Análisis Discriminante Evolutivo

Alejandro Echeverría Rey

Director: Dr. Alejandro Sierra Urrecho

Escuela Politécnica Superior - Universidad Autónoma de Madrid

Madrid, Febrero de 2009

II

Resumen

En este trabajo se propone una nueva extensión no lineal del Análisis Discriminante de Fisher (ADF) denominada Análisis Discriminante Evolutivo (ADE).

ADE es un nuevo algoritmo de clasificación que comparte con su análogo clásico características como su naturaleza libre de etiquetas, pero que supera limitaciones como su incapacidad para aprovechar posibles relaciones no lineales entre las variables de entrada.

R. Fisher introdujo ADF en 1936 como un procedimiento de reducción dimensional supervisada que obtiene, para problemas de clasificación binarios, un nuevo atributo como combinación lineal de los originales, independientemente del número de éstos. Este atributo es el que maximiza la separación entre las medias de las clases, manteniendo los ejemplos tan agrupados como es posible alrededor de sus medias. Este criterio se puede expresar sucintamente en términos de las matrices de covarianza y posee una solución analítica muy elegante.

ADF es un método de reducción dimensional, no un algoritmo de clasificación. Cuando se utiliza como clasificador, la regla más utilizada consiste en asignar cada ejemplo a la clase de la media proyectada más próxima. No es necesario decir que esta regla no tiene por qué ser la regla óptima y, de hecho, es también habitual utilizar una red neuronal para aprovechar adecuadamente la proyección o proyecciones (en problemas con más de dos clases) de Fisher. Esta situación no es del todo satisfactoria pues supone optimizar dos criterios diferentes: el de Fisher en primer lugar y posteriormente el error cuadrático de clasificación.

El algoritmo propuesto en este trabajo soluciona este problema pues minimiza directamente el error de clasificación sin criterios intermedios. Al igual que en ADF, no se aprenden etiquetas como hacen las redes neuronales, sino que los ejemplos se clasifican por proximidad a las medias proyectadas. Sin embargo, a diferencia de lo que ocurre con ADF, en ADE la proyección es

óptima para esta regla de clasificación pues el único criterio que se optimiza es el número de ejemplos mal clasificados.

Ya que este criterio es discreto hemos de recurrir a un algoritmo capaz de optimizar cantidades no diferenciables. En concreto, ADE utiliza una estrategia evolutiva. La versión más elemental del algoritmo utiliza una estrategia evolutiva $1 + 1$ y se puede expresar de la siguiente manera:

- Se genera una proyección inicial.
- El valor o fitness de la proyección es el número de errores cometidos al asignar ejemplos a la clase de la media más cercana en el espacio proyectado.
- Se repiten los siguientes pasos hasta que se alcanza un número determinado de generaciones sin mejoras:
  - Se genera una nueva proyección añadiendo a los coeficientes de la actual una perturbación gaussiana.
  - Se calcula el número de ejemplos mal clasificados por la nueva proyección.
  - La nueva proyección sustituye a la actual cuando reduce el error.
- La proyección actual es la respuesta del algoritmo.

Una de las grandes ventajas de este algoritmo es que el número de proyecciones puede escogerse libremente. En el caso de ADF y de sus principales extensiones no lineales, como el Análisis Discriminante de Fisher con Núcleos (ADFN) y el Análisis Discriminante No Lineal (ADNL), el número de proyecciones es necesariamente igual al de clases menos uno. Esto es así debido al rango de una de las matrices de covarianza que intervienen en el criterio de Fisher. Sin embargo, ADE no optimiza este criterio y se libera de esta limitación de forma muy natural. Ahora sí es posible obtener proyecciones bidimensionales de problemas de clasificación con un número arbitrario de clases, lo cual tiene mucho interés para la visualización de problemas complejos.

A lo largo del trabajo se introducen distintos algoritmos que comparten el núcleo básico que acabamos de describir. ADEL es la versión lineal, es decir, busca una combinación lineal de atributos o proyección lineal. Una forma de

construir proyecciones no lineales consiste en combinar proyecciones lineales de forma jerárquica. Esta idea da lugar al Análisis Discriminante Evolutivo Jerárquico (ADEJ). Sin embargo, la extensión no lineal general de ADEL se denomina Análisis Discriminante Evolutivo (ADE) y se basa en una red neuronal con una capa oculta de pesos. Todos estos algoritmos se describen en las siguientes secciones y se comparan con los principales algoritmos de clasificación y discriminación de la literatura.

Este libro está dedicado a
mi familia (Elena y Alejandro)
y a mi madre Isabel.

Prefacio y Agradecimientos

"Generalmente, una tesis doctoral no es más que un traslado de huesos de una tumba a otra."
J. Frank Dobie

Una de las principales dificultades que afronté al realizar el doctorado fue la dispersión de mi labor investigadora. Al principio se abren, de forma atrayente, diversos campos de investigación y aplicaciones que invitan a picotear y ojear numerosos trabajos y publicaciones, y experimentar con múltiples herramientas y lenguajes. A medida que avancé en el conocimiento de cada área vi más difícil profundizar y aportar algo en todas ellas a la vez. Una vez superado el problema de tener que centrarme en un único tema concreto y desarrollarlo en detalle, la dificultad apareció al intentar gestionar, resumir y transmitir toda la cantidad de información, experimentos y conocimientos que se iban generando. Para ello seguí el enfoque de la *Navaja de Occam*, compartido y pontenciado en todo momento por mi director de tesis, el Dr. Alejandro Sierra. De esta forma aposté por algoritmos y soluciones lo más simples y sencillas posible que contra todo pronóstico pueden producir resultados igual de competentes que otros métodos más complejos pero de una forma mucho más intuitiva y clara.

El marco de investigación de este trabajo de tesis engloba como campos principales: el reconocimiento de patrones como objeto de estudio, centrándonos en el análisis discriminante clásico; y la computación evolutiva como

herramienta de investigación, haciendo especial uso de las estrategias evolutivas. El nuevo enfoque obtenido se denomina Análisis Discriminante Evolutivo (ADE) y conforma el pilar central de este libro. Al escribir el trabajo de tesis he intentado centrarme fundamentalmente en la parte innovadora y evitar "...trasladar huesos de una tumba a otra", como irónicamente resume la cita de J. Frank Dobie[1] que abre este prefacio. Para ello, también he reducido el estado del arte evitando así repetir conceptos e ideas demasiado básicas y extensas que con un buen número de referencias bien escogidas no sería necesario volverlas a explicar.

Siguiendo este enfoque, se ha dividido el libro en tres partes. El primer capítulo, titulado "Análisis Discriminante de Fisher y extensiones", está dedicado al estado del arte y en él se revisan los algoritmos ADF, ADFR, ADFN y ADNL que posteriormente se compararán con ADE. La frase anónima que lo encabeza "Los problemas ni se crean ni se destruyen, sólo se transforman" (modificación cómica de la famosísima cita de A. Einstein) resume de forma sencilla lo que en esencia hacen los algoritmos propuestos en ese capítulo: Transformar (o proyectar) de una forma u otra el problema original en otro diferente más sencillo. La segunda parte describe la versión lineal de ADE (denoninada ADEL) y comienza con la cita de Thomas Fuller "Todo es muy difícil antes de ser sencillo", con la que se quiere resumir la capacidad de ADE para convertir un problema original muy complejo en una representación sencilla e intuitiva. Finalmente se describe el algoritmo ADE completo en la última parte del libro y asocio el contenido de este capítulo con la cita anónima "Demasiada rectitud impide saborear el detalle". Aunque en su contexto se refiere al comportamiento del ser humano, en este trabajo intenta señalar la limitación lineal de ADEL (tratada en la segunda parte del libro) al no poder tratar satisfactoriamente problemas no lineales, y que sin embargo la versión completa de ADE (no lineal) sí puede tratar sin problemas. Todos los algoritmos tratados en este trabajo se han implementado en C/C++ y Matlab para obtener resultados (excepto ADNL que ha sido facilitado por la Dra. Ana González Marcos)

Este trabajo de tesis ha dado lugar a 5 publicaciones: 2 publicaciones en revista internacional [43] [45] y 3 publicaciones en congreso internacional [42] [44] [46]. La finalización de este trabajo de tesis significa para el autor, más que el final de un ciclo, el comienzo de una nueva etapa investigadora llena de posibilidades y proyectos.

[1] de su libro "Un tejano en Inglaterra" (*A texan in England*, 1944)

Me gustaría agradecer, en primer lugar, a mi director de tesis, Dr. Alejandro Sierra, todo su tiempo empleado en este trabajo y su gran dedicación. Ha trabajado conmigo desde el principio y me ha guiado durante todo el proceso de doctorado. Su gran experiencia, exigencia y perfeccionismo que le caracterizan, han sido uno de los grandes motores que me ha permitido completar esta tesis y a su vez encontrar un sentimiento de satisfacción con el trabajo realizado.

Este trabajo de tesis y sus publicaciones han sido subvencionados por el proyecto CICYT (Comisión Interdepartamental de Ciencia y Tecnología) número TIN2004-07676-C02-02, dirigido por el Dr. Alberto Suárez, al que estoy muy agradecido por su disponibilidad y atención. También quiero agradecer a la Dra. Ana González Marcos por facilitarme el software y resultados del algoritmo ADNL (Análisis Discriminante No Lineal) que detalla de forma central en su trabajo de tesis.

Quiero agradecer también al Dr. Manuel Sánchez-Montañés por ofrecerse como lector de la tesis y por sus detallados comentarios e instructivos consejos que han sido de gran ayuda para mi.

También quiero agradecer al Dr. Pablo Varona y al Dr. Francisco De Borja Rodríguez, por hacerme tan amáblemente un hueco en su laboratorio de investigación del Grupo de Neurocomputación Biológica (GNB), en el cual siempre he estado rodeado de grandes investigadores con los que he podido intercambiar ideas y argumentos en cualquier momento. A Elisa Barceló, Marisol Orta, Ana Barcenilla y al resto de personal de la Biblioteca Politécnica de la UAM por su ayuda en todo momento con mis necesidades bibliográficas y por permitirme acceder a las obras más actualizadas del momento. A Juana Calle que ha arrojado luz, siempre con una sonrisa, a normativas, procedimientos y papeleo, cuando me encontraba en la completa oscuridad. También a Marisa Moreno, por su disponibilidad y amabilidad en todo momento ayudándome a completar el proceso final de la entrega de la tesis. A Pilar y Jaci por convertir mi trabajo digital en un verdadero libro impreso, siempre muy agradables y eficientes.

Agradezco a mi hermana Elena sus consejos referentes a la redacción y sus interesantes conversaciones sobre la redacción de textos científicos. También agradezco profundamente el gran apoyo que me ha brindado mi madre Isabel, preguntándome día a día y animándome en todo momento a continuar y terminar este trabajo. A ella le dedico este trabajo, junto con mi mujer Elena y mi hijo Alejandro, a los que debo mi buen humor de cada día, mi ilusión

para hacer cualquier cosa desde que comienza hasta que termina el día y mi confianza para intentar conseguir todo lo que me propongo.

Hay muchas otras personas que me han ayudado, animado e influido durante mi doctorado y a los que agradezco brevemente a continuación. A los más cercanos, mis hermanos y familiares. A Los Muchachos (Soberano, Auténtico, Pete, Luisja, Luisco, Lerus, Sergio, Txete, Ra, Javi, Dani, Blancoso y PC), fieles seguidores de mis progresos en la docencia e investigación universitarias, y compañeros inseparables de todo tipo de fatigas. A mis compañeros del C-15, sobre todo a Borch, Frank, Luis y al Guaje, por sus interesantes conversaciones científicas y sobre el c-universo. También a mis compañeros de la UAM: Víctor T., Javier T., Fernando H., Daniel M., Fabiano B., Fernando L., Jordi P. y un largo etcétera, con los que he compartido mis 3 últimos años de doctorado.

Un agradecimiento final a muchos otros investigadores y docentes con los que he tenido productivos diálogos durante algún congreso o "summer school", y de quienes he aprendido numerosos conceptos y algoritmos nuevos de una forma sencilla y cercana.

Madrid, Enero 2009.
Alejandro Echeverría Rey

Indice

INDICE DE TABLAS

Indice de figuras

Capítulo 1

Análisis Discriminante de Fisher y extensiones

"Los problemas ni se crean ni se destruyen,
sólo se transforman."
Anónimo

Este capítulo está dedicado al Análisis Discriminante de Fisher (ADF) y a sus extensiones no lineales más importantes como son el Análisis Discriminante de Fisher con Núcleos (ADFN) y el Análisis Discriminante No Lineal (ADNL). La primera sección (1.1) introduce ADF, el punto de partida de esta tesis doctoral. Describe su ingenioso planteamiento que a pesar de su sencillez sigue haciendo que ADF sea uno de los métodos de discriminación más utilizados en la práctica. A pesar de su éxito, ADF es un método lineal y, por lo tanto, no puede resolver adecuadamente problemas de clasificación no lineales. En estos casos, es necesario recurrir a generalizaciones de la idea original como, por ejemplo, ADFN, que se estudia en la segunda sección (1.2). ADFN es un método basado en núcleos que transforma el espacio de atributos original en otro con mayor o incluso infinito número de dimensiones en el que un algoritmo lineal de clasificación es capaz de separar las clases correctamente. La siguiente sección (1.3) es una introducción del método ADNL que dispone de la capacidad de aproximación no lineal de un perceptrón multicapa aprovechando también la naturaleza libre de etiquetas del análisis discriminante de Fisher. La sección 1.4 está dedicada al análisis de los resultados experimentales y la última sección (1.5) a las conclusiones.

1.1. Análisis Discriminante de Fisher

ADF fue introducido por Fisher en 1936 [7] y posteriormente generalizado por Rao [34] para cualquier número de clases. Es un método de reducción dimensional que devuelve para dos clases un único atributo como combinación lineal de los originales sin importar el número de éstos. Este atributo "reducido" es óptimo en el siguiente sentido: las medias de las clases proyectadas sobre dicho atributo se encuentran situadas lo más lejos posible la una de la otra, mientras que los ejemplos proyectados de cada clase quedan lo más agrupados posible en torno a las proyecciones de las medias.

Es importante tener en mente que ADF es una técnica supervisada de reducción dimensional, no un algoritmo de clasificación. Es muy frecuente, sin embargo, utilizarlo como preprocesamiento antes de aplicar un algoritmo de clasificación. En estos casos, la regla de clasificación más sencilla consiste en asignar cada ejemplo a la clase cuya media esté más cercana en el espacio proyectado. Obviamente, esta regla no es necesariamente la óptima y, en general, es necesario recurrir a una red neuronal u otro algoritmo de clasificación para explotar al máximo la proyección de Fisher. Así, el proceso completo de clasificación acaba optimizando dos criterios diferentes: primero la medida de Fisher y después el error de clasificación en el espacio proyectado. En el próximo capítulo se discutirá cómo fundir estos dos criterios en uno solo y convertir el análisis discriminante en un verdadero algoritmo de clasificación.

La estructura de esta sección es la siguiente. Primero se analizará en detalle el funcionamiento de ADF para dos clases (1.1.1) y posteriormente su generalización a problemas con un número arbitrario de clases (1.1.2).

1.1.1. ADF para 2 clases

Supongamos que disponemos de un conjunto de ejemplos, cada uno de ellos caracterizado por d atributos $(x_1, \ldots, x_d)$ y marcado como perteneciente a una de las dos clases con medias $\mathbf{m}_1$ y $\mathbf{m}_2$. ADF busca una proyección $\mathbf{w}$ de d dimensiones a 1 de la siguiente forma:

$$\mathbf{x} = (x_1, \ldots, x_d) \rightarrow w_1 x_1 + \ldots + w_d x_d. \tag{1.1}$$

Para empezar, parece razonable exigir que $\mathbf{w}$ maximice la separación entre

las medias proyectadas. El cuadrado de esta distancia es el siguiente

$$\begin{aligned} J(\mathbf{w}) &= (\mathbf{w}'\mathbf{m}_2 - \mathbf{w}'\mathbf{m}_1)^2 = (\mathbf{w}'(\mathbf{m}_2 - \mathbf{m}_1))^2 \\ &= (\mathbf{w}'(\mathbf{m}_2 - \mathbf{m}_1))(\mathbf{w}'(\mathbf{m}_2 - \mathbf{m}_1))', \end{aligned} \tag{1.2}$$

donde $\mathbf{w}'$ es el vector $\mathbf{w}$ transpuesto. Teniendo en cuenta la siguiente relación

$$(\mathbf{w}'(\mathbf{m}_2 - \mathbf{m}_1))' = (\mathbf{m}_2 - \mathbf{m}_1)'\mathbf{w},$$

la distancia entre medias adopta la siguiente forma

$$J(\mathbf{w}) = \mathbf{w}'(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)'\mathbf{w} = \mathbf{w}'\mathbf{S}_b\mathbf{w} \tag{1.3}$$

donde $\mathbf{S}_b$ es la matriz de dispersión *entre clases* definida como:

$$\mathbf{S}_b = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)', \tag{1.4}$$

con $\mathbf{m}_i = \frac{1}{n_i}\sum_{j=1}^{n_i}\mathbf{x}_j$, siendo n_i el número de ejemplos de la clase i.

Este criterio se puede hacer tan grande como queramos aumentando la magnitud de $\mathbf{w}$, lo cual no tiene sentido dado que $\mathbf{w}$ es una proyección. Por esta razón reformulamos el criterio de la siguiente manera

$$J(\mathbf{w}) = \frac{\mathbf{w}'\mathbf{S}_b\mathbf{w}}{\mathbf{w}'\mathbf{w}}. \tag{1.5}$$

Podemos encontrar la proyección óptima de este criterio derivando esta cantidad respecto de $\mathbf{w}$ e igualando el gradiente a cero obteniendo

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{2\mathbf{S}_b\mathbf{w}(\mathbf{w}'\mathbf{w}) - 2\mathbf{w}\mathbf{w}'\mathbf{S}_b\mathbf{w}}{(\mathbf{w}'\mathbf{w})^2} = 0. \tag{1.6}$$

Teniendo en cuenta que las expresiones $(\mathbf{w}'\mathbf{w})$ y $(\mathbf{w}'\mathbf{S}_b\mathbf{w})$ son escalares, obtenemos la siguiente relación

$$\mathbf{w} \propto \mathbf{S}_b\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)'\mathbf{w}. \tag{1.7}$$

La expresión $[(\mathbf{m}_2 - \mathbf{m}_1)'\mathbf{w}]$ también es un escalar, de modo que la proyección óptima queda finalmente expresada como

$$\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1), \tag{1.8}$$

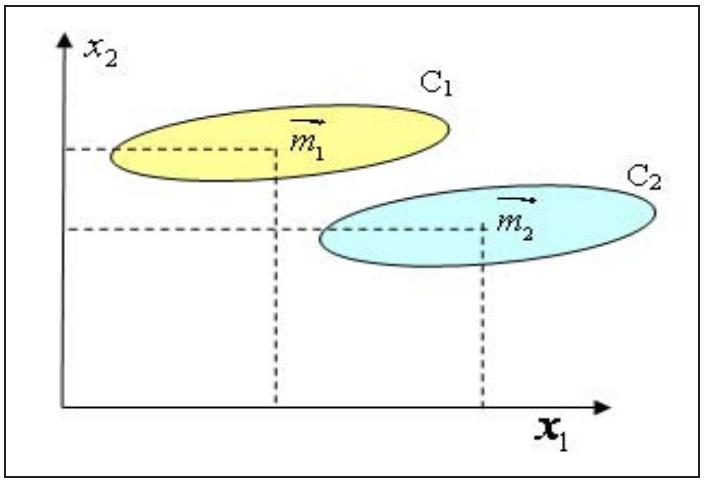

Figura 1.1: Efecto de la dispersión en ADF
En esta figura se muestra un conjunto de ejemplos dividido en dos clases con medias m_1 y m_2. La distancia entre las medias es mayor al proyectar en el eje x_1 que en el eje x_2. Sin embargo, el eje x_2 es preferible porque la dispersión de los ejemplos alrededor de su media es menor.

correspondiente al eje que une las medias y maximiza la distancia entre ellas.

En algunos problemas esta proyección no sirve para separar las clases correctamente por lo que ADF corrige este criterio buscando la proyección óptima que, además de maximizar la distancia entre las medias proyectadas como en el criterio anterior, minimiza la dispersión de los ejemplos proyectados de cada clase alrededor de su media (figura 1.1). Esto es equivalente a minimizar la varianza dentro de cada clase. Ambos objetivos se pueden satisfacer conjuntamente maximizando el cociente de la distancia entre medias y la varianza a lo largo de la proyección, lo que nos lleva al siguiente criterio:

$$J(\mathbf{w}) = \frac{\mathbf{w}'\mathbf{S}_b\mathbf{w}}{\mathbf{w}'\mathbf{S}_w\mathbf{w}}, \tag{1.9}$$

donde $\mathbf{S}_w$ es la matriz de dispersión *intra-clase* definida como:

$$\mathbf{S}_w = \sum_{i=1,2}\sum_{j=1}^{n_i}(\mathbf{x}_j - \mathbf{m}_i)(\mathbf{x}_j - \mathbf{m}_i)', \tag{1.10}$$

con $\mathbf{m}_i = \frac{1}{n_i}\sum_{j=1}^{n_i}\mathbf{x}_j$, siendo n_i el número de ejemplos de la clase i.

De nuevo, la proyección óptima se obtiene maximizando J como sigue

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \propto (\mathbf{w}'\mathbf{S}_w\mathbf{w})\mathbf{S}_b\mathbf{w} - (\mathbf{w}'\mathbf{S}_b\mathbf{w})\mathbf{S}_w\mathbf{w} = 0, \tag{1.11}$$

expresión de la que es fácil despejar el vector óptimo

$$\mathbf{w} \propto \mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w}. \tag{1.12}$$

De (1.7) y (1.8) sabemos que $\mathbf{S}_b\mathbf{w}$ es proporcional a la diferencia entre las medias de las clases, de tal forma que la expresión para la proyección óptima corregida por $\mathbf{S}_w$ es la siguiente:

$$\mathbf{w} \propto \mathbf{S}_w^{-1}(\mathbf{m}_2 - \mathbf{m}_1). \tag{1.13}$$

1.1.2. ADF multiclase

La forma más habitual de generalizar el criterio de Fisher (1.9) para más de dos clases consiste en sustituir varianzas por covarianzas y cocientes simples por cocientes de determinantes. Este procedimiento se basa en el hecho de que el determinante de una matriz de covarianza, conocido como varianza generalizada, es el producto de las varianzas a lo largo de las direcciones de las componentes principales. Puede verse [34] que la proyección $\mathbf{W}$ de d a d' dimensiones maximizando el siguiente criterio generalizado

$$J(\mathbf{W}) = \frac{|\mathbf{W}'\mathbf{S}_b\mathbf{W}|}{|\mathbf{W}'\mathbf{S}_w\mathbf{W}|}, \tag{1.14}$$

se obtiene resolviendo el siguiente problema de autovectores generalizado

$$\mathbf{S}_b\mathbf{w}_i = \lambda_i\mathbf{S}_w\mathbf{w}_i. \tag{1.15}$$

La dimensión d' de la proyección puede ser como máximo $c-1$ (siendo c el número de clases) debido a que como mucho $c-1$ de los λ_i autovalores son distintos de cero. Esto es debido a la limitación impuesta por el rango de la matriz $\mathbf{S}_b$ que se construye como la suma de c matrices cuyo rango es 1 como máximo. Los correspondientes autovectores normalizados forman las $c-1$ columnas de la proyección óptima $\mathbf{W}$. Este problema generalizado puede transformarse en un problema convencional de autovalores invirtiendo $\mathbf{S}_w$. No obstante, no es necesario dado que los autovalores se pueden calcular primero como raíces del siguiente polinomio característico

$$|\mathbf{S}_b - \lambda_i\mathbf{S}_w| = 0. \tag{1.16}$$

Posteriormente hay que resolver (1.15) únicamente para los autovalores no nulos.

ADF, como acabamos de ver, es una medida inteligente de separación de clases con una elegante solución analítica en términos de la inversa de la matriz de covarianza intra-clases, y que ha sido aplicado con éxito a muchos problemas prácticos: reconocimiento de caras [24] [53], reconocimiento de voz [21][32], etc. A pesar de ser uno de los métodos más utilizados en la práctica, el análisis discriminante clásico presenta diversas limitaciones. Una de ellas es la singularidad de la matriz de covarianza intra-clases $\mathbf{S}_w$ que dificulta la obtención del vector de proyección. Otra de las limitaciones de ADF es debida a que la dimensión del espacio sobre el que se proyecta es igual al número de clases menos uno. Como ya hemos visto anteriormente esta limitación la impone el rango de la matriz S_b. Esto implica que la dimensión del espacio de proyección está previamente fijado y no puede ser elegida libremente. Esta situación dificulta la visualización del resultado obtenido por ADF para problemas con más de tres clases. Por último, ADF es un método lineal y, por lo tanto, no puede resolver correctamente problemas no lineales que son tan frecuentes en la práctica.

1.2. Análisis Discriminante de Fisher con Núcleos

Recientemente el análisis discriminante de Fisher con Núcleos (ADFN) ha cobrado gran protagonismo [28] [52]. Los resultados de clasificación de ADFN son comparables a los que se obtienen utilizando máquinas de vectores de soporte (MVS) [38]. ADFN optimiza el criterio de Fisher en un espacio transformado, denominado espacio de características, definido implícitamente por una función de núcleo encontrando de esta forma un discriminante no lineal en el espacio de entrada. En el primer apartado (1.2.1) se describe ADFN para dos clases y posteriormente (1.2.2) se generaliza para un número arbitrario de clases.

1.2.1. ADFN para 2 clases

El objetivo del análisis discriminante de Fisher con núcleos (ADFN) es optimizar el criterio de Fisher en un espacio de características $\mathcal{F}$, produciendo a la vez un discriminante no lineal en el espacio de características original. Si Φ es una transformación no lineal que genera un espacio de características

$\mathcal{F}$, podemos escribir el criterio de Fisher para dos clases (1.9) en $\mathcal{F}$ como

$$J^{\Phi}(\mathbf{w}) = \frac{\mathbf{w}'\mathbf{S}_b^{\Phi}\mathbf{w}}{\mathbf{w}'\mathbf{S}_w^{\Phi}\mathbf{w}}, \tag{1.17}$$

donde ahora $\mathbf{w}$ es una proyección en $\mathcal{F}$, y $\mathbf{S}_b^{\Phi}$ y $\mathbf{S}_w^{\Phi}$ son las matrices $\mathbf{S}_b$ y $\mathbf{S}_w$ en $\mathcal{F}$ respectivamente, es decir

$$\mathbf{S}_b^{\Phi} = (\mathbf{m}_2^{\Phi} - \mathbf{m}_1^{\Phi})(\mathbf{m}_2^{\Phi} - \mathbf{m}_1^{\Phi})' \tag{1.18}$$

$$\mathbf{S}_w^{\Phi} = \sum_{i=1,2}\sum_{j=1}^{n_i}(\Phi(\mathbf{x}_j) - \mathbf{m}_i^{\Phi})(\Phi(\mathbf{x}_j) - \mathbf{m}_i^{\Phi})', \tag{1.19}$$

donde $\mathbf{m}_i^{\Phi} = \frac{1}{n_i}\sum_{j=1}^{n_i}\Phi(\mathbf{x}_j)$ y n_i es el número de ejemplos de la clase i. Si el espacio transformado $\mathcal{F}$ es de alta dimensión, será prácticamente imposible resolver este problema directamente. Sin embargo, sí es posible hacerlo expresando el producto escalar en el espacio transformado como una función del producto escalar en el espacio original mediante la siguiente expresión

$$\Phi'(\mathbf{x})\Phi(\mathbf{y}) = k(\mathbf{x}, \mathbf{y}), \tag{1.20}$$

siendo $\mathbf{x}$ e $\mathbf{y}$ ejemplos en el espacio original, $\Phi(\mathbf{x})$ y $\Phi(\mathbf{y})$ sus respectivas transformaciones en $\mathcal{F}$ y k la función denominada función de núcleo. De esta forma podremos calcular productos escalares en $\mathcal{F}$ sin llegar a calcular explícitamente las transformaciones de los ejemplos originales, permitiéndonos resolver el problema original en el espacio transformado.

Para optimizar el criterio de Fisher en el espacio de características $\mathcal{F}$ necesitamos una formulación de (1.17) únicamente en términos de productos escalares entre los ejemplos de entrenamiento. Posteriormente serán sustituidos por una función de núcleo adecuada. Sabemos que cualquier proyección $\mathbf{w}$ puede escribirse en $\mathcal{F}$ como [27]:

$$\mathbf{w} = \sum_{j=1}^{n}\alpha_j\Phi(\mathbf{x}_j), \tag{1.21}$$

donde n es el número total de ejemplos y α_j $(j = 1, \ldots, n)$ son números reales. Utilizando esta expresión y la definición de $\mathbf{m}_i^{\Phi}$ podemos escribir

$$\begin{aligned}\mathbf{w}'\mathbf{m}_i^{\Phi} &= \frac{1}{n_i}\sum_{j=1}^{n}\sum_{k=1}^{n_i}\alpha_j k(\mathbf{x}_j, \mathbf{x}_k) \\ &= \boldsymbol{\alpha}'\mathbf{M}_i \end{aligned} \tag{1.22}$$

donde hemos definido $(\mathbf{M}_i)_j = \frac{1}{n_i}\sum_{k=1}^{n_i} k(\mathbf{x}_j, \mathbf{x}_k^i)$, con $j = 1, \ldots, n$, y hemos sustituido los productos escalares por la función de núcleo correspondiente.

Mediante (1.22) y (1.18) podemos reescribir el numerador de (1.17) como

$$\begin{aligned} \mathbf{w}'\mathbf{S}_b^\Phi\mathbf{w} &= \mathbf{w}'(\mathbf{m}_2^\Phi - \mathbf{m}_1^\Phi)(\mathbf{m}_2^\Phi - \mathbf{m}_1^\Phi)'\mathbf{w} \\ &= (\mathbf{w}'\mathbf{m}_2^\Phi - \mathbf{w}'\mathbf{m}_1^\Phi)(\mathbf{w}'\mathbf{m}_2^\Phi - \mathbf{w}'\mathbf{m}_1^\Phi)' \\ &= (\boldsymbol{\alpha}'\mathbf{M}_2 - \boldsymbol{\alpha}'\mathbf{M}_1)(\boldsymbol{\alpha}'\mathbf{M}_2 - \boldsymbol{\alpha}'\mathbf{M}_1)' \\ &= \boldsymbol{\alpha}'(\mathbf{M}_2 - \mathbf{M}_1)(\mathbf{M}_2 - \mathbf{M}_1)'\boldsymbol{\alpha} \\ &= \boldsymbol{\alpha}'\mathbf{F}_b\boldsymbol{\alpha}, \end{aligned} \tag{1.23}$$

donde $\mathbf{F}_b = (\mathbf{M}_2 - \mathbf{M}_1)(\mathbf{M}_2 - \mathbf{M}_1)'$. Por otro lado, también podemos reescribir el denominador de (1.17) utilizando las expresiones (1.22) y (1.19), de la siguiente manera

$$\mathbf{w}'\mathbf{S}_w^\Phi\mathbf{w} = \boldsymbol{\alpha}'\mathbf{F}_w\boldsymbol{\alpha} \tag{1.24}$$

donde $\mathbf{F}_w = \sum_{i=1,2}\mathbf{K}_i(\mathbf{I} - \mathbf{1}_{n_i})\mathbf{K}_i'$, $\mathbf{I}$ es la matriz identidad, $\mathbf{1}_{n_j}$ es la matriz con todos sus elementos igual a $\frac{1}{n_j}$, y $\mathbf{K}_i$ es la matriz de núcleo para la clase i de dimensión $n \times n_i$

$$(\mathbf{K}_i)_{jk} = \sum_{j=1}^{n}\sum_{k=1}^{n_i} k(\mathbf{x}_j, \mathbf{x}_k^i),$$

siendo $\mathbf{x}_k^i$ los ejemplos pertenecientes a la clase i. Combinando (1.23) y (1.24) podemos reescribir (1.17) nuevamente como

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}'\mathbf{F}_b\boldsymbol{\alpha}}{\boldsymbol{\alpha}'\mathbf{F}_w\boldsymbol{\alpha}}. \tag{1.25}$$

De manera análoga a la realizada en el espacio original, podemos maximizar este cociente resolviendo el problema de autovectores generalizado $\mathbf{F}_b\boldsymbol{\alpha} = \lambda\mathbf{F}_w\boldsymbol{\alpha}$, es decir, calculando el autovector principal de $\mathbf{F}_w^{-1}\mathbf{F}_b$. Este nuevo enfoque no lineal es el que hemos denominado anteriormente como análisis discriminante de Fisher con núcleos (ADFN). La proyección de un nuevo ejemplo $\mathbf{z}$ sobre $\mathbf{w}$ viene dada por

$$\mathbf{w}'\Phi(\mathbf{z}) = \sum_{j=1}^{n}\alpha_j k(\mathbf{x}_j, \mathbf{z}). \tag{1.26}$$

Posteriormente necesitaremos un criterio de clasificación que normalmente asignará un nuevo ejemplo a la clase cuya media proyectada sea la más cercana.

1.2.2. ADFN multiclase

ADFN se puede generalizar a problemas con más de dos clases de la misma forma que ADF. La proyección vectorial $\mathbf{w}$ se sustituye por una matriz $\mathbf{W}$ que contiene $c-1$ direcciones, siendo c el número de clases del problema, y el cociente simple (1.17) se sustituye por un cociente de determinantes como sigue

$$J(\mathbf{W}) = \frac{|\mathbf{W}'\mathbf{S}_b^{\Phi}\mathbf{W}|}{|\mathbf{W}'\mathbf{S}_w^{\Phi}\mathbf{W}|}, \tag{1.27}$$

donde

$$\begin{aligned} \mathbf{S}_b^{\Phi} &= \sum_{i=1}^{c} n_i(\mathbf{m}_i^{\Phi} - \mathbf{m}^{\Phi})(\mathbf{m}_i^{\Phi} - \mathbf{m}^{\Phi})', \\ \mathbf{S}_w^{\Phi} &= \sum_{i=1}^{c}\sum_{j=1}^{n_i}(\Phi(\mathbf{x}_j) - \mathbf{m}_i^{\Phi})(\Phi(\mathbf{x}_j) - \mathbf{m}_i^{\Phi})', \end{aligned} \tag{1.28}$$

con $\mathbf{m}^{\Phi} = \frac{1}{n}\sum_{j=1}^{n}\Phi(\mathbf{x}_j)$, $\mathbf{m}_i^{\Phi} = \frac{1}{n_i}\sum_{j=1}^{n_i}\Phi(\mathbf{x}_j)$, donde n_i es el número de ejemplos de la clase número i, n es el número total de ejemplos e $i = 1, \ldots, c$. Utilizando las transformaciones (1.23) y (1.24) del apartado anterior podemos expresar (1.27) como

$$J(\mathbf{A}) = \frac{|\mathbf{A}'\mathbf{F}_b\mathbf{A}|}{|\mathbf{A}'\mathbf{F}_w\mathbf{A}|} \tag{1.29}$$

donde $\mathbf{A}$ es la matriz formada por los vectores $(\boldsymbol{\alpha}_1', \ldots, \boldsymbol{\alpha}_{c-1}')$.

Al igual que en el espacio original, también en el transformado podemos encontrar la matriz $\mathbf{A}^*$ que maximiza el criterio de Fisher gracias a los $c-1$ autovectores de $\mathbf{F}_w^{-1}\mathbf{F}_b$ correspondientes a los $c-1$ autovalores no nulos. Una vez hayamos encontrado $\mathbf{A}^*$ podremos proyectar un ejemplo $\mathbf{z}$ sobre el espacio discriminante $(c-1)$-dimensional definido por las columnas de $\mathbf{A}^*$ obteniendo el vector proyectado $\mathbf{y}_\mathbf{z}$ de dimensión $(c-1)$ como se muestra a continuación

$$\mathbf{y}_\mathbf{z} = \mathbf{A}^{*\prime}\mathbf{K}^\mathbf{z}, \tag{1.30}$$

donde $\mathbf{K}^\mathbf{z} = \sum_{j=1}^{n} k(\mathbf{x}_j, \mathbf{z})$. Finalmente necesitaremos una regla de clasificación en el espacio proyectado que asigne cada ejemplo $\mathbf{z}$ a una de las c clases. Lo más habitual es clasificar por distancia a las medias proyectadas, es decir

$$\text{clase}(\mathbf{z}) = \underset{i}{\operatorname{argmin}}\, d(\mathbf{y}_\mathbf{z}, \overline{\mathbf{m}}_i), \tag{1.31}$$

donde $\overline{\mathbf{m}}_i$ es la media proyectada de la clase i y $d(\mathbf{y_z}, \overline{\mathbf{m}}_i)$ es la distancia entre el vector $\mathbf{y_z}$ y $\overline{\mathbf{m}}_i$.

1.3. Análisis Discriminante No Lineal

El análisis discriminante no lineal (ADNL) [37] es un algoritmo de extracción de características que combina la proyección no lineal de un perceptrón multicapa (PMC) [2] con el análisis discriminante de Fisher (ADF). De esta forma ADNL dispone de la capacidad de aproximación no lineal de un PMC aprovechando también la naturaleza libre de etiquetas de ADF. Aunque el PMC es un *aproximador universal* puede encontrarse con serias dificultades al tratar problemas con clases de tamaños muy descompensados y un grado muy alto de mezcla entre ellas. Es en esos casos en los que ADNL obtiene resultados especialmente buenos debido a que ADF es menos sensible al desequilibrio entre clases.

En el primer apartado (1.3.1) se muestra la arquitectura de ADNL y en 1.3.2 se describe su algoritmo de entrenamiento.

1.3.1. Arquitectura de ADNL

ADNL mantiene gran similitud en su arquitectura con un perceptrón multicapa tradicional. Dicha arquitectura consiste en diversas capas de neuronas (nodos) con interconexiones unidireccionales de todas las neuronas de una capa con cada una de las neuronas de la siguiente. La primera y última corresponden a la capa de entrada y salida respectivamente. El resto son denominadas *capas ocultas* que, en el caso de haber sólo una, daría lugar a la arquitectura más sencilla de ADNL formada por tres capas en total. La capa de entrada, al igual que en un PMC tradicional, tiene $d+1$ nodos correspondientes a la dimensión d del problema más una entrada adicional que siempre vale 1. Las capas ocultas también son iguales que en un PMC, es decir, cada capa oculta i ($i = 1, \ldots, l$), siendo l el número de capas ocultas de la red, tiene un número h_i de neuronas con funciones de activación dadas por $\varphi(x) = 1/(1+e^{-x})$. En todas ellas, menos en la que conecta directamente con la capa de salida, se incluye a su vez una neurona extra con valor 1. La capa de salida, principal diferencia con un PMC, contiene un número de neuronas igual al de clases menos 1. Esta es la dimensión de la proyección obtenida por ADF.

Las conexiones entre las neuronas de la red, denominadas pesos, se representan mediante la matriz $\mathbf{W}$. En ADNL esta matriz se descompone en dos partes, $\mathbf{W} = (\mathbf{W}^e, \mathbf{W}^s)$, facilitando la representación en los cálculos que se llevarán a cabo durante el aprendizaje. La primera, $\mathbf{W}^e$, denota los pesos que terminan en la capa oculta e, con $e = 1, \ldots, l$. La segunda parte es $\mathbf{W}^s$ y denota los pesos conectados con la capa de salida. Para una mejor comprensión del algoritmo en el siguiente apartado, vamos a centrarnos a partir de ahora en la arquitectura de una única capa oculta con h neuronas como se muestra en la figura 1.2.

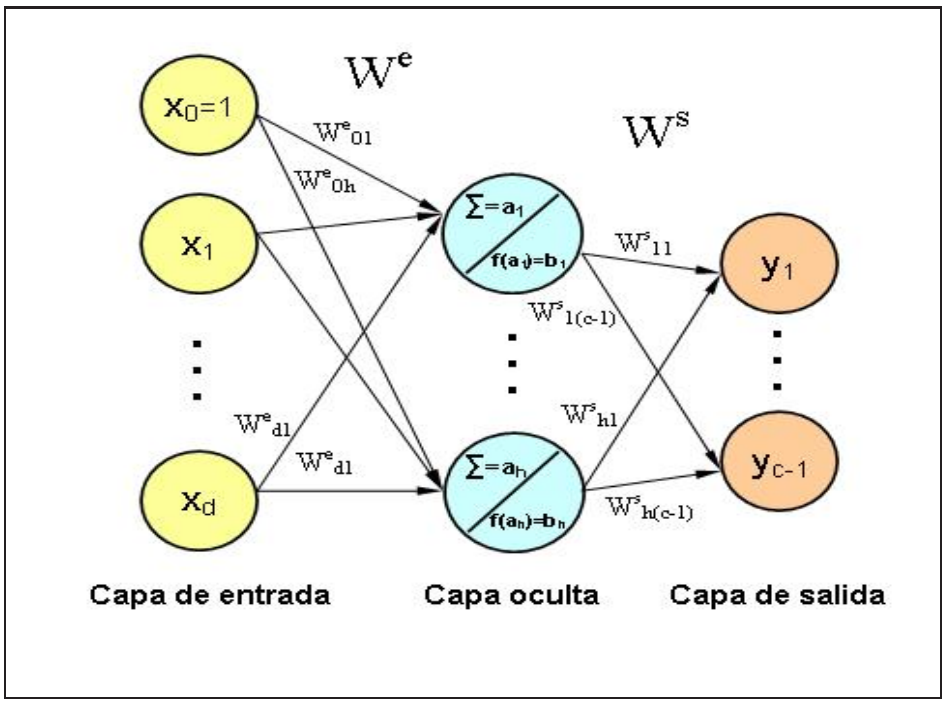

Figura 1.2: Arquitectura de una red ADNL
Arquitectura de una red ADNL formada por una capa de entrada (en amarillo) con $d + 1$ neuronas de entrada, siendo d la dimensión del problema; una capa intermedia denominada capa oculta (en azul) con h neuronas y una capa de salida (en rojo) con $c - 1$ neuronas de salida. Los pesos $\mathbf{W}^e$ conectan cada neurona de entrada con la capa oculta y los pesos $\mathbf{W}^s$ conectan las neuronas ocultas con las neuronas de la capa de salida.

1.3.2. Algoritmo de entrenamiento de ADNL

ADNL también mantiene gran similitud en su funcionamiento con un PMC tradicional. Las diferencias residen en la función objetivo utilizada en el aprendizaje, la codificación de los vectores objetivo y la actualización de los pesos de la red, como veremos a continuación.

En un PMC tradicional la función objetivo viene definida como

$$\mathcal{E}(\mathbf{W}) = \sum_{j=1}^{n} \|\hat{\mathbf{y}}_j - \mathcal{F}(\mathbf{x}_j, \mathbf{W})\|^2,$$

donde $\mathbf{x}_j$ es el ejemplo número j, n es el número total de ejemplos y el vector objetivo $\hat{\mathbf{y}}_j$ representa la clase a la que pertenece el ejemplo $\mathbf{x}_j$. $\mathcal{F}(\mathbf{x}_j, \mathbf{W})$ es la función de transferencia o salida propuesta por la red para el vector $\mathbf{x}_j$. En una arquitectura con una capa oculta donde y_i es la salida número i de la red ($i = 1, \ldots, v$) y v es el número de neuronas de salida, la función de transferencia $\mathcal{F}_i$ se calcula mediante la expresión

$$y_i = \sum_{p=0}^{h} \varphi(\sum_{k=0}^{d} x_k w_{kp}^e) w_{pi}^s,$$

donde d es la dimensión del problema, h es el número de neuronas de la capa oculta, w_{kp}^e es el peso que conecta la neurona k de entrada con la neurona p de la capa oculta, y w_{pi}^s es el peso que conecta la neurona p de la capa oculta con la neurona i de la capa de salida. En ADNL la función objetivo $\mathcal{E}$ se sustituye por el inverso del criterio original de Fisher (1.14), es decir, ADNL minimiza el siguiente criterio

$$\mathcal{J}(\mathbf{W}) = \frac{|\tilde{\mathbf{S}}_w|}{|\tilde{\mathbf{S}}_b|}, \tag{1.32}$$

donde $\mathbf{W}$ son los pesos de la red y las matrices de covarianza tienen las siguientes expresiones

$$\begin{aligned} \tilde{\mathbf{S}}_w &= \mathbf{W}'\mathbf{S}_w\mathbf{W} \\ \tilde{\mathbf{S}}_b &= \mathbf{W}'\mathbf{S}_b\mathbf{W}. \end{aligned} \tag{1.33}$$

El algoritmo de aprendizaje de ADNL, al igual que el PMC tradicional, es iterativo. Cada una de sus iteraciones recibe el nombre de época. En cada

época t se presentan todos los ejemplos de entrenamiento a la red y se obtienen los pesos $\mathbf{W}_{t+1}$ de la época siguiente. La actualización se realiza en dos etapas. Primero se calculan los pesos $\mathbf{W}^s$ resolviendo el criterio de Fisher $\mathcal{J}(\mathbf{W}^s)$ explicado en el apartado 1.1.2 y posteriormente se calculan los pesos óptimos $\mathbf{W}^e$ minimizando $\mathcal{J}(\mathbf{W}^e)$ mediante un procedimiento *cuasi-Newton*. Este método es una aproximación del método de Newton que es computacionalmente más costoso. La regla de actualización de los pesos en el método de Newton es la siguiente:

$$\mathbf{W}^e_{t+1} = \mathbf{W}^e_t + \mathbf{H}^{-1}\nabla\mathcal{J}(\mathbf{W}^e), \tag{1.34}$$

donde $\mathbf{H}$ es la matriz del Hessiano y sus elementos son las segundas derivadas de la función de error respecto de los pesos de la red, es decir,

$$H_{ij} = \frac{\partial^2\mathcal{J}(\mathbf{W})}{\partial w_i \partial w_j}.$$

En el método *cuasi-Newton* se construye una aproximación de la inversa de $\mathbf{H}$ sin necesidad de calcular las derivadas de segundo orden. La regla de actualización de los pesos es la siguiente:

$$\mathbf{W}^e_{t+1} = \mathbf{W}^e_t + \alpha_t\mathbf{G}_t\nabla\mathcal{J}(\mathbf{W}^e), \tag{1.35}$$

donde α_t es un factor que depende del método de minimización y $\mathbf{G}_t$ es la aproximación a la matriz del Hessiano en la iteración t. Al comienzo del entrenamiento $\mathbf{G}_0$ es una matriz simétrica y definida positiva, de modo que normalmente se elige la matriz identidad que hace que la primera iteración se realice en la dirección marcada por el gradiente $-\nabla\mathcal{J}(\mathbf{W}^e)$. Posteriormente se van calculando las matrices $\alpha_t\mathbf{G}_t$ de cada *época* del entrenamiento mediante la fórmula *Broyden-Fletcher-Goldfar-Shanno* [29]. El gradiente para cada ejemplo de entrenamiento presentado a la red se puede expresar de la siguiente forma, siguiendo la codificación utilizada en la figura 1.2:

$$\begin{aligned}
\frac{\partial\mathcal{J}}{\partial W^e_{kl}} &= \sum_{p=1}^{h}\frac{\partial\mathcal{J}}{\partial b_p}\frac{\partial b_p}{\partial a_p}\frac{\partial a_p}{\partial W^e_{kl}} \\
&= \sum_{p=1}^{h}\frac{\partial\mathcal{J}}{\partial b_p}f'(a_p)x_k\delta_{pl} \\
&= \frac{|\tilde{\mathbf{S}}_b|\frac{\partial|\tilde{\mathbf{S}}_w|}{\partial b_l} - |\tilde{\mathbf{S}}_w|\frac{\partial|\tilde{\mathbf{S}}_b|}{\partial b_l}}{|\tilde{\mathbf{S}}_b|^2}f'(a_l)x_k,
\end{aligned} \tag{1.36}$$

donde $k = 0, \ldots, d$, $l = 1, \ldots, h$ y δ_{pl} es la función *delta de Kronecker*. La derivada del determinante de una matriz simétrica $\mathbf{A}$ puede calcularse de la siguiente manera:

$$\frac{\partial |\mathbf{A}|}{\partial x} = |\mathbf{A}| Tr(\mathbf{A}^{-1} \frac{\partial \mathbf{A}}{\partial x}).$$

Como $\tilde{\mathbf{S}}_b$ y $\tilde{\mathbf{S}}_w$ son simétricas, sus derivadas se pueden expresar como sigue:

$$\begin{aligned} \frac{\partial |\tilde{\mathbf{S}}_b|}{\partial b_l} &= |\tilde{\mathbf{S}}_b| Tr(\tilde{\mathbf{S}}_b^{-1} \frac{\partial \tilde{\mathbf{S}}_b}{\partial b_l}), \\ \frac{\partial |\tilde{\mathbf{S}}_w|}{\partial b_l} &= |\tilde{\mathbf{S}}_w| Tr(\tilde{\mathbf{S}}_w^{-1} \frac{\partial \tilde{\mathbf{S}}_w}{\partial b_l}). \end{aligned} \tag{1.37}$$

Para calcular $\partial \tilde{\mathbf{S}}_b / \partial b_l$ y $\partial \tilde{\mathbf{S}}_w / \partial b_l$ utilizamos (1.33) y el hecho de que los pesos W^s están fijos

$$\begin{aligned} \frac{\partial \tilde{\mathbf{S}}_b}{\partial b_l} &= (W^s)' \frac{\partial \mathbf{S}_b}{\partial b_l} W^s \\ \frac{\partial \tilde{\mathbf{S}}_w}{\partial b_l} &= (W^s)' \frac{\partial \mathbf{S}_w}{\partial b_l} W^s. \end{aligned} \tag{1.38}$$

Por lo tanto, para obtener finalmente $\partial \mathcal{J} / \partial W_{kl}^e$ son necesarias las derivadas $\partial \mathbf{S}_b / \partial b_l$ y $\partial \mathbf{S}_w / \partial b_l$ que podremos calcular si reescribimos las matrices $\mathbf{S}_b$ y $\mathbf{S}_w$ en función del vector $\mathbf{b}$ de la capa oculta de la red como sigue

$$\begin{aligned} \mathbf{S}_b &= \sum_{i=1}^{c} n_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})' \\ \mathbf{S}_w &= \sum_{i=1}^{c} \sum_{j=1}^{n_i} (\mathbf{b}_{ij} - \mathbf{m}_i)(\mathbf{b}_{ij} - \mathbf{m}_i)', \end{aligned} \tag{1.39}$$

donde $\mathbf{m}_i = 1/n_i \sum_{j=1}^{n_i} \mathbf{b}_{ij}$ y $m = 1/n \sum_{i=1}^{c} \sum_{j=1}^{n_i} \mathbf{b}_{ij}$.

Una vez actualizados los pesos se clasifican los ejemplos de entrenamiento y *validación* en cada época del entrenamiento. La regla más sencilla de

clasificación consiste en asignar cada ejemplo $\mathbf{x}$ a la clase v cuya media esté más cercana en el espacio de salida de la red como sigue

$$\text{clase}(\mathbf{x}) = \arg\min_{v}(\sum_{i=1}^{c-1}(y_i - \tilde{m}_i^v)^2), \tag{1.40}$$

donde

$$\begin{aligned} y_i &= \mathcal{F}_i(\mathbf{x}, \mathbf{W}) \\ &= \sum_{p=1}^{h} \varphi(\sum_{k=0}^{d} x_k w_{kp}^e) w_{pi}^s \end{aligned} \tag{1.41}$$

es la salida producida por ADNL para el ejemplo $\mathbf{x}$, y donde

$$\begin{aligned} \tilde{m}_i^v &= \mathcal{F}_i(\mathbf{m}^v, \mathbf{W}) \\ &= \sum_{p=1}^{h} \varphi(\sum_{k=0}^{d} m_k^v w_{kp}^e) w_{pi}^s \end{aligned} \tag{1.42}$$

es la salida producida por ADNL para la media de la clase v, $\mathbf{m}^v$.

Los ejemplos de validación no intervienen en la actualización de los pesos y, de esta forma, el error cometido sobre ellos sirve para prevenir el sobreaprendizaje. La suma del error cometido sobre los ejemplos de entrenamiento y validación servirá para medir la calidad de dicha combinación de pesos y deberá disminuir de una época a la siguiente. Cuando dicho valor aumente, el algoritmo ADNL habrá finalizado y tomará el anterior valor de $\mathbf{W}$ como resultado final.

1.4. Resultados experimentales

Hemos aplicado los algoritmos ADF, ADFN y ADNL, presentados en este capítulo, a una batería de bases de datos elegidas de diferentes repositorios y descritas en detalle en el apéndice A. Forman una selección heterogénea compuesta por problemas con distinto número de clases, número de atributos y número de ejemplos. Las bases de datos se han dividido en tres partes: entrenamiento, validación y test.

La tabla 1.1 muestra un resumen de las bases de datos utilizadas en este trabajo. La primera columna indica el nombre de la base de datos y

BD	C	Ejem	Atr	Z_0	RNAl	RNAnl	Wilcoxon
cancer1	2	699	9	34.5	1.72	(5) 1.44	significativa
card1	2	690	51	44.0	14.97	(5) 15.26	no significativa
diabetes1	2	768	8	34.9	25.36	(5) 28.00	significativa
digitos35	2	681	225	49.3	2.67	(10) 2.61	no significativa
mushroom1	2	8124	125	48.0	0.12	(5) 0.00	significativa
spam1	2	4601	57	39.4	7.20	(5) 7.49	no significativa
gene1	3	3175	120	50.0	12.54	(10) 13.18	significativa
horse1	3	364	58	38.0	30.83	(5) 30.06	no significativa
iris	3	150	4	66.7	5.33	(3) 2.00	significativa
thyroid2	3	7200	21	7.4	4	(5) 1.67	significativa
car	4	1728	6	30.0	18.79	(5) 5.81	significativa
glass1	6	214	9	64.5	31.70	(10) 29.15	significativa
satellite	6	6434	36	76.2	22.39	(10) 11.68	significativa
soybean1	19	683	82	86.5	16.41	(15) 7.29	significativa

Tabla 1.1: Resumen de Bases de datos

Esta tabla resume las bases de datos utilizadas en este trabajo de tesis. La primera columna contiene el nombre de la base de datos y las columnas 2, 3 y 4 muestran sus características: número de clases, número de ejemplos y número de atributos, respectivamente. La columna 5 contiene el error base y las columnas 6 y 7 muestran los resultados promediados de 20 ejecuciones de una red neuronal lineal (RNAl) y no lineal con una capa oculta (RNAnl), respectivamente. Los números entre paréntesis indican el número de unidades de la capa oculta. La última columna revela, mediante un test de Wilcoxon no paramétrico [40], si la diferencia entre los resultados obtenidos en las columnas RNAl y RNAnl es significativa o no.

las columnas 2, 3 y 4 muestran el número de clases, ejemplos y atributos de cada una de ellas, respectivamente. La columna 5 indica el error de test que se obtiene al clasificar todos los ejemplos en la clase mayoritaria. Las dos columnas siguientes, 6 y 7, contienen los resultados promediados de 20 ejecuciones de una red neuronal lineal (RNAl) y no lineal con una capa oculta (RNAnl) [33], respectivamente. Los números entre paréntesis presentes en la columna RNAnl indican el número de unidades de la capa oculta que ha utilizado el algoritmo para obtener su mejor solución. La última columna indica si la diferencia entre los resultados obtenidos en las columnas RNAl y RNAnl es significativa. Para ello hemos aplicado un test de Wilcoxon no paramétrico [40] utilizado también en todos los experimentos de este trabajo para analizar la significación de los resultados obtenidos.

Los algoritmos utilizan exclusivamente los conjuntos de entrenamiento y validación durante el proceso de aprendizaje. De esta forma, el error cometido en el conjunto de test es una medida de la capacidad de generalización de los

algoritmos. La regla de clasificación es la misma para todos los algoritmos: los ejemplos se asignan a la clase cuya media proyectada se encuentra más próxima. Estas medias se calculan utilizando exclusivamente los ejemplos del conjunto de entrenamiento.

Los resultados de los experimentos se muestran al final de este capítulo en la tabla 1.4, que es una ampliación de la tabla 1.1 en la que los valores de las 7 últimas columnas representan porcentajes de error de clasificación en el conjunto de test. La primera columna es el nombre de la base de datos. Las columnas 2, 3 y 4 contienen el número de clases, número de ejemplos y número de atributos del problema, respectivamente. La columna 5 muestra el error que se obtiene al clasificar todos los ejemplos como pertenecientes a la clase mayoritaria. Las dos columnas siguientes, 6 y 7, corresponden a resultados obtenidos mediante una red neuronal lineal y no lineal (con una capa oculta), respectivamente. La diferencia entre ambos resultados puede constituir un indicador del nivel de no-linealidad del problema planteado por cada base de datos. Las cuatro últimas columnas, de la 8 a la 11, pertenecen a los algoritmos estudiados en este capítulo. Se ha aplicado un test de Wilcoxon no paramétrico [40] entre los porcentajes de error de clasificación de todos los algoritmos para saber si la diferencia es o no significativa. En la tabla sólo se muestran los resultados del test entre ADFN y el resto de algoritmos, debido a que ADFN obtiene mejores resultados que ADF, ADFR y ADNL, por término medio, tanto en las bases de datos binarias como multiclase. Concretamente, las cifras en negrita indican que la diferencia entre ADFN y el algoritmo correspondiente no es significativa. El mejor resultado en cada base de datos aparece subrayado en la tabla.

Los resultados correspondientes a ADF se encuentran en la columna 8. El símbolo "-", que acompaña a las bases de datos *card1*, *digitos35*, *mushroom1*, *horse1* y *soybean1*, indica que no se pudo invertir la matriz de covarianza *intra-clase* $\mathbf{S}_w$. Una de las formas más sencillas de aliviar este problema consiste en regularizar la matriz mediante la suma de una matriz proporcional a la identidad como sigue

$$\mathbf{S}_w^R = \mathbf{S}_w + \mu\mathbf{I}, \tag{1.43}$$

donde $\mathbf{I}$ es la matriz identidad y μ es la constante de regularización, que ha sido fijada como $\mu = 10^{-3}$ para todos los experimentos. En la tabla se muestran los resultados para esta variante de ADF en la columna número 9 (ADFR). Gracias a esta pequeña modificación se pueden obtener resultados para todas las bases de datos, siendo en la mayoría de ellas iguales a

los obtenidos por ADF. En algunos casos ADFR empeora, como sucede en *diabetes1* y *glass1*. Sin embargo, mejora ligeramente en *satellite* y *thyroid2*.

A partir de ahora nos referiremos a ADF como la conjunción del análisis discriminante de Fisher (método de reducción dimensional) y la regla de clasificación por distancia a medias proyectadas explicada anteriormente, para poder referirnos a ADF como un algortimo de clasificación. Por lo tanto, los resultados obtenidos por ADF son, en general, mejores para problemas binarios que para problemas con un número de clases mayor que dos. Sin duda, la principal limitación de ADF es su naturaleza lineal. Cuando el problema exige combinaciones no lineales de los atributos, su funcionamiento queda claramente por debajo del óptimo. Para resolver adecuadamente estos problemas, es necesario recurrir a métodos no lineales de reducción dimensional como los estudiados en este capítulo. Como vemos en la tabla 1.4 y detallamos seguidamente, el análisis discriminante de Fisher con núcleos (ADNF) y el análisis discriminante no lineal (ADNL), mejoran los resultados obtenidos por ADF y RNAl en todas las bases de datos multiclase.

Para obtener los resultados de ADFN hemos utilizado la familia de núcleos parametrizables [16] que se muestran en la tabla 1.2 cuyos parámetros han sido elegidos mediante una búsqueda iterativa. Concretamente, el algoritmo selecciona la función de núcleo y valor de su parámetro correspondientes al mínimo error de validación. Los parámetros c de la función Gaussiana y s de la sigmoidal se exploran en el intervalo $[0, 20]$ en incrementos de 0.1. El parámetro θ, presente en las funciones de núcleo sigmoidal y polinómico, se ha dejado fijo e igual a 0 para ambas funciones de núcleo. Para el núcleo polinómico, el intervalo $[2, 20]$ se recorre en incrementos de 1. El límite inferior de este intervalo equivale al análisis discriminante cuadrático (ADC)[9][39]. El valor 1 equivale al análisis discriminante de Fisher (ADF).

FBR Gaussiana	$k(\mathbf{x}, \mathbf{y}) = \exp \frac{-\|\mathbf{x}-\mathbf{y}\|^2}{c}$
Polinómica	$(\mathbf{x}'\mathbf{y} + \theta)^d$
Sigmoidal	$\tanh(s\mathbf{x}'\mathbf{y} + \theta)$

Tabla 1.2: Lista de núcleos utilizados en ADFN
Se muestra una lista de las funciones de núcleo utilizadas en ADFN: FBR Gaussiana ($c \in \mathbb{R}$), polinómica ($d \in \mathbb{N}, \theta \in \mathbb{R}$) y sigmoidal ($s, \theta \in \mathbb{R}$).

Durante la exploración de los parámetros, los intervalos susceptibles de contener un mínimo local son nuevamente explorados con incrementos de 0.01. Si aparecen nuevos mínimos locales, los intervalos adyacentes se vuelven a explorar con un nivel máximo de precisión de 0.001. En la tabla 1.3 se recogen los resultados de la exploración para cada base de datos. La primera columna indica el nombre de la base de datos, la segunda es la función de núcleo utilizada y la tercera columna muestra el valor del parámetro de la función.

La función de núcleo Gaussiana es la que conduce a mejores resultados en general. Al igual que ocurre con ADF, ADFN se encuentra con problemas cuando la matriz de covarianza intra-clase es singular en el espacio proyectado. Sin embargo, en todas las bases de datos estudiadas, siempre se encuentra algún valor de los parámetros de núcleo que evitan este problema.

Base de datos	Función de núcleo	Parámetro
cancer1	Gaussiana	c = 0.324
card1	Gaussiana	c = 14.04
diabetes1	Polinómica	d = 2
digitos35	Polinómica	d = 2
mushroom1	Sigmoidal	s = 0.75
spam1	Gaussiana	c = 8.2
gene1	Gaussiana	c = 8.4
horse1	Sigmoidal	s = 2.1
iris	Gaussiana	c = 0.5
thyroid2	Gaussiana	c = 0.07
car	Gaussiana	c = 1.04
glass1	Polinómica	d = 5
satellite	Gaussiana	c = 28.8
soybean1	Polinómica	d = 2

Tabla 1.3: Valores de los parámetros utilizados por ADFN
Valores de los parámetros de la función de núcleo utilizada por ADFN y correspondientes al mínimo error de validación. La primera columna es el nombre de la base de datos, la segunda es la función de núcleo y la tercera muestra el valor óptimo del parámetro correspondiente.

Al ser ADFN un método no lineal, sus resultados mejoran de manera significativa los obtenidos mediante ADF y ADFR para los problemas con

una mayor componente no lineal (*car*, *thyroid2*). ADFN es el algoritmo que más veces obtiene el mejor resultado de todos, comparado con todos los algoritmos propuestos en la tabla 1.4. Obtiene el mejor resultado en 4 de las 6 bases de datos binarias y en 5 de las 8 bases de datos multiclase. A pesar de este éxito, al igual que otros métodos de clasificación basados en núcleos, su rendimiento depende críticamente de la elección del núcleo y de los parámetros del mismo. La búsqueda del núcleo óptimo para cada problema concreto es una tarea con un coste computacional muy elevado [10] [25] [22], que en algunos casos puede llegar a convertirse en un problema con la misma entidad que el problema original que intentamos solucionar.

La arquitectura de la red utilizada por ADNL en los experimentos consta de tres capas. La capa de entrada tiene un número de unidades igual al número de atributos del problema, la capa de salida tiene un número de unidades igual al número de clases menos uno, y la capa oculta tiene un número de unidades parametrizable. Se recorre dicho parámetro en incrementos de 1 en el intervalo definido entre el número de clases y el número de atributos, $[c, n]$, calculando el error mínimo de clasificación de validación. Para afinar más la búsqueda, se sigue incrementando unitariamente el número de unidades de la capa oculta hasta que el error de clasificación de validación empeora. En ese momento ADNL toma como mejor solución el valor anterior y calcula el error de clasificación de test correspondiente a dicha arquitectura, como se muestra en la columna ADNL de la tabla 1.4. El número final de unidades ocultas aparece entre paréntesis.

A pesar de que ADNL no obtiene el mejor resultado para ninguna base de datos, excepto para *mushroom1* que consigue un 0 % al igual que ADFR, RNAnl y ADNL, obtiene resultados muy buenos tanto en las bases de datos binarias como en los problemas multiclase. Sin embargo puede encontrarse con los mismos problemas que el algoritmo lineal clásico al obtener la proyección de Fisher para los pesos de la última capa. Es decir, si la matriz de covarianza intra-clase que se necesita invertir es singular, ADNL no podrá obtener ningún resultado para ese conjunto de datos. En dichos casos, se obtiene la pseudoinversa mediante la Descomposición en Valor Singular (SVD, Singular Value Decomposition) [14] o bien regularizando la matriz de covarianza intra-clase mediante (1.43). Si mediante dichos procedimientos tampoco puede finalizar el cálculo, ADNL vuelve a comenzar el algoritmo confiando en que una nueva inicialización aleatoria de los pesos de la red permita obtener un resultado válido.

1.5. Conclusiones

Los experimentos realizados en este primer capítulo permiten ver que ADF tiene problemas cuando la matriz de covarianza es singular. Esto ocurre con varias de las bases de datos seleccionadas en este capítulo (*car*, *digitos35*, *mushroom*, *horse* y *soybean*). Este problema se soluciona mediante la regularización de la matriz de covarianza y ADF regularizado no empeora los resultados del algoritmo original. Sin duda, la principal limitación de ADF es su naturaleza lineal. Cuando el problema exige combinaciones no lineales de los atributos, su funcionamiento queda claramente por debajo del óptimo. Para resolver adecuadamente estos problemas, es necesario recurrir a métodos no lineales de reducción dimensional como el Análisis Discriminante de Fisher con Núcleos y el Análisis Discriminante No Lineal. Ambos métodos alcanzan resultados similares a los de las redes neuronales no lineales, como cabía esperar.

Por término medio ADFN consigue los mejores resultados en las bases de datos binarias, aunque sorprenden los resultados tan competitivos obtenidos por los algoritmos lineales como ADF regularizado y la red neuronal artificial lineal. Esto es debido al sobreaprendizaje de los algoritmos no lineales como la red neuronal no lineal y ADNL, que obtienen porcentajes de entrenamiento y validación muy bajos, pero un error más elevado en test. ADFN, sin embargo, no consigue unos errores de validación tan bajos pero sin embargo consigue una generalización más acertada.

En las bases de datos multiclase el mejor algoritmo por término medio es la red neuronal no lineal, seguida muy de cerca por ADFN y ADNL. Los algoritmos lineales encuentran muchas dificultades para conseguir resultados competitivos. En las bases de datos con una componente no lineal más acentuada (*iris*, *thyroid2*, *car*, *satellite* y *soybean1*) los algoritmos lineales obtienen errores cercanos al doble de los obtenidos por los algoritmos no lineales. El caso más claro lo encontramos en el conjunto *car*, en el que los algoritmos no lineales son entre 3 y 7 veces mejores que los lineales. A pesar del éxito obtenido por los algoritmos no lineales, su coste computacional es muy elevado.

BD binarias	C	Ejem	Atr	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL
cancer1	2	699	9	34.5	**1.72**	(5) 1.44	2.30	2.30	1.72	(3) 1.75
card1	2	690	51	44.0	14.97	(5) 15.26	-	13.95	14.53	(25) 14.22
diabetes1	2	768	8	34.9	25.36	(5) 28.00	21.88	23.96	20.83	(7) 25.49
digitos35	2	681	225	49.3	2.67	(10) 2.61	-	2.27	1.14	(5) 3.41
mushroom1	2	8124	125	48.0	0.12	(5) **0.00**	-	**0.00**	0.00	(3) **0.00**
spam1	2	4601	57	39.4	7.20	(5) 7.49	10.42	10.42	8.42	(5) 7.65
			Promedio:	41.68	8.67	9.13	11.53	8.81	7.77	8.75

BD multiclase	C	Ejem	Atr	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL
gene1	3	3175	120	50.0	12.54	(10) 13.18	12.86	12.86	8.70	(5) 11.14
horse1	3	364	58	38.0	30.83	(5) 30.06	-	38.46	36.26	(7) 34.67
iris	3	150	4	66.7	5.33	(3) 2.00	3.33	3.33	0.00	(15) 1.10
thyroid2	3	7200	21	7.4	4.00	(5) 1.67	27.33	22.00	3.22	(7) 2.99
car	4	1728	6	30.0	18.79	(5) 5.81	23.12	23.12	3.18	(25) 5.03
glass1	6	214	9	64.5	31.70	(10) 29.15	39.62	33.96	35.85	(15) 33.30
satellite	6	6434	36	76.2	22.39	(10) 11.68	19.02	18.92	10.39	(9) 14.32
soybean1	19	683	82	86.5	16.41	(15) 7.29	-	10.59	7.06	(21) 10.00
			Promedio:	52.41	17.75	12.60	20.88	20.40	13.08	14.07

Tabla 1.4: Resultados para ADF, ADFR, ADFN y ADNL

Consultar la tabla 1.1 para la descripción de las 5 primeras columnas. Las cuatro últimas columnas muestran los porcentajes de error de clasificación de los algoritmos ADF, ADFR, ADFN y ADNL. El símbolo "-"de la columna ADF indica la singularidad de la matriz de covarianza. Los números entre paréntesis de las columnas RNAnl y ADNL son el número de unidades de la capa oculta. Los mejores resultados para cada base de datos aparecen subrayados para facilitar la lectura. Las cifras en negrita corresponden a las únicas diferencias no significativas de acuerdo con un test de Wilcoxon no paramétrico [40] entre ADFN y el resto de los algoritmos.

Capítulo 2

ADEL: Análisis Discriminante Evolutivo Lineal

"Todo es muy difícil antes de ser sencillo."
Thomas Fuller

Este capítulo está dedicado al Análisis Discriminante Evolutivo Lineal (ADEL). Este algoritmo es una extensión evolutiva del Análisis Discriminante de Fisher que prescinde de las matrices de covarianza del algoritmo original. La proyección obtenida por ADEL optimiza el número de ejemplos clasificados erróneamente mediante una estrategia evolutiva. La sección 2.3 introduce el nuevo algoritmo. La sección 2.4 está dedicada al estudio de la función de fitness. *En la sección 2.5, el algoritmo se aplica al problema wine del repositorio UCI, un problema de tres clases que, aunque puede ser tratado adecuadamente mediante ADF. ilustra la importancia de la normalización en el proceso evolutivo de ADEL. La sección 2.6 trata sobre la selección del número de proyecciones. Como caso de prueba, se estudia en detalle el problema de la soja del repositorio UCI. En la sección 2.7 probamos que, envolviendo la estrategia evolutiva en un algoritmo genético de selección de atributos, se pueden tratar problemas de mayor dificultad. Se utiliza como ejemplo el problema tiroides del repositorio UCI. En 2.9 se compara ADEL con los algoritmos estudiados en el primer capítulo y se extraen algunas conclusiones.*

2.1. Introducción

Una de las primeras tareas que debemos realizar cuando intentamos resolver un problema de reconocimiento de patrones es inspeccionar visualmente los ejemplos en una o dos dimensiones. Esta es una de las mejores formas de comprender la dificultad del problema de clasificación, es decir, el solapamiento entre clases. Dado que la mayoría de los problemas de interés práctico están escritos en espacios de alta dimensión, sus atributos tienen que ser seleccionados o combinados para reducir su número. Existen dos maneras de combinar atributos dependiendo de si el problema es supervisado o no. Cuando el problema es no supervisado, el análisis de componentes principales (ACP) [19] devuelve combinaciones de atributos ordenadas por la cantidad de información o de varianza que contienen. En situaciones supervisadas, que son nuestro principal interés, el análisis discriminante de Fisher (ADF) [7] sigue siendo uno de los algoritmos más utilizados en la práctica. Ambos algoritmos, ACP y ADF, hacen uso de medidas de información de las cuales proponemos deshacernos por razones que dentro de poco serán evidentes.

Es importante tener en cuenta que ADF es una técnica supervisada de reducción dimensional, no un algoritmo de clasificación. Es muy frecuente utilizarlo con fines de visualización y como pre-procesamiento antes de aplicar un algoritmo de clasificación. En este último caso, la regla de clasificación más sencilla consiste en asignar cada ejemplo a la clase cuya media esté más cercana en el espacio proyectado. Obviamente, esta regla no es necesariamente la óptima y, en general, es necesario recurrir a una red neuronal u otro algoritmo de clasificación para explotar al máximo la proyección de Fisher. Así, el proceso completo de clasificación acaba optimizando dos criterios diferentes: primero el criterio de Fisher y después el error de clasificación en el espacio proyectado. Lo que se propone en este capítulo es minimizar directamente el error de clasificación sin utilizar ningún criterio intermedio.

Se podría pensar que ese trabajo es el que realizan precisamente las redes neuronales [2], es decir, reducir directamente el error de clasificación. Sin embargo, el error minimizado por una red neuronal (y las técnicas de regresión relacionadas) es el error cuadrático de clasificación. Para calcular este error es necesario asignar códigos de salida para cada clase. Nuestro enfoque es más cercano a ADF que a la regresión dado que no necesita una codificación de salida. Sin embargo, a diferencia de lo que ocurre con ADF, la proyección es óptima con respecto a la regla de clasificación debido a que la función de

coste es precisamente el número de ejemplos mal clasificados al aplicar la regla.

Dado que el número de errores es una cantidad discreta no derivable, recurrimos a un método de optimización como las estrategias evolutivas [1] para minimizarlo. Antes de entrar en detalles veamos cómo podemos hacer que una estrategia evolutiva simple busque una proyección unidimensional discriminatoria en un conjunto de ejemplos etiquetados por la clase a la que pertenecen [42]:

- Se genera una proyección inicial aleatoria.
- El valor (fitness) de la proyección es el número de errores cometidos al asignar cada ejemplo de entrenamiento a la clase cuya media sea la más cercana en el espacio proyectado.
- Se repiten los siguientes pasos hasta que se alcanza un número predefinido de generaciones sin mejoras:
  - Se genera una nueva proyección añadiendo perturbaciones normales independientes a cada componente de la proyección actual.
  - Se calcula el número de ejemplos mal clasificados con la nueva proyección.
  - La nueva proyección pasa a ser la actual si se reduce el error.
- La proyección actual es la respuesta del algoritmo.

Como veremos en la siguiente sección, debemos tomar algunas precauciones para controlar el sobre-aprendizaje, problema muy común en aprendizaje automático. Aún así, nuestro algoritmo es muy simple y fácil de aplicar. Aunque consume más tiempo que su homólogo tradicional, tiene ventajas claras debido a que carece de medidas de separación entre clases.

Aunque la deficiencia de rango ha sido tratada en la literatura [5], nuestro enfoque no sólo se ocupa de ella sino que permite encontrar proyecciones bidimensionales en conjuntos de datos multiclase como el problema de la soja [3]. Esta libertad a la hora de elegir la dimensión del espacio proyectado es una cualidad difícil de encontrar en los algoritmos de clasificación. Normalmente se suelen utilizar códigos de salida o medidas de separación de clases, ambos

restringidos por el número de clases del problema. En la siguiente sección analizamos estas representaciones y buscamos un nuevo enfoque que solucione sus limitaciones.

2.2. ¿Medias proyectadas o códigos de salida fijos?

Consideremos un conjunto de ejemplos $\{(\mathbf{x}, t)\}$ en d dimensiones. Cada ejemplo $\mathbf{x}$ está etiquetado con la clase $\mathbf{t}(\mathbf{x})$ a la que pertenece. Nuestro objetivo es construir una función $\mathbf{y}(\mathbf{x})$ del espacio de entrada al espacio de clases que minimice el error de clasificación para ejemplos no vistos durante el entrenamiento. En particular, los perceptrones multicapa aprenden estas funciones mediante la minimización de errores cuadráticos [2] como

$$L = \sum_{\mathbf{x}} (\mathbf{y}(\mathbf{x}) - \mathbf{t}(\mathbf{x}))^2. \tag{2.1}$$

Es muy común utilizar una codificación 1-de-c para los códigos de salida pues no impone relaciones de orden espúreas:

$$\mathbf{t}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbf{t}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \ldots \mathbf{t}_c = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{2.2}$$

Veamos si podemos prescindir de estos códigos. A fin de simplificar las ecuaciones, consideremos una red neuronal sin capas ocultas y sin funciones de activación. En este caso, las salidas son las siguientes:

$$y_j = \sum_{i=0}^{d} x_i w_{ij}, \tag{2.3}$$

donde $x_0 = 1$ para todos los ejemplos. El gradiente de la función de error cuadrático para un ejemplo $\mathbf{x}$ es el siguiente:

$$\frac{\partial (\mathbf{y}(\mathbf{x}) - \mathbf{t}(\mathbf{x}))^2}{\partial w_{ij}} = 2(y_j - t_j)x_i. \tag{2.4}$$

Este gradiente conduce a la regla delta clásica o regla de retro-propagación cuando existen unidades ocultas.

Intentemos ahora sustituir los códigos de salida en (2.1) por las medias de las clases. Para empezar, remplazar $\mathbf{t}(\mathbf{x})$ por $\mathbf{m}(\mathbf{x})$ (la media de la clase de $\mathbf{x}$) en la ecuación 2.1 es peligroso cuando existen solapamientos no lineales entre las clases. Por ejemplo, en el problema XOR, podríamos terminar utilizando códigos idénticos para todas las clases. Incluso en problemas linealmente separables, si utilizáramos las medias como objetivos, la dimensión de salida resultaría ser igual a la dimensión de entrada, perdiendo la propiedad de reducción dimensional que buscamos. Esta es la razón por la que debemos proyectar las medias junto con sus ejemplos.

Al proyectar los ejemplos junto con las medias de las clases, la función de error se convierte en

$$L = \sum_{\mathbf{x}} (\mathbf{y}(\mathbf{x}) - \mathbf{y}(\mathbf{m}))^2. \tag{2.5}$$

El nuevo gradiente es proporcional a los pesos que intentamos aprender

$$\frac{\partial (\mathbf{y}(\mathbf{x}) - \mathbf{y}(\mathbf{m}))^2}{\partial w_{ij}} = 2w_{ij}(x_i - m_i)^2. \tag{2.6}$$

Por lo tanto, para evitar que los pesos tiendan a anularse, la función de coste en la ecuación 2.5 debe modificarse de alguna forma. Tal vez, la distancia entre las medias podría ayudar. Sin embargo, en vez de intentar mejorar esta función de coste, hemos tratado el problema con un nuevo enfoque. Hemos renunciado a las funciones de error cuadrático y utilizamos el número de ejemplos mal clasificados como nueva función de coste. Sólo existe un inconveniente: el descenso por gradiente no puede ser utilizado para minimizar este error debido a su naturaleza discreta. Debemos recurrir a un algoritmo como, por ejemplo, una estrategia evolutiva [1].

No obstante, el proceso seguido por la estrategia evolutiva es muy intuitivo. En su versión más simple puede ser descrita como sigue. Comenzamos con un modelo elegido aleatoriamente. A fin de calcular el fitness de este modelo, es decir, el número de ejemplos mal clasificados, debemos proyectar los ejemplos y las medias de las clases, y entonces clasificar los ejemplos como correspondientes a la clase cuya media proyectada esté más próxima. La estrategia evolutiva, después de mutar los pesos del modelo, los actualizará cuando el número de errores decrezca. Este mecanismo tan simple y la ausencia de errores cuadráticos nos permite elegir el número de proyecciones sin restricciones. Los códigos de salida son ahora las proyecciones de las medias de las clases que son aprendidas durante el proceso. La siguiente sección está

dedicada a explicar el funcionamiento detallado del algoritmo discriminante evolutivo.

2.3. ADEL: Análisis Discriminante Evolutivo Lineal

Sea $\{(\mathbf{x}, t)\}$ un conjunto de ejemplos en d dimensiones, cada uno etiquetado con su clase t, $(t = 1, \ldots, c)$. Estamos buscando proyecciones lineales en p dimensiones $(p < d)$,

$$\bar{x}_j = \sum_{i=1}^{d} x_i w_{ij}, \qquad j = 1, \ldots, p, \tag{2.7}$$

con el fin de clasificar de manera óptima (los ejemplos proyectados serán marcados con una barra).

La proyección propuesta por ADF es la que maximiza el criterio de separación de clases 1.14. Esencialmente, esta medida es la distancia entre las medias dividida por la dispersión de cada clase en el espacio proyectado. Por tanto, tiene sentido clasificar los ejemplos como pertenecientes a la clase con la media proyectada más cercana como sigue

$$\text{clase}(\mathbf{x}) = \min_k \left(\sum_{j=1}^{p} (\bar{x}_j - \bar{m}_j^k)^2\right), \tag{2.8}$$

donde

$$\bar{m}_j^k = \sum_{i=1}^{d} m_i^k w_{ij}, \qquad k = 1, \ldots, c, \tag{2.9}$$

son las componentes proyectadas de la media de la clase k.

Nuestro algoritmo se inspira en el análisis discriminante clásico. Es decir, los ejemplos son proyectados más cerca de su propia media que de las demás. Sin embargo, en vez de optimizar un estadístico de segundo orden de la separación entre clases como 1.14, se utiliza el error medio de clasificación

$$J(\mathbf{w}) = \frac{1}{N} \sum_{\mathbf{x}} L\left(t(\mathbf{x}), \min_k \left(\sum_{j=1}^{p} (\bar{x}_j - \bar{m}_j^k)^2\right)\right), \tag{2.10}$$

donde N es el número de ejemplos, y el valor devuelto por $L(t, z)$ es 0 cuando $t = z$, y 1 en caso contrario. Al fin y al cabo, este error, o su versión validada, es el objetivo último en cualquier algoritmo de clasificación.

Dado que el número de ejemplos mal clasificados no es una cantidad continua, necesitamos utilizar un algoritmo de optimización capaz de tratar cantidades discretas, como la estrategia evolutiva [1]. En la sección introductoria se ha descrito una versión simplificada del algoritmo. El algoritmo completo, denominado Análisis Discriminante Evolutivo Lineal, utiliza una estrategia evolutiva general $(\mu, \lambda|\rho)$-ES, y funciona como se describe a continuación:

1. Se elige la dimensión (p) del espacio proyectado que permanecerá fija durante la evolución. En la siguiente sección se estudiará cómo elegir este parámetro apropiadamente.

2. Se normalizan los ejemplos, es decir, sus atributos son objeto de la siguiente transformación

$$x_i \longrightarrow \frac{x_i - m_i}{\sigma_i}, \qquad i = 1, \ldots, d, \tag{2.11}$$

   donde m_i y σ_i son la media y la desviación estándar, respectivamente, del atributo número i.

3. Se elige el tamaño de la población de progenitores (μ), el tamaño de la población de descendientes (λ) y el tamaño de la familia (ρ). La familia de un descendiente es el subconjunto de los progenitores que, después de la recombinación, dan lugar al descendiente. La notación $(\mu, \lambda|\rho)$ especifica el tamaño de las poblaciones, incluida la familia, y el esquema de remplazamiento (el remplazamiento *coma* selecciona las mejores μ proyecciones, de entre los λ descendientes, como la nueva población de progenitores).

4. Se elige el paso de mutación (σ) que permanecerá fijo durante el proceso evolutivo.

5. Se eligen los pesos iniciales aleatoriamente en el intervalo $[-0{,}5, 0{,}5]$ y se repiten los siguientes pasos hasta que se ejecute un número prescrito de generaciones sin mejoras:

   - Se genera un grupo de λ pesos (w_{ij}^l con $l = 1, \ldots, \lambda$) mediante la recombinación discreta de los pesos de ρ progenitores. La recombinación discreta funciona de la siguiente manera: para cada descendiente se eligen aleatoriamente ρ individuos de la población

de progenitores. Seguidamente, cada componente (alelo) del descendiente se iguala al alelo correspondiente de uno de los ρ progenitores, elegidos aleatoriamente del subconjunto que forma la familia. Este esquema de recombinación no introduce nuevos alelos en la población.

- Cada individuo es mutado añadiendo desviaciones aleatorias independientes σ_{ij} a cada componente. Estas desviaciones son extraídas de una distribución normal $N(0, \sigma)$. Por ejemplo, el peso número l es mutado de la siguiente forma:

$$w_{ij}^{l} \rightarrow w_{ij}^{l} + \sigma_{ij}. \tag{2.12}$$

- Se crea una nueva población de progenitores con los μ mejores individuos de entre los λ descendientes.

Esta es una estrategia evolutiva clásica cuya principal característica es su enfoque tradicional respecto al paso de mutación, que es el mismo para todos los atributos y no se adapta [1]. Como veremos después, esta configuración tan simple puede tratar problemas de reducción dimensional siempre que los ejemplos estén normalizados. Sin embargo, nuestro algoritmo básico puede beneficiarse de algoritmos evolutivos más potentes como CMA-ES [15], IEA [17], StGA [49], o aquellos en los cuales las mutaciones Gaussianas se ven superadas por mutaciones de Lévy al tratar funciones con muchos óptimos locales [23].

2.4. Fitness

Anteriormente hemos explicado que el fitness es el error cometido al asignar cada ejemplo a la clase con la media más cercana en el espacio proyectado. En la práctica, este error debe ser estimado dado que muy pocas veces se dispone de la población de ejemplos completa. En la literatura de aprendizaje automático normalmente se utiliza validación cruzada o *bootstrap* como técnicas de estimación. Para simplificar los cálculos hemos decidido romper el conjunto de ejemplos en tres subconjuntos (conjunto de entrenamiento, validación y test) y hemos utilizado los conjuntos de entrenamiento y validación en el proceso evolutivo como explicamos a continuación.

Para calcular el fitness de una proyección, primero se calculan las medias proyectadas utilizando los ejemplos de entrenamiento exclusivamente. Los

ejemplos de validación y los de test no formarán parte de este cálculo. A continuación, se clasifican los conjuntos de entrenamiento y validación por distancia a las medias proyectadas. La contribución del conjunto de validación al error tenderá a controlar el sobre-aprendizaje. Finalmente, dado que los ejemplos de test no han sido utilizados en ningún momento durante el proceso evolutivo, el error cometido por la proyección sobre el conjunto de test será una estimación de la capacidad de generalización de dicha proyección.

El número de errores de entrenamiento y validación pueden ser combinados de diferentes formas. Hemos realizado un experimento preliminar que muestra que el proceso evolutivo no se ve afectado por una combinación concreta siempre y cuando ambos conjuntos sean tenidos en cuenta. La tabla 2.1 muestra los resultados de este experimento que encuentra proyecciones bidimensionales para el conjunto *wine* [3] compuesto por tres clases. Este problema consiste en encontrar cuál de los tres diferentes cultivos produce cada uno de los 178 vinos de una región de Italia. Tenemos a nuestra disposición 13 atributos como la intensidad del color o el matiz, y las concentraciones de alcohol, ácido málico o magnesio. El conjunto original ha sido partido aleatoriamente en conjunto de entrenamiento (95 vinos), validación (42 vinos) y test (41 vinos).

Cada fila de la tabla 2.1 muestra los resultados promediados sobre 30 ejecuciones de ADEL para un valor específico de α en la función de fitness:

$$J_{ADEL}(\mathbf{w}) = (1-\alpha)J_{en}(\mathbf{w}) + \alpha J_{va}(\mathbf{w}), \tag{2.13}$$

donde $J_{en}(\mathbf{w})$ y $J_{va}(\mathbf{w})$ vienen dados por la ecuación

$$J_S(\mathbf{w}) = \frac{1}{N}\sum_{\mathbf{x}} L(t(\mathbf{x}), \arg\min_r(\sum_{k=1}^{n}(\bar{x}_k - \bar{m}_k^r)^2)), \tag{2.14}$$

donde N es el número de ejemplos y S es el conjunto de entrenamiento (tr) o validación (va) sobre el que se calcula $J(\mathbf{w})$ definida anteriormente mediante la ecuación 2.10. El valor devuelto por $L(t,t')$ es 0 cuando $t = t'$, y 1 en caso contrario. Los valores que se muestran son los porcentajes de error medios para los conjuntos de entrenamiento, validación y test, junto con el número medio de generaciones utilizadas para encontrar la mejor proyección y el fitness medio de la mejor proyección (última columna). El resultado sólo empeora cuando no se tienen en cuenta el conjunto de entrenamiento ($\alpha = 1$) o el conjunto de validación ($\alpha = 0$). Por tanto, hemos decidido

α	Entrenamiento %	Validación %	Test %	# Gen.	Fitness
0	0.00	9.17	3.89	927	9.17
0.1	0.00	0.00	2.78	1402	0.00
0.2	0.00	0.00	2.92	1284	0.00
0.3	0.00	0.00	2.92	1222	0.00
0.4	0.00	0.00	3.19	1293	0.00
0.5	0.00	0.00	2.92	1404	0.00
0.6	0.00	0.00	2.92	1753	0.00
0.7	0.00	0.00	2.78	1660	0.00
0.8	0.00	0.00	2.92	1757	0.00
0.9	0.00	0.00	3.06	1683	0.00
1.0	10.75	0.00	10.69	584	10.75

Tabla 2.1: Diseño de la función de fitness
Resultados promediados de 30 ejecuciones de ADEL sobre el conjunto de datos *wine* para 11 definiciones diferentes de la función de fitness (11 valores distintos de α en $J_{ADEL}(\mathbf{w}) = (1-\alpha)J_{en}(\mathbf{w}) + \alpha J_{va}(\mathbf{w})$). Las columnas 2, 3 y 4 muestran el porcentaje de ejemplos mal clasificados para cada conjunto. La ejecución se ve deteriorada sólo cuando el conjunto de validación ($\alpha = 0$) o el de entrenamiento ($\alpha = 1$) no son tenidos en cuenta. Nótese que en esos casos necesita menos generaciones (columna 5) para encontrar la mejor solución, dado que ADEL está resolviendo un problema más simple.

utilizar $\alpha = 0{,}5$. Llegados a este punto podemos describir cómo se asigna el fitness a cada proyección ($\mathbf{w} = (w_{ij})$):

- El conjunto de ejemplos se divide en tres subconjuntos: entrenamiento, validación y test. Se calculan las medias de cada clase (m_i^k) exclusivamente mediante los ejemplos de entrenamiento. Ambos conjuntos, entrenamiento y validación, son clasificados por distancia a dichas medias. El conjunto de test sólo se utiliza para valorar la capacidad de generalización de las proyecciones finales.

- Se proyectan las medias de entrenamiento (m_j^k) con los pesos w_{ij} como sigue:

$$\bar{m}_j^k = \sum_{i=1}^{d} m_i^k w_{ij}, \qquad j = 1, \ldots, p. \tag{2.15}$$

- Se proyectan los ejemplos ($\mathbf{x}$) de los conjuntos de entrenamiento y validación como sigue:

$$\bar{x}_j = \sum_{i=1}^{d} x_i w_{ij}, \qquad j = 1, \ldots, p. \tag{2.16}$$

- Se asignan los ejemplos a la clase cuya media proyectada esté más cerca:

$$\text{clase}(\mathbf{x}) = \min_k (\sum_{j=1}^{p} (\bar{x}_j - \bar{m}_j^k)^2). \tag{2.17}$$

- El valor de fitness $J_{ADEL}(\mathbf{w})$ asignado a $\mathbf{w} = (w_{ij})$ es el porcentaje de ejemplos mal clasificados para dicha proyección. De hecho, el error en el conjunto de entrenamiento $J_{tr}(\mathbf{w})$ y el error en el conjunto de validación $J_{va}(\mathbf{w})$ se unen en una única cifra $J_{ADEL}(\mathbf{w}) = J_{tr}(\mathbf{w}) + J_{va}(\mathbf{w})$, que será utilizada como función de fitness por ADEL.

2.5. Normalización

Continuamos nuestro razonamiento por medio de problemas cada vez más exigentes, comenzando por el problema de los viticultores italianos (*wine*) [3], introducido previamente. Este conjunto de datos, a pesar de no representar un desafío importante dado que es casi linealmente separable, nos ayudará a entender bajo qué circunstancias nuestro algoritmo puede obtener mejores resultados que el algoritmo clásico (ADF).

Analizaremos primero los resultados obtenidos por el enfoque tradicional. Dado que hay tres clases en este problema, ADF encuentra dos combinaciones lineales de los 13 atributos originales. La proyección está formada por los dos autovectores no nulos de $S_w^{-1} S_b$. El resultado se muestra en la figura 2.1 (panel de la izquierda). Cuando los vinos se asignan al viticultor cuya media es la más cercana en el espacio proyectado, se consigue un error igual a 2,44 % en el conjunto de test. El error se puede reducir posteriormente a 0 % si los ejemplos están normalizados antes de aplicar el algoritmo de Fisher.

Comparemos esta proyección con la encontrada por ADE. Los resultados se muestran en la tabla 2.2. Recordemos que el número de proyecciones debe ser elegido antes de ejecutar la estrategia evolutiva. Para proyecciones de una dimensión el mejor fitness medio de las 30 ejecuciones de una estrategia

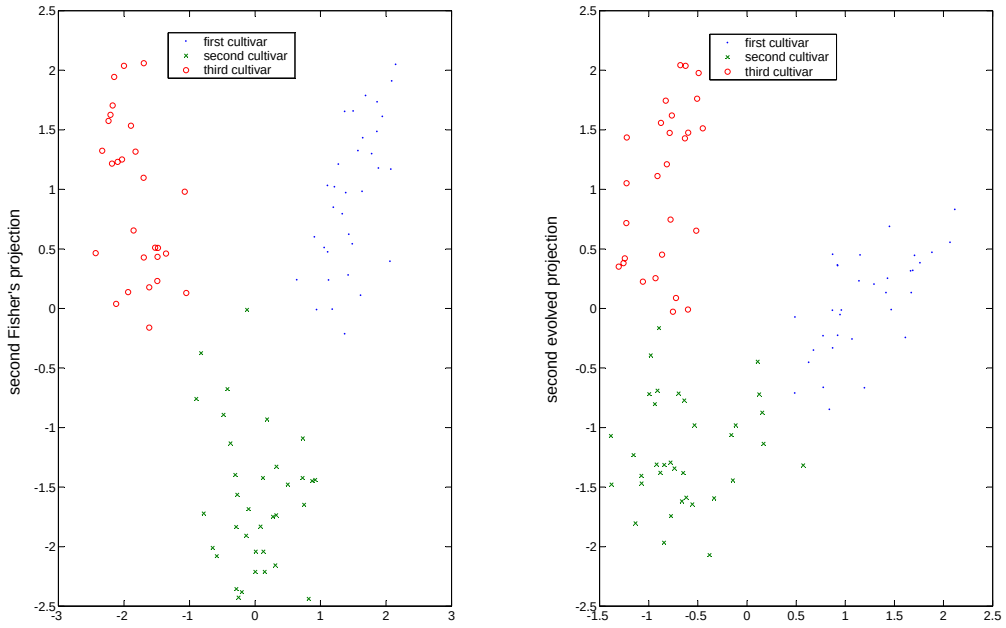

Figura 2.1: El problema *wine* proyectado mediante ADF y ADEL
Conjunto de datos de entrenamiento proyectado por ADF (izquierda) y ADEL (derecha). Se muestran dos combinaciones lineales de los 13 atributos originales del conjunto de entrenamiento, para las tres clases de viticultores.

# proy.	Estrategia	% En.	% Val.	% Test	% Desv.	Mejor Fit.	Eval.
1	$(1+1)$	5.75	2.30	10.98	2.87	8.06	416
1	$(15, 100\|2)$	1.86	0.00	7.24	2.16	1.86	7925
2	$(1+1)$	0.00	0.00	2.11	2.23	0.00	313
2	$(15, 100\|2)$	0.00	0.00	1.30	1.63	0.00	1207

Tabla 2.2: Resultados de ADEL para el problema *wine*
Valores promediados de 30 ejecuciones de ADEL aplicado a la búsqueda de proyecciones de una y dos dimensiones para el problema de los viticultores (*wine*). El error medio de entrenamiento, validación y test de las mejores proyecciones de cada ejecución se muestran en las columnas 3, 4 y 5, respectivamente, junto con la desviación estándar del error de test (columna 6). El mejor fitness medio y el número medio de evaluaciones para la mejor solución se muestran en las columnas 7 y 8, respectivamente.

evolutiva $(1+1)$-ES es 8.06. El mejor fitness medio de las 30 ejecuciones de una estrategia evolutiva $(15, 100|2)$-ES es 1.86. Con respecto al porcentaje

de error en test, el promedio para la primera estrategia es 10,98 % mientras que para la segunda es 7,24 %. Parece obvio que necesitamos incrementar la dimensión del espacio de proyección. Como era de esperar, las proyecciones en dos dimensiones consiguen mejorar mucho el fitness. De hecho, el mejor fitness medio por validación cruzada decrece hasta 0 %. El error medio en test de 30 ejecuciones de una estrategia evolutiva $(1+1)$-ES es 2,11 % y el error medio en test para una estrategia evolutiva $(15, 100|2)$-ES es 1,30 %.

La figura 2.1 (panel derecho) muestra las proyecciones encontradas evolutivamente por una estrategia $(1+1)$-ES, de los ejemplos del conjunto de entrenamiento. El resultado evolutivo es muy similar al obtenido por la proyección clásica (panel izquierdo). ADEL deja de mejorar la solución al alcanzar el mínimo de la función de coste, lo que ocurre rápidamente en este ejemplo debido a su naturaleza lineal. Por tanto, en general, no se aconseja la minimización directa del error de clasificación para problemas sencillos linealmente separables, por lo menos con nuestro método sencillo de validación cruzada. Es muy probable que una técnica más robusta de validación cruzada de 10 hojas mejore nuestros resultados.

La normalización de los ejemplos es un factor decisivo en la convergencia para problemas con más de dos clases. De la figura 2.2 podemos deducir que es mucho más fácil encontrar el mínimo de la función de error cuando los atributos están normalizados. Cuando utilizamos los atributos originales el fitness se mantiene por encima de 5 unidades incluso después de 10^5 generaciones. Esto no nos sorprende dado que, como comentamos anteriormente, los pesos son inicializados aleatoriamente en el intervalo $[-0{,}5, 0{,}5]$. En general, ésta es una mejor elección cuando los atributos están normalizados que cuando no lo están. Utilizando diferentes mecanismos de inicialización, adaptados de alguna manera a cada atributo, la evolución se desarrollaría de otra manera.

2.6. Selección del número óptimo de proyecciones

Uno de los propósitos de este trabajo es desarrollar un procedimiento capaz de visualizar en dos dimensiones conjuntos de datos con un número arbitrario de dimensiones y clases. Sin embargo, encontrar la mejor proyección es también un objetivo clave, que obviamente incluye buscar el número óptimo de dimensiones proyectadas. Aunque es tentadora la posibilidad de evolucionar el número de proyecciones y las propias proyecciones al mismo

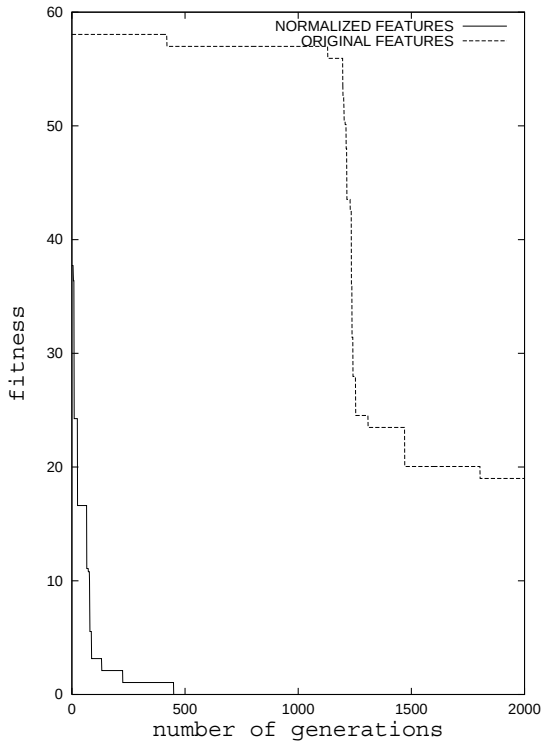

Figura 2.2: Importancia de la normalización
Fitness frente al número de generaciones para dos ejecuciones (atributos originales y normalizados) de ADEL aplicado a la discriminación de vinos en un espacio proyectado bidimensional. Para buscar las proyecciones se utiliza una estrategia simple $(1 + 1)$-ES. La normalización de los ejemplos afecta drásticamente a la convergencia.

tiempo, hemos optado por seguir un procedimiento paso a paso. Este procedimiento consiste en ejecutar nuestro algoritmo básico para un número creciente de dimensiones:

1. Se inicializa el número de proyecciones ($p = 1$). Se elige el número máximo de proyecciones (p_{max}). Se elige el número N de ejecuciones para promediar los resultados, típicamente $N = 20$.

2. Se realizan N ejecuciones de ADEL para el número actual de proyecciones (p). Se estima el mejor fitness para este número de proyecciones

como el siguiente promedio:

$$\bar{\phi}_p = \frac{1}{N} \sum_{i=1}^{N} \phi_p^i, \tag{2.18}$$

donde ϕ_p^i es el mejor fitness para la ejecución número i.

3. Se incrementa en una unidad el número de proyecciones.

4. Se ejecuta nuevamente el punto 2 hasta que se alcanza el número máximo de proyecciones (p_{max}).

5. El mejor número de proyecciones ($\hat{p}$) es la dimensión con el mínimo fitness medio:

$$\hat{p} = \min_p \bar{\phi}_p. \tag{2.19}$$

Vamos a ilustrar este procedimiento con la base de datos sobre enfermedades de la soja del repositorio UCI [3]. Aunque la soja viene caracterizada por 35 atributos continuos en el respositorio, haremos uso de la codificación de Prechelt [33] mediante 82 atributos reales, incluyendo códigos especiales para tratar la gran cantidad de atributos desconocidos.

El problema de la soja es un problema muy difícil por varias razones. Primero, la probabilidad a priori de las distintas clases es muy irregular como podemos ver en la tabla 2.3. Por ejemplo, hay hasta 12 veces más ejemplos de *brown-spot* que de *herbicide-injury*. Segundo, la codificación tradicional de salida 1-de-19 supondría 19×82 parámetros ajustables que obviamente representan demasiados coeficientes libres.

Ambos problemas se pueden resolver fácilmente con nuestro enfoque evolutivo. Primero, al igual que en el análisis discriminante clásico, hacemos uso de las medias de las clases, que son menos sensibles a probabilidades a priori irregulares que los códigos de salida. Sin embargo, ADF no puede ser aplicado a este problema dado que la matriz de covarianza intra-clases, S_w, es singular, incluso después de eliminar los atributos (atributos 66 y 82) con varianza cero. Además, la minimización directa del error de clasificación permite elegir la dimensión de salida a voluntad, y podremos construir incluso proyecciones unidimensionales para este problema de 19 clases. Esta es una clara ventaja sobre las redes neuronales que, por ejemplo, dependen de códigos de salida.

Enfermedad de la soja	# ejemplos	Prob. a priori
herbicide-injury	8	0.01
cyst-nematode	14	0.02
diaporthe-pod-and-stem-blight	15	0.02
2-4-d-injury	16	0.02
bacterial-blight	20	0.03
bacterial-pustule	20	0.03
charcoal-rot	20	0.03
diaporthe-stem-canker	20	0.03
downy-mildew	20	0.03
phyllosticta-leaf-spot	20	0.03
powdery-mildew	20	0.03
purple-seed-stain	20	0.03
rhizoctonia-root-rot	20	0.03
anthracnose	44	0.06
brown-stem-rot	44	0.06
phytophthora-rot	88	0.13
alternarialeaf-spot	91	0.13
frog-eye-leaf-spot	91	0.13
brown-spot	92	0.13

Tabla 2.3: Probabilidad a priori de las clases en el conjunto *soybean1*

La primera columna muestra las 19 enfermedades de la soja recogidas en esta base de datos. Las columnas 2 y 3 muestran el número de ejemplos de cada clase y la probabilidad a priori, respectivamente.

Vemos así que nuestro algoritmo es una alternativa viable a otros algoritmos de proyección tradicionales para problemas multiclase.

La estrategia utilizada ha sido una estrategia evolutiva $(15, 100|15)$-ES. Se utiliza el remplazamiento *coma* dado que generalmente es recomendado para problemas de optimización con valores reales. Se utiliza un tamaño de paso de mutación igual a 1.0 para todos los coeficientes. Esta decisión está íntimamente ligada a la normalización del conjunto de ejemplos, como se ha discutido en la sección anterior. Se ha elegido la recombinación discreta para mantener la variabilidad dentro de la población. La condición de terminación consiste en detener la evolución después de 1000 generaciones sin mejora. La

tabla 2.4 muestra los resultados de 10 ejecuciones para proyecciones bidimensionales. Las columnas 2, 3 y 4 muestran los errores de entrenamiento, validación y test, respectivamente. La columna 5 muestra el número de evaluaciones del fitness hasta encontrar la mejor solución. La cifra de la columna 6 es el fitness de la mejor solución. Los valores medios de todas estas cantidades se encuentran en la última fila de la tabla.

Ejecución	% Entr.	% Val	% Test	Eval. (x 10^3)	Fitness
0	3.51	2.92	10.59	65	6.43
1	6.14	6.43	10.59	215	12.57
2	3.51	3.51	12.35	60	7.02
3	2.63	1.17	12.35	213	4.38
4	2.92	3.51	11.76	36	7.02
5	2.92	3.51	11.18	92	6.73
6	5.56	7.02	12.35	27	12.57
7	7.02	8.19	14.12	82	15.20
8	6.43	7.02	12.35	66	13.45
9	2.92	2.92	10.00	213	5.85
Media	4.36	4.62	11.76	107	9.12

Tabla 2.4: Clasificación de *soybean1* mediante proyecciones bidimensionales de ADEL

Diez ejecuciones de ADEL aplicadas a la búsqueda de proyecciones bidimensionales en el problema *soybean1*. Se ha utilizado una estrategia evolutiva $(15, 100|15)$-ES. Los porcentajes de error en entrenamiento, validación y test de las mejores proyecciones de cada ejecución se muestran en las columnas 2, 3 y 4, respectivamente. El número de evaluaciones necesario para obtener la mejor proyección y el mejor fitness se muestran en la quinta y sexta columnas, respectivamente.

El error medio para el conjunto de test de las 10 ejecuciones es 11,8 %. Este porcentaje no está muy lejos de los porcentajes medios obtenidos con redes neuronales lineales (9,5 %) [33]. Sin embargo, estamos haciendo uso únicamente de dos dimensiones y por lo tanto podremos visualizar cómo se relacionan las clases proyectadas con cada una de las demás. La figura 2.3 muestra una de las visualizaciones bidimensionales del conjunto de entrenamiento *soybean1*. Dado que no tenemos códigos de salida, la proximidad entre los grupos nos da mucha información sobre la naturaleza de las en-

fermedades de la soja. Hay clases perfectamente separadas de las demás y también aparecen clases solapadas. Aunque esto se podría ver igualmente en la matriz de clasificación, la imagen añade información incluso cuando la matriz de clasificación es diagonal, al mostrar distancias relativas entre las diferentes clases.

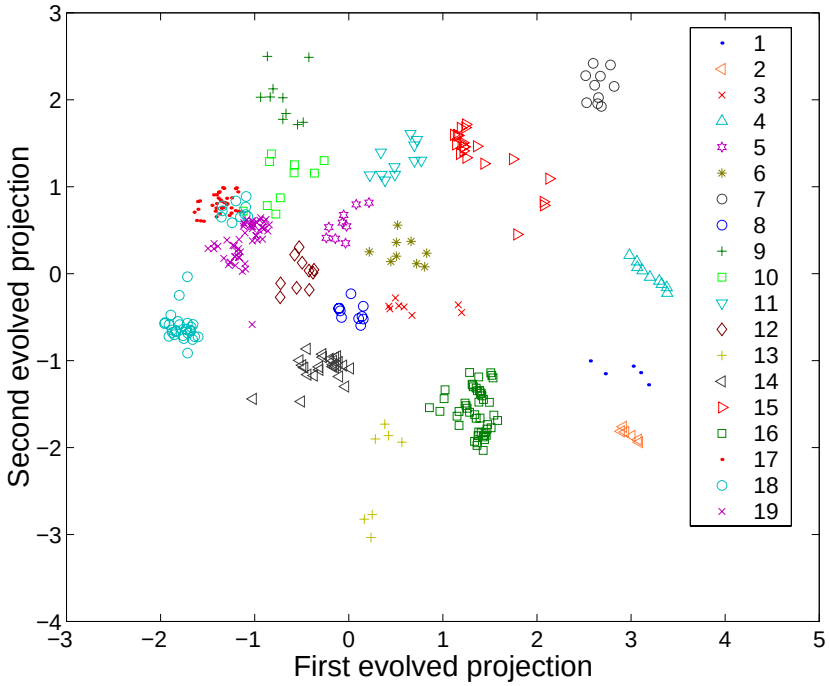

Figura 2.3: Proyección bidimensional del conjunto de entrenamiento *soybean1*
Proyección en dos dimensiones del problema *soybean1* (únicamente el conjunto de entrenamiento) encontrado por ADEL utilizando una estrategia evolutiva $(15, 100|15)$-ES. Uno de los puntos clave de nuestro enfoque es que permite visualizar problemas multiclase como este (19 clases) en dos dimensiones.

Antes de cerrar esta sección debemos contestar la siguiente pregunta: ¿cuál es la dimensión de la mejor proyección? En este sentido, la figura 2.4 muestra el promedio del mejor fitness frente al número de proyecciones para el problema soybean1. Cada punto es la media de 20 ejecuciones de una estrategia evolutiva $(15, 100|15)$-ES. La segunda curva de la figura representa el promedio del error para el conjunto de test y es, por lo tanto, una medida de la capacidad de generalización de cada dimensión. El primer mínimo local se obtiene con una proyección de 5 dimensiones tras la cual se produce un

pequeño incremento al pasar a 6 dimensiones. Sin embargo, el espacio óptimo de proyección corresponde a 12 dimensiones, que a su vez da lugar al mejor resultado de generalización (7,97 %).

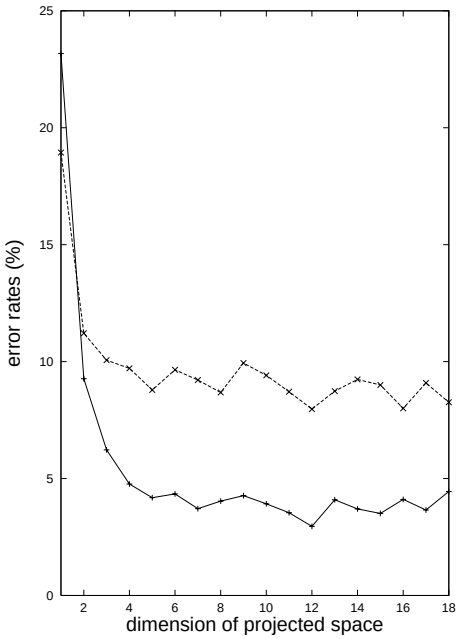

Figura 2.4: Dimensión óptima de la mejor proyección de ADEL en soybean1
El mejor fitness (línea continua) y el error de clasificación sobre el conjunto de test (línea discontinua) frente al número de proyecciones para el problema soybean1. Los valores están promediados sobre 20 ejecuciones de ADEL con una estrategia evolutiva $(15, 100|15)$-ES. El número óptimo de dimensiones es aquel que se corresponde con el menor fitness (12 dimensiones). El error de test permite apreciar la capacidad de generalización de cada proyección.

Otro ejemplo para ilustrar este procedimiento es la base de datos *satellite* del respositorio UCI [3]. Comencemos estudiando cómo trata ADF este problema con 6 clases. ADF encuentra una proyección de 5 dimensiones con un

19,02 % de error en el conjunto test. La matriz de confusión es la siguiente:

$$C_{sat} = \begin{pmatrix} 194 & 0 & 5 & 0 & 30 & 0 \\ 1 & 88 & 0 & 7 & 9 & 0 \\ 0 & 0 & 178 & 24 & 0 & 1 \\ 0 & 0 & 8 & 73 & 0 & 12 \\ 5 & 0 & 1 & 8 & 84 & 8 \\ 0 & 0 & 5 & 56 & 3 & 162 \end{pmatrix} \tag{2.20}$$

El valor C_{ij} de la matriz es el número de ejemplos de la clase i clasificados como pertenecientes a la clase j. Se ve claramente que las clases 3, 4 y 6, correspondientes a diferentes tipos de tierra gris (*grey soil*), se confunden mucho entre sí. Las clases 1 y 5 se confunden también entre sí. La clase número 2 (*cotton crop*) es la más limpia de todas.

Analicemos ahora el rendimiento de las proyecciones lineales encontradas por ADEL. En la tabla 2.5 se muestra el fitness y los porcentajes de error de los mejores individuos encontrados en 20 ejecuciones de ADEL con dos dimensiones de salida. El error de test medio de esas 20 ejecuciones es 16,67 %, más de un 10 % de mejora sobre ADF, incluso cuando ambos algoritmos son proyecciones lineales y clasifican por distancia a medias. Esta diferencia es debida al hecho de que estamos optimizando la regla de clasificación directamente. Por el contrario, ADF primero maximiza el criterio de separación entre clases y sólo después, clasifica los ejemplos por su distancia a las medias. La proyección, como ocurre en este caso, puede terminar en un subóptimo con respecto a la clasificación por distancia a medias.

A fin de comprobar la dependencia del rendimiento de nuestro algoritmo con el número de unidades de salida hemos ejecutado ADEL con un número creciente de dimensiones de salida. El resultado se puede observar en la figura 2.5. Cada punto de la figura es el resultado de promediar 20 ejecuciones de ADEL y muestra el fitness medio y el error de test frente al número de dimensiones de salida. El número de dimensiones varía desde 1 hasta 8. Debido a la naturaleza lineal de las proyecciones, hay una fuerte correlación entre el fitness y la capacidad de generalización, es decir, el porcentaje de error de test. Tres dimensiones de salida producen un error de test del 15 %, que supone una mejora del 20 % respecto de ADF.

Modelo	Entr.	Valid.	Test	Eval. (x10^3)	Fitness
1	15.08	11.43	15.90	59.64	26.51
2	15.01	11.75	17.15	32.18	26.76
3	15.43	11.85	16.42	18.02	27.28
4	15.21	10.71	16.53	64.28	25.92
5	15.37	11.43	17.05	25.74	26.80
6	15.41	11.23	17.05	52.51	26.64
7	15.06	11.12	16.11	52.65	26.18
8	15.10	11.43	16.94	47.58	26.53
9	15.06	11.43	16.94	40.57	26.49
10	15.34	11.12	16.94	41.69	26.47
11	15.59	11.12	16.74	35.28	26.71
12	15.28	11.43	16.22	39.39	26.71
13	15.34	11.85	16.01	21.97	27.19
14	15.30	11.12	17.67	59.43	26.42
15	15.43	11.54	17.46	22.37	26.97
16	15.37	11.64	16.53	41.93	27.01
17	15.37	11.54	16.11	47.68	26.90
18	15.28	11.43	16.63	42.05	26.71
19	15.48	10.71	16.53	47.02	26.18
20	15.14	11.54	16.42	29.00	26.68
Mean	15.28	11.37	16.67	41.05	26.65
Dev	0.16	0.31	0.47	12.94	0.33

Tabla 2.5: 20 ejecuciones de ADEL para el problema *satellite*
20 ejecuciones de ADEL y dos dimensiones de salida para la base de datos *satellite*. Se muestran los errores de entrenamiento, validación y test para el mejor individuo de cada ejecución. En las columnas 5 y 6, se muestran el número de evaluaciones hasta encontrar este mejor individuo y su fitness, respectivamente.

2.7. ADELM: Análisis discriminante evolutivo a varios niveles

En esta sección analizaremos el problema tiroides (*thyroid*) del repositorio UCI [3]. En este caso, aunque ADF puede ser aplicado dado que S_w no es

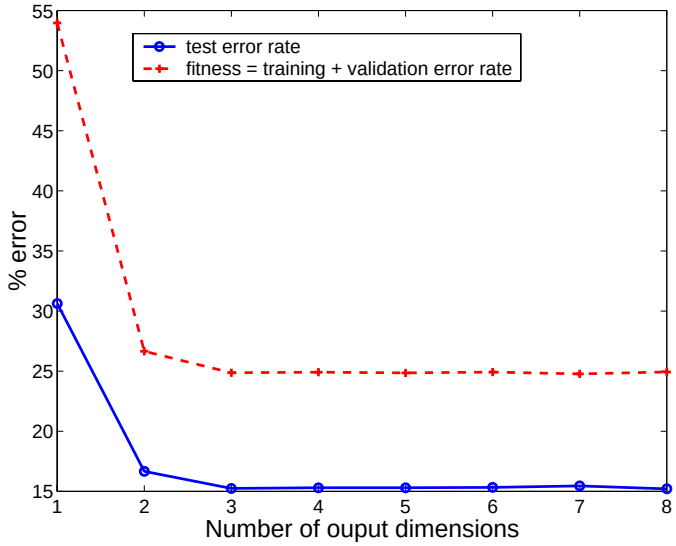

Figura 2.5: ADEL y *satellite*, con dimensión de salida creciente
Fitnes medio y error de test para 20 ejecuciones de ADEL y el conjunto *satellite* con un número creciente de unidades de salida. Tres unidades de salida producen un error de test del 15 %.

singular, produce resultados muy pobres. El problema consiste en determinar si un paciente ingresado en una clínica sufre hipotiroidismo. El conjunto contiene 7200 pacientes de 21 atributos cada uno, 15 binarios (de x_2 a x_{16}) y 6 continuos (x_1, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}). Utilizaremos la permutacióon thyroid2 proporcionada por Prechelt [33]. Los atributos son los siguientes: edad, sexo, paciente tratado con tiroxina, posible tratamiento con tiroxina, bajo medicación anti-tiroidea, enfermo (el paciente informa de malestar), embarazada, cirugía en la glándula tiroides, tratamiento I131, test de hipotiroidismo, test de hipertiroidismo, tratamiento con litio, bocio, tumor, hipopituitaria, síntomas psicológicos, TSH (thyroid-stimulating hormone), T3 (*triiodothyronine*), TT4 (*total thyroxine*), T4U *(thyroxine resin uptake*) y FTI (*free thyroxine index*). Las probabilidades a priori de las tres clases son muy irregulares. Dado que el 92 % de los pacientes pertenece a la misma clase, cualquier clasificador razonable debería mejorar el 8 % de error de clasificación sobre el conjunto de test.

La principal lección de esta sección es que una estrategia evolutiva simple $(1+1)$-ES, quizá el algoritmo evolutivo más sencillo, puede tratar problemas como *thyroid* siempre que se aplique sobre el conjunto de atributos adecua-

do. La figura 2.6 muestra el fitness frente al número de generaciones para una estrategia evolutiva $(1 + 1)$-ES aplicada a la búsqueda de proyecciones unidimensionales para el problema thyroid2 y dos conjuntos de atributos: el conjunto completo de los 21 atributos del problema y el subconjunto (x_3, x_8, x_{17}) que utiliza los siguientes atributos: paciente tratado con tiroxina, cirugía en la glándula tiroides y TSH. Hay cuatro curvas debido a que la estrategia se ejecuta con ambos conjuntos de atributos para sus valores originales y sus correspondientes valores normalizados. Resulta evidente que eliminando ciertos atributos el aprendizaje se acelera significativamente. La normalización mejora ligeramente la situación pero definitivamente el ingrediente clave es la selección de atributos. En la figura 2.6 se aprecia que la estrategia aplicada a los tres atributos sin normalizar es capaz de encontrar la solución óptima, a pesar de partir de la peor situación inicial.

Por lo tanto, lo que se propone en esta sección es utilizar un algoritmo genético [13] para explorar subconjuntos de atributos y utilizar una estrategia evolutiva simple (1+1)-ES para encontrar proyecciones para cada subconjunto. Todo esto comenzando por una población compuesta por las soluciones más simples, es decir, los atributos individuales. Estamos hablando de un algoritmo evolutivo con dos niveles de evolución, de los cuales el superior corresponde a un algoritmo genético. Sus cromosomas binarios codifican los subconjuntos de atributos mientras que los cromosomas de la estrategia evolutiva codifican los coeficientes reales de la proyección. Para recordar los detalles de codificación de la estrategia evolutiva el lector deberá volver a la sección 2.3. Respecto al nivel del algoritmo genético (AG), el modo más natural de codificar subconjuntos de atributos consiste en asignar un bit por atributo (1 indicando presencia y 0 ausencia de dicho atributo).

El algoritmo ADELM procede de la siguiente manera:

1. Se elige y se fija la dimensión del espacio proyectado (p). El algoritmo busca proyecciones de d a p dimensiones.

2. Se normalizan los ejemplos, es decir, los atributos x_i $(i = 1, \ldots, d)$ son objeto de la siguiente transformación:

$$x_i \rightarrow \frac{x_i - m_i}{\sigma_i}, \tag{2.21}$$

donde m_i y σ_i son la media y la desviación estándar, respectivamente, del atributo número i.

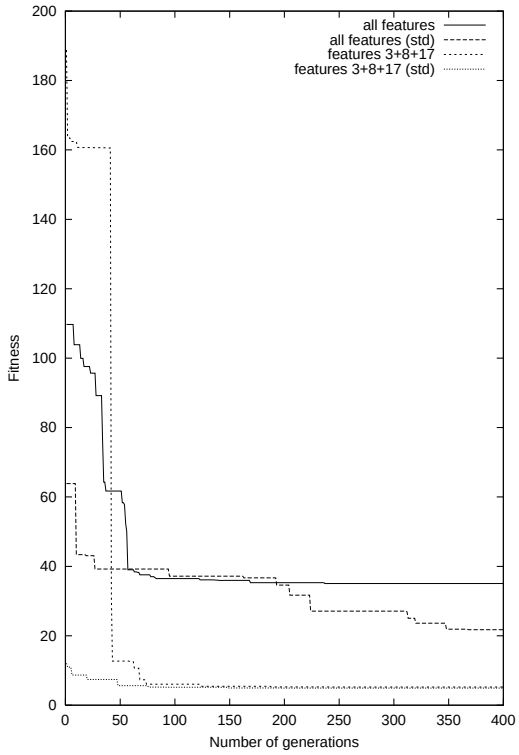

Figura 2.6: Normalización frente a selección de atributos
Fitness frente al número de generaciones para cuatro ejecuciones de ADEL aplicadas a la búsqueda de proyecciones unidimensionales del problema *thyroid2*. Se utiliza una estrategia evolutiva (1+1)-ES y dos conjuntos de atributos: el conjunto completo con los 21 atributos y el subconjunto (x_3, x_8, x_{17}). Ambos se prueban para sus valores originales y sus correspondientes valores normalizados. La normalización mejora el resultado pero la clave radica en encontrar el subconjunto adecuado de atributos.

3. La población inicial, es decir, los subconjuntos de atributos iniciales, son los siguientes d cromosomas binarios (uno para cada atributo):

$$\begin{aligned} (1,0,0,\ldots,0) &\rightarrow x_1, \\ (0,1,0,\ldots,0) &\rightarrow x_2, \\ &\ldots \\ (0,0,0,\ldots,1) &\rightarrow x_d, \end{aligned}$$

representando d subconjuntos unidimensionales.

4. Se repiten los siguientes pasos hasta que se ejecuten 100 generaciones sin mejora:

   - El fitness de cada cromosoma genético se calcula ejecutando ADEL (sección 2.3) pero aplicada a los atributos especificados en el cromosoma binario. El fitness se calcula por validación cruzada del error (sección 2.4).
   - Se construye una nueva población seleccionando los individuos proporcionalmente a su fitness, utilizando SUS (*stochastic universal sampling*), y combinándolos mediante cruce de un punto y mutación [13].

5. El mejor subconjunto se utiliza para calcular la matriz de clasificación sobre el conjunto de test.

Una característica importante de nuestro algoritmo es la población inicial del algoritmo genético. Esta población se construye mediante los subconjuntos más simples, aquellos que hacen uso únicamente de un atributo. Si se desea conseguir una población mayor que el número de atributos se pueden utilizar copias. Estas poblaciones permiten que el algoritmo genético identifique rápidamente los atributos relevantes, pues éstos se utilizan aisladamente. Ambos niveles trabajan conjuntamente para combinar atributos produciendo sucesivamente discriminantes más y más precisos. La tabla 2.6 muestra los sucesivos mejores individuos encontrados por una ejecución típica de este algoritmo para el problema *thyroid2*.

Hemos dedicado esta sección a envolver nuestra estrategia básica en un algoritmo de selección de atributos. Esta no es la primera vez que se mejoran modelos lineales reconociendo atributos relevantes mediante un algoritmo genético. Tanto las redes de enlaces funcionales [47] como incluso los discriminantes de Fisher [41] han sido envueltos mediante un algoritmo genético con anterioridad. Sin embargo, esta es la primera vez que el algoritmo completo es evolutivo, combinando un algoritmo genético y una estrategia evolutiva.

2.8. Trabajo relacionado

Como hemos visto a lo largo de este capítulo, ADEL es una nueva extensión del análisis discriminante de Fisher que tiene como ingrediente clave la minimización directa del error de clasificación. La minimización directa

Gen.	Atributos	% En.	% Val.	% Test
0	x_1	53.31	53.50	53.00
0	x_9	9.06	8.11	9.17
0	x_{17}	4.78	4.17	3.89
1	$x_9\ x_{10}\ x_{17}\ x_{20}$	4.11	3.56	3.83
2	$x_8\ x_{14}\ x_{17}$	3.81	3.50	3.72
3	$x_8\ x_{17}$	3.61	3.44	3.33
5	$x_8\ x_9\ x_{14}\ x_{17}$	3.64	3.39	3.44
8	$x_3\ x_8\ x_9\ x_{17}\ x_{18}$	2.64	2.67	2.33
11	$x_3\ x_{17}\ x_{18}$	2.47	2.83	2.50
14	$x_3\ x_8\ x_{17}\ x_{18}$	2.50	2.72	2.61
14	$x_3\ x_{15}\ x_{17}$	2.36	2.50	2.11
16	$x_3\ x_7\ x_8\ x_9\ x_{17}$	2.33	2.44	2.17
19	$x_3\ x_8\ x_9\ x_{17}$	2.28	2.44	2.17
20	$x_3\ x_8\ x_{17}$	2.17	2.44	2.06
25	$x_3\ x_8\ x_{17}$	2.22	2.39	2.06
28	$x_3\ x_8\ x_{17}$	2.25	2.33	1.94
31	$x_3\ x_8\ x_{17}$	2.22	2.33	2.00

Tabla 2.6: Progreso de ADEL a dos niveles
Sucesivas mejoras encontradas por ADEL en dos niveles para el problema thyroid2. Se muestra el número de generaciones, el subconjunto de atributos y los errores de clasificación sobre los conjuntos de entrenamiento, validación y test. La población inicial corresponde a la generación número 0.

del error ha sido tratada anteriormente mediante algoritmos de búsqueda de proyecciones (*projection pussuit*) [8]. La búsqueda de proyecciones es un procedimiento para encontrar proyecciones interesantes con un número bajo de dimensiones como combinaciones de funciones no lineales (ϕ_j) que a su vez son funciones de combinaciones lineales ($\mathbf{w}_j\mathbf{x}$) de los atributos:

$$\mathbf{x} \rightarrow \sum_{j=1}^{m} \phi_j(\mathbf{w}_j'\mathbf{x}), \tag{2.22}$$

siendo m el número de atributos. Esta proyección es más compleja que nuestras proyecciones lineales, y más relacionada con las proyecciones encontradas por los perceptrones multicapa [18]. La solución se encuentra mediante la

optimización de una función criterio conocida como índice de proyección, el equivalente al criterio de Fisher (ver ecuación 1.14). La probabilidad total de error de clasificación se utiliza como índice en [31], haciendo uso de estimaciones del núcleo de los datos proyectados en vez de los datos originales. En [30], la pérdida total esperada se ha utilizado como índice para buscar proyecciones de clasificación óptimas. Sin embargo, en vez de optimizar el error de clasificación real, sólo se utilizan errores de proyecciones unidimensionales para guiar la búsqueda de proyecciones óptimas. Además, la función indicador que aparece en la pérdida se sustituye por una función continua. En ADEL, dicha sustitución no es necesaria dado que se utiliza una estrategia evolutiva, capaz de optimizar cantidades discontinuas. Existe un intento reciente [36] de optimizar el error de clasificación directamente mediante una técnica de enfriamiento simulado. Según los autores, la técnica puede incorporar peticiones del usuario mediante términos de penalización. Nosotros también utilizamos un algoritmo de optimización global cuya diferencia principal respecto al enfriamiento simulado es el uso de una población de soluciones en lugar de un único candidato.

El perceptrón multicapa es probablemente la extensión más conocida del análisis discriminante clásico [11]. Las proyecciones encontradas por las capas ocultas de un perceptrón multicapa son proyecciones no lineales. Sin embargo, el perceptrón minimiza el error cuadrático y hace uso de códigos de salida. ADEL, por el contrario, no necesita códigos de salida y la dimensión de salida se puede escoger libremente. Nuestra función objetivo discreta tiene un inconveniente al tratar con problemas linealmente separables. Una vez que el fitness es cero, no es posible mejorarlo más. Este fenómeno ocurre al solucionar el problema de los vinos en la sección 2.5 donde diferentes ejecuciones no generalizan bien a pesar de que consiguen el mínimo fitness posible. Esto es menos probable que ocurra al minimizar errores cuadráticos o distancias entre clases. El problema puede ser solventado mediante una técnica de validación cruzada más robusta que haga más difícil que el algoritmo alcance un error de entrenamiento nulo antes de tiempo. Por lo tanto, es recomendable probar primero el algoritmo clásico para comprobar si el problema es linealmente separable. Nuestro algoritmo es especialmente útil en problemas con mezcla de atributos, probabilidades a priori irregulares y distribuciones multimodales.

2.9. Resultados experimentales

Hemos aplicado el análisis discriminante evolutivo lineal al conjunto de bases de datos descrito en el apéndice A. Los resultados de los experimentos se muestran en la tabla 2.7 que completa la tabla 1.4 del capítulo anterior añadiendo los resultados correspondientes a ADEL. Todos los valores de la tabla son porcentajes de error de clasificación del conjunto de test. La primera columna es el nombre de la base de datos. Las columnas 2, 3 y 4 contienen el número de clases, número de ejemplos y número de atributos del problema, respectivamente. La columna 5 muestra el error que se obtiene al clasificar todos los ejemplos como pertenecientes a la clase mayoritaria. Las dos columnas siguientes, 6 y 7, corresponden a resultados obtenidos mediante una red neuronal lineal y no lineal (con una capa oculta), respectivamente. La diferencia entre ambos resultados puede constituir un indicador del nivel de no-linealidad del problema planteado por cada base de datos. Las cuatro columnas siguientes, de la 8 a la 11, corresponden a los algoritmos estudiados en el capítulo anterior: ADF, ADFR, ADFN y ADNL. La última columna está dedicada al algoritmo estudiado en este capítulo, ADEL. El número entre corchetes en esa columna indica la dimensión del espacio proyectado de la mejor solución obtenida por ADEL. Se ha aplicado un test de Wilcoxon no paramétrico [40] entre los porcentajes de error de clasificación de todos los algoritmos para averiguar si la diferencia es o no significativa. En la tabla sólo se muestran los resultados del test entre ADEL (última columna) y el resto de algoritmos. Un valor en negrita indica que la diferencia entre ADEL y el algoritmo correspondiente no es significativa. El mejor resultado en cada base de datos aparece subrayado en la tabla.

Para obtener los resultados de la tabla 2.7, ADEL utiliza una estrategia evolutiva $(15, 50|2)$-ES, es decir, una estrategia evolutiva que utiliza una población de 50 descendientes generados a partir de 15 progenitores y un tamaño de familia igual a 2. El algoritmo tiene un paso de evolución fijo igual a 0,15 y termina cuando transcurren 100 generaciones sin mejoras en el mejor valor de fitness. El número óptimo de proyecciones (dimensión del espacio de salida) se calcula mediante el procedimiento paso a paso, introducido en la sección 2.6 y fijando el número máximo de proyecciones en 20.

En el conjunto de bases de datos binarias, ADEL consigue porcentajes de error de test muy similares a los algoritmos no lineales, llegando a superar por término medio a RNAnl y ADNL. Lo más llamativo es los buenos resultados

que obtiene también en el grupo de bases de datos multiclase, acercándose incluso a los resultados de los algoritmos no lineales. En la base de datos *horse1* consigue el mejor resultado de todos los algoritmos, proyectando los ejemplos sobre un espacio de dos dimensiones. La causa de que los algoritmos no lineales se vean superados por ADEL es seguramente el sobreaprendizaje en el que incurren en la fase de entrenamiento pues se trata de una base de datos con sólo 364 ejemplos.

2.10. Conclusiones

Aunque ADEL ha sido inspirado por el análisis discriminante clásico, ha sido dotado de características que le hacen diferente del algoritmo original y que podrían explicar la mejora que experimenta sobre ADF y ADFR en todas las bases de datos multiclase menos *gene1*. Como en el análisis discriminante clásico, se buscan combinaciones lineales de los atributos originales de modo que los ejemplos sean proyectados más cerca de las medias de sus clases que del resto. De hecho, como hemos mencionado anteriormente, la función de coste de ADEL es el error de clasificación calculado asignando ejemplos a la clase cuya media de entrenamiento proyectada esté más próxima. Sin embargo, en contraste con el enfoque clásico, ADEL minimiza este coste directamente en vez de optimizar un criterio de separación de clases tradicional.

Dado que el número de ejemplos mal clasificados es una cantidad discreta, hemos renunciado a las técnicas de optimización tradicionales como el descenso por gradiente. En cambio, recurrimos a una estrategia evolutiva cuyos cromosomas son los propios pesos de la proyección. El fitness asignado a cada cromosoma es el error cometido al asignar ejemplos a la clase de la media más cercana en el espacio proyectado por el cromosoma. De esta forma, el proceso evolutivo tenderá a aumentar la separación entre clases sin utilizar un criterio explícito de separación de clases. Los beneficios de renunciar a un criterio de separación de clases son muchos. Sobre todo, el nuevo algoritmo es más flexible y puede ser aplicado a situaciones en las que el algoritmo tradicional no puede debido a la singularidad de las matrices de covarianza. Este es el caso del problema *soybean1* de UCI [3] de 19 clases, para el cual obtenemos proyecciones en dos dimensiones con porcentajes de error muy razonables.

Para finalizar, resumimos todos los enfoques expuestos a lo largo de este

capítulo, junto con ADF como punto de partida, como sigue:

- ADF debe ser usado como punto de partida. Es un algoritmo de proyección lineal supervisado ampliamente utilizado que produce excelentes resultados cuando la matriz de covarianza no es singular y el número de clases es bajo. Debe comprobarse si la normalización de los ejemplos mejora el rendimiento.

- ADEL constituye una buena alternativa a ADF, especialmente para problemas multiclase y cuando ADF encuentra dificultades debido a matrices de covarianza singulares. Se pueden obtener proyecciones en dos dimensiones independientemente del número de clases. Si el error de clasificación es la principal preocupación, se puede encontrar el número óptimo de proyecciones siguiendo el procedimiento paso a paso explicado en la sección 2.6.

- ADELM envuelve a ADEL en un algoritmo genético de selección de atributos. Puede mejorar los resultados de ADEL en presencia de ruido o con muchos atributos irrelevantes, que ocurre normalmente en problemas de clasificación con alto número de dimensiones. El problema *thyroid* es un buen ejemplo.

- El procedimiento paso a paso, introducido en la sección 2.6, para encontrar el número óptimo de proyecciones, puede ser aplicado tanto a ADEL como a ADELM.

La clave principal de ADEL (y por tanto de ADELM también) es la minimización evolutiva directa del error de clasificación por distancia a medias en el espacio reducido. Nuestra experiencia muestra que ADEL es un algoritmo muy flexible capaz de dar resultados competitivos incluso en problemas multiclase con una gran componente no lineal.

En el siguiente capítulo de este trabajo investigaremos cómo generalizar ADEL para poder encontrar proyecciones no lineales que sigan sin necesitar de códigos de salida como ocurre con las proyecciones lineales que busca ADEL.

Base de datos	C	Ejem	Atr	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL	ADEL
cancer1	2	699	9	34.5	1.72	(5) 1.44	**2.30**	**2.30**	1.72	(3) 1.75	[1] 2.30
card1	2	690	51	44.0	14.97	(5) 15.26	-	13.95	14.53	(25) 14.22	[1] 14.48
diabetes1	2	768	8	34.9	25.36	(5) 28.00	21.88	23.96	20.83	(7) 25.49	[1] 21.09
digitos35	2	681	225	49.3	2.67	(10) 2.61	-	2.27	1.14	(5) 3.41	[1] 3.12
mushroom1	2	8124	125	48.0	0.12	(5) **0.00**	-	**0.00**	**0.00**	(3) **0.00**	[1] 0.00
spam1	2	4601	57	39.4	7.20	(5) 7.49	10.42	10.42	8.42	(5) **7.65**	[1] 7.23
			Promedio:	41.68	8.67	9.13	11.53	8.81	7.77	8.75	8.04

BD multiclase	C	Ejem	Atr	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL	ADEl
gene1	3	3175	120	50.0	12.54	(10) 13.18	12.86	12.86	8.70	(5) 11.14	[2] 15.13
horse1	3	364	58	38.0	30.83	(5) 30.06	-	38.46	36.26	(7) 34.67	[2] 28.02
iris	3	150	4	66.7	5.33	(3) 2.00	3.33	3.33	0.00	(15) 1.10	[2] 0.67
thyroid2	3	7200	21	7.4	4	(5) **1.67**	27.33	22.00	3.22	(7) 2.99	[2] 1.94
car	4	1728	6	30.0	18.79	(5) 5.81	23.12	23.12	3.18	(25) 5.03	[3] 19.23
glass1	6	214	9	64.5	31.70	(10) **29.15**	39.62	33.96	35.85	(15) 33.30	[5] 30.00
satellite	6	6434	36	76.2	22.39	(10) 11.68	19.02	18.92	10.39	(9) 14.32	[3] 15.25
soybean1	19	683	82	86.5	16.41	(15) **7.29**	-	10.59	7.06	(21) 10.00	[12] 7.67
			Promedio:	52.41	17.75	12.60	20.88	20.40	13.08	14.07	14.74

Tabla 2.7: Resultados para ADEL

Esta tabla completa la tabla 1.4 añadiendo los resultados de ADEL en la última columna. El símbolo "-"que aparece en la columna ADF indica la singularidad de la matriz de covarianza. Los números entre paréntesis de las columnas RNAnl y ADNL indican el número de unidades de la capa oculta de las redes correspondientes. En la última columna (ADEL) se indica entre corchetes la dimensión óptima de la proyección. Los errores en negrita indican diferencias no significativas entre ADEL y el resto de los algoritmos de acuerdo con un test de Wilcoxon no paramétrico [40].

Capítulo 3

ADE: Análisis Discriminante Evolutivo

"Demasiada rectitud impide saborear el detalle."
Anónimo

Este capítulo está dedicado principalmente al Análisis Discriminante Evolutivo no lineal (ADE). Este algoritmo utiliza la potencia no lineal de una red neuronal junto con la naturaleza libre de etiquetas del análisis discriminante clásico. La minimización directa del error de clasificación permite obtener proyecciones no lineales en un número arbitrario de dimensiones. Antes de estudiar este algoritmo en la sección 3.3, se introduce primero en la sección 3.2 un modelo de discriminación no lineal basado en la composición de proyecciones lineales denominado ADE Jerárquico (ADEJ). Este algoritmo solo puede ser aplicado a problemas multiclase. Los resultados de los experimentos que comparan ADE con el resto de los algoritmos estudiados en este trabajo se detallan en la sección 3.5 y, por último, finalizamos con la sección dedicada a las conclusiones y trabajo futuro.

3.1. Introducción

El análisis discriminante de Fisher (ADF) [7], en vez de imponer códigos de salida fijos, hace uso de las medias de las clases como objetivos. En lugar de aprender códigos de salida mediante la minimización del error cuadrático, maximiza una medida de separación de clases. En términos generales, esta medida es la distancia entre las medias dividida por la dispersión alrededor de las medias. Típicamente, una vez encontrada la proyección, los ejemplos se clasifican como pertenecientes a la clase con la media proyectada más cercana. El número de dimensiones del espacio de salida es necesariamente igual al número de clases menos uno.

Aparte de esta restricción en el número de dimensiones de salida, una de las principales limitaciones de este enfoque clásico es su naturaleza lineal. Para solucioanr problemas no lineales se necesitan transformaciones no lineales. Una de las razones clave de la popularidad de los perceptrones multicapa (PMC) es precisamente su habilidad para aproximar relaciones entrada-salida no lineales. La composición de funciones sigmoidales unidimiensionales, $\varphi(x) = 1/(1+e^{-x})$, está detrás de esta capacidad de aproximación no lineal [2]. Nuestra propuesta consiste en aprovechar la aproximación funcional de los PMC con las medias de las clases como códigos de salida.

Como probamos en el capítulo anterior (sección 2.2), no hay una forma sencilla de entrenar perceptrones multicapa con descenso por gradiente con las medias de las clases como códigos de salida. En vez de intentar resolver este problema añadiendo términos de penalización a la función de error, proponemos renunciar a los errores cuadráticos y utilizar el número de ejemplos mal clasificados como objetivo [42]. Puesto que esta función de coste es una cantidad discreta, se ha utilizado una estrategia evolutiva [1] para aprender los pesos de la red. El algoritmo en su versión más simple funciona como sigue:

- Se elige el número de dimensiones de salida y se genera aleatoriamente un PMC inicial.
- Se proyectan los ejemplos y las medias de cada clase.
- El fitness de este PMC es el número de errores cometidos al asignar ejemplos a la clase cuya media esté más cercana en el espacio proyectado.

- Se repiten los siguientes pasos hasta que se alcance un número predefinido de generaciones sin mejoras:
  - Se genera un nuevo PMC añadiendo perturbaciones normales a cada peso del PMC actual.
  - Se calcula el número de ejemplos mal clasificados por el nuevo PMC.
  - Cuando el error se reduce, el nuevo PMC se convierte en el actual.
- El PMC actual es la respuesta del algoritmo.

Una de las claves de este algoritmo es que no impone restricciones sobre el número de dimensiones de salida. De esta forma, podremos obtener visualizaciones en dos dimensiones de los conjuntos de datos sin importar el número de clases al que nos enfrentemos. En la siguiente sección, antes de describir con detalle el algoritmo no lineal, estudiamos otra forma de afrontar problemas no lineales componiendo modelos lineales de forma jerárquica.

3.2. ADEJ: Discriminación jerárquica

El esfuerzo realizado hasta ahora en ADEL está dirigido a liberar la discriminación de un criterio intermedio mediante la utilización del error de clasificación. Esto ha sido posible evitando las matrices de covarianza utilizadas por Fisher y obteniendo así resultados positivos en problemas de clasificación como *soybean1*, en los que, de lo contrario, las matrices utilizadas hubieran sido singulares. No obstante, seguimos necesitando tratar una de las principales limitaciones del análisis discriminante clásico, es decir, su naturaleza lineal. Ante problemas no lineales debemos recurrir a proyecciones no lineales o combinaciones de proyecciones lineales como proponemos en esta sección.

La composición de aplicaciones lineales es probablemente el modo más sencillo de construir proyecciones no lineales. Esta es la idea detrás de muchos algoritmos de aprendizaje, como algunos árboles de decisión (model trees) [4], y la mezcla modular de expertos [20]. En esta última se combinan un conjunto de expertos neuronales lineales mediante lo que se denomina *gating network*. Los pesos de cada experto se actualizan proporcionalmente al error cometido y a su responsabilidad sobre el ejemplo. De esta forma, los expertos tienden a especializarse aprendiendo diferentes regiones del espacio de características.

Igualmente, los árboles de decisión adjuntan modelos de regresión lineal a las hojas del árbol durante la fase de poda. De esta forma, la estructura del árbol juega el papel de *gating network*.

El procedimiento descrito en esta sección es también un enfoque de este tipo especialmente diseñado para problemas multiclase. En nuestro caso, las piezas lineales corresponden a particiones del espacio de clases. Por ejemplo, en un problema de tres clases, podemos separar primero las dos primeras clases de la tercera y posteriormente separar los ejemplos que han terminado en el grupo compuesto. Podemos llevar a cabo ambas tareas con proyecciones lineales como las vistas en secciones anteriores. Sin embargo, la proyección combinada terminará siendo un modelo no lineal.

El algoritmo denominado Análisis Discriminante Evolutivo Jerárquico (ADEJ), funciona como sigue:

- Se divide el espacio de clases jerárquicamente. Esta división puede representar mediante un árbol cuyos nodos contienen subconjuntos de las clases. El nodo raíz contiene el conjunto completo de clases y las hojas son las clases aisladas. Se adjunta una proyección lineal a cada nodo. La proyección de cada nodo clasifica en tantos subconjuntos como ramas parten de dicho nodo.

- Se ejecutan ADEL o ADELM para cada nodo del árbol. Las clases utilizadas para calcular cada modelo lineal son los grupos especificados por los nodos descendientes.

- Los ejemplos son clasificados aplicando los modelos de los nodos correspondientes empezando por el nodo raíz hasta que se alcanza una de las hojas.

El problema *thyroid2*, del repositorio UCI [3], es un ejemplo extremadamente bueno de la efectividad de este procedimiento. Definitivamente, éste es un problema no lineal y en secciones previas se muestra que el mejor error de clasificación que se puede obtener mediante proyecciones lineales es un 2 %. A fin de sobrepasar este umbral, debemos hacer uso de proyecciones no lineales. Existen sólo tres formas de partir un problema de tres clases y sólo se necesita obtener dos proyecciones lineales para cada partición. La tabla 3.1 compara el rendimiento de las tres particiones. Todos los valores mostrados en esta tabla son el promedio de 20 ejecuciones del algoritmo. Cada modelo lineal (dos por partición) se construye con el algoritmo ADELM (sección

2.7). De la tabla se deduce claramente que la partición que separa primero pacientes normales (tercera clase) de pacientes enfermos (primera y segunda clases) es la mejor. Tiene el mejor fitness y el mejor error de generalización, un 0,77 % de error medio de test. Esta cifra representa una mejora media del 60 % sobre las mejores proyecciones lineales.

Partición	% Entr.	% Val	% Test	Fitness
$(123) \rightarrow (3)(12) \rightarrow (1)(2)$	0.95	0.81	0.77	1.76
$(123) \rightarrow (2)(13) \rightarrow (1)(3)$	3.36	2.95	3.01	6.31
$(123) \rightarrow (1)(23) \rightarrow (2)(3)$	1.47	1.36	1.49	2.84

Tabla 3.1: Partición de las clases de thyroid2
Porcentajes de error y fitness de los modelos construidos mediante las tres particiones del espacio de clases del problema thyroid2. Se evolucionan dos proyecciones lineales de dos dimensiones para cada partición. Los valores son promediados de 30 ejecuciones de ADELM para cada partición.

Analicemos la clave del éxito de esta partición y su correspondiente proyección no lineal. La tabla 3.2 muestra una de las 30 ejecuciones de la primera partición de la tabla 3.1. Se muestran los sucesivos mejores discriminantes junto con los atributos involucrados para los dos nodos de la partición. El hecho más sorprendente es que los pacientes con hipotiroidismo e hipertiroidismo se pueden separar perfectamente haciendo uso únicamente de un atributo, x_{21}.

Esto es posible debido a que se han descartado los pacientes normales al construir esta proyección. En otras palabras, el problema se vuelve mucho más sencillo de resolver si lo enfocamos como una combinación de dos subproblemas. Pensamos que esta misma es la razón del éxito de las soluciones a este problema obtenidas previamente mediante árboles de decisión y la dificultad de alcanzar resultados de reconocimiento similares con, por ejemplo, redes neuronales [33]. Finalmente, la selección de atributos es también una cuestión clave para este conjunto de datos. En el segundo nodo se obtiene un 0 % de error en entrenamiento mientras que si se utilizan todos los atributos se obtiene un error de test mayor. Esto es debido a que se minimiza directamente el error y por tanto, una vez que se alcanza un 0 % de error en entrenamiento, el proceso evolutivo se detiene. Sin embargo, esto no ocurre cuando la proyección involucra únicamente el atributo x_{21}. Con este modelo tan sencillo se consigue una correlación perfecta entre los errores de

Nodo	Entr. %	Val %	Test %	Atributos	Fitness
(123) → (12)(3)	50.28	50.61	49.89	x_1	100.89
	8.94	8.06	9.11	x_9	17.00
	8.89	8.00	8.28	x_{12}	16.89
	5.61	5.17	4.94	x_{17}	10.78
	3.69	2.83	2.94	x_{10} x_{17}	6.52
	3.28	2.56	3.00	x_{10} x_{17}	5.83
	1.36	1.17	1.06	x_3 x_8 x_{13} x_{17}	2.53
	1.08	0.94	1.00	x_3 x_8 x_{13} x_{15} x_{17}	2.03
	0.94	0.78	0.72	x_3 x_8 x_{13} x_{15} x_{17}	1.72
(12) → (1)(2)	52.50	56.45	59.23	x_1	108.95
	28.21	26.61	28.46	x_3	54.83
	15.36	17.74	17.69	x_{17}	33.10
	5.71	4.03	6.92	x_{19}	9.75
	0.00	0.00	0.00	x_{21}	0.00

Tabla 3.2: Primera partición de las clases de thyroid2
El problema thyroid2 se parte en dos subproblemas, cada uno solucionado mediante ADELM. Este primer problema consiste en separar pacientes normales del resto. El error de test para esta tarea es del 0,72 %. El segundo problema consiste en romper los pacientes enfermos en hiper e hipotiroideos. Sorprendentemente, esta tarea se resuelve con el atributo x_{21} (TSI: *thyroid stimulating immunoglobulin*) y un error del 0 %.

entrenamiento, validación y test.

En la tabla 3.3 se puede comprobar que las jerarquías de clases son capaces de resolver el problema *thyroid2* satisfactoriamente. El éxito está relacionado con el de los árboles (CART tree) y las reglas simples (C-MLP2LN) para este problema, que en esencia utilizan el enfoque divide y vencerás, y con el fallo de las redes neuronales (MLP) para alcanzar porcentajes de error similares.

La capacidad no lineal de ADEJ reside, en gran parte, en el número de clases del problema. Cuanto mayor sea su número mayor será el de jerarquías de clasificadores binarios posibles, obteniendo a su vez un modelo global más complejo. Como contrapartida, el espacio de búsqueda de la jerarquía óptima crece también rápidamente pudiendo llegar a convertirse en un problema en sí mismo. Por otro lado, para problemas binarios existe únicamente una manera de combinar las clases, de modo que estos problemas nunca podrán ser

Algoritmo	Error entr.	Error test	Ref.
Decision tree	-	0.50	[48]
C-MLP2LN rules	0.11	0.64	[6]
CART tree	0.21	0.64	[50]
PVM	0.20	0.67	[50]
ADEJ	0.95	0.76	[45]
MLP (8+4)	1.00	1.28	[33]
ADEL	1.25	1.94	[45]
HOFDA	-	2.27	[41]

Tabla 3.3: Varios algoritmos aplicados al problema *thyroid2*
La clave para solucionar este problema no lineal parece encontrarse en partir el problema en subtareas. Tanto reglas como árboles y ADEJ se benefician de ello.

tratados de forma no lineal mediante este algoritmo. Por esta y otras razones continuaremos en la siguiente sección con la extensión no lineal del análisis discriminante evolutivo para obtener un único clasificador no lineal.

3.3. ADE: Análisis Discriminante Evolutivo

Uno de los ingredientes principales de ADE es la sustitución del error cuadrático tradicional por el número de ejemplos mal clasificados. Esta cantidad discreta será minimizada mediante una estrategia evolutiva, que se describe detalladamente en esta sección. Empecemos describiendo el esquema de codificación.

3.3.1. Esquema de codificación

Los valores reales de los pesos de los PMCs conducen claramente a la elección de una estrategia evolutiva como algoritmo de optimización, en lugar de un algoritmo genético, por ejemplo. Además, dado que la mayoría de los experimentos realizados en este capítulo hacen uso de redes con dos capas de pesos, ilustraremos el esquema de codificación con una de estas redes.

Se utilizarán PMCs con una capa de neuronas ocultas, es decir, dos capas de pesos: d+1 unidades de entrada (d es el número de atributos del problema), m unidades ocultas, y n unidades de salida. Los pesos de la primera capa se denominan w_{ij}, donde $i = 0, \ldots, d$ y $j = 1, \ldots, m$. Los pesos de la segunda

capa o pesos de salida se denominan v_{jk}, donde $j = 0, \ldots, m$ y $k = 1, \ldots, n$. En las unidades ocultas se utilizan funciones de activación sigmoidales $\varphi(z) = 1/(1 + \exp^{-z})$. La componente número k de la función representada por este PMC es la siguiente:

$$y_k(\mathbf{x}) = \sum_{j=0}^{m} v_{jk}\varphi(\sum_{i=0}^{d} x_i w_{ij}). \tag{3.1}$$

Los pesos (w_{ij}, v_{jk}) se sitúan en los cromosomas de la estrategia en el siguiente orden. Primero, los pesos w_{ij}, desde $i = 0$ hasta $i = d$, y después los pesos v_{jk}, desde $j = 0$ hasta $j = n$:

$$(\ldots, w_{i1}, w_{i2}, \ldots, w_{im}, \ldots, v_{j1}, v_{j2}, \ldots, v_{jn}, \ldots). \tag{3.2}$$

Éste es un esquema de codificación directo y probablemente la forma más sencilla de codificar funciones de proyección.

3.3.2. Fitness

El valor de fitness asignado a un cromosoma (dado por la ecuación 3.2) es la suma del porcentaje de ejemplos mal clasificados en entrenamiento y validación por la red representada por este cromosoma. Este porcentaje de error de clasificación se calcula proyectando primero los ejemplos y después asignándolos a la clase con la media proyectada más cercana. El conjunto de ejemplos se divide en tres partes: conjunto de entrenamiento, validación y test. Las medias de cada clase se calculan con los ejemplos de entrenamiento exclusivamente. Ambos conjuntos, el de entrenamiento y el de validación, se clasifican por distancia a dichas medias proyectadas. Los detalles del algoritmo para una proyección de n a d dimensiones son los siguientes:

- Las medias de entrenamiento (m_i^r), donde $i = 1, \ldots, d$ y $r = 1, \ldots, c$, se proyectan de acuerdo con los pesos $(\mathbf{w}, \mathbf{v})$ como sigue

$$\bar{m}_k^r = \sum_{j=0}^{m} v_{jk}\varphi(\sum_{i=0}^{d} m_i^r w_{ij}) \qquad k = 1, \ldots, n, \tag{3.3}$$

  donde $\varphi(z)$ es la función sigmoidal.

- Los ejemplos ($\mathbf{x}$) de los conjuntos de entrenamiento y validación se proyectan de acuerdo con

$$\bar{x}_k = \sum_{j=0}^{m} v_{jk}\varphi(\sum_{i=0}^{d} x_i w_{ij}) \qquad k = 1, \ldots, n. \tag{3.4}$$

- Los ejemplos se asignan a la clase con la media de entrenamiento más cercana en el espacio proyectado

$$\text{clase}(\mathbf{x}) = \arg \min_{r=1,\ldots,c} (\sum_{k=1}^{n} (\bar{x}_k - \bar{m}_k^r)^2) \tag{3.5}$$

- El fitness $J_{ADE}(\mathbf{w})$ asignado a $(\mathbf{w}, \mathbf{v})$ es el porcentaje de ejemplos mal clasificados para esta proyección. Dicho valor lo obtenemos de forma similar a $J_{ADEL}(\mathbf{w})$ en la sección 2.4.

3.3.3. Algoritmo de aprendizaje

Una vez estudiado cómo asignar fitness a los cromosomas, estamos preparados para escribir el algoritmo denominado Análisis Discriminante Evolutivo (ADE):

- Se elige el número de dimensiones de la proyección.
- Se elige una arquitectura para la red: el número de capas ocultas y el número de unidades en cada capa. Nuestros experimentos utilizan dos capas de pesos.
- Se elige una estrategia evolutiva $(\mu, \lambda|\rho)$: μ es el tamaño de la población de progenitores, λ es el tamaño de la población de descendientes, y ρ es el tamaño de la familia, es decir, el número de progenitores que formarán parte de la recombinación de cada descendiente.
- El paso de mutación ($\sigma = 0{,}15$) se mantiene fijo durante todo el proceso evolutivo.
- Se genera una población inicial de μ redes neuronales con pesos extraídos aleatoriamente en $[-0{,}5, 0{,}5]$.

- Los siguientes pasos se repiten hasta alcanzar un número predefinido de generaciones (100 en nuestros experimentos) sin mejoras del mejor cromosoma:

  - Se genera un grupo de λ redes neuronales por recombinación discreta de ρ redes progenitoras. La recombinación discreta funciona como sigue. Para cada descendiente, se eligen aleatoriamente ρ individuos de la población de progenitores (familia). Seguidamente, cada alelo de cada descendiente será igual al alelo correspondiente de uno de los ρ progenitores, elegidos aleatoriamente del subconjunto formado por la familia.
  - Se muta cada red recombinada añadiendo desviaciones normales independientes a cada uno de los pesos. La desviación de la distribución normal es $\sigma = 0{,}15$.
  - Se utiliza el remplazamiento *coma*, es decir, se crea una nueva población de progenitores formada por los μ mejores de entre los λ descendientes.

Las siguientes secciones informan sobre los resultados obtenidos al aplicar este algoritmo a diversos conjuntos de ejemplos y problemas de clasificación.

3.4. Un ejemplo ilustrativo: satellite

En esta sección, antes de analizar los resultados para el resto de las bases de datos, se estudia con cierto detalle el conjunto *satellite*, de imágenes tomadas por satélite [3]. De las bases de datos multiclase detalladas en la sección A.2, *satellite* ha sido elegida por su alto número de clases (6) y por su dificultad. Aunque ADF obtiene proyecciones lineales en cinco dimensiones, ADE obtiene proyecciones no lineales en dos dimensiones con porcentajes de reconocimiento comparables a los del estado del arte.

La tabla 3.4 muestra los resultados de 20 ejecuciones de ADE con una arquitectura 36:8:2, es decir, ocho unidades ocultas y dos dimensiones de salida. El error medio de test es 13,8 %, que claramente mejora el error medio de test de ADEL. Las desviaciones medias de todas las ejecuciones casi doblan las lineales, lo que significa que existe una mayor variedad de soluciones. El valor medio del número de evaluaciones hasta el mejor individuo también se

Modelo	Entr.	Valid.	Test	Eval. ($x10^3$)	Fitness
1	11.55	8.32	13.31	86.54	19.87
2	12.39	8.00	14.45	113.06	20.40
2	11.37	8.00	12.27	85.63	19.38
4	12.37	8.21	13.72	97.66	20.58
5	13.81	10.60	13.93	23.99	24.42
6	12.66	8.63	13.72	67.91	21.29
7	11.95	8.00	14.35	71.08	19.95
8	12.62	9.04	15.07	77.10	21.66
9	11.97	8.21	12.99	119.78	20.18
10	11.88	7.28	13.41	122.78	19.16
11	12.82	7.59	13.93	71.77	20.40
12	11.84	7.80	13.31	73.14	19.64
13	12.84	9.25	14.14	82.58	22.09
14	12.51	9.67	14.24	60.28	22.17
15	13.22	9.04	14.66	72.64	22.26
16	11.62	8.32	14.35	95.57	19.93
17	12.93	9.56	14.86	74.56	22.49
18	12.57	8.94	12.58	83.28	21.51
19	12.04	7.48	13.20	73.89	19.52
20	12.93	8.32	13.62	80.14	21.24
Media	12.39	8.51	13.80	81.67	20.91
Desv.	0.60	0.81	0.72	21.32	1.30

Tabla 3.4: 20 ejecuciones de ADE aplicado a *satellite*
20 ejecuciones de ADE con ocho unidades ocultas y dos dimensiones de salida para la base de datos *satellite*. Se muestran los errores de entrenamiento, validación y test de los mejores individuos de cada ejecución. El número de evaluaciones de la función de fitness necesarias para encontrar el mejor individuo y su fitness se muestran en las columnas 5 y 6, respectivamente.

dobla. La nueva matriz de confusión es la siguiente (que debe ser comparada

con la matriz de confusión de ADEL (ecuación 2.20):

$$C_{sat} = \begin{pmatrix} 225 & 1 & 2 & 0 & 1 & 0 \\ 6 & 95 & 0 & 0 & 4 & 0 \\ 4 & 0 & 192 & 6 & 0 & 1 \\ 1 & 1 & 18 & 55 & 0 & 18 \\ 9 & 8 & 1 & 1 & 75 & 12 \\ 0 & 1 & 10 & 25 & 4 & 186 \end{pmatrix} \tag{3.6}$$

La figura 3.1 muestra cómo varían el fitness y el error de test con el número de unidades ocultas, cuando se utilizan dos unidades de salida. La arquitectura 36:2:2, como se puede ver en el pico de la figura, parece quedar atrapada en un mínimo local. En general, el rendimiento mejora con el número de unidades ocultas como era de esperar pero, alrededor de 22 unidades, tanto el fitness como el error de test empiezan a empeorar. El error de test correspondiente al mejor fitness está por debajo del 13 %.

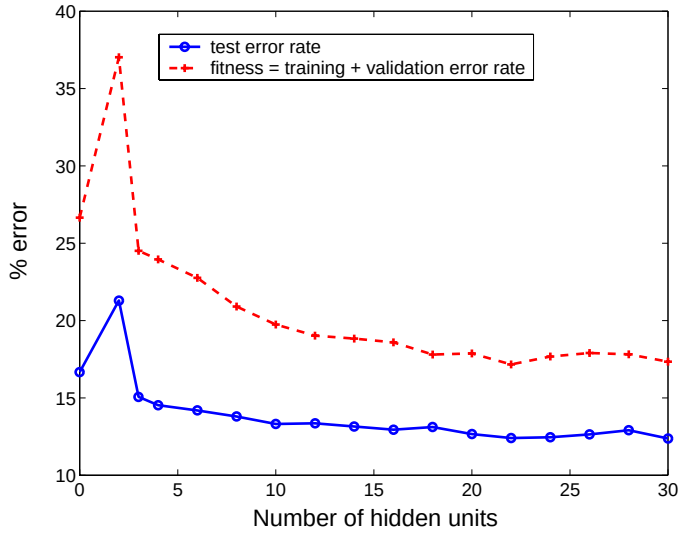

Figura 3.1: ADE aplicado a la clasificación de satellite con un número creciente de unidades ocultas
Valores de fitness y error de test promediados sobre 20 ejecuciones de ADE para el conjunto *satellite* con dos unidades de salida y un número creciente de unidades ocultas.

En la tabla 3.5 se compara ADE con otros algoritmos de clasificación aplicados al conjunto *satellite*. Como se puede ver, la clasificación uno contra

todos (OVA) obtiene los mejores resultados (7,8 %). OVA hace uso de $c = 6$ clasificadores SVM binarios, cada uno entrenado para distinguir los ejemplos de una clase del resto de ejemplos. Los nuevos ejemplos se clasifican como pertenecientes a la clase del clasificador con el mayor valor de salida para la clase aislada. El rendimiento en el problema satellite es excelente principalmente debido a la combinación de clasificadores [35]. Entre los modelos de clasificación individuales, la regla $k = 1$ vecinos próximos obtiene el mejor resultado, con un 9,4 % de error de test [26]. Los resultados obtenidos por ADE y ADEL son los correspondientes a las proyecciones con mejor fitness. El resultado no lineal está cerca del obtenido por 1NN incluso habiendo utilizado 2 proyecciones de salida únicamente.

Algoritmo	Entr. %	Test %	Ref.
OVA (SVM)	-	7.8	[35]
KNN	8.9	9.4	[26]
ADF	17.49	19.02	[45]
ADEL	14.07	15.45	[45]
ADE	11.04	12.41	[44]

Tabla 3.5: Comparación de algoritmos aplicados a la clasificación de *satellite*

KNN es $k = 1$ vecinos próximos. OVA es clasificación uno contra todos (*one-versus-all*) con redes SVM como clasificadores binarios.

3.5. Resultados experimentales

Hemos aplicado el análisis discriminante evolutivo no lineal al conjunto de bases de datos del apéndice A. Los resultados de los experimentos se muestran en la tabla 3.6 que completa la tabla 2.7 añadiendo los resultados de ADE en la última columna. Todos los valores son porcentajes de error del conjunto de test. La primera columna es el nombre de la base de datos. Las columnas 2, 3 y 4 contienen el número de clases, número de ejemplos y número de atributos del problema. La columna 5 muestra el error que se obtiene al clasificar todos los ejemplos como pertenecientes a la clase mayoritaria. Las dos columnas siguientes, 6 y 7, corresponden a resultados obtenidos mediante una red neuronal lineal y no lineal (con una capa oculta), respectivamente. La diferencia entre ambos resultados puede constituir un indicador del nivel

de no-linealidad del problema planteado por cada base de datos. Las cuatro columnas siguientes, de la 8 a la 11, pertenecen a los algoritmos estudiados en el primer capítulo: ADF, ADFR, ADFN y ADNL. La penúltima columna está dedicada a los resultados obtenidos por ADEL. ADE, estudiado en este capítulo, se puede encontrar en la última columna (ADE). Se ha aplicado un test de Wilcoxon no paramétrico [40] entre los porcentajes de error de clasificación de todos los algoritmos para saber si la diferencia es o no significativa. En la tabla sólo se muestran los resultados del test entre ADE (última columna) y el resto de algoritmos. Las cifras en negrita indican que la diferencia entre ADE y el algoritmo correspondiente no es significativa. El mejor resultado para cada base de datos aparece subrayado en la tabla.

Para obtener los resultados de la tabla 3.6, ADE utiliza una estrategia evolutiva $(15, 50|2)$-ES, es decir, una estrategia que utiliza una población de 50 descendientes a partir de 15 progenitores y un tamaño de familia igual a 2. El remplazamiento de tipo coma utilizado hace que los progenitores se elijan de entre los mejores descendientes. El algoritmo tiene un paso de mutación fijo igual a 0,15 y termina cuando transcurren 100 generaciones sin mejoras en el valor de fitness. La arquitectura de la red utilizada por ADE en los experimentos consta de tres capas. La capa de entrada tiene un número de unidades igual al número de atributos del problema, la capa de salida tiene un número de unidades igual al número de proyecciones elegido, y la capa oculta tiene un número de unidades también parametrizable. El número óptimo de dimensiones (unidades de la capa de salida) se calcula mediante el procedimiento paso a paso, introducido en la sección 2.6 con un máximo de 20. El número de unidades de la capa oculta se determina de la siguiente manera. El valor inicial corresponde al número de unidades del intervalo $[1, n]$ (n es el número de atributos) con el error de validación mínimo. A continuación se exploran valores a partir de n hasta que el error supere el del valor inicial. ADE se queda con el número de unidades con un error de validación más bajo, que puede ser el inicial.

Los resultados obtenidos por el análisis discriminante evolutivo no lineal son bastante satisfactorios. Para las bases de datos binarias únicamente necesita una proyección y un número no muy alto de unidades en la capa oculta para conseguir mejorar a ADEL. En las bases de datos multiclase, ADE mejora notablemente a ADEL debido a la complejidad de dichos conjuntos. Por término medio, ADE obtiene los mejores resultados tanto en las bases de datos binarias como en las multiclase. Además, ADE es el mejor algoritmo

para la mitad de las bases de datos, siendo sólo superado por ADNL.

A medida que aumenta la complejidad del problema, la solución obtenida por ADE es también más compleja. Esto se refleja en la dimensión de la proyección (número de unidades de salida) y en el número de unidades de la capa oculta de la mejor solución obtenida por el algoritmo. Es importante destacar el buen resultado obtenido por ADE en el conjunto *horse1* utilizando únicamente una dimensión de salida. Otra característica destacable es que la proyección obtenida por ADE en todos los conjuntos de datos multiclase, tiene menor dimensión que la obtenida por ADEL y sin embargo, ADE obtiene mejores resultados de clasificación.

La regla de clasificación utilizada por ADE, que asigna los ejemplos a la clase cuya media proyectada esté más cerca, no soluciona el problema XOR debido a que las medias de las clases se solapan completamente y al proyectarlas seguirán solapándose, impidiendo la discriminación de los ejemplos en el espacio proyectado. Esto se soluciona volviendo a calcular las medias en el espacio proyectado en vez de proyectar las obtenidas en el espacio inicial. Esta nueva versión de ADE genera resultados muy similares a los obtenidos por el algoritmo original pero incurriendo en un coste computacional sumamente elevado. Por esta razón deo eficiencia se ha utilizado ADE como la versión central de este trabajo. Si el solapamiento entre las medias de las clases es total se puede utilizar la nueva versión aunque sea mucho más lenta. La razón por la que ADE sigue obteniendo resultados muy competitivos, aunque no es capaz de solucionar el problema XOR, es que optimiza la proyección obtenida para que obtenga el menor porcentaje de ejemplos mal clasificados, sea cual sea la regla de clasificación utilizada. En nuestro caso, siempre que las medias de las clases no estén completamente solapadas, ADE podrá obtener un resultado muy similar al obtenido por la nueva versión con un ahorro computacional muy grande.

3.6. Conclusiones

Como vimos en el primer capítulo, una de las principales limitaciones del análisis discriminante clásico es su naturaleza lineal. De hecho, las proyecciones de ADF son combinaciones lineales de los atributos del problema. Existen problemas como *thyroid2* que definitivamente exigen modelos no lineales. Nuestro enfoque inicial de la construcción de proyecciones no lineales utilizaba una composición jerárquica de proyecciones lineales basada en la

descomposición del espacio de clases. Como ejemplo de prueba, el problema thyroid ha sido resuelto mediante una composición de dos proyecciones lineales. La primera separa pacientes sanos de pacientes enfermos. La segunda proyección se aplica a aquellos ejemplos que caen en el primer grupo (pacientes enfermos), separándolos en hipertiroideos e hipotiroideos. El modelo compuesto mejora por término medio en un 60 % el modelo lineal y consigue obtener un error de test medio del 0,76 %.

La composición jerárquica de proyecciones lineales solo se puede utilizar cuando el problema tiene más de dos clases. Por esta y otras razones se ha introducido ADE, un algoritmo que permite entrenar perceptrones multicapa sin necesidad de imponer códigos de salida. Esta flexibilidad se consigue renunciando al error cuadrático de clasificación y minimizando el número de errores de clasificación por distancia a medias mediante una estrategia evolutiva. Los resultados son muy satisfactorios pues por término medio, ADE supera a otros algoritmos de discriminación como, por ejemplo, ADF, ADFR, ADFN y ADNL.

Uno de los principales objetivos de este trabajo ha sido desarrollar un algoritmo capaz de obtener visualizaciones no lineales en dos dimensiones de problemas multiclase. Este trabajo completa un hueco importante en la reducción dimensional, debido a que en la mayor parte de los algoritmos de este tipo no se puede escoger el número de proyecciones. Por ejemplo, el análisis discriminante clásico proyecta en un número de dimensiones igual al número de clases menos uno. Aunque es cierto que este número puede ser reducido descartando algunas combinaciones lineales, los atributos rechazados producirán una pérdida de información. ADE, desde el principio, puede buscar en dos dimensiones la mejor forma de proyectar los ejemplos para optimizar el error de clasificación por distancia a medias.

Nuestro procedimiento puede ser generalizado de diferentes formas. Lo primero a tener en cuenta es que no hay necesidad de restringirnos a las medias de las clases. Las medias pueden ser sustituidas por códigos de proyección evolucionados que se aprenderán junto con las propias proyecciones. Nuestros resultados preliminares muestran que este enfoque puede ser útil en problemas de 2 ó 3 clases.

Otra generalización razonable que, además, puede servir como método de aproximación no lineal, consiste en utilizar más de una proyección por clase. Las medias utilizadas en todos los algoritmos propuestos son el único representante utilizado por cada clase. Sin embargo, si utilizáramos dos o más re-

presentantes por clase, se podrían resolver problemas no lineales sin necesidad de recurrir a la capacidad de aproximación no lineal de los perceptrones multicapa.

Otro aspecto que no ha sido investigado en este trabajo es la determinación de la partición óptima del espacio de clases en ADEJ. Tan solo se ha estudiado un caso sencillo, thyroid2, que solo tiene tres clases y, por lo tanto, dos posibles particiones. El número de particiones crece rápidamente con el número de clases y se vuelve necesario disponer de algún método de búsqueda razonable. Esta es una de las líneas de investigación que se seguirán en el futuro.

Base de datos	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL	ADEL	ADE
cancer1	34.5	1.72	(5) 1.44	2.30	2.30	1.72	(3) **1.75**	[1] 2.30	(4)[1] 1.46
card1	44.0	14.97	(5) 15.26	-	13.95	14.53	(25) 14.22	[1] 14.48	(4)[1] 12.03
diabetes1	34.9	25.36	(5) 28.00	21.88	23.96	**20.83**	(7) 25.49	[1] **21.09**	(5)[1] 20.85
digitos35	49.3	2.67	(10) 2.61	-	2.27	1.14	(5) 3.41	[1] 3.12	(8)[1] 2.19
mushroom1	48.0	0.12	(5) **0.00**	-	**0.00**	**0.00**	(3) **0.00**	[1] **0.00**	(6)[1] 0.00
spam1	39.4	**7.20**	(5) 7.49	10.42	10.42	8.42	(5) 7.65	[1] **7.23**	(6)[1] 7.12
Promedio:	41.68	8.67	9.13	11.53	8.81	7.77	8.75	8.04	7.27

BD multiclase	Z_0	RNAl	RNAnl	ADF	ADFR	ADFN	ADNL	ADEL	ADE
gene1	50.0	12.54	(10) 13.18	12.86	12.86	8.70	(5) **11.14**	[2] 15.13	(9)[2] 11.44
horse1	38.0	30.83	(5) 30.06	-	38.46	36.26	(7) 34.67	[2] 28.02	(7)[1] 25.67
iris	66.7	5.33	(3) 2.00	3.33	3.33	0.00	(15) 1.10	[2] **0.67**	(5)[1] 0.67
thyroid2	7.4	4	(5) **1.67**	27.33	22.00	3.22	(7) 2.99	[2] 1.94	(5)[2] 1.33
car	30.0	18.79	(5) 5.81	23.12	23.12	3.18	(25) 5.03	[3] 19.23	(7)[2] 2.95
glass1	64.5	31.70	(10) 29.15	39.62	33.96	35.85	(15) 33.30	[5] 30.00	(10)[7] 27.73
satellite	76.2	22.39	(10) **11.68**	19.02	18.92	10.39	(9) 14.32	[3] 15.25	(12)[2] 11.82
soybean1	86.5	16.41	(15) 7.29	-	10.59	7.06	(21) 10.00	[12] 7.67	(21)[5] 6.29
Promedio:	52.41	17.75	12.60	20.88	20.40	13.08	14.07	14.74	10.99

Tabla 3.6: Resultados para ADE

Esta tabla completa la tabla 2.7 añadiendo los resultados de ADE en la última columna. El símbolo "-"que aparece en la columna ADF indica la singularidad de la matriz de covarianza. Los números entre paréntesis de las columnas RNAnl, ADNL y ADE indican el número de unidades de la capa oculta. En las columnas ADEL y ADE se indica entre corchetes la dimensión óptima de la proyección. Se ha aplicado un test de Wilcoxon no paramétrico [40] entre los porcentajes de error de clasificación de la última columna (ADE) y el resto de algoritmos. Los resultados en negrita indican que la diferencia entre ambos valores no es significativa. Este test no paramétrico se ha utilizado previamente para analizar el comportamiento de algoritmos evolutivos [12], como también lo son ADEL y ADE.

Apéndice A

Bases de datos

En este apéndice describimos las bases de datos utilizadas a lo largo del trabajo. La mayor parte de los experimentos han sido realizado sobre una selección de conjuntos de ejemplos pertenecientes al repositorio de la Universidad de California (UCI) [3], junto con algunas bases de datos de otras fuentes [33]. Los conjuntos pertenecientes al repositorio de UCI han sido preprocesados y se encuentran disponibles en varias permutaciones marcadas por el número final que acompaña al nombre [33]. Por ejemplo, el conjunto *diabetes* está disponible como *diabetes1*, *diabetes2* y *diabetes3*. Se trata de diferentes particiones del mismo conjunto en conjunto de entrenamiento, validación y test.

Los subconjuntos de entrenamiento y validación son los únicos que se utilizan durante el entrenamiento de los algoritmos de este trabajo. Por ejemplo, en los algoritmos evolutivos, las medias de las clases se calculan exclusivamente con el conjunto de entrenamiento. Los ejemplos de validación se utilizan para el cálculo del fitness de los cromosomas. Sin embargo, el conjunto de test no interviene en el aprendizaje, de tal forma que constituye una estimación de la capacidad de generalización de las proyecciones resultantes.

Se ha aplicado un test de Wilcoxon no paramétrico [40] para averiguar si la diferencia entre los resultados de clasificación obtenidos mediante una red neuronal lineal y una red no lineal son significativos. Si el valor de p del test es menor que 0.05, el nivel de significación, la diferencia se acepta como significativa. Este análisis permite detectar posibles componentes no lineales en los problemas estudiados. A continuación se describen las bases de datos divididas en dos grandes grupos: bases de datos binarias (con dos clases) y multiclase (con más de dos clases).

A.1. Bases de datos binarias

A.1.1. cancer1

La base de datos *cancer1* contiene ejemplos de tumores de cáncer de pecho. Los ejemplos vienen marcados como malignos o benignos basándose en las descripciones de las células obtenidas mediante examen microscópico. Los datos se obtuvieron en el hospital univeristario de Wisconsin (Madison) por el Dr. William H. Wolberg [51]. La base de datos consta de 699 ejemplos caracterizados por 9 atributos con valores continuos. El 65 % de los tumores son benignos, lo que supone un error por mayoría del 34.5 %.

Una red neuronal lineal arroja un error de clasificación para el conjunto de test del 1.72 %, mientras que una red neuronal con una capa oculta de 5 unidades comete un error del 1.44 %. Estos resultados son la media de 20 ejecuciones de los algoritmos correspondientes. El test de Wilcoxon no paramétrico [40] da un valor de p de 0.000048, que indica una diferencia significativa y, por lo tanto, la posible existencia de una componente no lineal en este problema.

A.1.2. card1

Los ejemplos de esta base de datos están relacionados con operaciones de tarjetas de crédito. Los nombres de los atributos y sus valores fueron modificados inicialmente por el autor para proteger la confidencialidad de los datos. La base de datos está formada por 690 ejemplos que constan de 51 atributos, agrupados en dos clases. El 55.5 % de los ejemplos pertenecen a la clase negativa. Este conjunto tiene una mezcla interesante de atributos: continuos, nominales con pocos valores posibles, y nominales con un gran número de valores posibles. Existe un 5 % de valores desconocidos.

Se trata de un conjunto con pocos ejemplos (690) pero esta vez con un número muy alto de atributos. El error de clasificación de test para una red neuronal lineal es del 14.97 %, mientras que para una red neuronal con una capa oculta de 5 unidades, es del 15,26 %. El test de Wilcoxon no paramétrico [40] obtiene un p de 0.255 indicando que la diferencia no es significativa. Todo parece indicar que el problema no posee una naturaleza no lineal.

A.1.3. diabetes1

La base de datos *diabetes1* contiene ejemplos de indias Pima clasificadas en dos categorías según que padezcan o no de diabetes. Los 8 atributos continuos se reparten entre datos personales (edad, número de embarazos, etc) y resultados de exámenes médicos (presión sanguínea, índice de masa corporal, resultado de test de tolerancia a la glucosa, etc).

La base de datos contiene 768 ejemplos, de los cuales el 65.1 % no son diabéticos, lo que representa un error por mayoría del 34.9 %. Aunque no hay valores desconocidos en la base de datos, hay muchos valores nulos que podrían indicar pérdida de información.

La red neuronal lineal obtiene un error de test del 25.36 % y la red neuronal no lineal alcanza el 28 %. La diferencia entre ambos porcentajes es significativa, dado que el test de Wilcoxon no paramétrico [40] obtiene un p nulo. Será necesario controlar adecuadamente el sobreajuste de los modelos no lineales para evitar lo que ocurre con estos perceptrones multicapa.

A.1.4. digitos35

La base de datos *digitos35* contiene ejemplos de los dígitos 3 y 5 dibujados mediante un ratón en la pantalla de un ordenador. El conjunto contiene 681 ejemplos, cada uno caracterizado por 225 atributos binarios correspondientes a los 15×15 pixeles de la imagen. Esta base de datos está equilibrada pues el 49.3 % de los ejemplos corresponden al dígito 3.

Se trata de un conjunto con un número reducido de ejemplos pero con el mayor número de atributos de todos los utilizados. Esta característica ralentiza significativamente la mayoría de los algoritmos salvo los basados en núcleos, cuyo coste computacional depende del número de ejemplos, en lugar del número de atributos.

El error de clasificación de test que comete una red neuronal lineal es del 2.67 %, casi el mismo que el cometido por una red neuronal con una capa oculta con 10 unidades, 2.61 %. El test de Wilcoxon no paramétrico [40] devuelve un valor de p igual a 0.41, demostrando que la diferencia no es significativa. Este resultado es un claro indicio de que el problema no tiene una componente no lineal.

A.1.5. mushroom1

La base de datos mushroom1 contiene 8124 ejemplos de setas caracterizadas por la descripción de su hábitat, el color, el olor y la textura. El número de atributos es 125 y sólo uno de ellos tiene valores ausentes. El 52 % de los ejemplos corresponden a setas comestibles, frente al 48 % de setas venenosas.

Este conjunto es especial por varias razones. En primer lugar es el que más ejemplos tiene de todos los utilizados en este trabajo. En segundo lugar, se trata de un problema separable linealmente y, finalmente, ha sido creado sintéticamente a partir de las descripciones de las especies en un libro.

A.1.6. spam1

La base de datos *spam1* contiene ejemplos de correos electrónicos clasificados en dos categorías: correo basura y correo personal. Contiene un total de 4601 ejemplos con 57 atributos cada uno. La clase minoritaria tiene un 39 % de ejemplos.

La red neuronal lineal consigue un 7.2 % de error en el conjunto de test y la red neuronal no lineal un 7.49 %. La diferencia no es significativa ($p = 0{,}062$) de acuerdo con el test de Wilcoxon no paramétrico [40].

A.2. Bases de datos multiclase

A.2.1. gene1

Conjunto de datos formado por 3175 secuencias de nucleótidos. Cada secuencia puede pertenecer a una de tres clases, según que la secuencia de 60 nucleótidos corresponda a un límite intrón/exón (donante), a un límite exón/intrón (receptor), o a ninguno de los dos. Cada ejemplo viene caracterizado por 120 atributos. Esto es así porque cada nucleótido, que tiene cuatro posibles valores nominales, está codificado mediante dos atributos binarios. Hay un 25 % de donantes y un 25 % de receptores, con lo que el error cometido al clasificar en la clase mayoritaria es del 50 %.

La red neuronal lineal obtiene un 12.54 % de error para el conjunto de test. La red neuronal no lineal con una capa oculta de 10 unidades lo supera llegando al 13.18 %. Esta diferencia sí es significativa ($p = 0{,}003$) según el test de Wilcoxon no paramétrico [40], lo cual indica que se ha cometido sobreaprendizaje en el entrenamiento de la red neuronal no lineal.

A.2.2. horse1

Esta base de datos contiene ejemplos de caballos que sufrieron un cólico. Cada caballo se asigna a una de tres categorías de acuerdo con la suerte que corrió tras el cólico: sobrevivir, morir o ser sacrificado. La base de datos contiene 364 ejemplos con 58 atributos cada uno. Los caballos sobrevivieron en el 62 % de los casos, murieron en el 24 %, y el resto (14 %) tuvieron que ser sacrificados. Esto significa que el error cometido al clasificar en la clase mayoritaria es del 38 %. En este problema hay muchos valores omitidos, que se representan explícitamente mediante atributos adicionales.

El error cometido por la red neuronal lineal es 30.83 %, y el de la red no lineal es del 30.06 %. La diferencia no es significativa ($p = 0{,}18$) según el test de Wilcoxon no paramétrico [40].

A.2.3. iris

Esta base de datos contiene ejemplos de flores iridáceas. Contiene 150 ejemplos caracterizados cada uno de ellos por 4 atributos: la longitud del sépalo, la anchura del sépalo, la longitud del pétalo y la anchura del pétalo, todo en centímetros. Las tres clases de iridácea son: setosa, versicolor y virgínica.

La variedad setosa se puede separar linealmente de las otras dos, que no son separables linealmente entre sí. Esto explica que la red neuronal lineal obtenga un 5.33 % de error de clasificación para el conjunto de test y que la red neuronal no lineal obtenga un 2 %. Todo parece indicar que el problema iris es un problema no lineal.

A.2.4. thyroid2

Este conjunto está formado por pacientes de un hospital clasificados según su función tiroidea: hipertiroidismo, normal o hipotiroidismo. La base de datos consta de 7200 ejemplos, cada uno caracterizado por 21 atributos. Faltan valores para algunos de los atributos. El número de ejemplos de cada clase está muy desequilibrado, y el error al asignar todos los pacientes a la clase mayoritaria es del 7.9 %.

La red neuronal lineal obtiene un error de clasificación de test del 4 %, mientras que la red no lineal obtiene un 1.67 %. La diferencia es significativa

($p = 0$) según un test de Wilcoxon no paramétrico [40], lo que puede indicar una componente no lineal en este conjunto.

A.2.5. car

Esta base de datos contiene 1728 ejemplos de coches caracterizados cada uno de ellos por 6 atributos y agrupados en 4 clases. Los ejemplos han sido producidos por un modelo de decisión jerárquico subyacente, lo cual hace que esta base de datos sea especialmente útil para probar métodos de inducción constructiva.

El error que se comete al clasificar todos los ejemplos en la clase mayoritaria es del 30 %. La red neuronal lineal obtiene un 18.79 % de error de clasificación de test, mientras que la red neuronal no lineal llega a un 5.81 %. Esta diferencia es significativa ($p = 0$) de acuerdo con el test de Wilcoxon no paramétrico [40]. Este resultado apunta a una clara componente no lineal en este problema.

A.2.6. glass1

Esta base de datos contiene 214 ejemplos de cristales procedentes de investigaciones forenses de accidentes de tráfico. Cada muestra viene caracterizada por 9 atributos continuos correspondientes al análisis de 8 elementos químicos más el índice de refracción del cristal. El número de clases es 6 dependiendo de la procedencia de la muestra: ventanas, faros delanteros, etc.

El error por clasificación en la clase mayoritaria es del 64.5 %. La red neuronal lineal obtiene un error del 31.70 % y la red neuronal no lineal alcanza un 29.15 %. La diferencia es significativa ($p = 0{,}002$) de acuerdo con el test de Wilcoxon no paramétrico [40].

A.2.7. satellite

La base de datos *satellite* contiene imágenes de satélite clasificadas según el tipo de terreno en 6 grupos: red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, y very damp grey soil. Contiene un total de 6434 ejemplos con 36 atributos enteros cada uno.

La tabla A.1 muestra la distribución del número de ejemplos por clase. El error de test por clasificación en la clase mayoritaria es 76.2 %. La red

neuronal lineal obtiene un 22.39 % de error en el conjunto de test mientras que la red no lineal alcanza un 11.68 %. La diferencia es significativa ($p = 0$) según el test de Wilcoxon no paramétrico [40], lo cual indica una clara componente no lineal en este problema.

type of soil	número	prob. a priori
red soil	1533	0.24
cotton crop	703	0.11
grey soil	1358	0.21
damp grey soil	626	0.10
soil with vegetation stubble	707	0.11
mixture class	-	-
very damp grey soil	1508	0.23

Tabla A.1: Conjunto de datos satellite del repositorio de UCI
Conjunto de datos con imágenes de satélite: 36 atributos enteros y seis clases. La columna 1 muestra los nombres de 7 tipos de terreno. Las columnas 2 y 3 muestran el número de ejemplos de cada clase y la probabilidad a priori, respectivamente.

A.2.8. soybean1

Este conjunto contiene 683 ejemplos de semillas de soja caracterizadas por 35 atributos continuos y marcadas por el tipo de lesión que sufren. Existen un total de 19 enfermedades, cuyos nombres y probabilidades a priori se pueden consultar en la tabla A.2. En este trabajo se ha utilizado una codificación mediante 82 atributos enteros denominada *soybean1* [33], que incluye códigos especiales para el abundante número de datos que faltan.

El error que se comete al clasificar las semillas en la clase mayoritaria es 85 %. La red neuronal lineal obtiene un 16.41 % de error de test, mientras que la red neuronal no lineal con una capa oculta de 15 unidades obtiene un 7.29 %. Esta diferencia es significativa ($p = 0$) según el test de Wilcoxon no paramétrico [40]. El gran número de clases de este problema junto con su naturaleza no lineal lo convierten en un conjunto interesante para poner a prueba a ADE y su habilidad para obtener proyecciones no lineales en un número bajo de dimensiones.

Enfermedad de la soja	número	prob. a priori
herbicide-injury	8	0.01
cyst-nematode	14	0.02
diaporthe-pod-and-stem-blight	15	0.02
2-4-d-injury	16	0.02
bacterial-blight	20	0.03
bacterial-pustule	20	0.03
charcoal-rot	20	0.03
diaporthe-stem-canker	20	0.03
downy-mildew	20	0.03
phyllosticta-leaf-spot	20	0.03
powdery-mildew	20	0.03
purple-seed-stain	20	0.03
rhizoctonia-root-rot	20	0.03
anthracnose	44	0.06
brown-stem-rot	44	0.06
phytophthora-rot	88	0.13
alternarialeaf-spot	91	0.13
frog-eye-leaf-spot	91	0.13
brown-spot	92	0.13

Tabla A.2: Conjunto de datos soybean de UCI
Este conjunto tiene 82 atributos enteros y 19 clases. La columna 1 muestra los nombres de las 19 lesiones del problema. Las columnas 2 y 3 muestras el número de ejemplos de cada clase y las probabilidades a priori, respectivamente.

Bibliografía

[1] H. G. Beyer and H. P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[2] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[3] C. L. Blake and C. J. Merz. UCI repository of machine learning databases [www.ics.uci.edu/mlearn/mlrepository.html], 1998.

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

[5] Y. Q. Cheng, Y. M. Zhuang, and J. Y. Yang. Optimal fisher discriminant analysis using the rank decomposition. *Pattern Recognition*, 25:101–111, 1992.

[6] W. Duch, R. Adamczack, and K. Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, 12:277–306, 2001.

[7] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[8] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9):881–884, 1974.

[9] K. Fukunaga. *Introduction to statistical pattern recognition (2nd ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[10] G. Fung, M. Dundar, J. Bi, and B. Rao. A fast iterative algorithm for fisher discriminant using heterogeneous kernels. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 40, New York, NY, USA, 2004. ACM Press.

[11] P. Gallinari, S. Thiria, F. Badran, and F. F. Soulie. On the relation between discriminant analysis and multilayer perceptrons. *Neural Networks*, 4(3):349–360, 1991.

[12] S. Garcia, D. Molina, M. Lozano, and F. Herrera. Un estudio experimental sobre el uso de test no paramétricos para analizar el comportamiento de los algoritmos evolutivos en problemas de optimización. In *Proceedings Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB07)*, February 2007.

[13] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[14] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205–224, 1965.

[15] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, pages 159–195, 2001.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[17] S.Y. Ho, L. S. Shu, and J. H. Chen. Intelligent evolutionary algorithms for large parameter optimization problems. *IEEE Transactions on Evolutionary Computation*, 8(6):522–541, 2001.

[18] J. N. Hwang, S. R. Lay, M. Maechler, R. D. Martin, and J. Schimert. Regression modelling in back-propagation and projection pursuit learning. *IEEE Transactions on Neural Networks*, 5(3):342–353, 1994.

[19] J. E. Jackson. *A User's Guide to Principal Components.* New York: Wiley, 1991.

[20] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

[21] M. Katz. Robustness of linear discriminant analysis in automatic speech recognition. 2002.

[22] S.-J. Kim, A. Magnani, and S. Boyd. Optimal kernel selection in kernel fisher discriminant analysis. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 465–472, New York, NY, USA, 2006. ACM Press.

[23] C. Y. Lee and X. Yao. Evolutionary programming using mutations based on the levy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8(1):1–13, 2004.

[24] C. Liu and H. Wechsler. Enhanced fisher linear discriminant models for face recognition. In *Proc. the 14th International Conference on Pattern Recognition*, pages 17–20, 1998.

[25] C. A. Micchelli and M. Pontil. Learning the kernel function via regularization. *J. Mach. Learn. Res.*, 6:1099–1125, 2005.

[26] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Prentice Hall, 1994.

[27] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. Muller. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–48, 1999.

[28] S. Mika, G. Rätsch, and K. R. Müller. A mathematical programming approach to the kernel fisher algorithm. *Proc. Conf. Neural Information Processing Systems*, 13:591–597, 2001. T.K. Leen, T.G. Dietterich, and V. Tresp, eds.

[29] E. Polak. *Computational Methods in Optimization.* Academic Press, New York, 1971.

[30] J. Polzehl. Projection pursuit discriminant analysis. *Computational Statistics and Data Analysis*, 20:141–157, 1995.

[31] C. Posse. Projection pursuit discriminant analysis for two groups. *Computational Statistics*, 21(1):1–19, 1992.

[32] G. Potamianos and H. P. Graf. Linear discriminant analysis for speechreading. 1998.

[33] L. Prechelt. Some notes on neural learning algorithm benchmarking. *Neurocomputing*, 9(3):343–347, 1995.

[34] C. R. Rao. The utilization of multiple measurements in problems of biological classification (with discussion). *Journal of the Royal Statistical Society series B*, 10:159–203, 1948.

[35] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[36] M. Rohl, C. Weihs, and W. Theis. Direct minimization of error rates in multivariate classification. *Computational Statistics*, 17:29–46, 2002.

[37] C. Santa Cruz and J.-R. Dorronsoro. A nonlinear discriminant algorithm for feature extraction and data classification. *IEEE Transactions on Neural Networks*, 9(6):1370–1376, 1998.

[38] B. Schölkopf. *Support Vector Learning*. Munich, Germany: Oldenbourg-Verlag, 1997.

[39] J. R. Schott. Dimensionality reduction in quadratic discriminant analysis. *Computational Statistics and Data Analysis*, 16(2):161–174, 1993.

[40] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures.* CRC Press, third edition, 2004.

[41] A. Sierra. High order fisher's discriminants. *Pattern Recognition*, 35(1):1291–1302, 2002.

[42] A. Sierra and A. Echeverría. Skipping fisher's criterion. In *Proceedings of the First Iberian Conference on Pattern Recognition and Image Analysis, Mallorca, Spain, IbPRIA 2003, LNCS 2652*, pages 962–969, 2003.

[43] A. Sierra and A. Echeverría. Neural networks trained by distance to means. *WSEAS Transactions on Information Science and Applications*, 2(9):1446–1453, 2005.

[44] A. Sierra and A. Echeverría. Perceptrons without output codes. In *Proceedings of the 5th WSEAS International Conference on Simulation,*

Modelling and Optimization, Corfu, Greece, volume 10, pages 331–336, 2005.

[45] A. Sierra and A. Echeverría. Evolutionary discriminant analysis. *IEEE Transactions on Evolutionary Computation*, 10(1):81–92, 2006.

[46] A. Sierra and A. Echeverría. The polar evolution strategy. In *Proceedings of the IEEE Congress On Evolutionary Computation. Vancouver, BC, Canada*, 2006.

[47] A. Sierra, J. A. Macías, and F. Corbacho. Evolution of functional link networks. *IEEE Trasnsactions on Evolutionary Computation*, 5(1):54–65, 2001.

[48] L. Tjen-Sien, L. Wei-Yin, and Yu-Shan S. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classfication algorithms. *Machine Learning*, 40:203–228, 2000.

[49] Z. Tu and Y. Lu. A robust stochastic genetic algorithm (stga) for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 8(5):456–470, 2004.

[50] S. M. Weiss and I. Kakpouleas. An empirical comparison of pattern recognition, neural nets and machine learning classification methods. In *Readings in Machine Learning. J. W. Shavlik, T. G. Dietterich, eds., San Francisco: Morgan Kaufmann*, 1990.

[51] W. H. Wolberg and O. L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences, U.S.A.*, 87:9193–9196, 1990.

[52] J. Yang, A. Frangi, J. Y. Yang, D. Zhang, and Z. Jin. Kpca plus lda: A complete kernel fisher discriminant framework for feature extraction and recognition. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 27:230–244, 1989.

[53] W. Zhao. Discriminant component analysis for face recognition. *International Conference on Pattern Recognition*, 2, 2000.