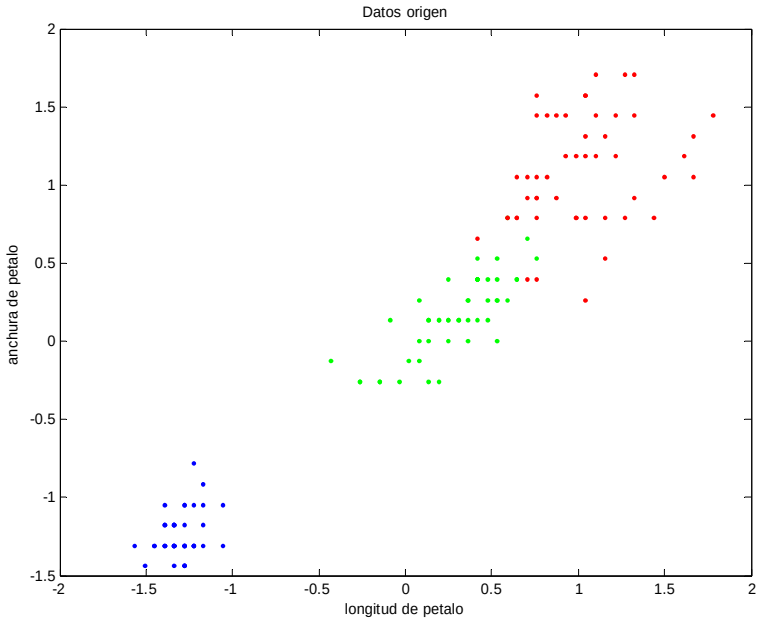


Figura 5-14 Problema Iris. Proyección del conjunto de entrenamiento sobre sus dos dimensiones principales, que indican la longitud y anchura de pétalo, respectivamente. Cada color indica una variedad de lirio.

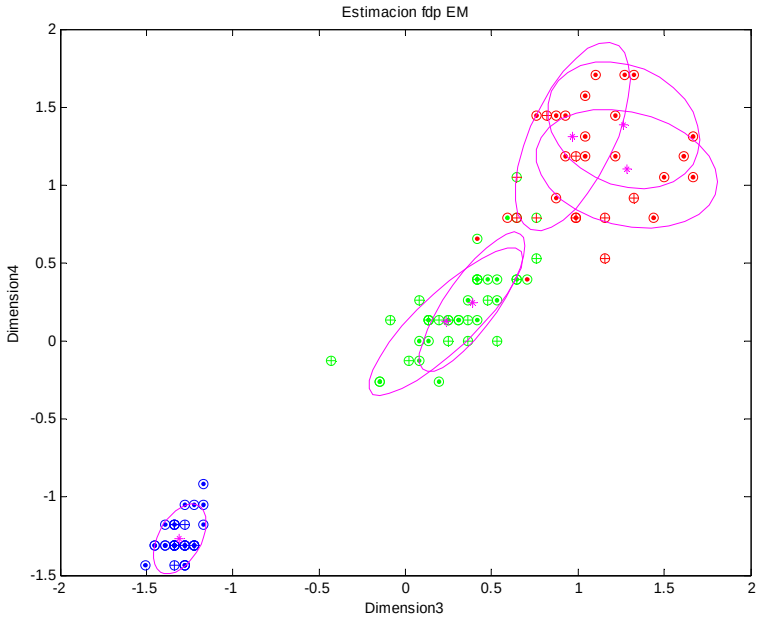


Figura 5-15 Estimación de la mezcla de gaussianas proyectada sobre el subespacio que forman las dimensiones tercera y cuarta.

Tras realizar un barrido de estimaciones EM con números de componentes de 1 a 6, en la figura 5-15 podemos ver el diagrama de elipses que ilustran la forma de la mezcla de gaussianas definitiva estimada sobre el conjunto de entrenamiento. Observamos que cada muestra está representada por un punto o una cruz rodeados de un círculo. Los puntos indican muestras del conjunto de entrenamiento, mientras que las cruces son muestras del conjunto de validación. El color de los puntos o cruces indica el valor original de las muestras, y el color del círculo que los rodea nos informa sobre la clase que el algoritmo EM les asigna de acuerdo con la mezcla de gaussianas estimada. Al contemplar esta figura, podemos pensar que la clase asignada a algunas muestras no es demasiado lógica, pero no debemos olvidar que hay otras dos dimensiones que no estamos representando. Lo que ocurre es que en esas dimensiones que no vemos, estas muestras se encuentran rodeadas por muestras de la clase asignada.

En la figura 5-16 podemos analizar la evolución de las mezclas estimadas. Hemos hecho un barrido de componentes, calculando un máximo de 20 épocas para cada componente, y hemos seleccionado la época que produce un menor error de clasificación sobre el conjunto de validación. En caso de que haya varias con el mismo error, seleccionamos entre ellas la que ofrezca una mayor verosimilitud en validación. Por último, para elegir el número de componentes apropiado empleamos el mismo criterio.

Componentes	6
Época	4
Verosimilitud entrenamiento	-1.39
Verosimilitud validación	-2.55
Verosimilitud test	-3.45
Error entrenamiento	2.99 %
Error validación	3.03 %
Error test	2 %

Tabla 5-3 Características de la mezcla seleccionada por EM tras realizar varias pasadas para diferentes cantidades de componentes (de 1 a 6) para el ejercicio de la figura 5-15.

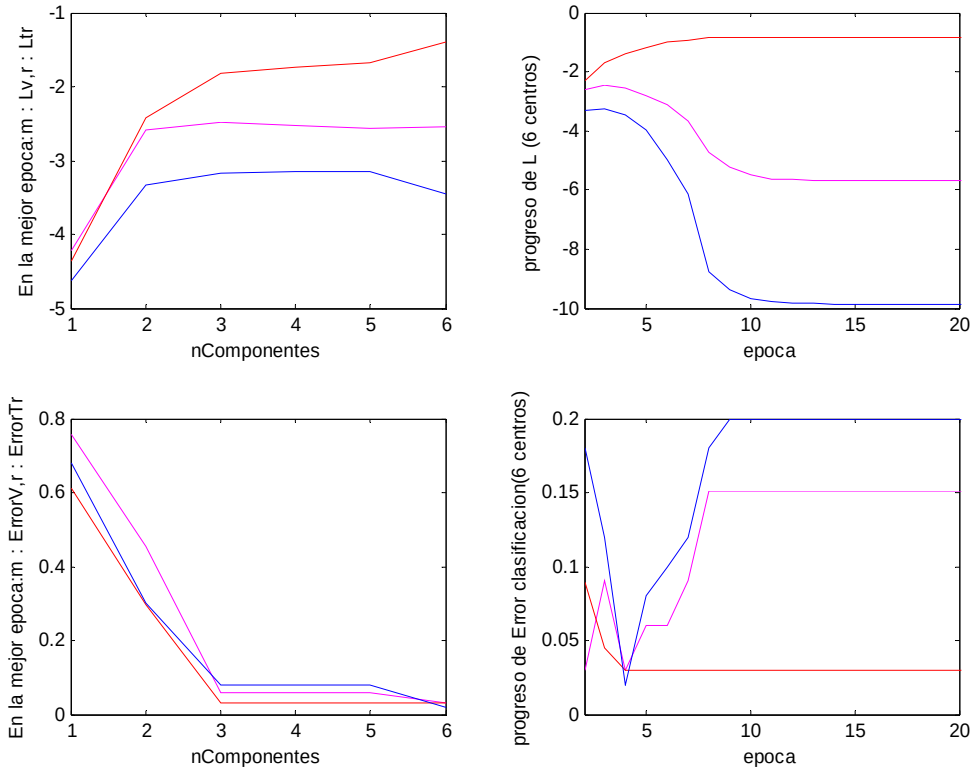


Figura 5-16 Evolución de la función de verosimilitud (arriba) y del error de clasificación (abajo) en función del número de componentes de la mezcla (izquierda) o de las épocas del algoritmo (derecha) con el número de componentes seleccionado. En rojo conjunto de entrenamiento, magenta para validación y azul para test.

En la figura 5-17 hemos representado las densidades de probabilidad marginales para cada dimensión, tanto condicionadas a clase como no condicionadas. Con estos gráficos podemos hacernos una idea de las dimensiones que más información aportan (aquellas con las densidades condicionadas a clase más separadas). Por último, en la figura 5-18 podemos ver una representación tridimensional de la función de densidad, en función de las dimensiones 3 y 4.

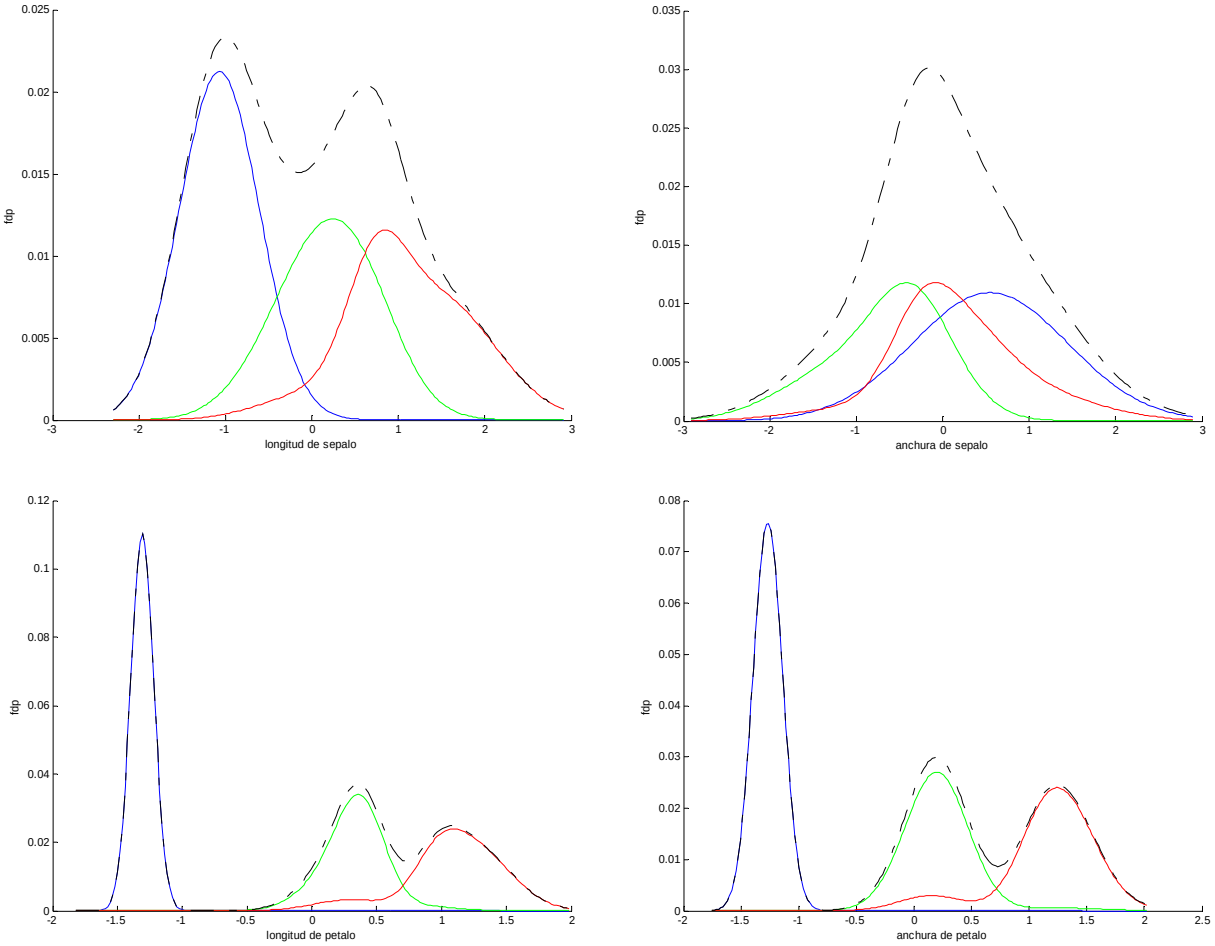


Figura 5-17 Funciones de densidad de distribución marginal para las cuatro dimensiones. Incondicional (en negro, a trazos) y condicionadas a las clases (en los colores correspondientes).

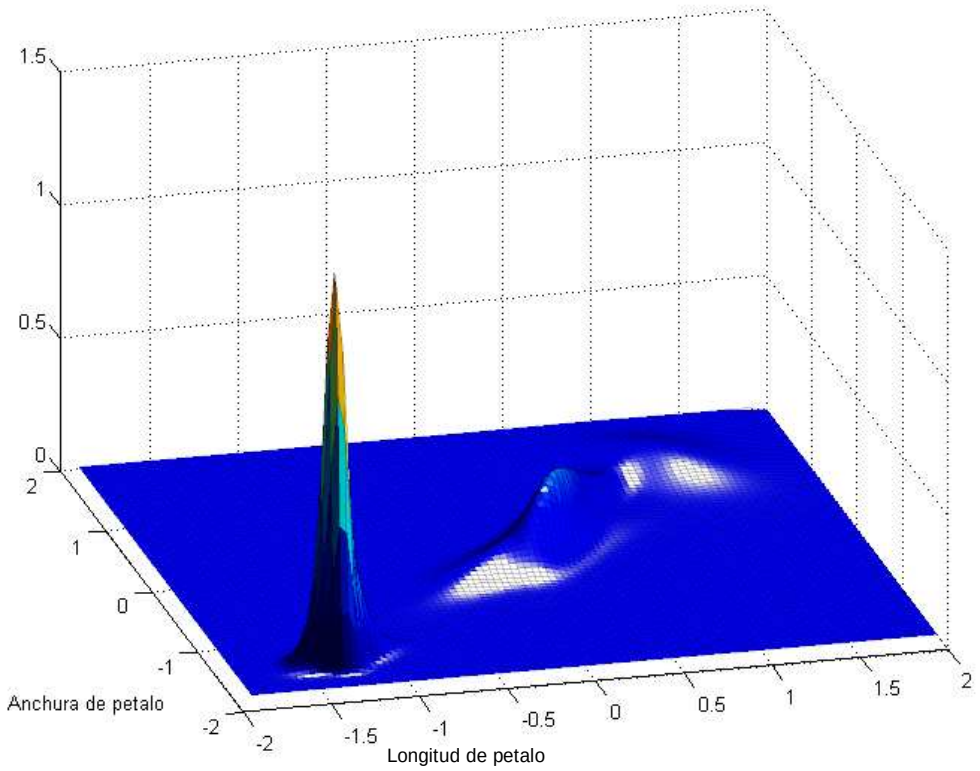


Figura 5-18 Asepecto tridimensional que presenta la fdp estimada para las dimensiones 3 y 4.

Hasta ahora hemos ilustrado un caso particular, pero si queremos obtener conclusiones generales es necesario probar el algoritmo varias veces (con distintas inicializaciones) sobre varias variantes del problema (distinta distribución de los conjuntos de entrenamiento, validación y test) . De este modo podemos estimar de forma más precisa el error de clasificación del algoritmo EM para el problema IRIS. En la tabla 5-4 se muestran los resultados medios obtenidos tras promediar 10 simulaciones.

Por otro lado, veremos como afecta la existencia de valores perdidos al algoritmo EM. En la figura 5-19 podemos observar el mismo problema que antes, aunque ahora el 50% de la información para cada dimensión de los conjuntos de entrenamiento y validación está incompleta. Del mismo modo que para el caso anterior, en la figura 5-20 podemos analizar la evolución de las estimaciones del algoritmo.

Verosimilitud entrenamiento	-1,03
Verosimilitud validación	-2,28
Verosimilitud test	-2,02
Error entrenamiento	2,68%
Error validación	1,12%
Error test	3,20%

Tabla 5-4 Resultados medios obtenidos por el algoritmo EM frente al problema IRIS tras realizar 10 simulaciones.

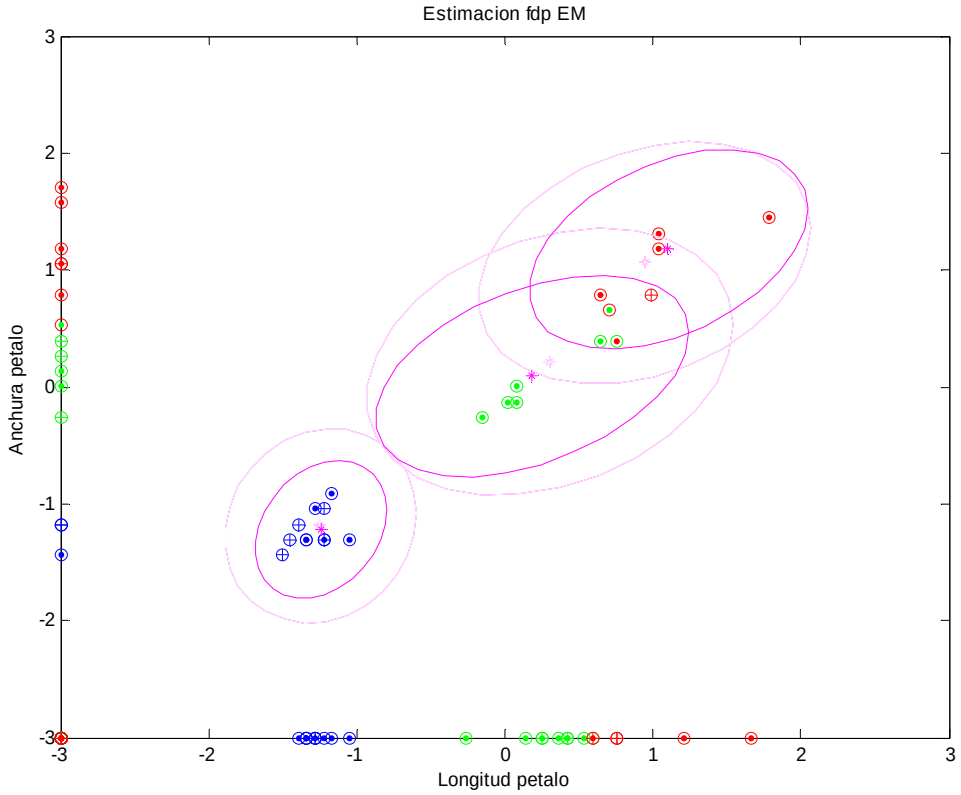


Figura 5-19 Proyección de la estimación de la mezcla de gaussinas para las dimensiones tercera y cuarta con un 50% de valores perdidos (solo 12 muestras mantienen las cuatro dimensiones).

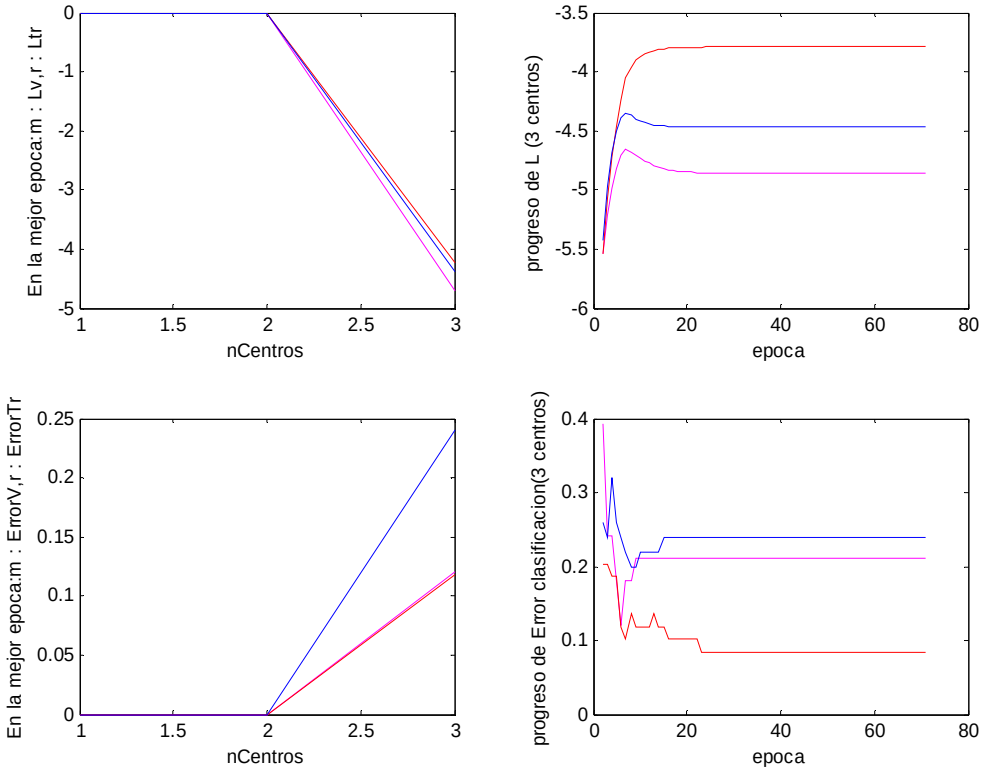


Figura 5-20 Evolución de la función de verosimilitud (arriba) y del error de clasificación (abajo) en función del número de componentes de la mezcla (izquierda) o de las épocas del algoritmo (derecha). En rojo conjunto de entrenamiento, magenta para validación y azul para test. El 50% de datos de los conjuntos de entrenamiento y validación están perdidos.

5.1.6 Experimento 6. Problema Voting

Hasta ahora todos los experimentos en los que hemos trabajado constaban de variables continuas. El objetivo de este problema será comprobar la capacidad de trabajo del EM con valores discretos. Esta habilidad no es obvia, ya que la versión standard del algoritmo EM solamente cuenta con una mezcla de gaussianas, mientras que para poder trabajar con datos discretos es necesario implementarlo para una mezcla multinomial.

El problema Voting está basado en un registro de votaciones del congreso de los Estados Unidos. El conjunto de datos consta de 435 muestras correspondientes a otros tantos congresistas. Cada muestra incluye el valor de los votos efectuados por cada

congresista en 16 votaciones claves. Los congresistas deberán ser clasificados en dos clases: republicanos o demócratas. Los votos pueden tomar tres valores: a favor, en contra o desconocido (dato incompleto). En la siguiente tabla se muestra la proporción de votos desconocidos para cada votación. Por otro lado, disponemos las etiquetas de clase para todo el conjunto de datos.

Atributo	Valores perdidos
1	0,00%
2	2,76%
3	11,03%
4	2,53%
5	2,53%
6	3,45%
7	2,53%
8	3,22%
9	3,45%
10	5,06%
11	1,61%
12	4,83%
13	7,13%
14	5,75%
15	3,91%
16	6,44%

Tabla 5-5 Porcentaje de valores perdidos para cada característica del conjunto de datos.

Siguiendo el mismo criterio que en otros experimentos, dividiremos el conjunto de datos en tres subconjuntos, uno para entrenamiento (1/2 del total), otro para validación (1/6) y otro para test (1/3).

Para calcular los resultados de forma más fiable, promediaremos sobre 10 conjuntos distintos, escogidos aleatoriamente, de entrenamiento, validación y test. Para cada uno de estas 10 grupos de conjuntos, los resultados medios obtenidos tras realizar 10 veces el experimento están expuestos en la tabla 5-6.

Verosimilitud entrenamiento	-6,51
Verosimilitud validación	-7,23
Verosimilitud test	-7,17
Error entrenamiento	5,72%
Error validación	4,65%
Error test	5,71%

Tabla 5-6 Resultados medios obtenidos por el algoritmo EM frente al problema Voting tras realizar 10 simulaciones.

5.2. Combinación de GMM-EM y MTL

Tras comprobar las fortalezas y debilidades del modelado de funciones de densidad de probabilidad mediante GMM entrenadas con el algoritmo EM, como último objetivo, vamos a aplicar este método para guiar la imputación de lar redes MTL en problemas de datos incompletos. En concreto, queremos probar, en una primera aproximación, que el procedimiento propuesto en el apartado 4.4.1**¡Error! No se encuentra el origen de la referencia.** mejora generalmente los resultados obtenidos con la red MTL. Por ello, nos centraremos en la arquitectura de red MTL mostrada en la figura 4-2.

En este apartado pretendemos combinar este potente método para estimación de fdp con redes neuronales MTL para clasificación de vectores de entrada incompletos [43], tal y como propusimos en el apartado 4.4.1. En concreto, optaremos por la función de error propuesta en la ecuación (4.11), donde para la ponderación de las diferencias se tiene en cuenta la etiqueta de clase en el cálculo de la fdp.

Para los experimentos, utilizaremos distintos problemas de clasificaciones de patrones incompletos, ya sean reales o artificiales, comparando los resultados obtenidos por el método desarrollado en este PFC con los resultados de la red MTL. Estas comparaciones se realizarán en términos de la probabilidad de error de clasificación, tanto en entrenamiento como en test, y a través de una medida de la calidad de las imputaciones.

Esta medida es el valor medio de las probabilidades de cada valor estimado, siendo estas probabilidades obtenidas a partir de la fdp estimada por la mezcla de gaussianas. Por tanto, cuanto mayor sea este valor, más probables son las imputaciones de datos que se han realizado.

5.2.1 Experimento 7. Problema Artificial 2-D

Este problema bidimensional (d=2) de clasificación (c=2) ha sido generado artificialmente a partir de cinco nubes gaussianas, donde las matrices de covarianza para las cinco nubes son igual a $\mathbf{\Sigma} = 0.03 \cdot \mathbf{I}$, siendo $\mathbf{I}$ la matriz identidad, mientras que los vectores de media para cada nube son los siguientes,

$$\boldsymbol{\mu}^1 = (1,1) \quad \boldsymbol{\mu}^2 = (-0.7, 0.3) \quad \boldsymbol{\mu}^3 = (0.3, 0.3) \quad \boldsymbol{\mu}^4 = (-0.3, 0.7) \quad \boldsymbol{\mu}^5 = (0.4, 0.7)$$

En la figura 5-21 se muestra la representación en el espacio de entrada del problema de clasificación (distinguiendo con rojo y azul las dos posibles clases) a resolver en este experimento. Contamos con 1250 casos de entrada, con igual proporción de casos para cada nube.

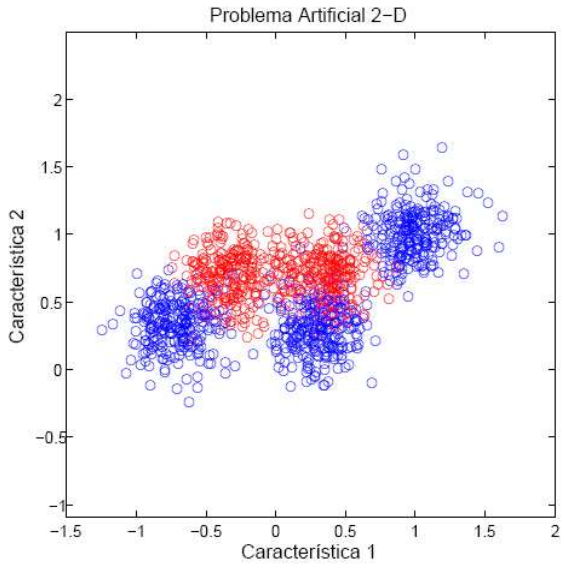


Figura 5-21 Problema Artificial 2-D (problema original generado con datos completos), mostramos un conjunto de entrenamiento con un 5% de valores perdidos en la característica 1.

Como queremos probar las prestaciones sobre conjuntos de datos incompletos, se eliminan de manera aleatoria valores perdidos en la primera característica para distintos porcentajes (5%, 10%, 20%, 30% y 40%), generando aleatoriamente diez grupos distintos de conjuntos de entrenamiento y test, compuestos respectivamente por 250 y 1000 vectores de entrada. Como conjunto de validación se escoge una tercera parte del conjunto de entrenamiento.

Antes de presentar y analizar los resultados obtenidos, vamos a explicar, usando este problema sintético, cual es el procedimiento propuesto para modificar la función de error en redes MTL usando GMM entrenados con el algoritmo EM. A partir de ahora, esta red MTL, cuya función de error a optimizar está modificada por la GMM, la denotaremos como red MTL-GMM. Supongamos que partimos del conjunto de entrenamiento mostrado en la figura 5-22, con valores perdidos en la primera característica. Cabe notar que, en esta figura, el conjunto de datos de entrada está normalizado con media cero y varianza unidad. En particular, para explicarlo nos centraremos en los tres casos incompletos dibujados con líneas discontinuas más gruesas. Además, en la figura 5-23, se muestra la proyección de la fdp estimada mediante la mezcla de gaussianas para este conjunto de entrenamiento.

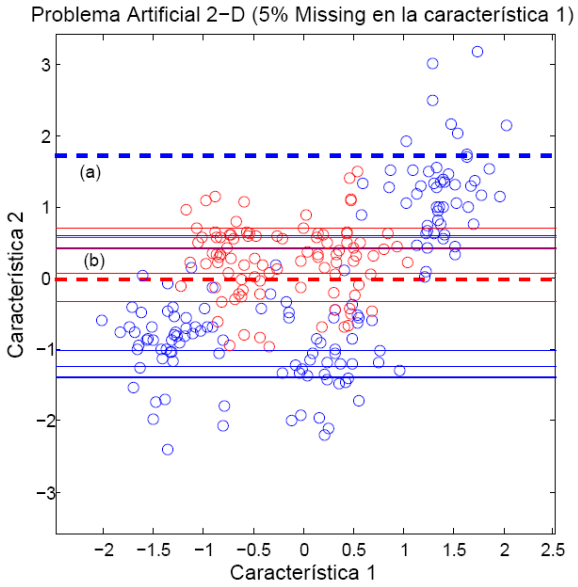


Figura 5-22 Conjunto de entrenamiento para el problema Artificial 2-D con un 5% de valores perdidos en la característica 1. En líneas discontinuas más gruesas se muestran los vectores (a), (b) y (c) que usaremos para explicar el funcionamiento del método MTL-GMM.

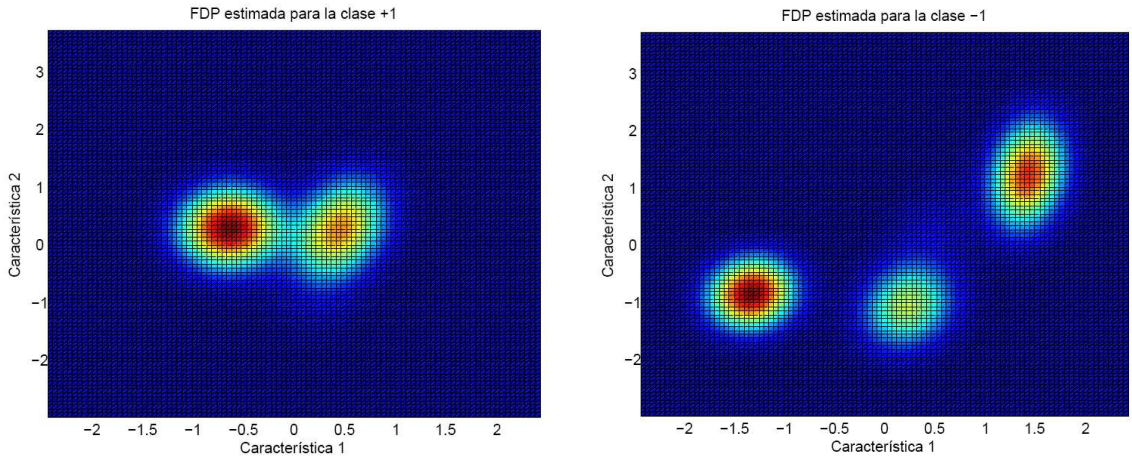


Figura 5-23 FDPs calculadas mediante el modelado de mezclas de gaussianas para el conjunto de entrenamiento incompleto mostrado en la figura 5-22.

Como se puede observar en la figura 5-22, en el caso del patrón incompleto (a), la red MTL no tendrá problemas en realizar una buena estimación del valor perdido. En cambio, para el patrón (b), la estimación no es sencilla, y como ya se comento en el capítulo anterior, la red MTL tenderá a realizar una estimación entre las nubes asociadas a su clase, por la manera intrínseca de minimizar la función de error cuadrático medio. Al realizar una ponderación de la función de error asociada a las tareas de imputación mediante la fdp calculada previamente mediante una mezcla de gaussianas, se está guiando la imputación hacia zonas más probables teniendo en cuenta la fdp estimada. A continuación se muestra la fdp normalizada para el patrón (b) teniendo en cuenta la información de su etiqueta.

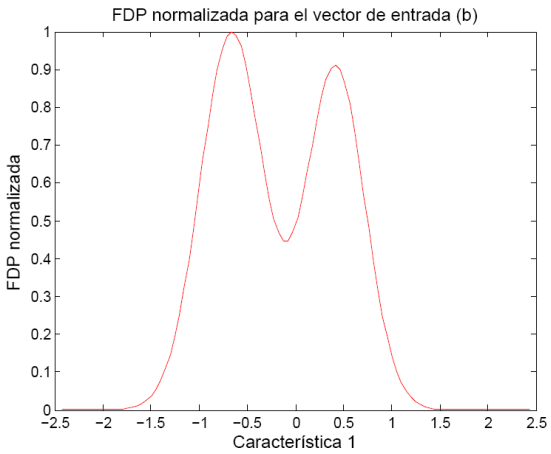


Figura 5-24 FDP normalizada obtenida para el vector de entrada incompleto (b).

Si se emplea esta FDP normalizada para ponderar las estimaciones sobre el vector (b) podemos evitar que en lugar de imputar a zonas poco probables, a zonas más probables cercanas a los máximos de la fdp. Así, cuando tengamos imputaciones cercanas a x_1=0, los errores asociados a dicha imputación contarán mucho más, provocando así la correspondiente actualización de pesos de la red, mientras que en el caso de realizar imputaciones más probables, estos errores contarán menos, ya que en realidad se está realizan una 'buena' imputación, por lo que los pesos de la red asociados a la tarea que aprende x_1 apenas se actualizarán.

Una vez explicada la forma de adaptar el aprendizaje MTL mediante la mezcla de gaussianas, se presentan los resultados obtenidos. Para cada grupo de conjuntos de entrenamiento y test, hemos realizado diez simulaciones distintas (con distintas inicilializaciones aleatorias de los vectores de pesos) usando la red MTL mostrada en la figura 4-2. Tras entrenar todas las redes simuladas, cada una de ellas se vuelve a entrenar (partiendo de los mismos pesos iniciales) teniendo en cuenta la función de error ponderada con la mezcla de gaussianas, obtenida previamente sobre el conjunto de entrenamiento. Esto se hace así para comprobar el efecto real del uso de GMM en la ponderación la función de error de las tareas de imputación, al partir de la misma inicialización de pesos tanto la red MTL como la red MTL-GMM. Este procedimiento se mantiene en el resto de experimentos.

En la tabla siguiente se muestran la probabilidad de error de clasificación, tanto sobre el conjunto de entrenamiento como sobre el conjunto de test, usando la red MTL y la red MTL-GMM. Podemos observar como en la mayoría de los casos se produce una mejora en el aprendizaje del conjunto de entrenamiento usando la red MTL-GMM que en el caso de usar la red MTL, pero en cambio, los resultados sobre el conjunto de test son levemente mejores. Esto es debido a que estamos ante un problema bidimensional al que llegamos a eliminar el 40% de los datos en una de sus componentes. En este problema podemos observar como la ausencia de datos es un fenómeno muy perjudicial en conjuntos de datos reducidos.

a=[1]	**Probabilidad de Error en Clasificación**					
	Entrenamiento			**Test**		
	Media ± Desviación típica			Media ± Desviación típica		
% Missing	**MTL**	**MTL+GMM**	**Mejora(%)**	**MTL**	**MTL+GMM**	**Mejora(%)**
5	11.58 ± 2.12	11.39 ± 1.96	+1.64	11.83 ± 0.52	11.57 ± 0.52	+2.19
10	10.62 ± 2.09	10.52 ± 2.72	+0.94	13.01 ± 1.22	13.02 ± 1.12	-0.07
20	12.17 ± 1.93	12.29 ± 1.84	-0.98	13.70 ± 1.17	13.74 ± 1.21	-0.29
30	16.11 ± 1.48	15.05 ± 0.92	+6.57	15.54 ± 1.14	15.21 ± 0.83	+2.12
40	15.61 ± 0.43	15.35 ± 0.84	+1.66	17.16 ± 0.64	17.30 ± 0.70	-0.82

Tabla 5-7 Resultados para el Problema Artificial 2-D. Probabilidad de Error de Clasificación para distintos porcentajes de valores perdidos en la característica 1 usando la red MTL y la red MTL-GMM.

En los siguientes experimentos comprobaremos como aumenta la calidad de los valores imputados, es decir, la red MTL imputa valores más probables al estar guiada el aprendizaje de las tareas de imputación por la fdp estimada mediante la mezcla de gaussianas. También veremos como se mejora la velocidad de convergencia y la capacidad de generalización.

5.2.2 Experimento 8. Problema Iris Artificial

Este problema de clasificación ha sido generado a partir del problema Iris usado en el quinto experimento, para ello hemos realizado un modelado de la fdp mediante la mezcla de gaussianas implementada en este trabajo, y a partir de la mezcla obtenida, hemos generado un conjunto de datos artificial con 750 casos. Como partimos de un conjunto de datos completo, se generan, al igual que en el experimento anterior, diez grupos de conjuntos de entrenamiento (350 casos) y test (400 casos) insertando valores perdidos aleatoriamente para distintos porcentajes y características.

Para este problema únicamente hemos implementado la red neuronal MTL con una única capa oculta común que aprende todas las tareas de clasificación e imputación, y la hemos comparado con la resultante de emplear la mezcla de gaussianas. En las tablas siguientes se muestran los distintos resultados obtenidos en el problema Iris artificial. En la

mayoría de los casos, el método propuesto en este PFC mejora los resultados obtenidos por la red MTL, además, en este experimento, podemos comprobar la gran robustez que aportan las redes basadas en MTL con respecto al incremento del porcentaje de datos incompletos. Por otro lado, las imputaciones que realiza la red MTL+GMM son más probables que las realizadas por la red MTL, por lo cual, se consigue el doble objetivo de conseguir unas imputaciones más probables y obtener un mejor resultado en la clasificación final.

a=[1]	**Probabilidad de Error en Clasificación**					
	Entrenamiento			**Test**		
	Media ± Desviación típica			Media ± Desviación típica		
% Missing	**MTL**	**MTL+GMM**	**Mejora(%)**	**MTL**	**MTL+GMM**	**Mejora(%)**
5	3.03 ± 0.47	3.04 ± 0.46	-0.33	2.93 ± 0.54	3.07 ± 0.41	-4.77
10	2.69 ± 0.63	2.68 ± 0.63	-0.37	3.26 ± 0.55	3.22 ± 0.51	+1.23
20	2.87 ± 0.47	2.79 ± 0.75	+2.78	3.25 ± 0.56	3.11 ± 0.57	+4.30
30	3.05 ± 0.59	3.12 ± 0.62	-2.29	3.40 ± 0.58	3.16 ± 0.50	+7.06
40	2.99 ± 1.01	2.92 ± 1.07	+2.34	3.54 ± 0.59	3.40 ± 0.60	+3.95

Tabla 5-8. Probabilidad de Error de Clasificación en el Problema Iris Artificial para distintos porcentajes de valores perdidos en la característica 1 usando la red MTL y la red MTL-GMM.

a=[1]	**Probabilidad de los Valores Imputados**					
	Entrenamiento			**Test**		
	Media ± Desviación típica			Media ± Desviación típica		
% Missing	**MTL**	**MTL+GMM**	**Mejora(%)**	**MTL**	**MTL+GMM**	**Mejora(%)**
5	68.5 ± 7.29	69.2 ± 7.55	+1.02	69.0 ± 5.33	70.1 ± 5.30	+1.59
10	64.5 ± 7.30	65.1 ± 7.78	+0.93	68.3 ± 5.91	69.0 ± 6.97	+1.02
20	66.0 ± 7.29	68.4 ± 4.98	+3.63	65.5 ± 4.62	67.7 ± 4.81	+3.36
30	63.1 ± 3.96	65.1 ± 4.31	+3.17	62.4 ± 3.54	63.9 ± 4.02	+2.40
40	58.3 ± 4.10	60.8 ± 4.80	+4.29	59.4 ± 4.53	61.3 ± 0.47	+3.13

Tabla 5-9 Probabilidad media de los valores imputados en el Problema Iris Artificial para distintos porcentajes de valores perdidos en la característica 1 usando la red MTL y la red MTL-GMM.

Además, hemos observado durante la implementación de la red MTL+GMM que, en muchos casos, ésta provoca un aumento en la velocidad de convergencia comparando con la red MTL. Esto se puede apreciar en la siguiente figura, donde se muestra la evolución del coste de entrenamiento durante el aprendizaje para las tareas principal y secundaria (asumiendo en este caso un 20% de valores perdidos) usando la red MTL y la red MTL+GMM. Como podemos ver, hay un incremento en la velocidad de convergencia de la tarea principal de clasificación.

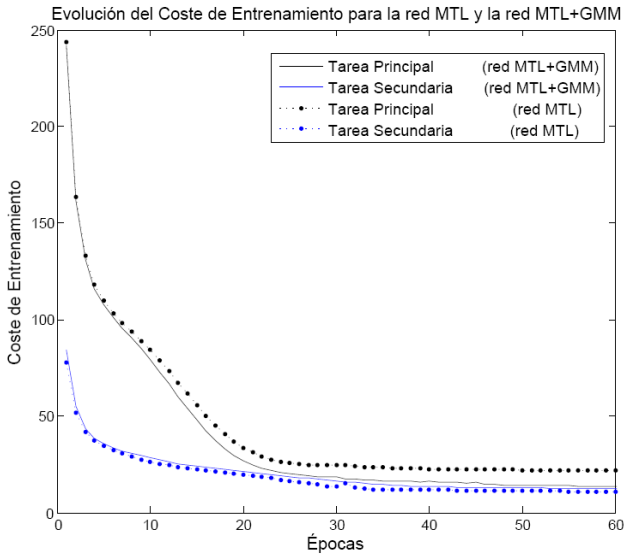


Figura 5-25 Evolución de la función de coste durante el entrenamiento para la red MTL y la red MTL+GMM. Podemos observar como la red MTL modificada mediante la fdp estimada por la GMM consigue una convergencia más rápida.

5.2.3 Experimento 9. Problema Pima Indians

El objetivo del problema Pima Indians es diagnosticar la enfermedad de la diabetes a partir de un conjunto de datos obtenido sobre una población de indios 'pima' americanos. Cada vector de entrada está compuesto por ocho características.

El problema consta de 632 patrones divididos previamente en un conjunto de entrenamiento (formado por 200 patrones completos y 100 patrones incompletos) y un conjunto de test (332 patrones completos). En el conjunto de entrenamiento incompleto, los valores perdidos están presentes en tres de las ocho características de entrada, siendo el

vector de características incompletas **a**=[3,4,5], presentando respectivamente los siguientes porcentajes de valores perdidos: 4.33%, 32.67% y 1.00%.

Este problema lo utilizaremos para comprobar las prestaciones del método propuesto sobre un problema de clasificación real con entradas incompletas. Como en los casos anteriores, compararemos los resultados obtenidos por la red MTL, además de hacer un barrido para distinto número de neuronas, desde cuatro hasta diez neuronas. En las tablas siguientes se muestran los resultados obtenidos en la probabilidad de error de clasificación y en la calidad de las imputaciones.

Neuronas	Probabilidad de Error de Clasificación en Entrenamiento (%)			Probabilidad de Error de Clasificación en Test (%)		
	Media ± Desviación típica			Media ± Desviación típica		
	MTL	**MTL+GMM**	**Mejora(%)**	**MTL**	**MTL+GMM**	**Mejora(%)**
4	24.90 ± 0.90	25.86 ± 1.96	-3.85	22.83 ± 2.06	22.04 ± 1.11	+0.54
6	24.63 ± 0.63	25.16 ± 1.21	-2.15	22.31 ± 1.73	21.17 ± 1.87	+5.11
8	22.76 ± 1.36	24.70 ± 1.42	-8.52	23.46 ± 0.61	22.62 ± 1.60	+3.58
10	22.80 ± 0.84	24.66 ± 0.72	-8.15	21.23 ± 1.40	21.59 ± 0.95	-1.69

Tabla 5-10. Probabilidad de Error de Clasificación en el Problema Pima Indians para distinto número de neuronas de la capa oculta usando la red MTL y la red MTL-GMM.

Neuronas	Probabilidad de los Valores Imputados		
	Entrenamiento		
	Media ± Desviación típica		
	MTL	MTL+GMM	Mejora(%)
4	83.40 ± 2.06	92.43 ± 3.70	+10.83
6	80.77 ± 2.44	88.96 ± 5.72	+10.14
8	82.58 ± 3.64	88.49 ± 3.78	+7.15
10	78.09 ± 2.37	83.26 ± 4.16	+6.62

Tabla 5-11 Probabilidad media de los valores imputados en el Problema Pima Indians para distinto número de neuronas de la capa oculta usando la red MTL y la red MTL-GMM.

Capítulo 6

Conclusión y trabajos futuros

En este proyecto se ha implementado un potente estimador de funciones de densidad de probabilidad (fdp) mediante el modelado basado en mezclas de funciones de distribución gaussianas (“Gaussianas Mixture Models”, GMM) entrenadas mediante el algoritmo EM (“Expectation-Maximization”). La razón fundamental de usar este algoritmo es su gran capacidad para el modelado de funciones de densidad de probabilidad cuando el conjunto de datos de entrada presenta valores perdidos (“missing data”). Hemos realizado además un análisis extenso del estado del arte del algoritmo EM, comprobando en distintos problemas las ventajas e inconvenientes que presenta. Por último, basándonos en los resultados obtenidos con el modelado de mezclas de gaussianas mediante EM, hemos propuesto un nuevo procedimiento para mejorar la capacidad de aprendizaje de las redes de aprendizaje multitarea (“Multitask Learning”, MTL). En particular, en una primera aproximación, el objetivo es mejorar las estimaciones realizadas por una red MTL, basándonos en que la fdp estimada mediante el modelo GMM sirva de guía durante el aprendizaje de las distintas tareas secundarias de imputación asociadas a cada característica de entrada incompleta.

6.1. Conclusiones principales

A continuación se describen brevemente las conclusiones principales a las que se ha llegado en este proyecto:

1. El proceso de modelado de mezclas (gaussianas o multinomiales) mediante el algoritmo EM es robusto y flexible, pudiendo aplicarse en infinidad de problemas. Entre ellos, destacamos problemas de clasificación, clustering o agrupamiento, y problemas de aproximación de funciones o regresión.

2. El modelado de funciones de densidad de probabilidad es otra de las aplicaciones para este método, permitiéndonos trabajar de una manera sencilla y eficiente sobre conjuntos de datos complejos. Es importante resaltar que el algoritmo es capaz de modelar densidades de probabilidad en el caso de que existan vectores de entrada incompletos, un problema muy frecuente en aplicaciones reales de reconocimiento de patrones.

3. El algoritmo desarrollado puede modelar conjuntos de datos con atributos continuos o discretos. Para el caso de los atributos continuos se emplean mezclas de gaussianas, mientras que para los atributos discretos nos valemos de mezclas de funciones multinomiales. Si un conjunto de datos presenta simultáneamente atributos discretos y continuos, empleamos una mezcla que combina funciones gaussianas y multinomiales.

4. La incorporación del modelado de densidades de probabilidad a las redes MTL consigue que éstas proporcionen una mejor estimación de los valores perdidos, ya que la mezcla de gaussianas guía las tareas de imputación durante el proceso de aprendizaje, evitando imputaciones de valores poco probables teniendo en cuenta la densidad de probabilidad estimada. Además, generalmente, el uso de las GMM como guía a las tareas de imputación, implica una mayor velocidad de convergencía.

5. Los resultados obtenidos, tanto en problemas reales como artificiales, muestran las ventajas de combinar el aprendizaje multitarea con los los modelos de mezclas de gaussianas entrenados mediante el algoritmo EM.

6.2. Trabajos futuros

Con la experiencia acumulada durante el desarrollo del proyecto, se pueden plantear las siguientes líneas futuras:

1. Extender el método propuesto para combinar la mezcla de gaussianas con la red MTL. Una alternativa directa es introducir como entrada a la red MTL una variable indicadora de la pertenencia de las muestras a cada componente de la mezcla. De esta forma se proporciona a la red una información adicional que permite distinguir entre distintos grupos en el espacio de entrada, evitando imputaciones a zonas improbables. Siguiendo esta filosofía, es posible implementar la estimación de esta información de pertenencia a una componente de la mezcla como tarea secundaria, que guié tanto la imputación como la clasificación.

2. Desarrollo de nuevas técnicas que, además de estimar los parámetros de la mezcla (medias, covarianzas y proporciones de mezcla, para funciones gaussianas) calculen también el dimensionado del modelo, es decir, incluyan el número de componentes de la mezcla como parámetro a optimizar por el algoritmo EM.

3. Implementar el proceso de entrenamiento y actualización de pesos de una red MTL mediante el algoritmo EM, llegando incluso al dimensionado automático de la red.

4. Evaluar los distintos métodos implementados en una mayor cantidad de problemas de clasificación reales con entradas incompletas, como problemas de diagnosis médica o reconocimiento de voz mediante espectrogramas incompletos.

Referencias bibliograficas

[1] P. James McCracken. “Selective representational transfer using stochastic noise”. Acadia University. Bachelor Thesis. Abril 2003.

[2] R. Caruana. “Multitask learning: a knowledge-based source of inductive bias”. Proceedings of the 10th International Conference of Cognitive Science. 1993.

[3] P.J. García Laencina, J.L. Sancho Gómez y A.R. Figueiras Vidal. “Pattern classification with missing values using multitask learning”, en Proc. International Joint-Conference on Neural Networks 2006, Julio 2006.

[4] H. J. Ritter, y K. J. Schulten, “Kohonen’s self-organizing maps: exploring their computational capabilities”, en Proc. IEEE International Conference on Neural Networks, 1, pp. 109-116, 1988.

[5] Sandra P. Daza. “Redes neuronales artificiales. Fundamentos, modelos y aplicaciones”. Universidad Militar Nueva Granada.

[6] M. Cañizares, I. Barroso y K. Alfonso. “Datos incompletos: una mirada crítica para su uso en estudios sanitarios”. Instituto nacional de Higiene, Epidemiología y Microbiología (INHEM). La Habana. Cuba. Mayo 2003.

[7] R. J. A. Little y D. B. Rubin. “Statistical Analysis with Missing Data”. Wiley, New York. 1987.

[8] Z. Ghahramani y M. I. Jordan. “Learning from incomplete data”. MIT, artificial intelligence laboratory y Center for biological and computational learning, department of brain and cognitive sciences. Diciembre 1994.

[9] J. B. Navarro Pastor. “Aplicación de redes neuronales artificiales al tratamiento de datos incompletos”. U. Autònoma de Barcelona. Tesis Doctoral. 1998.

[10] P. J. Sharpe y R. J. Solly. “Dealing with Missing values in neural Network-Based Diagnostic Systems”. Neural Computing and Aplications. vol. 3, no. 2, pp 73-77. USA: Springer, 1995.

[11] S. Y. Yoon y S. Y. Lee. “Training algorithm with incomplete data for feed-foward neural networks”. Neural Processing Letters, no. 10, pp. 171-179. Netherlands: Kluwer, 1999.

[12] G. E. Batista y M. C. Monard. “A study of K-Nearest Neighbour as an Imputation Method”. Second International Conference on Hybrid Intelligent Systems. Netherlands: IOS Press, v. 87, p. 251-260. Santiago, Chile. 2002.

[13] M. K. Markey, G. D. Tourassi, M. Margolis y D. M. DeLong. “Impacto of missing data in evaluating artificial neural networks trained on complete data”, Computers in Biology and Medicine. (accepted paper, in press).

[14] V. Tresp, R. Neuneir y S. Ahmad. “Eficient methods for dealing with missing data in supervised learning”. Advances in Neural Information Processing Systems 7. MIT Press, Cambridge 1995.

[15] H. Ishibuchi, A. Miyazaki, K. Kwon y H. Tanaka. “Learning from incomplete training data with missing values and medical application”. Proceedings of 1993 International Joint Conference on Neural Networks, pp 1871-1874. Nagoya, Japan, 1994.

[16] Zs. J. Viharos, K. Novaki y T. Vincze, “Training application of artificial neural networks with incomplete data”. Proceedings of 15th International Conference on Industrial & Engineering Aplications of Artificial Intelligence & Experts Systems, LNCS, pp. 64-659. Caims, Australia, 2002.

[17] Y. Bengio y F Gingras. “Recurrent neural networks for missing or asynchronous data”. Advances in Neural Information Processing Systems 8. pp, 395-401. Cambridge, UK: MIT Press, 1996.

[18] S. Parveen, P. Green y Y. Abdel-Haleem, “Combining imputation and classification in a single recurrent neural network for robust ASR with missing data”, Proccedings of Fourth International ICSC Symposium on Engineering of Intelligent Systems. Madeira, Portugal, 2004.

[19] D. R. Cox y D. V. Hinkley. “Theoretical statistics. NuevaYork: John Wiley & Sons. 1974.

[20] D. Peña. “Estadística, modelos y métodos: fundamentos”. Vol I. Alianza Editorial. Madrid. 1991.

[21] C.M. Bishop. “Neural Networks for Pattern Recognition”. Oxford University Press. 1995.

[22] R. Jacobs, M. Jordan, S. Nowlan y G. Hilton. “Adaptive mixture of local experts”. Neural computation, 3:79-87. 1991.

[23] J. H. Friedman. “Multivariate adaptive regression splines”. The Annals of Statistics, 19:1-141. 1991.

[24] C. Liu y D. Rubin. “The ECME algorithm: a simple extension of EM and ECM with faster monotone convergence”. Biometrika, 81:633--648, 1994.

[25] V. Tresp, S. Ahmad y R. Neuneier. “Training neural networks with deficient data”. Advances in Neural Information Processing Systems 6. San Francisco. CA. Morgan Kaufman Publishers.

[26] A. P. Dempster, N. M. Laird y D. B. Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. J. Royal Statistical Society Series B, 39:1-38.

[27] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone. “Classification and Regression Trees”. Wadsworth International Group, Belmont, CA.

[28] C. M. Bishop. “Mixture density networks”. Technical Report NCRG/4288. AStON University. Birmingham, U. K. 1994.

[29] G. J. McLachlan y T.Krishnan. “The EM algorithm and extensions”. Wiley-Interscience. 1997.

[30] J.J. Verbeek, N. Vlassis y B. Kröse. “Efficient Greedy Learning of Gaussian Mixture Models”. University of Amsterdam. MIT press. 2002.

[31] P. Paalanen. “Bayesian classification using gaussian mixture model and EM estimation: implementations and comparisons”. Lappeentanta University of Technologie. 2004.

[32] P.J. García Laencina. “Mejora en la detección de microcalcificaciones en mamografías digitalizadas mediante la aplicación de arquitecturas neuronales”. U. Politécnica de Cartagena. Proyecto Fin de Carrera. Septiembre 2004.

[33] A.K. Jain, J. Mao, and K.M. Mohiuddin. “Artificial Neural Networks: A Tutorial“. IEEE Computer Magazine, 29(3):31.44, 1996.

[34] W.S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent innervous activity”. Bulletin of Mathematical Biophysics, 5:115.133, 1943.

[35] Simon Haykin. “Neural Networks: A Comprehensive Foundation”. Prentice Hall, Upper Saddle River, N.J., 1999.

[36] F. Rosenblatt. “Principles of Neurodynamics”. Spartan Books, Washington, D.C., 1962.

[37] M.L. Minsky and S.A. Papert. “Perceptrons: An Introduction to Computational Geometry”. MIT Press, Cambridge, Mass., 1969.

[38] J. Hopfield. “Neural networks as physical systems with emergent collective computational abilities”. In Proceedings of the National Academy of Sciences, USA, volume 79, pages 2554.2558, 1982.

[39] J. Werbos. “Beyond Regression: New tools for prediction and analysis in the behavioral sciences”. PhD thesis, Dept. of Applied Mathematics, Harvard University, Cambridge, Mass., 1974.

[40] David E. Rumelhart and James L. McClelland, editors. “Parallel Distributed Processing: Explorations in the Microstructure of Cognition”, volume 1. MIT Press, Cambridge, Mass., 1986.

[41] R. Caruana. “Multitask Learning”. Tesis doctoral. Carnegie Mellon University. 1997.

[42] J. Baxter. “Learning internal representations”. Tesis doctoral. Flinders University of South Australia. Adelaide, 1994.

[43] P. J. García Laencina, R. Verdú Monedero y J. L. Sancho Gómez. “Aprendizaje Multitarea mediante Arquitecturas Neuronales basadas en Subredes Privadas”. Actas del Congreso Nacional URSI 2005. Septiembre 2005.

[44] P. J. García Laencina y J. L. Sancho Gómez. “Imputación de Datos Incompletos y Clasificación de Patrones mediante Aprendizaje Multitarea”. Actas del Congreso Nacional URSI 2005. Septiembre 2005.

[45] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability"*, Berkeley, University of California Press, 1:281-297