UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería Técnica de Telecomunciación

Sistemas de Telecomunicación

PROYECTO FIN DE CARRERA

# DESARROLLO Y EVALUACIÓN DE MÉTODOS CONSTRUCTIVOS BASADOS EN EL DISCRIMINANTE LINEAL DE FISHER

Autor: Francisco Javier Guzmán Rivas

Tutor: Manuel Ortega Moral

Madrid, Julio de 2006

# PROYECTO FIN DE CARRERA

Departamento de Teoría de la Señal y Comunicaciones

Universidad Carlos III de Madrid

**Título:** Desarrollo y evaluación de métodos constructivos basados en el Discriminante Lineal de Fisher
**Autor:** D. Francisco Javier Guzmán Rivas
**Tutor:** D. Manuel Ortega Moral

La defensa del presente proyecto fin de carrera se realizó el día 18 de julio de 2006 siendo calificada por el siguiente tribunal:

**Presidente:**

**Vocal:**

**Secretario:**

Habiendo obtenido la siguiente calificación

| **Presidente** | **Vocal** | **Secretario** |
|---|---|---|

*Este proyecto lo dedico a mi familia, a toda la gente que coincidió conmigo en la universidad y a mi tutor Manuel por todo su apoyo.*

# Índice general

**1. Introducción** **1**

1.1. Motivación . . . . . 1

1.2. Objetivos . . . . . 2

1.3. Organización del trabajo . . . . . 2

**2. Extracción de conocimiento** **3**

2.1. Introducción . . . . . 4

2.2. Proceso de extracción del conocimiento . . . . . 4

2.3. Problemas en la extracción de patrones . . . . . 7

**3. Introducción a las Redes Neuronales Artificiales** **9**

3.1. Introducción: Neurona biológica y artificial . . . . . 10

3.1.1. Aprendizaje en las redes neuronales artificiales . . . . . 14

3.1.2. Aplicaciones de las redes neuronales artificiales . . . . . 15

3.2. Redes Monocapa . . . . . 17

3.2.1. Discriminantes lineales . . . . . 17

3.2.2. Discriminación logística . . . . . 18

3.2.3. Técnicas de aprendizaje . . . . . 19

3.2.4. Perceptrón . . . . . 20

3.2.5. Adaline . . . . . 22
3.3. Discriminante Lineal de Fisher . . . . . 24
3.4. Redes Multicapa . . . . . 27
3.4.1. Retropropagación . . . . . 28
3.5. Funciones de Base Radial . . . . . 33
3.5.1. Topología y entrenamiento de las RBF . . . . . 33

**4. Estado del Arte** **37**
4.1. Máquinas de Vector Soporte . . . . . 38
4.1.1. Generalización a problemas no lineales . . . . . 40
4.2. Sistemas de múltiples redes . . . . . 41
4.2.1. Comités . . . . . 41
4.2.2. Sistemas Modulares . . . . . 45

**5. Arquitecturas Propuestas** **47**
5.1. Arquitectura_1 . . . . . 48
5.1.1. Modificación del umbral de clasificación . . . . . 50
5.2. Arquitectura_2 . . . . . 53
5.2.1. Fisher . . . . . 54
5.2.2. Proyección . . . . . 54
5.2.3. PCA . . . . . 59
5.2.4. Bloques opcionales . . . . . 63
5.2.5. Cálculo de la salida global . . . . . 63
5.3. Comités . . . . . 64
5.4. Fisher-Modular . . . . . 67
5.4.1. Obtención del gate . . . . . 68
5.4.2. Clasificación de las muestras . . . . . 69

**6. Resultados** **71**
6.1. Descripción de problemas reales . . . . . 72

6.2. Arquitectura_1 . . . . . 73
6.2.1. Resultados . . . . . 74
6.2.2. Tiempo de cómputo . . . . . 75
6.2.3. Conclusiones . . . . . 79
6.3. Arquitectura_2 . . . . . 81
6.3.1. Resultados . . . . . 82
6.3.2. Tiempo de cómputo . . . . . 85
6.3.3. Conclusiones . . . . . 86
6.4. Comités . . . . . 87
6.4.1. Resultados y Conclusiones Bagging . . . . . 87
6.4.2. Resultados y Conclusiones AdaFisher . . . . . 87
6.5. Fisher Modular . . . . . 91
6.5.1. Resultados . . . . . 91
6.5.2. Tiempo de Cómputo . . . . . 93
6.5.3. Conclusiones . . . . . 93
6.6. Resultados y Conclusiones finales . . . . . 95

**A. Duración y presupuesto del proyecto** **101**

**Bibliografía** **103**

# Índice de figuras

3.1. Estructura de una neurona biológica . . . . . 10
3.2. Modelo de neurona artificial . . . . . 11
3.3. Funciones de Activación . . . . . 12
3.4. Modelo de red monocapa . . . . . 13
3.5. Modelo de red multicapa . . . . . 14
3.6. Diagrama de una red neuronal monocapa . . . . . 18
3.7. Estructura de un Perceptrón . . . . . 21
3.8. Frontera de decisión lineal, H . . . . . 21
3.9. Estructura típica de un SLP . . . . . 24
3.10. Diferentes direcciones de proyección . . . . . 25
3.11. Diagrama de una red neuronal multicapa . . . . . 28
3.12. Regiones de Decisión para redes monocapa y multicapa . . . . . 29
3.13. Diagrama de una red neuronal multicapa . . . . . 32
3.14. Ejemplo de red RBF . . . . . 34

4.1. Ejemplo de SVM . . . . . 39
4.2. Problema no separable linealmente . . . . . 39

5.1. Esquema Arquitectura_1 . . . . . 48
5.2. Ejemplo de cálculo del *DLF* . . . . . 49

5.3. Función de densidad estimada mediante ventanas de parzen . . . . . 53
5.4. Esquema Arquitectura_2 - Método Constructivo . . . . . 54
5.5. Esquema final Arquitectura_2 . . . . . 64
5.6. Esquema entrenamiento Adaboost . . . . . 65
5.7. Esquema evaluación Adaboost . . . . . 65
5.8. Esquema Bagging . . . . . 66
5.9. Esquema Fisher-Modular . . . . . 67

6.1. Tiempo de cómputo arq1 - y0, ejemplo . . . . . 76
6.2. Tiempo de cómputo arq1 - y0', ejemplo . . . . . 76
6.3. Tiempo de cómputo arq1 - y0", ejemplo . . . . . 77
6.4. Tiempo de cómputo SLP, ejemplo . . . . . 77
6.5. Esquema Arquitectura_2 - Método Constructivo . . . . . 81
6.6. Esquema completo Arquitectura_2 . . . . . 82
6.7. Evolución de las medias aplicando énfasis . . . . . 89
6.8. ZOOM Evolución de las medias aplicando énfasis . . . . . 90
6.9. Ejemplo aprendedores débiles . . . . . 90
6.10. Frontera de decisión - problema Kwok . . . . . 92
6.11. Frontera de decisión - problema Ripley . . . . . 92

# Índice de tablas

6.1. Caracterísiticas de los conjuntos de datos *reales* . . . . . 72
6.2. Evaluación Arquitectura_1 - Problemas Sintéticos . . . . . 74
6.3. Evaluación Arquitectura_1 - Problemas *reales* . . . . . 75
6.4. Tiempo de cómputo - problema kwok . . . . . 78
6.5. Tiempo de cómputo - problema phoneme . . . . . 78
6.6. Tiempo de cómputo - problema ripley . . . . . 78
6.7. Tiempo de cómputo - problema spam . . . . . 79
6.8. Evaluación Arquitectura_2 - problema kwok . . . . . 82
6.9. Evaluación Arquitectura_2 - problema ripley . . . . . 83
6.10. Evaluación Arquitectura_2 - problema phoneme . . . . . 83
6.11. Evaluación Arquitectura_2 - problema spam . . . . . 84
6.12. Tiempo de entrenamiento - Arquitectura_2 . . . . . 85
6.13. Tiempo de cómputo - Arquitectura_2 . . . . . 86
6.14. Evaluación Fisher Modular - Problemas *reales* . . . . . 91
6.15. Tiempo de Entrenamiento - Fisher-Modular . . . . . 93
6.16. Tiempo de Evaluación - Fisher-Modular . . . . . 93
6.17. Resumen de resultados de evaluación . . . . . 95
6.18. Resumen Tiempos de cómputo - problema kwok . . . . . 96
6.19. Resumen Tiempos de cómputo - problema phoneme . . . . . 96

6.20. Resumen Tiempos de cómputo - problema ripley . . . . . 97
6.21. Resumen Tiempos de cómputo - problema spam . . . . . 97

A.1. Presupuesto del proyecto . . . . . 102

Capítulo 1

# Introducción

## 1.1. Motivación

Las técnicas de aprendizaje máquina han demostrado un buen rendimiento en campos como la decisión o clasificación. Si bien a lo largo del tiempo la complejidad de las máquinas ha ido aumentando, también lo ha hecho su eficacia a la hora de dar una solución a los problemas planteados. Lo que hay que pagar a cambio de esta mejora en eficiencia y el aumento de complejidad es un aumento del tiempo de cómputo, principalmente en el periodo de aprendizaje de la máquina. Por otro lado, los clasificadores lineales siguen siendo de utilidad en aplicaciones donde prevalece la obtención de una solución rápida frente a una pequeña mejora en la tasa de acierto. Existirá, generalmente, un compromiso entre complejidad y coste computacional o, de manera análoga, entre tasa de acierto y rapidez en el aprendizaje.

En este proyecto se propone la construcción de clasificadores o conjuntos de ellos basados en el discriminante lineal de Fisher (DLF). Es uno de los discriminantes lineales más conocidos por sus buenas propiedades de generalización y robustez frente a outlayers, pero sobre todo, debido a que dicho discriminante se calcula de forma analítica y es óptimo en el caso de que las distribuciones de las clases sean gaussianas.

Dado que habrá problemas no lineales que requieran clasificadores más potentes se han implementado varios métodos de crecimiento para formar sistemas de múltiples redes, teniendo especial cuidado de no degradar la eficiencia computacional del DLF.

Todos los algoritmos se han probado con conjuntos de datos aportados por el departamento de Teoría de la Señal y Comunicaciones y con algunos conjuntos de datos sintéticos generados para destacar aspectos del algoritmo, también se ha comparado con algoritmos de similares características a los implementados.

## 1.2. Objetivos

Los objetivos que cubre este proyecto se enumeran a continuación:

- Desarrollo de diversas arquitecturas basadas en el Discriminante Lineal de Fisher
- Evaluación de prestaciones y de tiempo de cómputo de cada arquitectura. Comparativa con arquitecturas de complejidad similar.
- Valorar la idoneidad de utilizar el Discriminante Lineal de Fisher para la construcción de sistemas de múltiples redes.

## 1.3. Organización del trabajo

En esta sección se resumen los apartados que se encuentran en este trabajo. En primer lugar se presenta una introducción a los procesos de extracción de conocimiento posteriormente nos centramos en el ámbito de las redes neuronales como técnica de extracción de conocimiento desde sus formas básicas hasta el estado del arte, del cual forman parte las máquinas de vectores soporte y los sistemas de múltiples redes. A continuación comienza el desarrollo principal de este proyecto mostrando las arquitecturas desarrolladas a lo largo del mismo para acabar en el último capítulo con los resultados de la experimentación y las conclusiones obtenidas.

Capítulo 2

# Extracción de conocimiento

Las empresas, organizaciones, instituciones y particulares toman decisiones basándose en información de experiencias pasadas, de la situación actual, etc..., es decir, de información almacenada. Esta información generalmente va a estar almacenada de muy diversas formas. Por lo tanto, parece lógico realizar un análisis de esta información con el objetivo de extraer la información realmente útil para mejorar el proceso de toma de decisiones en el ámbito en el cual se han extraído los datos.

En muchas ocasiones este análisis se ha realizado de forma manual por un especialista, por ejemplo un médico, que realizaba un análisis de los datos para obtener pautas de enfermedades infecciosas, etc... Pero esta manera de actuar es costosa y lenta y no se puede llevar a cabo cuando la cantidad de datos crece de forma rápida. Es conveniente por tanto generar un proceso automático o semi-automático (asistido) que extraiga el conocimiento que nos permita tomar mejores decisiones. ¿Cómo califico automáticamente los mensajes de correo entre más o menos probables de ser correo basura (*spam*)?

## 2.1. Introducción

Generalmente el proceso de extracción del conocimiento sigue una secuencia iterativa de 4 etapas o fases que son: *preparación de datos*; *fase de extracción del conocimiento*; *evaluación* y *difusión y uso*, las cuales describimos a continuación.

## 2.2. Proceso de extracción del conocimiento

### 1. Preparación de datos

Para iniciar el proceso de extracción del conocimiento debemos recopilar los datos a analizar, estos pueden provenir de muy diferentes fuentes tanto internas a la organización como externas. Toda esta recopilación puede ser muy complicada, más cuando cada fuente (de datos) puede utilizar distintas formas de almacenar los datos, por lo que será necesario una integración o unificación de los datos recopilados.

La relevancia del conocimiento extraído no sólo depende de la técnica utilizada sino también de la calidad de los datos recopilados. Es necesario también realizar una selección de los datos más adecuados. Podemos encontrarnos con datos que presenten un comportamiento muy diferente al general (*outliers*) o datos incompletos o parciales. Estos y otros problemas muestran la necesidad de realizar una limpieza de los datos.

Sabemos que no todos los atributos van a ser igual de relevantes, seleccionaremos generalmente los más relevantes, ya que, la información redundante puede perjudicar la construcción del modelo.

Dependiendo de la técnica utilizada o del problema a resolver puede ser necesaria una transformación de los datos. Por ejemplo, una transformación de un atributo nominal en un atributo numérico si este es el tipo de atributos que acepta nuestra técnica.

En el caso de este proyecto esta primera etapa o fase de preparación de datos no ha sido necesaria llevarla a cabo, ya que, los datos han sido aportados por el departamento. Estos datos ya han sido utilizados en otros proyectos de extracción de patrones y son utilizados por la comunidad de aprendizaje automático. Este proyecto no intenta mejorar

la calidad de los mismos.

### 2. Extracción de conocimiento

Fase principal del proceso de extracción del conocimiento que da nombre a todo el proceso. Esta fase pretende extraer el conocimiento a partir de los datos recopilados en la fase anterior para que pueda ser utilizado por el usuario. Aquí se propone el modelo que va a descubrir patrones y relaciones entre los datos con el objetivo de poder hacer predicciones, entender la situación actual o explicar situaciones pasadas.

Debemos determinar la tarea a realizar con los datos. Entre estas tareas tenemos las *tareas predictivas*, que pretenden estimar valores futuros o desconocidos de variables de interés. Las *tareas descriptivas* identifican patrones que explican o resumen la información contenida en los datos.

Entre las tareas descriptivas se encuentra el **agrupamiento** (o *clustering*), las **reglas de asociación** y entre las tareas predictivas tenemos la **clasificación** (*o discriminación*) y la **regresión**.

Una vez definida la tarea a realizar, que en el caso de este proyecto es una clasificación de los datos, también debemos definir la técnica utilizada para abordar el problema. Entre las técnicas más utilizadas para clasificación tenemos los **árboles de decisión** y las **redes neuronales**.(Clasificación obtenida de [Hernández Orallo *et al.*, 2004, página 12])

En una tarea de clasificación cada instancia del conjunto de datos pertenece a una clase que se indica mediante una variable de clase. El resto de variables se utilizan para estimar la clase. El objetivo en el diseño de un clasificador es predecir la clase a la que pertenecen los nuevos datos de los cuales se desconoce su clase.

Para la tarea de clasificación en este proyecto recurrimos a utilizar *redes neuronales*, esta técnica presenta una buena capacidad para modelar problemas complejos y muchas alternativas para su desarrollo.

En la fase de construcción del modelo a utilizar se observa el carácter iterativo de todo el proceso. En el caso de las redes neuronales no hay una arquitectura general que

resuelva cualquier problema de manera satisfactoria, por lo que, una vez construido un primer modelo y en base a los resultados obtenidos podemos optar por modificar los parámetros del modelo o por derivar nuevas arquitecturas o modelos más complejos. Esto queda patente en este proyecto al presentar diferentes arquitecturas que parten de una arquitectura simple común, con el objetivo de encontrar un modelo que presente buenas características y prestaciones.

Una vez construido un modelo debemos llevar acabo una evaluación del mismo.

### 3. Etapa de evaluación

Medir la calidad del modelo obtenido en función de los patrones que descubre no es algo nada fácil y depende mucho del contexto del problema, de la tarea a realizar y de la técnica utilizada. Generalmente se espera de los patrones descubiertos tres cualidades: precisión, comprensibilidad y utilidad.

En el contexto de nuestro proyecto, es decir, el contexto de la clasificación de muestras evaluaremos la calidad del modelo en función de su precisión a la hora de predecir la clasificación de nuevos datos. En las tareas de clasificación para entrenar y probar el modelo se parten los datos en dos conjuntos: conjunto de entrenamiento y conjunto de test o prueba. La precisión de nuestro modelo se calcula como el número de muestras del conjunto de test clasificadas correctamente dividido por el total de muestras que componen el conjunto de prueba.

### 4. Etapa de difusión y uso

Una vez creado el modelo y medidas sus capacidades, este podría ser incorporado a aplicaciones automáticas como los filtros *anti-spam* o podría asistir al ser humano en la toma de decisiones por ejemplo al evaluar los correos electrónicos de ser susceptible de ser *spam*. Otro tipo de aplicaciones podrían ser el reconocimiento de caracteres o el reconocimiento de voz, así como muchos otros.

Difundiríamos la aplicación pero cuidando de evolucionarla, ya que la naturaleza variante de algunos problemas hacen que de vez en cuando el modelo deba ser re-calculado,

re-entrenado e incluso re-construido.

Esta etapa no va a ser cubierta en este proyecto aunque si podría cubrirse en otros proyectos de extensión o ampliación del mismo si los resultados son satisfactorios.

## 2.3. Problemas en la extracción de patrones

El ser humano tiene una capacidad enorme de extraer patrones o pautas de nuestro alrededor. Las técnicas de extracción de patrones y reconocimiento de formas intentan emular estas capacidades. Un primer problema de estas técnicas es que son computacionalmente muy costosas y más cuanto más interesantes, comprensibles y precisos queremos que sean los patrones extraídos. Para nosotros es fácil reconocer la voz de una persona conocida o reconocerla en una foto; pero estos problemas son muy complejos de resolver de forma automática.

Por ejemplo, una técnica de clasificación va a emplear más tiempo generalmente que una técnica de extracción de reglas de asociación. A su vez, no requiere el mismo coste computacional realizar una clasificación mediante una red monocapa que utilizando una red multicapa.

De entre las distintas tareas con las que se puede extraer conocimiento a partir de datos tenemos: la regresión, el agrupamiento, las reglas de asociación, etc. Este proyecto pretende resolver problemas de clasificación. Todas estas tareas requieren de unos métodos, técnicas o algoritmos para su resolución. Lógicamente, una tarea puede ser resuelta por diferentes métodos, pero también, un mismo método puede resolver distintas tareas.

La diferencia fundamental entre los diversos métodos es su capacidad para extraer conocimiento de los datos. De ahí, que unos métodos funcionen mejor que otros para un problema concreto; igual que, un mismo método no tiene porque resolver distintos problemas de forma correcta. Por lo tanto, si un método falla a la hora de resolver nuestro problema particular podremos utilizar otro método.

Capítulo 3

# Introducción a las Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) puede verse como un grafo dirigido con muchos nodos (elementos de proceso) y arcos entre ellos (interconexiones). Cada uno de estos nodos va a funcionar de manera independiente de los demás. Las redes de neuronas aparecen con el objetivo de imitar las capacidades que presenta el cerebro humano, por ejemplo a la hora de reconocer rostros o formas. El cerebro es una estructura muy compleja con un funcionamiento no-lineal y en el que se realizan múltiples operaciones en paralelo.

Las RNA son máquinas diseñadas de manera que su comportamiento se asemeje al comportamiento del cerebro a la hora de abordar diferentes problemas.

A lo largo de esta sección presentamos una introducción a las redes neuronales artificiales presentando su similitud con las neuronas biológicas, las distintas estructuras de las RNA y los métodos de aprendizaje de una manera general; para acabar nombrando las distintas aplicaciones de estas redes en la actualidad.

## 3.1. Introducción: Neurona biológica y artificial

El cerebro está formado por un conjunto enorme de unidades interconectadas entre sí, las *neuronas*. La Figura 3.1 muestra la estructura de una neurona. Las neuronas reciben información procedente de otras neuronas o de receptores externos a través de las sinapsis de sus dendritas. Cada sinapsis representa la unión entre un par de neuronas. La información que llega a través de estas conexiones se propaga hasta el cuerpo de la célula. En el cuerpo de la neurona se suceden unos procesos químicos, que modelaremos como una *función de activación*, y provocarán que la neurona siga propagando la información a otra u otras neuronas a través de su axón o cese la propagación, dependiendo de la información que le llegue de las neuronas adyacentes y de su estado de activación en ese momento.

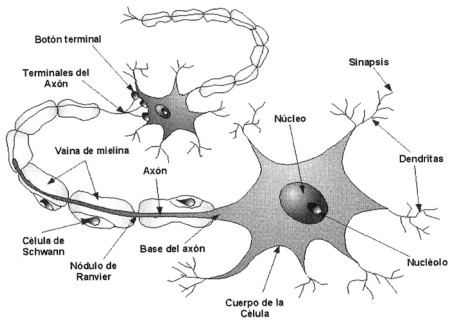



Figura 3.1: Estructura de una neurona biológica

Y ésta es la forma en la que la información viaja por una red de neuronas. Pero se tiene que tener en cuenta que las sinapsis tienen diferente rendimiento y que este rendimiento varía a lo largo del tiempo. También en el cerebro se producen nuevas conexiones entre neuronas o se suprimen conexiones existentes.

Una neurona artificial constituye una unidad de procesado de información y aparece con el objetivo de imitar el comportamiento de las neuronas biológicas. Las redes neuronales artificiales están constituidas por estas unidades de proceso, cuyo diagrama fun-

cional se muestra en la siguiente figura[1]

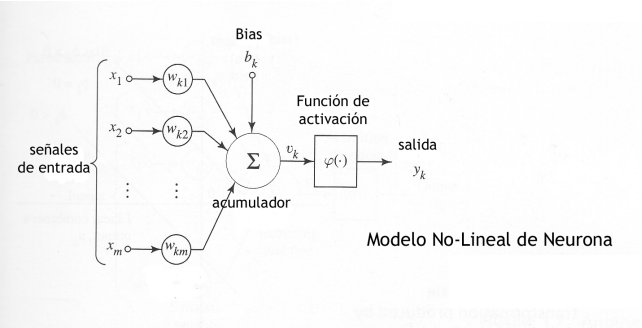



Figura 3.2: Modelo de neurona artificial

En este modelo de neurona artificial se pueden distinguir tres partes:

- ***conexiones*** caracterizadas por un peso $w_{kj}$, donde $k$ se corresponde con la neurona a la que se conecta la variable de entrada $j$ o otra neurona $j$, vendrían a simular las conexiones sinápticas.

- ***acumulador*** realiza una combinación lineal de las entradas

$$\sum_{j=i}^{m} w_{kj}x_j + b_k \tag{3.1}$$

- ***función de activación*** $\varphi(v)$, intenta simular junto al acumulador el proceso que tiene lugar en el cuerpo de la neurona.

El comportamiento de la neurona $k$-ésima queda descrito por la siguiente expresión

$$y_k = \varphi(\sum_{j=i}^{m} w_{kj}x_j + b_k) \tag{3.2}$$

donde $x_1, \cdots, x_m$ son las variables de entrada; $w_{k1}, \cdots, w_{km}$ los pesos sinápticos de cada conexión a la neurona $k$; $b_k$ es el *bias* e $y_k$ la salida de la unidad de proceso.

[1]Imagen obtenida de [Haykin, 1999]

**Funciones de activación** La función de activación determina la salida de cada neurona en función de $v(3.1)$ que representa la contribución de las neuronas adyacentes o de las variables de entrada. La figura 3.3[2] muestra las funciones de activación más populares.

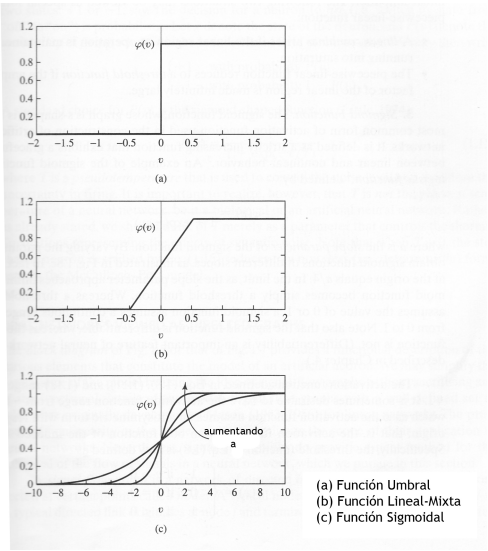



Figura 3.3: Funciones de Activación

Entre estas funciones de activación cabe destacar las funciones *sigmoide* la expresión de este tipo de funciones es

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{3.3}$$

Siendo $a$ el parámetro que controla la pendiente cuyo valor es generalmente igual a 1. Si aumentamos el valor de $a$ hasta infinito la función *sigmoide* tiende a la función umbral. Este tipo de funciones de activación son las más empleadas ya que, combinan un comportamiento lineal entorno a 0 con un comportamiento no-lineal. Otra ventaja de estas funciones es que son funciones derivables en función de los pesos. La función sigmoide genera valores en el entorno $[0, 1]$, pero también puede ser habitual presentar valores en el entorno $[-1, 1]$ por lo que la función de activación a utilizar sería la función tangente hiperbólica:

$$\varphi(v) = \frac{1 - \exp(-av)}{1 + \exp(-av)} = \tanh(\frac{av}{2}) \tag{3.4}$$

[2]Imagen obtenida de [Haykin, 1999]

la cual presenta las mismas características que (3.3).

Vemos que una única neurona es una unidad de proceso muy simple, por lo que, a semejanza del modelo biológico se constituyen estructuras más complejas agrupando neuronas en capas o niveles, con el objetivo de aumentar su capacidad de procesado de información. Las estructuras generales serían:

**Redes de una capa.** Constituidas por una única capa de neuronas de salida. Figura 3.4[3].

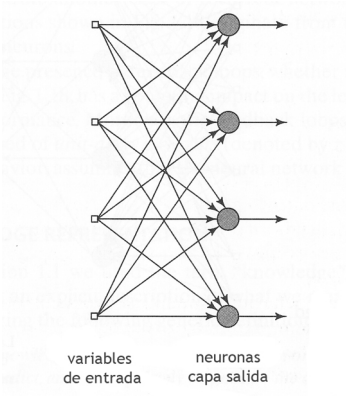



Figura 3.4: Modelo de red monocapa

**Redes multicapa.** Arquitecturas en las que al menos está presente una capa de neuronas ocultas, las cuales no son accesibles desde fuera de la red. Es decir, esta capa de neuronas ocultas no recibe información directamente del exterior a través de las variables de entrada ni tampoco generan información que salga directamente fuera de la red. Figura 3.5[4]

**Redes recurrentes.** Estructura que a diferencia de las redes *feedforward*, con únicamente conexiones hacia delante, presentan conexiones salida-entrada.

Para completar la definición de la red neuronal debemos definir

[3]Imagen obtenida de [Haykin, 1999]
[4]Imagen obtenida de [Haykin, 1999]

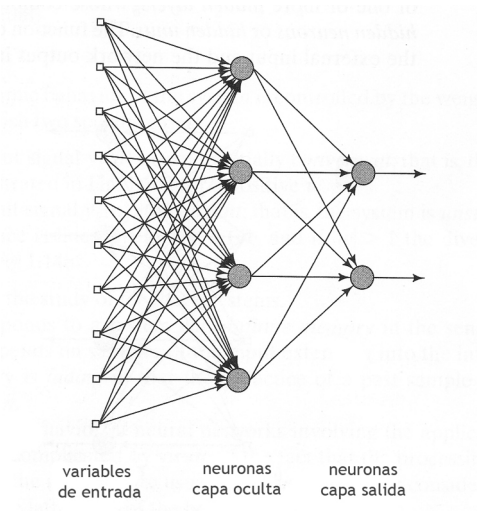



Figura 3.5: Modelo de red multicapa

**La regla de propagación.** Define cómo se propaga la activación a lo largo de la red.

**La regla de aprendizaje.** Define cómo se modifican los pesos de cada una de las neuronas que constituyen la red.

### 3.1.1. Aprendizaje en las redes neuronales artificiales

Las redes neuronales optimizan su comportamiento mediante un proceso de aprendizaje. En este proceso de aprendizaje se realiza un ajuste en los valores de los parámetros de la red, generalmente en el valor de los pesos de las conexiones de las neuronas, y con este aprendizaje se adquiere el conocimiento que queda almacenado en los valores de estos parámetros.

Según aparece en [Haykin, 1999, página 50] el aprendizaje de una red neuronal se define como

> *"Learning is a process by which the free parameters of a neural network are adapted through a process of simulation by the enviroment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place."*

Resumiendo, nos dice que el proceso de aprendizaje es una adaptación de los parámetros de la red a partir de los datos de un entorno. Y el tipo de aprendizaje determinará la manera en que se adaptan los valores de esos parámetros.

Las principales técnicas de aprendizaje en las redes neuronales son

**Aprendizaje supervisado.** Para realizar este aprendizaje se dispone de un conjunto de datos de entrada y su salida correspondiente (conjunto de entrenamiento), y el objetivo es que la red proporcione una salida lo más parecida a la deseada para cada uno de los datos. Para ello se ajustan los pesos de forma que la red genere una salida correcta cuando se presente a la red un patrón de entrada idéntico o similar. Este tipo de aprendizaje es útil en tareas de clasificación.

**Aprendizaje no supervisado.** En este caso a la red solo se le proporciona un conjunto de datos de entrada, no se proporciona la salida correspondiente, por lo que la red debe auto-organizarse en función de la estructura de los datos. Este tipo de aprendizaje es útil para tareas de agrupamiento y reducción de la dimensionalidad.

### 3.1.2. Aplicaciones de las redes neuronales artificiales

En lo que a la aplicación práctica de estas redes se refiere su principal ventaja radica en su capacidad de procesado paralelo, adaptativo y no-lineal. Además son estructuras muy expresivas, y una vez correctamente ajustadas obtienen precisiones muy altas. Entre sus inconvenientes tenemos la necesidad de presentar muchos ejemplos para el aprendizaje y la lentitud relativa en la construcción del modelo (etapa de aprendizaje) y, fundamentalmente, la incomprensibilidad del modelo que sugieren.

Las RNA son usadas en aplicaciones tan diversas como:

- Obtención de patrones de uso fraudulento de tarjetas de crédito.
- Análisis de riesgos en créditos.
- Detección de piezas dañadas.

- Diagnóstico de enfermedades.
- Detección de correo *spam*.
- Reconocimiento de voz.
- Reconocimiento de caracteres manuscritos.

Todos estos problemas y muchos otros se agrupan en distintas tareas más generales que intentan resolver las redes de neuronas y son:

- Clasificación
- Regresión
- Asociación

## 3.2. Redes Monocapa

En este apartado nos introducimos con más detalle en las funciones discriminantes cuyos parámetros van a ser calculados a partir de un conjunto de datos mediante un algoritmo de aprendizaje.

Empezaremos presentando las funciones discriminantes más simples que se constituyen como una combinación lineal de las variables de entrada, este tipo de funciones han sido ampliamente estudiadas.

Partiendo de estas funciones lineales se pueden construir generalizaciones que mejoren su rendimiento. Una de las generalizaciones consiste en aplicar una función no-lineal a la combinación lineal o bien realizar una transformación no-lineal a las variables de entrada y a partir de las variables transformadas realizar la combinación lineal.

Algunas de estas funciones discriminantes se pueden considerar como redes neuronales con una única capa de parámetros ajustables (Redes Monocapa).

### 3.2.1. Discriminantes lineales

En esta primera sección consideramos varias formas de discriminantes lineales y discutimos sus propiedades.

#### Problemas bi-clase

Centrándonos en los problemas de clasificación binaria (2-clases) una función discriminante lineal se puede escribir como:

$$y(\mathbf{x}) = \mathbf{w^T x} + w_0 \tag{3.5}$$

Donde el vector $\mathbf{w}$ $d$-dimensional representa el vector de parámetros ajustables y $w_0$ el *bias* o umbral.

Una función discriminante $y(\mathbf{x})$ asigna el vector $\mathbf{x}$ a la clase $\mathcal{C}_1$ si $y(\mathbf{x}) \geq y_0$ y a la clase $\mathcal{C}_2$ si $y(\mathbf{x}) < y_0$, siendo $y_0$ un umbral. Sabemos que la ecuación (3.5) es óptima para

el caso de distribuciones normales con iguales matrices de covarianzas, ya que en este caso la frontera de decisión óptima es una frontera lineal.

Se observa la similitud entre la forma de (3.5) con la de una red neuronal de una única capa si expresamos (3.1) en forma matricial obtenemos la misma ecuación $y(\mathbf{x}) = \mathbf{w^T x} + w_0$. Si definimos $\mathbf{w_e} = (w_0, \mathbf{w})$ y $\mathbf{x_e} = (1, \mathbf{x})$ podemos reescribir (3.5) como

$$y(\mathbf{x_e}) = \mathbf{w_e^T x_e} \tag{3.6}$$

Ahora podemos representar (3.6) en términos de un diagrama de red como se ve en la figura 3.6

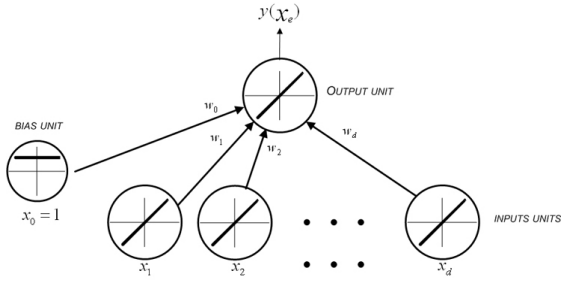



Figura 3.6: Diagrama de una red neuronal monocapa

### Problemas multiclase

Las funciones discriminantes se pueden extender en el caso de contar con $c$ clases, usando una función discriminante $y_k(\mathbf{x})$ para cada clase $\mathcal{C}_k$ de la forma

$$y_k(\mathbf{x}) = \mathbf{w_k^T x} + w_{k0} \tag{3.7}$$

En este tipo de problemas el vector $\mathbf{x}$ será asignado a la clase $\mathcal{C}_k$ si $y_k(\mathbf{x}) > y_j(\mathbf{x})$ para todo $j \neq k$.

## 3.2.2. Discriminación logística

Podemos generalizar las funciones discriminantes lineales utilizando una función no-lineal $g(\cdot)$, en la función discriminante para el problema bi-clase tendremos

$$y = g(\mathbf{w^T x} + w_0) \tag{3.8}$$

A $g(\cdot)$ la denominamos *función de activación* y normalmente va a ser una función monótona. Como consecuencia de ser una función monótona la frontera de decisión de la función discriminante seguirá siendo lineal. Como ya vimos las funciones de este tipo más utilizadas actualmente en las redes neuronales son las funciones de tipo *sigmoide*(3.3) y (3.4) ya que combinan un comportamiento lineal y no-lineal y son funciones derivables, esta última característica va a jugar un papel muy importante para su utilización en las redes neuronales multicapa.

### 3.2.3. Técnicas de aprendizaje

Hasta ahora hemos descrito la forma que presentan la funciones discriminantes pero no cómo calcular los coeficientes del vector $\mathbf{w}$. Como es lógico buscaremos unos valores para los coeficientes que minimicen una función de error. A continuación presentamos el algoritmo de minimización más usado en RNA.

#### Descenso por gradiente

La función de error $E = E(\mathbf{w})$ va a ser una función dependiente del valor de los pesos, si además es derivable en función de los mismos podemos utilizar el algoritmo de minimización de descenso por gradiente, el cual desarrollamos a continuación.

Comenzaríamos eligiendo de manera aleatoria un vector de pesos $\mathbf{w}$ y para actualizar el valor de estos pesos nos moveríamos en la dirección de máximo descenso en la función de error que se corresponde con $-\nabla_w E$, en busca del mínimo de la función de error $E(\mathbf{w})$. Iterando este algoritmo vamos generando vectores de pesos $\mathbf{w}^\tau$ calculados según

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau - \eta \frac{\partial E_{w_\tau}}{\partial \mathbf{w}} \tag{3.9}$$

donde $\eta$ representa cuanto nos movemos en la dirección de máximo descenso en cada iteración. Bajo ciertas condiciones la secuencia de vectores de pesos converge en un punto que minimiza $E$.

Esta regla de aprendizaje puede ser utilizada igualmente de manera secuencial, es decir, se calcula el gradiente de la función de error sobre la observación introducida en

la red y se van presentando todas las observaciones de manera secuencial. Sabiendo que $E$ se puede expresar como la suma del error para cada muestra

$$E(\mathbf{w}) = \sum_n E^n(\mathbf{w}) \tag{3.10}$$

la expresión secuencial del algoritmo esta definida por la siguiente expresión

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \frac{\partial E^n_{w_\tau}}{\partial \mathbf{w}} \tag{3.11}$$

### 3.2.4. Perceptrón

El Perceptrón es la forma más simplificada de red neuronal que puede ser empleada en problemas de clasificación. Básicamente, la estructura de un perceptrón consiste en una única neurona con unos parámetros ajustables (pesos) y un *bias*, y una función de activación no-lineal como es la función $sign(v)$, es idéntica a la función umbral que mostraba la figura 3.3 desplazada en el eje $y$. La Figura 3.7 muestra la estructura típica de un Perceptrón.

El Perceptrón fue investigado inicialmente por Rosenblatt en 1962 [Rosenblatt, 1962] además propuso un primer método para el ajuste de los parámetros de la red.

La Figura 3.8 muestra la frontera de decisión que genera este tipo de red y que viene dada por (3.12), es decir, una frontera lineal

$$g(\mathbf{x}) = sign(\mathbf{w}_e^T \mathbf{x}_e) = 0 \tag{3.12}$$

La Regla de aprendizaje del Perceptrón es una regla que intenta minimizar los errores de clasificación que comete la red modificando los valores de los pesos de manera secuencial. Se van a ir presentando de manera secuencial cada uno de los ejemplos de los que se compone el conjunto de datos de entrenamiento.

Tenemos el vector de entrada $\mathbf{x}_e^{(k)}$ de $(m+1)$-dimensiones; $\mathbf{w_e}^{(k)}$ es el vector de pesos de $(m+1)$-dimensiones. $y_k = sign(v^{(k)})$ es la salida actual cuyos valores pueden ser $\{+1, -1\}$ con $v = \mathbf{w_e}^{(k)T}\mathbf{x_e}^{(k)}$.

La regla de aprendizaje en su forma secuencial viene dada por:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \frac{\alpha}{2}[d^{(k)} - y^{(k)}]\mathbf{x}^{(k)} \tag{3.13}$$

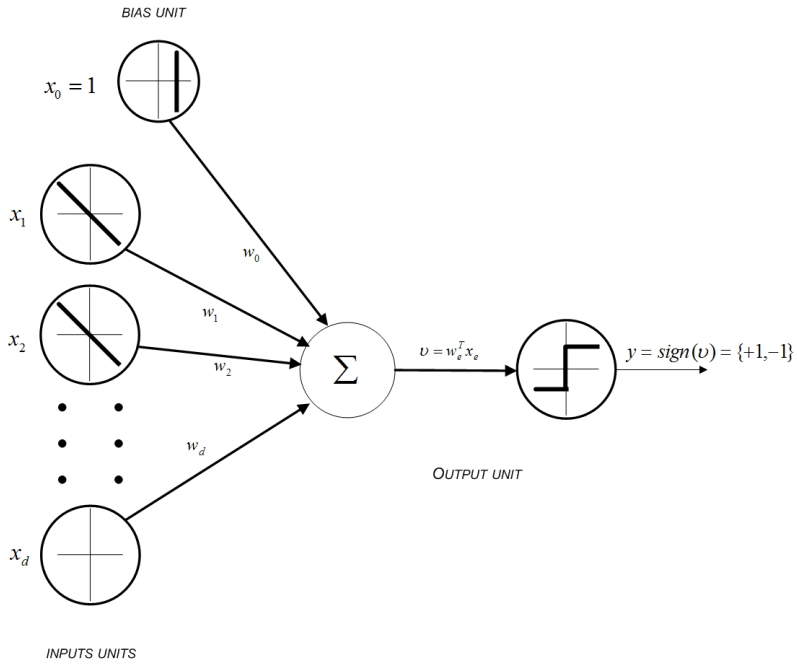



Figura 3.7: Estructura de un Perceptrón

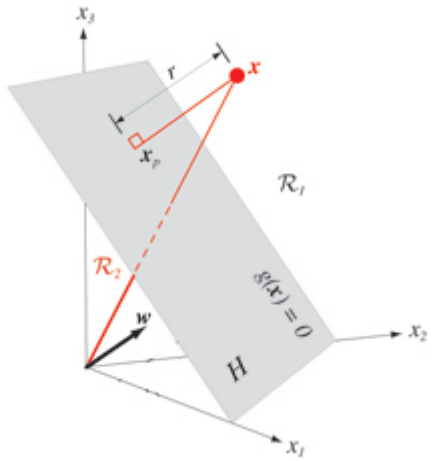



Figura 3.8: Frontera de decisión lineal, H

Si al presentar la muestra $\mathbf{x}^{(k)}$ tenemos que

$$d^{(k)} = y^{(k)}$$

la red clasifica correctamente ese ejemplo y el valor de los pesos no se actualiza. Sin embargo:

$$si\ d^{(k)} = -1, y^{(k)} = 1 \Rightarrow \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha \mathbf{x}^{(k)}$$

$$si\ d^{(k)} = 1, y^{(k)} = -1 \Rightarrow \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha \mathbf{x}^{(k)}$$

con lo que se intenta corregir el error cometido al intentar clasificar ese ejemplo.

Según esta regla el algoritmo se detiene siempre que los ejemplos de entrenamiento queden todos clasificados correctamente mediante una frontera del tipo 3.12, llegados a este punto el valor de los pesos no se volverá a modificar. El principal problema de esta regla es que solo se detiene en aquellos problemas que son linealmente separables. Pero no todos los problemas son linealmente separables, y esto es un claro inconveniente. Para solventar este problema presentamos la siguiente estructura en la cual al tener un valor continuo a la salida el valor de los pesos se modifica en función del error cometido; y esto permite una convergencia al menos a un mínimo local.

### 3.2.5. Adaline

Fue concebida por Widrow y sus colaboradores y su estructura es idéntica a la del Perceptrón (3.7), pero en este caso el entrenamiento se realiza mediante el algoritmo *Widrow-Hoff* aplicado a la entrada del decisor $(d - v)^2$. Con lo que la función de actualización de los pesos pasa a ser:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \alpha[d^{(\tau)} - v^{(\tau)}]\mathbf{x}^{(\tau)} \tag{3.14}$$

con $v^{(\tau)} = \mathbf{w}^{(\tau)\mathbf{T}}\mathbf{x}^{(\tau)}$ la entrada del decisor.

Si además sustituimos la decisión dura (función de activación *sign*) por una aproximación derivable como puede ser la tangente hiperbólica (3.4), tendríamos una decisión blanda.

Con lo que se puede llevar a acabo una minimización del error cuadrático mediante un algoritmo de descenso por gradiente 3.2.3. La regla a iterar se corresponde con (3.11) pero en este caso el valor del error para la muestra $n$ es

$$E^n = E^n(\mathbf{w}) = \frac{1}{2}(d^{(n)} - y^{(n)})^2 \tag{3.15}$$

con $y^{(n)}$ la salida de la red cuando tenemos a la entrada $\mathbf{x^{(n)}}$.

Desarrollando la derivada del error en función de los pesos y aplicando la regla de la cadena tenemos

$$\frac{\partial E^\tau}{\partial \mathbf{w}} = -(d^{(\tau)} - y^{(\tau)})\frac{\partial y^{(\tau)}}{\partial \mathbf{w}} \tag{3.16}$$

utilizando como función de activación la tangente hiperbólica sabemos que

$$\frac{\partial \tanh(v^{(\tau)})}{\partial \mathbf{w}} = (1 - \tanh\left(v^{(\tau)}\right)^2)\mathbf{x}^{(\tau)} \tag{3.17}$$

sustituyendo estos valores en (3.11) tenemos

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \alpha(d^{(\tau)} - y^{(\tau)})(1 - \tanh\left(v^{(\tau)}\right)^2)\mathbf{x}^{(\tau)} \tag{3.18}$$

Esta regla se denomina LMS (Least Mean Square).

Asociando estructuras de este tipo que denominamos SLP (*Single Layer Perceptron*) se pueden crear los perceptrones multicapa, ya que permite que los parámetros de las capas ocultas sean ajustados gracias al uso de funciones de activación derivables. Se ha demostrado que estas estructuras con una capa intermedia de neuronas ocultas en las que se utilizan funciones de activación derivables dan lugar a aproximadores universales [Kolmogorov, 1957]. La Figura 3.9 muestra la estructura típica de un SLP en el que se utiliza como función de activación la tangente hiperbólica.

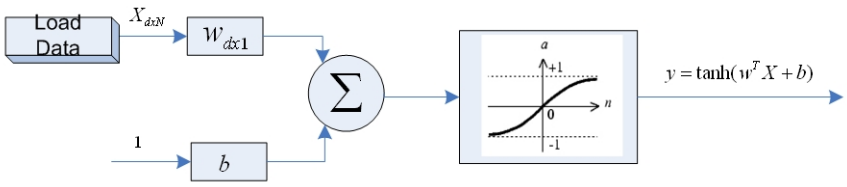



Figura 3.9: Estructura típica de un SLP

## 3.3. Discriminante Lineal de Fisher

El discriminante de *Fisher* no es un discriminante propiamente dicho pero nos ayuda a construirlos. Lo vamos a usar para realizar una proyección de los datos en un espacio de una dimensión, de la forma:

$$y = \mathbf{w^T x} \tag{3.19}$$

Donde $\mathbf{w}$ representa el vector de parámetros ajustables. La proyección de los datos en una única dimensión puede tener como consecuencia una gran pérdida de información, por lo que va a ser fundamental una elección de $\mathbf{w}$ en la que la información de clasificación no se pierda, buscamos que la proyección de los datos maximice la separación entre clases que es la información que necesitamos para diseñar un clasificador. Esto es lo que intenta el *Discriminante Lineal de Fisher* (DLF).

Nos centraremos en un problema de clasificación binario, suponemos que contamos con un conjunto de $k$ ejemplos de $d$-dimensiones $\mathbf{x_1} \ldots \mathbf{x_k}$ de los cuales $k_1$ pertenecen a las clase $\mathcal{C}_1$ y $k_2$ pertenecen a las clase $\mathcal{C}_2$. En primer lugar calculamos el vector $d$-dimensional de medias de los ejemplos de cada clase, dado por:

$$\mathbf{m}_k = \frac{1}{n_k} \sum_{\mathbf{x} \in \mathcal{C}_k} \mathbf{x}, \tag{3.20}$$

y la media de los ejemplos proyectados esta dada por:

$$m_k = \frac{1}{n_k} \sum_{y \in \mathcal{Y}_k} y = \frac{1}{n_k} \sum_{\mathbf{x} \in \mathcal{C}_k} \mathbf{w^T x} = \mathbf{w^T m}_k \tag{3.21}$$

que representa simplemente la proyección del vector de medias $\mathbf{m_i}$.

Podríamos pensar en maximizar la distancia entre las proyecciones de los vectores de las medias, es decir, maximizar $|m_1 - m_2| = \left|\mathbf{w^T}\left(\mathbf{m}_1 - \mathbf{m}_2\right)\right|$ pero como vemos en la imagen 3.10[5] no es suficiente. Debemos contar con información sobre la dispersión de las clases en las distintas direcciones, con el objetivo de seleccionar una proyección que proporcione una mayor separación entre clases. Para ello, *Fisher* propone maximizar una función que representa una medida de separación entre la proyección de las medias de las clases normalizada por una medida de dispersión dentro de cada clase de datos a lo largo de la dirección $\mathbf{w}$.

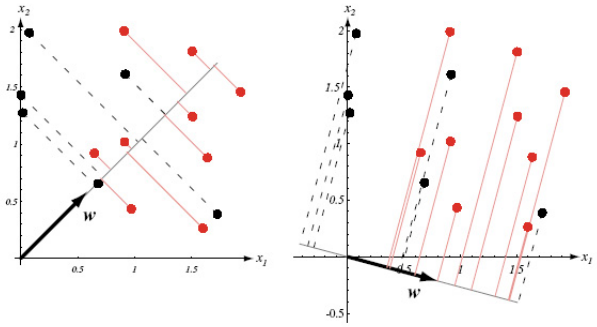



Figura 3.10: Diferentes direcciones de proyección. La de la derecha muestra una mayor separación entre los puntos rojos y negros proyectados.

La dispersión de la proyección de cada clase $\mathcal{C}_k$ va a estar representada con la matriz de covarianzas de cada clase dada por:

$$s_k^2 = \sum_{y \in \mathcal{Y}_k} (y - m_k)^2 \tag{3.22}$$

Si definimos la matriz de covarianzas para todo el conjunto de datos como $s_1^2 + s_2^2$ el

[5]Imagen obtenida de [Duda *et al.*, 2001]

criterio de *Fisher* queda de la siguiente manera:

$$J(\mathbf{w}) = \frac{|m_2 - m_1|^2}{s_1^2 + s_2^2} \tag{3.23}$$

Queremos obtener *J(·)* como una función de $\mathbf{w}$ por lo que definimos las matrices de covarianzas de los ejemplos $\mathbf{S}_i$ y $\mathbf{S_W}$.

$$\mathbf{S}_i = \sum_{\mathbf{x} \in \mathcal{C}_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T, \tag{3.24}$$

y

$$\mathbf{S_W} = \mathbf{S}_1 + \mathbf{S}_2 \tag{3.25}$$

$\mathbf{S_W}$ se denomina la matriz de covarianzas intraclase (*within-class scatter matrix*). Ahora podemos escribir:

$$s_1^2 + s_2^2 = \mathbf{w^T S_W w} \tag{3.26}$$

Si procedemos de igual manera con las medias de los datos proyectados tenemos

$$(m_1 - m_2) = \mathbf{w^T S_B w} \tag{3.27}$$

donde $\mathbf{S_B}$ se denomina la matriz de covarianzas interclase (*between-class scatter matrix*), dada por:

$$\mathbf{S_B} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \tag{3.28}$$

Reescribiendo el criterio de *Fisher* en función de $\mathbf{S_B}$ y $\mathbf{S_W}$.

$$J(\mathbf{w}) = \frac{\mathbf{w^T S_B w}}{\mathbf{w^T S_W w}} \tag{3.29}$$

Finalmente el valor de $\mathbf{w}$ que maximiza el criterio de Fisher *J(·)* es:

$$\mathbf{w} \propto \mathbf{S_W}^{-1} (\mathbf{m}_1 - \mathbf{m}_2) \tag{3.30}$$

En cuanto a la magnitud de $\mathbf{w}$ no nos afecta, lo que nos interesa es su dirección. Aunque es importante tener controlada la magnitud de $\mathbf{w}$, para la posterior utilización de los datos una vez proyectados. Esta magnitud influirá en los distintos métodos constructivos implementados por lo que una vez obtenido el valor óptimo de $\mathbf{w}$ se normalizará para tener $\|\mathbf{w}\| = 1$.

## 3.4. Redes Multicapa

En la sección anterior hemos estudiado estructuras de redes neuronales con una única capa de parámetros ajustables. Pero este tipo de redes tienen grandes limitaciones a la hora de aproximar relaciones no-lineales con los datos. Su utilización se limitaba a problemas que fueran linealmente separables, pero este tipo de problemas no son los más comunes.

En esta sección extenderemos las estructuras anteriores intercalando capas ocultas de parámetros ajustables. En estas estructuras las funciones de activación pueden ser diferentes para cada capa, aunque generalmente se suelen utilizar funciones sigmoide para todas las capas.

El potencial de las redes multicapa se descubrió pronto, pero no así se descubrió un método apropiado para el aprendizaje. La regla de aprendizaje más conocida parece que fue descubierta varias veces de manera independiente, aunque fue popularizada por *Rumelbart*, *Hinton* y *Williams* (1986), bajo el nombre de *back-propagation* (propagación hacia atrás). Este método se apoya en la derivabilidad de las funciones de activación para calcular las derivadas del error respecto de los pesos de las distintas capas. Aunque el algoritmo back-propagation no obtenga óptimas soluciones para todo tipo de problemas permite observar las capacidades de estas redes de neuronas.

Sólo vamos a considerar redes *feed-forward*, es decir, redes en las que la salida está determinada por los valores a la entrada a la red, los valores de los pesos y la entrada de la red (la señal) se propaga en dirección de entrada a salida. Este tipo de redes recibe el nombre de *Multi-Layer Perceptrons* (MLP - Perceptrones Multicapa).

El funcionamiento general de estas redes es el siguiente: la señal se propaga de la entrada, a través de los pesos, hasta la capa oculta donde se aplica una función de activación. Esta activación se propaga a través de los pesos de la capa de salida donde se aplica la función de activación de salida. Se pueden interconectar un número cualquiera de capas ocultas, pero el tiempo de entrenamiento puede ser excesivo en estructuras con muchas capas. Además se sabe que a partir de una estructura con 2 capas de pesos (capa

oculta y capa de salida) se puede aproximar cualquier función siempre que se utilicen funciones de activación no-lineales y derivables en la capa oculta [Kolmogorov, 1957].

La figura 3.11[6] muestra un ejemplo de una red MLP y la figura 3.12[7] muestras las diferentes fronteras de decisión que pueden generar las distintas redes vistas hasta ahora para un espacio de entrada bidimensional.

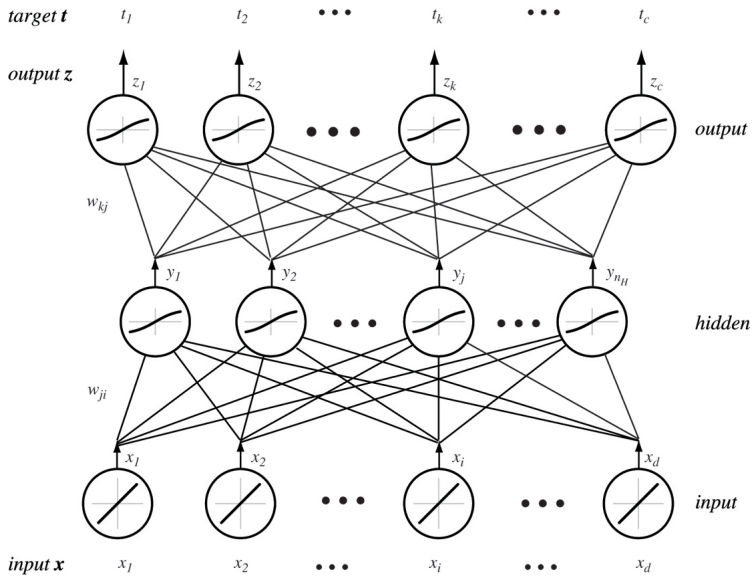



Figura 3.11: Diagrama de una red neuronal multicapa

### 3.4.1. Retropropagación

En este método de aprendizaje las redes se entrenan definiendo una función de error que minimizaremos ajustando los pesos de cada una de las capas. Este entrenamiento se puede dividir en dos etapas: una primera etapa en la que se presenta un ejemplo a la entrada de la red. Esta señal se propagará hacia la salida de la red y obtendremos una salida. La segunda etapa comienza calculando el error cometido sobre la muestra presentada evaluando una función de error y propagando este error hacia atrás (*Retro-*

[6]Imagen obtenida de [Duda *et al.*, 2001]
[7]Imagen obtenida de [Duda *et al.*, 2001]

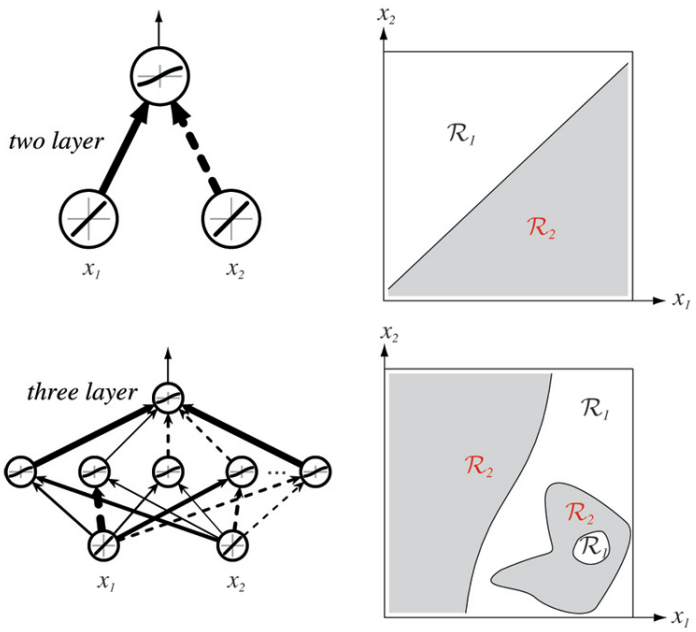



Figura 3.12: Regiones de Decisión para redes monocapa y multicapa

*propagación*) para obtener un ajuste de los pesos que permita que la proxima vez que se presente una entrada igual o similar se reduzca el error que comete la red.

A continuación desarrollamos un ejemplo a partir de la red de la figura 3.4.1[8] del algoritmo para una función de coste que va a ser la suma de los errores al cuadrado, para cada muestra, ya presentada en el entrenamiento de la red *Adaline* y definida por:

$$E(\mathbf{w}) = \frac{1}{2}\sum(t_k - o_k)^2 \tag{3.31}$$

con $t_k$ la salida deseada (*target*) y $o_k = \varphi_k(z_k)$ la salida generada por la neurona $k$-ésima de la capa de salida.

Y el ajuste se realiza por descenso por gradiente ya presentado en 3.2.3 y dado por:

$$\Delta\mathbf{w} = -\eta\frac{\partial E}{\partial \mathbf{w}} \tag{3.32}$$

Primero nos centramos en los pesos que conectan la capa oculta con la capa de salida,

[8]Imagen obtenida de [Duda *et al.*, 2001]

$w_{kj}$, aplicando la regla de la cadena tenemos

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial w_{kj}} = (t_k - o_k)\acute{\varphi}_k(z_k)y_j \tag{3.33}$$

Sabiendo que $z_k$ es la señal que recibe a su entrada la neurona $k$-ésima de la capa de salida

$$z_k = \sum_j y_j w_{kj} + w_{k0} = \mathbf{w_k^T y} \tag{3.34}$$

siendo $y_j$ la salida de la neurona $j$-ésima de la capa oculta. Por último, $\acute{\varphi}_k$ representa la derivada de la función de activación de la neurona k-ésima.

Y Finalmente el ajuste de los pesos de esta capa vendrá dado por la siguiente expresión

$$\Delta w_{kj} = \eta(t_k - o_k)\acute{\varphi}_k(\sum_j y_j w_{kj})y_j \tag{3.35}$$

Introduciendo

$$\delta_k = (t_k - o_k)\acute{\varphi}_k(\sum_j y_j w_{kj}) \tag{3.36}$$

obtenemos

$$\Delta w_{kj} = \eta\delta_k y_j \tag{3.37}$$

Para el caso de los pesos que conectan la entrada con la capa oculta calcularíamos

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \upsilon_j}\frac{\partial \upsilon_j}{\partial w_{ji}} \tag{3.38}$$

El primer término viene dado por

$$\begin{aligned}\frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_k (t_k - o_k)^2\right] \\ &= -\sum_k (t_k - o_k)\frac{\partial o_k}{\partial y_j} \\ &= -\sum_k (t_k - o_k)\frac{\partial o_k}{\partial z_k}\frac{\partial z_k}{\partial y_j} \\ &= -\sum_k (t_k - o_k)\acute{\varphi}(z_k)w_{kj}\end{aligned} \tag{3.39}$$

El segundo término

$$\frac{\partial y_j}{\partial \upsilon_j} = \acute{\varphi}_j(\upsilon_j) \tag{3.40}$$

donde $\upsilon_j$ representa la acumulación de señales en la neurona $j$-ésima de la capa oculta recibidas de las neuronas de entrada. Igual que $z_k$ para el caso de las neuronas de la capa de salida.

$$\upsilon_j = \sum_{i=1}^{m} x_i w_{ji} + w_{j0} = \mathbf{w_j^T x} \tag{3.41}$$

Y el último término

$$\frac{\partial \upsilon_j}{\partial w_{ji}} = x_i \tag{3.42}$$

El ajuste de los pesos de esta capa viene dado por

$$\Delta w_{ji} = \eta \left[ \sum w_{kj}(t_k - o_k)\acute{\varphi}_k(\sum_j y_j w_{kj}) \right] \acute{\varphi}_j(\sum_i x_i w_{ji}) x_i \tag{3.43}$$

Definiendo

$$\delta_j = \acute{\varphi}_j(\upsilon_j) \sum_k w_{kj}\delta_k \tag{3.44}$$

El ajuste de los pesos de la capa oculta es

$$\Delta w_{ji} = \eta x_i \delta_j \tag{3.45}$$

Sustituyendo en esta última ecuación el valor definido en (3.44) tenemos finalmente

$$\Delta w_{ji} = \eta \left[ \sum_k w_{kj}\delta_k \right] \acute{\varphi}_j(\sum_i x_i wji) x_i \tag{3.46}$$

y se observa el carácter iterativo de este algoritmo de salida a entrada.

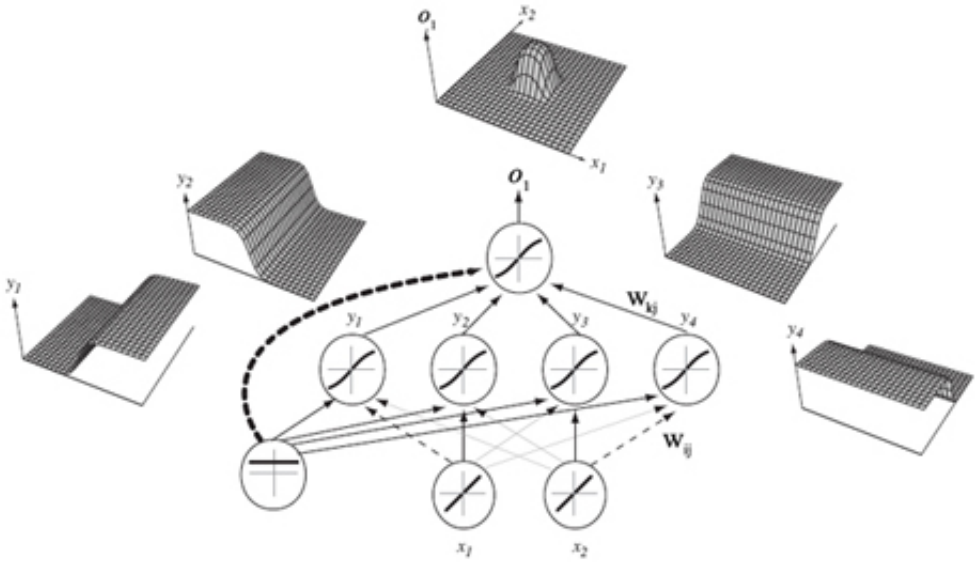



Figura 3.13: Diagrama de una red neuronal multicapa

## 3.5. Funciones de Base Radial (RBF)

Una red de Funciones de Base Radial (*Radial Basis Function*, *RBF*) presenta una arquitectura muy similar a la de un perceptron multicapa (MLP) como los vistos en la sección anterior. Pero el proceso de aprendizaje es considerablemente diferente.

Las RBF como ya mencionamos con los MLP son excelentes aproximadores universales, por lo tanto, son capaces de aproximar funciones continuas o de modelar problemas. El problema de los MLP es que en ocasiones los tiempos de entrenamiento son excesivos. Las RBF son redes neuronales que van a reducir estos tiempos de entrenamiento.

### 3.5.1. Topología y entrenamiento de las RBF

En una red RBF típica nos encontramos con una capa de neuronas de entrada que recibe los datos de entrada. La capa intermedia u oculta la forman unas neuronas que son las funciones base para los vectores de entrada. Estas neuronas de la capa oculta son la característica más importante de las RBF ya que calculan la *distancia* entre el vector de entrada y un vector dado, en lugar de evaluar el producto escalar de un vector de entrada y un vector de pesos en funciones de activación no lineales como hemos visto en el caso de los SLP y MLP. Finalmente las neuronas de la capa de salida realizan una combinación lineal de las salidas de las neuronas ocultas. La figura 3.14 muestra una estructura típica de una red RBF.

La neuronas de la capa oculta realizan una transformación no lineal de los vectores de entrada. Generalmente las funciones base son gaussianas y sus parámetros se determinan a partir del conjunto de datos de entrada. Siendo $\mathbf{x}$ el vector de entrada la salida de la neurona $i$-ésima de la capa intermedia viene dada por:

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2}\right) \tag{3.47}$$

Esta función gaussiana es una función local con la propiedad de que $\phi \to 0$ cuando $\|x\| \to$ ínf. Los vectores $\mathbf{c}_i$ se corresponden con el centro de la función base $\phi_i$, el valor de estos vectores se determina durante el entrenamiento de la red. Esta elección es uno

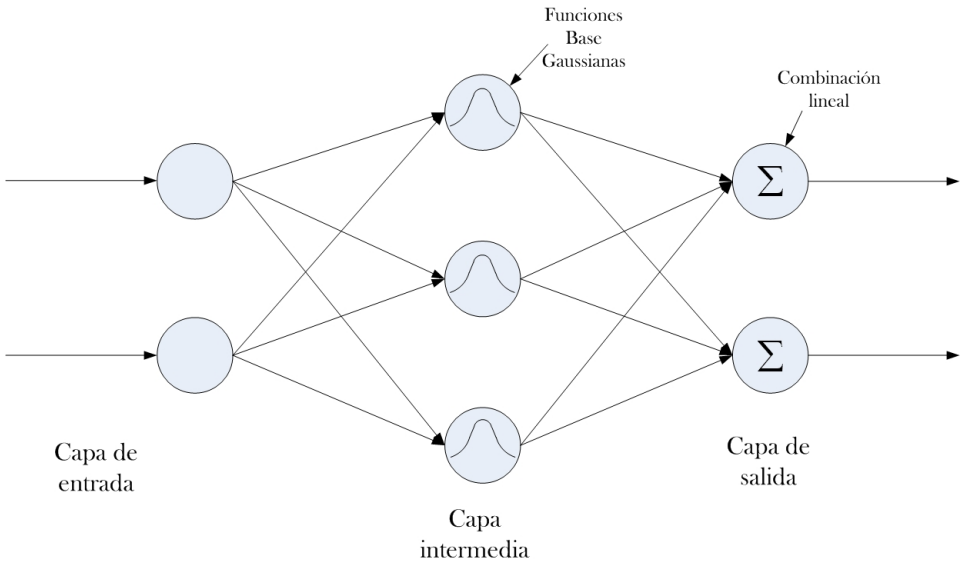



Figura 3.14: Ejemplo de una red RBF típica

de los puntos importantes en el entrenamiento de estas redes. Los términos $\|\mathbf{x} - \mathbf{c}_i\|$ representan la distancia Euclídea entre las entradas y el centro $i$-ésimo.

El entrenamiento de estas redes se puede dividir en 2 fases. En la primera se determinan los parámetros de las funciones base (por ejemplo $\mathbf{c}_i$ y $\sigma_i$). Una vez fijados estos parámetros se calculan los valores de los pesos de las neuronas de la capa de salida en la segunda fase del entrenamiento. Para la optimización de las funciones base se recomienda consultar [Bishop, 1995, Section 5.9] y/o también [Haykin, 1999, Chapter 5].

La obtención de los valores de los pesos de la capa de salida es equivalente al del cálculo de los pesos para las redes monocapa descrito en 3.2.3. Por lo que igualmente podemos optimizar el valor de los pesos minimizando una función de error, como es la suma del error cuadrático y calcular su gradiente como muestra la siguiente ecuación

$$\Delta \mathbf{w}_{jk} = -\alpha \frac{\partial E^i}{\partial \mathbf{w}_{jk}} = \alpha(e^i)\phi_j(\mathbf{x^i}) \tag{3.48}$$

Donde $\mathbf{w_{jk}}$ representa el peso de la conexión de la neurona $j$-ésima de la capa oculta y la neurona $k$-ésima de la capa de salida. Con $\mathbf{x}^i$ el vector de entrada $i$-ésimo, $e^i$ se

corresponde con el error de clasificación del vector de entrada $\mathbf{x}^i$ y por último $\alpha$ es un parámetro que sirve para regular la velocidad de convergencia.

Capítulo 4

# Estado del Arte

En esta capítulo introduciremos dos de las tecnologías que mejores resultados y que mayor desarrollo están teniendo en los últimos años en el ámbito del aprendizaje automático que son las **Máquinas de Vectores Soporte** y los **Sistemas de Múltiples Redes**.

## 4.1. Máquinas de Vector Soporte (SVM)

Los fundamentos de las Máquinas de Vector Soporte (*Support Vector Machines* , *SVM*) fueron presentados por Vapnik y otros autores durante los 80. El concepto de SVM se debe a Vapnik.

El modelo de SVM, como se entiende actualmente, fue presentado en el año 1992 y a partir de ese momento el interés por este método de aprendizaje no ha parado de crecer hasta el presente. Su desarrollo teórico, así como sus aplicaciones prácticas han avanzado notablemente.

Hoy en día, las SVM constituyen el referente dentro de las disciplinas del aprendizaje automático. Las SVM han tenido éxito en aplicaciones prácticas en campos tan distintos como son: la visión por computador, la recuperación de información, el procesamiento del lenguaje natural y el reconocimiento de formas entre otros.

Las SVM generan separadores lineales o hiperplanos en el espacio de entrada o bien en el espacio de características, dando lugar en este último caso a clasificadores no lineales.

El fundamento de las SVM es el siguiente: se parte de un conjunto de vectores de entrada de $R^n$, etiquetados con $+1$ y $-1$ , bajo la hipótesis de que sean linealmente separables por un hiperplano. Por lo tanto, existen infinitos planos en $R^n$ tal que separen los vectores de una clase de los de otra. Los puntos que pertenecen a uno de estos hiperplanos cumples la siguiente expresión:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{4.1}$$

donde $\mathbf{w}$ son los vectores normal a cada hiperplano, $\frac{|b|}{\|\mathbf{w}\|}$ es la distancia de estos al origen. Dicho hiperplano separará las muestras pertenecientes a clases diferentes.

$$\mathbf{x}_i \cdot \mathbf{w} + b > +1 \text{ para } y_i = +1$$
$$\mathbf{x}_i \cdot \mathbf{w} + b < -1 \text{ para } y_i = -1$$

De entre estos hiperplanos las SVM buscan aquel hiperplano que maximiza el mínimo margen entre los ejemplos del conjunto de datos y el hiperplano. Este máximo se

alcanzará cuando $\|\mathbf{w}\|$ sea mínima, sujeta a las restricciones anteriores. Este margen es la distancia entre los dos hiperplanos que contengan a las muestras más cercanas a la frontera (Ver Figura 4.1[1]).

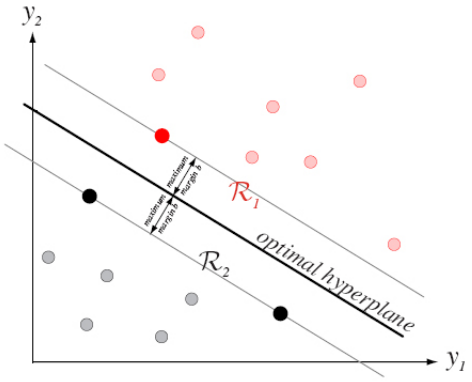



Figura 4.1: Ejemplo de SVM de margen máximo

En la práctica, este hiperplano separador de margen máximo ha demostrado una muy buena generalización, y por tanto una robustez frente al sobreajuste.

Siguiendo las hipótesis previas el clasificador resultante es lineal y como sabemos no todos los problemas admiten una solución de este tipo (Ver Figura 4.2[2]). Y además partimos de las hipótesis de que el conjunto de datos de entrada es linealmente separable lo que no tiene por qué ser cierto o fácil de conseguir en la mayoría de los casos.

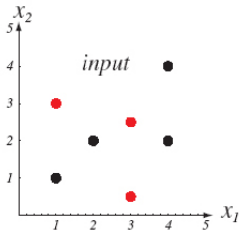



Figura 4.2: Ejemplo de problema no separable linealmente

[1]Imagen obtenida de [Duda *et al.*, 2001]
[2]Imagen obtenida de [Duda *et al.*, 2001]

### 4.1.1. Generalización a problemas no lineales

El funcionamiento de los SVM no lineales se basa en la idea de realizar una transformación no lineal de los atributos de entrada en un espacio de dimensión mayor denominado *espacio de características* donde es posible separar de manera lineal los ejemplos y es donde aplicamos las SVM lineales.

Las funciones que se utilizan para realizar la transformación no lineal son las denominadas *funciones núcleo* o *kernel*. Usando una función núcleo $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}).\Phi(\mathbf{y})$ la frontera de decisión no lineal presenta la forma siguiente:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{4.2}$$

Algunas funciones núcleo comunes son:

- Polinómica.

$$k(\mathbf{x}, \mathbf{y}) = (< x, y > +c)^d \qquad c \in \Re \tag{4.3}$$

- Gaussiana.

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\|x - y\|^2}{\gamma}\right) \tag{4.4}$$

- Sigmoidal

$$k(\mathbf{x}, \mathbf{y}) = \tanh(s < x, y > +r) \qquad s, r \in \Re \tag{4.5}$$

## 4.2. Sistemas de múltiples redes

La idea de combinar múltiples redes es bien conocida en problemas de decisión, es preferible consultar a varios expertos que tomar una decisión basándonos exclusivamente en la opinión de un único experto.

Bajo este supuesto en los últimos años ha surgido un creciente interés por el desarrollo de sistemas de múltiples redes (*multi-net systems*) con el objetivo de mejorar la precisión de los sistemas aislados.

En este proyecto nos centraremos en la combinación de redes neuronales, aunque no hay que descartar la idea de que un sistema de múltiples redes puede estar compuesto por distintas técnicas de aprendizaje automático como los árboles de decisión.

En está sección se revisan algunos de los principales métodos que han sido propuestos para construir los elementos del sistema y posteriormente los métodos de combinación de estos elementos.

Los sistemas de múltiples redes se pueden clasificar en dos categorías que son: los ***comités*** y los ***sistemas modulares***.

### 4.2.1. Comités

Son estructuras basadas en una combinación de máquinas donde cada una de las cuales genera una solución para el problema completo. Por esto se dice que son estructuras redundantes.

Hay dos puntos clave que afectan a la calidad del comité: el primero es la construcción de los elementos que compondrán el sistema, los cuáles deben ser suficientemente precisos y diversos; y el segundo, la forma de combinar los elementos obtenidos.

La diversidad de los componentes puede ser obtenida por ejemplo: variando los parámetros iniciales, variando la topología, variando el algoritmo de entrenamiento y/o variando los datos. Tal vez el último método sea el más utilizado. La variación de los datos se puede conseguir igualmente de muy diversas formas: muestreando los datos, creando conjuntos disjuntos de datos, con boosting o muestreo adaptativo, preprocesando

los datos. Estos métodos se pueden utilizar también de manera combinada.

Una vez obtenidos los elementos del comité llega la hora de combinarlos de forma óptima. A continuación algunos de los métodos de combinación más comunes (para ampliar información consultar [Wolpert, 1992],[Xu *et al.*, 1992]):

- Medias y medias ponderadas. Se realiza una combinación lineal de las salidas de cada elemento para determinar la salida del sistema global.
- Métodos de combinación no lineal. Se utilizan métodos de combinación no-lineal por ejemplo: *votación*.
- Supra Bayessian. La idea subyacente de este método es considerar la opinión de los expertos como datos en si mismos. Consultar [Jacobs, 1995].
- Stacked generalization. Una red no lineal aprende como combinar las redes con pesos que varían sobre el espacio de características. Consultar [Wolpert, 1992].

A continuación explicamos dos de los comités desarrollados en este proyecto no con mucho éxito.

### Bagging

Bagging es uno de los métodos más simples para obtener distintos clasificadores a partir de un único conjunto de datos. El término Bagging deriva de *"bootstrap aggregation"* y su funcionamiento se basa en obtener múltiples conjuntos de datos cada uno de los cuales se forma seleccionando aleatoriamente y con reemplazamiento $m'$ ejemplos del conjunto de datos de entrenamiento original de tamaño $m > m'$. Cada uno de estos conjuntos de datos se utiliza para entrenar y obtener un clasificador.

Una vez obtenidos cada uno de los componentes (clasificadores) la decisión final de clasificación se obtiene por votación mayoritaria. Es decir, se elige la clase con mayor número de votos entre todos los clasificadores.

No todos los algoritmos de aprendizaje son sensibles al conjunto de entrenamiento y esto puede limitar la diversidad de los componentes obtenidos.

### Boosting

En un conjunto de máquinas generadas por alguno de los métodos de Boosting, las máquinas son entrenadas partiendo de un conjunto de datos de entrenamiento modificado. Con lo que cada máquina no sé entrenará para resolver el problema original; sino que, intentará resolver el problema partiendo de una población modificada de cierta manera. En Boosting cada una de estas máquinas se denomina *weak learner* o aprendedor débil. El Boosting por tanto se puede incluir dentro de los conjuntos de redes que denominamos **comités**.

El Boosting es un método general que puede mejorar las prestaciones de cualquier algoritmo de aprendizaje.

En este proyecto vamos a desarrollar una de las variantes del Boosting denominada **AdaBoost** que resuelve algunos de los problemas prácticos de los primeros algoritmos de Boosting y que fue introducida en 1995 por Freund y Schapire [Freund & Schapire, 1997].

Una de las ideas principales de este algoritmo es que asigna un peso a cada ejemplo del conjunto de entrenamiento. Por lo que nuestro algoritmo de aprendizaje debe ser capaz de incorporar esta distribución de pesos al entrenamiento. El peso del ejemplo $i$ para el aprendedor débil $t$ se denota por $\mathcal{D}_t(i)$. Inicialmente los pesos de todos lo ejemplos son iguales, pero a medida que se incorpora un nuevo aprendedor se recalcula esta distribución. La distribución $\mathcal{D}_t$ se actualiza según muestra el pseudocódigo de la página 44, es decir, se incrementa el peso de los ejemplos mal clasificados por el último componente y se decrementa el peso de los ejemplos bien clasificados de este modo el algoritmo intenta que los componentes que se incorporan sucesivamente se centren en las muestras difíciles de clasificar, realiza un remuestreo. Esta distribución se va a ir adaptando (**Ada**ptative-**Boost**ing) según el error cometido por la máquina anterior. En al página 44 exponemos el algoritmo de Adaboost en pseudocódigo tomado de [Schapire, 1999].

La hipótesis final del conjunto $H$ se obtiene por suma ponderada de las hipótesis de cada aprendedor débil $h_t$ por el factor $\alpha_t$, el cual representa la importancia del componente. A menor error cometido por un componente durante el entrenamiento tendremos

**Pseudocódigo RealAdaBoost**

Dados: $(x_i, y_i), \ldots, (x_m, y_m)$ donde $x_i \in X$, $y_i \in Y = \{-1, +1\}$ Inicializamos $D_1(i) = \frac{1}{m}$.
Para t=1,. . . ,T:

- Entrenamos nuestra red utilizando $D_t$.
- Obtenemos las hipótesis $h_t : X \to \{-1, +1\}$ con error

$$\epsilon_t = Pr_{i \sim D_t} \left[ h_t \left( x_i \right) \neq y_i \right] = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$$

- Elegimos $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$.
- Actualizamos la distribución:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if} \quad h_t(x_i) = y_i \\ e^{-\alpha_t} & \text{if} \quad h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

donde $Z_t$ representa un factor de normalización elegido para que $D_t$ sea una distribución de probabilidad.

La hipótesis de salida final será:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

un valor de $\alpha_t$ mayor, de esta manera conseguimos dar más importancia a la hipótesis de aquellos componentes que presentaban un menor error. Debemos tener en cuenta que el error es medido con respecto a la distribución $D_t$ con la que el componente ha sido entrenado.

### 4.2.2. Sistemas Modulares

Estas estructuras se basan en el principio de *"divide y vencerás"* de modo que el problema se descompone en de subproblemas. Los distintos componentes del sistema se centran en resolver un *modulo* o subproblema y para obtener la solución completa se requiere la combinación de los distintos módulos. Cada subproblema puede ser resuelto por una arquitectura o algoritmo diferente aprovechando de este modo las capacidades de especialización.

Aparte de mejorar las prestaciones, los sistemas modulares se utilizan porque pueden reducir la complejidad del modelo y hacer que el sistema sea más fácil de comprender, modificar y extender. El tiempo de entrenamiento también puede verse reducido y el conocimiento previo puede ser incorporado de manera más sencilla en un subproblema (consultar [Sharkey, 1996]).

En cuanto a la combinación, contamos con diferentes formas de combinar los módulos como son: el modo cooperativo, el modo competitivo, el modo secuencial y el modo supervisor.

Por último destacar que los comités y los sistemas modulares no son excluyentes y pueden usarse de manera conjunta en el desarrollo de un sistema de múltiples redes.

Capítulo 5

# Arquitecturas propuestas

En este capítulo iremos introduciendo las distintas arquitecturas que se han desarrollado enfocadas a la realización de pruebas tanto sobre problemas sintéticos como sobre conjuntos de datos (*datasets*) aportados por el departamento de Teoría de la Señal y Comunicaciones, y que denominamos problemas reales, los cuales son utilizados para el análisis empírico de algoritmos de aprendizaje máquina desarrollados en dicho laboratorio.

Comenzaremos con la arquitectura que denominamos `Arquitectura_1` y que como veremos se corresponde con el *Discriminante Lineal de Fisher* (DLF). Posteriormente iremos aumentando la complejidad de la arquitectura con objetivo de aumentar su eficacia o variar sus prestaciones. Con la segunda arquitectura introducimos un método constructivo al que vamos a dar el nombre genérico de `Arquitectura_2`. Con este método constructivo mostramos una manera en la que es posible calcular sucesivos *DLF*'s, para posteriormente combinar los discriminantes de Fisher obtenidos y obtener una única salida global.

Más adelante presentaremos una serie de arquitecturas que se pueden clasificar como sistemas de múltiples redes. Presentando ejemplos de comités utilizando *Bagging* y *Boosting*; y un ejemplo de sistema modular que denominamos `Fisher-Modular`.

## 5.1. Arquitectura_1

Esta primera arquitectura que presentamos va a representar la base sobre la que se desarrolla este proyecto. En esta arquitectura se realizará el cálculo de la función discriminante que utilizaremos para la clasificación de ejemplos, el *DLF*. El siguiente esquema representa la estructura de esta primera arquitectura

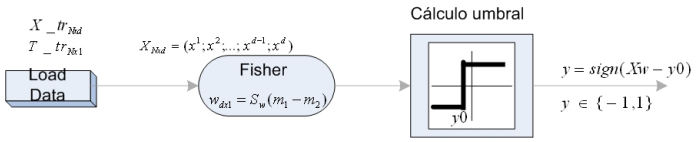



Figura 5.1: Esquema Arquitectura_1

En este proyecto el cálculo de esta función discriminante se corresponde con el cálculo del *DLF* el cual se desarrolló en el punto 3.3.

Como vimos, el *Discriminante Lineal de Fisher* $\mathbf{w}$ consiste en la optimización del siguiente criterio con el que se busca maximizar la separación entre clases (diferencia de medias) para la posterior clasificación de las muestras.

$$J(\mathbf{w}) = \frac{|m_2 - m_1|^2}{s_1^2 + s_2^2} \tag{5.1}$$

operando y dejando el criterio de Fisher $J(\cdot)$ en función de $\mathbf{w}$ tenemos

$$J(\mathbf{w}) = \frac{\mathbf{w^T S_B w}}{\mathbf{w^T S_W w}} \tag{5.2}$$

Recordando, se denomina $\mathbf{S_W}$ a la matriz de covarianzas intraclase (*within-class scatter matrix*) y $\mathbf{S_B}$ a la matriz de covarianzas interclase (*between-class scatter matrix*) definidas según (3.25) y (3.28) en la sección 3.3.

A partir del desarrollo presentado en 3.3 el valor de $\mathbf{w}$ que optimiza $J(\cdot)$ es:

$$\mathbf{w} \propto \mathbf{S_W}^{-1} (\mathbf{m}_1 - \mathbf{m}_2) \tag{5.3}$$

Los valores $\mathbf{m}_1$ y $\mathbf{m}_2$ representan los vectores de medias de los datos pertenecientes a cada una de las dos clases.

El *Discriminante Lineal de Fisher* $\mathbf{w}$ consigue una proyección de los datos en la cual se maximiza la separación entre las dos clases normalizada con respecto a la varianza dentro de las clases. El factor de escala de $\mathbf{w}$ no va a ser relevante para la clasificación, aunque puede influir en los resultados como veremos más adelante.

La Figura 5.2 muestra el resultado del cálculo de $\mathbf{w}$ a partir de unos datos de ejemplo, junto con la proyección de los datos sobre este vector $\mathbf{w}$

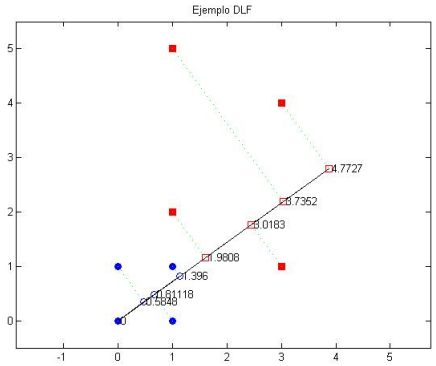



Figura 5.2: Ejemplo de cálculo del *DLF*

Este discriminante va a ser empleado para construir un clasificador lineal binario. Proyectando sobre la recta óptima respecto del Criterio de Fisher y luego comparando con un cierto umbral.

La clasificación de individuos $\mathbf{x}_k$ se realizará comparando la función discriminante $g(\mathbf{x}_k) = \mathbf{w}^\mathbf{T}\mathbf{x}_k$ con un cierto umbral $y_0$, de tal forma que si $g(\mathbf{x}_k) \geq y_0, \mathbf{x}_k$ lo clasificaremos como perteneciente a $C_1$, mientras que si $g(\mathbf{x}_k) < y_0, \mathbf{x}_k$ lo asignaremos a $C_2$.

Un primer valor de este umbral, con un cálculo sencillo, se corresponde con el punto medio de la proyección del valor medio de los ejemplos pertenecientes a cada una de las clases, para el caso de clasificación binaria tenemos:

$$y_0 = \frac{\mathbf{w}^\mathbf{T}\mathbf{m}_1 + \mathbf{w}^\mathbf{T}\mathbf{m}_2}{2} \tag{5.4}$$

También vimos que

$$m_k = \mathbf{w}^\mathbf{T}\mathbf{m}_k \tag{5.5}$$

es la media de los datos proyectados de la clase $\mathcal{C}_k$. Sustituyendo esta última expresión en (5.4) el valor de este primer umbral viene dado por

$$y_0 = \frac{m_1 + m_2}{2} \tag{5.6}$$

Por lo que, finalmente nuestro primer clasificador basado en el Criterio de Fisher realiza la siguiente asignación.

$$\text{si } g(\mathbf{x}_k) = \mathbf{w^T x}_k \geq y_0 \rightarrow \mathbf{x}_k \in C_1$$
$$\text{si } g(\mathbf{x}_k) = \mathbf{w^T x}_k < y_0 \rightarrow \mathbf{x}_k \in C_2$$

Se puede demostrar que la función discriminante de Fisher presenta la misma forma que la función discriminante $g(\mathbf{x})$ para el caso de dos clases con densidades de probabilidad gaussianas y con idéntica matriz de covarianzas $\mathbf{\Sigma} = \mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$ tenemos

$$g(\mathbf{x}) = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{\Sigma}^{-1} \mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \ln \frac{P(C_1)}{P(C_2)} \tag{5.7}$$

y con probabilidades a priori iguales para ambas clases tenemos

$$g(\mathbf{x}) = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{\Sigma}^{-1} \mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{5.8}$$

Sustituyendo $\mathbf{\Sigma}$ por $\mathbf{S_W}$ y $\boldsymbol{\mu}_1$ y $\boldsymbol{\mu}_2$ por $\mathbf{m}_1$ y $\mathbf{m}_2$ tenemos:

$$g(\mathbf{x}) = \mathbf{w^T x} - \frac{\mathbf{w^T m}_1 + \mathbf{w^T m}_2}{2} \tag{5.9}$$

Siendo $\mathbf{w} = \mathbf{S_W}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$

Por lo que salvo un factor de escala la función discriminante de Fisher coincide con la función discriminante con hipótesis de normalidad, equicovarianza y probabilidades a priori iguales. Por lo tanto, si se cumplen estas suposiciones la función discriminante de Fisher se corresponderá con el discriminante óptimo en ese problema.

### 5.1.1. Modificación del umbral de clasificación

Con lo mostrado hasta ahora podemos ver que tras aplicar la función discriminante $g(\mathbf{x}_k) = \mathbf{w^T x}_k$ sobre el conjunto de datos $\{\mathbf{x}_k\}_{k=1}^K$ nuestro problema de clasificación se

ha reducido a una clasificación unidimensional y en la que el umbral de decisión queda definido como

$$y_0 = \frac{\mathbf{w^T m}_1 + \mathbf{w^T m}_2}{2} = \frac{m_1 - m_2}{2} \tag{5.10}$$

el cuál puede ser una buena aproximación pero no tiene porque ser el umbral óptimo.

Por lo tanto, introducimos unas variantes del clasificador mostrado anteriormente que se diferencian exclusivamente en el cálculo de este umbral de decisión $y_0$. Estas variantes las denominamos como `Arquitectura_1 - Fisher (y0')` y `Arquitectura_1 - Fisher (y0'')`, y las desarrollamos a continuación.

### Arquitectura_1 - Fisher (y0')

Con el cálculo de este nuevo umbral $y_{0'}$ intentamos generalizar el umbral $y_0$ de la forma

$$y_{0'} = \frac{m_1 - m_2}{2} + \ln \frac{\hat{P}(\mathcal{C}_1)}{\hat{P}(\mathcal{C}_2)} \tag{5.11}$$

siendo $\hat{P}(\mathcal{C}_1)$ la proporción de ejemplos perteneciente a la clase $\mathcal{C}_1$ sobre el total de ejemplos y siendo $\hat{P}(\mathcal{C}_2)$ los pertenecientes a la clase $\mathcal{C}_2$, es decir, $\hat{P}(\mathcal{C}_1)$ y $\hat{P}(\mathcal{C}_2)$ representan las estimaciones a priori de la probabilidad de que una muestra pertenezca a uno u otra clase, con esto perseguimos incluir los efectos de encontrarnos con clases con distintas probabilidades a priori; efecto que no teníamos en cuenta con $y_0$.

### Arquitectura_1 - Fisher (yo")

Para el cálculo de este tercer umbral de decisión lo que buscamos no es solo estimar las probabilidades a priori de cada clase, sino que además estimaremos las densidades de probabilidad condicionales de cada clase, $f(y|\mathcal{C}_k)$.

Sabemos que la regla óptima de decisión es la *regla de Bayes*[1] que indica que asignaremos el individuo $y$ a la clase $j$ que presente mayor probabilidad a posteriori.

$$f(y|\mathcal{C}_j)P(\mathcal{C}_j) = \underset{k}{arg}\{max(f(y|\mathcal{C}_k)P(\mathcal{C}_k))\} \tag{5.12}$$

[1] para mayor detalle consulte [Duda *et al.*, 2001]

Esta regla se puede aplicar si conocemos tanto las probabilidades a priori como las densidades condicionales para cada una de las clases. En nuestro caso al no conocerlas recurrimos a la estimación, para la estimación de las probabilidades a priori utilizamos las frecuencias relativas de cada clase $\hat{P}(\mathcal{C}_k)$ que se corresponde con la proporción de elementos que pertenecen a la clase $k$ respecto del total de elementos. El mayor problema es la estimación de las funciones de densidad condicionales, esta estimación la vamos a realizar de forma *no-paramétrica*, ya que, de este modo no se presupone ninguna forma de la función de densidad y la estimación vendrá dada enteramente por los datos.

Para la estimación *no-paramétrica* utilizaremos estimadores núcleo en nuestro caso, ventanas de *Parzen*. La expresión de este tipo de estimador es

$$\hat{f}(y) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{y - y_i}{h}\right) \tag{5.13}$$

siendo $K$ la función núcleo que para el caso de la ventanas de *Parzen* es de la forma

$$\hat{f}(y) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{(2\pi h^2)^{1/2}} \exp\left(-\frac{y - y_i}{2h^2}\right) \tag{5.14}$$

con $h$ un parámetro de suavizado. El cálculo de este parámetro es de vital importancia. Muchos de los métodos de elección de $h$ pueden consultarse en [Wand & Jones, 1995], sin embargo nosotros optamos heurísticamente por elegir un valor que no genera una estimación demasiado irregular ni tampoco demasiado suave; $h = 0{,}4$.

La estimación de la densidad es una mezcla de $n$ densidades dada por la forma del estimador núcleo elegido, escaladas según $h$ y centradas cada una en una observación $y_i$. La densidad finalmente será la suma de estos núcleos. Un ejemplo de densidad estimada se muestra en la figura 5.1.1 para distintos valores del parámetro de suavizado

Siempre que estemos en un caso unidimensional como es nuestro caso.

En la anterior imagen se puede observar la influencia y por lo tanto la importancia del parámetro de suavizado $h$.

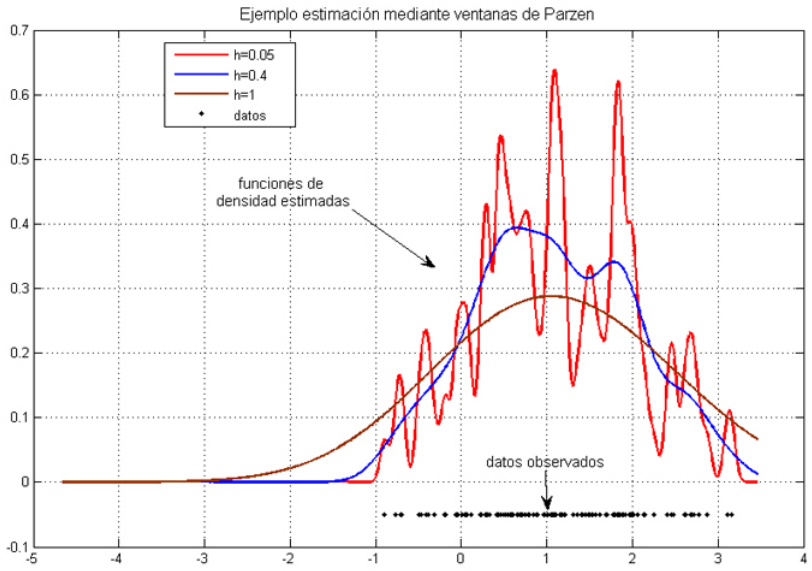



Figura 5.3: Función de densidad estimada mediante ventanas de parzen

## 5.2. Arquitectura_2

Con esta segunda arquitectura intentaremos combinar de forma constructiva sucesivos discriminantes de Fisher con el objetivo de mejorar los resultados de `Arquitectura_1 - Fisher`. Se buscan recurrentemente las direcciones que mejor discriminan los datos mediante el cálculo del DLF. La manera de calcular los sucesivos discriminantes de Fisher es común en esta arquitectura y será explicada a continuación, y para combinar cada uno de los distintos discriminantes que calculamos utilizaremos un SLP (Ver 3.2.5). El número de discriminantes que se calcularán varía de 1 a $d-1$ siendo $d$ el número de variables del problema dado, obteniéndose como máximo $d$ variables para la combinación.

En la figura 5.2 mostramos de forma esquematizada el cálculo de los sucesivos DLFs.

En los siguientes apartados se explica el desarrollo para el cálculo de este método constructivo presentando cada uno de los bloques que lo constituyen.

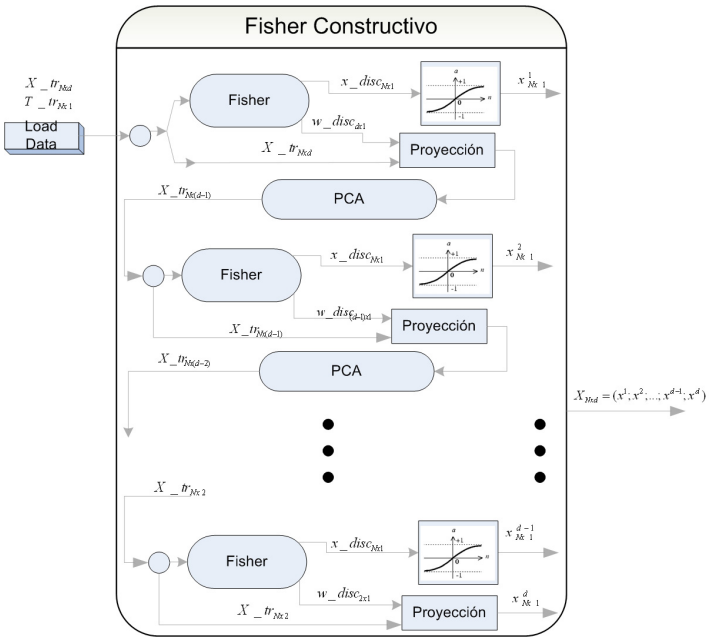



Figura 5.4: Método constructivo

### 5.2.1. Fisher

Este bloque se corresponde con la `Arquitectura_1` salvo que aquí únicamente necesitamos la dirección de proyección $\mathbf{w}$ que se obtiene como ya vimos de $\mathbf{w} = \mathbf{S_W^{-1}}(\mathbf{m}_1 - \mathbf{m}_2)$

La figura 5.2 presentada en la sección 5.1 mostraba el resultado del cálculo del DLF sobre unos datos de ejemplo junto con la proyección de los mismos.

### 5.2.2. Proyección

Como explicamos en la sección 3.2.1 las funciones de la forma:

$$y = \mathbf{w^T x} \tag{5.15}$$

se pueden considerar como una proyección de los datos $\mathbf{x}$ de dimensión $d$ en una dirección especificada por el vector $\mathbf{w}$ dando lugar a valores escalares $y$.

Recordamos que la idea del discriminante de Fisher es proyectar los datos en la dirección en la que la separación entre las clases de datos proyectados sea máxima.

Este bloque funcional ***proyección*** presentado en la figura (5.2) junto al resto de bloques no va a estar relacionado con la proyección que aplica el discriminante de Fisher, pero nos va a permitir aplicar sucesivos discriminantes de Fisher dando lugar en conjunción con el análisis de componentes principales a nuestro ***Método de Fisher Constructivo***.

Considerando un conjunto de datos de entrada $\mathbf{x}$ de dimensión $d$ a los que aplicamos un primer discriminante de Fisher que denominamos $\mathbf{w_1} = \mathbf{S_{W_1}^{-1}}(\mathbf{m}_1 - \mathbf{m}_2)$ y con el objetivo de aplicar sucesivos discriminantes lo que haremos será proyectar los datos de entrada $\mathbf{x}$ en el subespacio ortogonal del vector $\mathbf{w_1}$. Con esta proyección lo que pretendemos es descartar la dirección que según el *Criterio de Fisher* contiene la mayor información para realizar una clasificación. Para ello sabemos que cualquier vector $\mathbf{x}$ admite una única representación como combinación lineal de la forma:

$$\mathbf{x} = \sum_{i=1}^{n} z_i \mathbf{u_i} \tag{5.16}$$

donde los vectores $\mathbf{u_i}$ son ortonormales es decir cumplen que

$$\mathbf{u_i^T u_j} = \delta_{ij} \tag{5.17}$$

en la cual $\delta_{ij}$ se corresponde con la delta de Kronecker. En otras palabras $\delta_{ij} = 1$ si $i = j$ y $\delta_{ij} = 0$ en otro caso. Los coeficientes $z_i$ se pueden representar usando

$$z_i = \mathbf{u_i^T x} \tag{5.18}$$

Como hemos dicho, queremos representar los valores de entrada en el subespacio ortogonal al vector $\mathbf{w_1}$, es decir, que conocemos uno de los vectores ortogonales $\mathbf{w_1}$ y lo asociamos como $\mathbf{u_n} = \mathbf{w_1}$, por lo que los valores de entrada se pueden expresar ahora como

$$\mathbf{x} = \sum_{i=1}^{n-1} z_i \mathbf{u_i} + z_n \mathbf{u_n} \tag{5.19}$$

Lo que nos interesa es la representación de los datos de entrada en el subespacio ortogonal reescribimos la ecuación anterior para obtener

$$\sum_{i=1}^{n-1} z_i \mathbf{u_i} = \mathbf{x} - z_n \mathbf{u_n} \tag{5.20}$$

y de las ecuaciones anteriores obtenemos que

$$z_i = \frac{\mathbf{u_i^T x}}{\|\mathbf{u_i}\|^2} = \mathbf{u_i^T x}, \text{ donde } \|\mathbf{u_i}\| = 1 \tag{5.21}$$

Por lo que finalmente los datos proyectados al subespacio ortogonal se obtienen mediante

$$\mathbf{y} = \sum_{i=1}^{n-1} z_i \mathbf{u_i} = \mathbf{x} - \mathbf{u_n^T x u_n} = \mathbf{x} - \mathbf{w_1^T x w_1} \tag{5.22}$$

Una vez que tenemos los datos proyectados en el subespacio ortogonal, vemos que la dimensión de los datos $\mathbf{y}$ es la misma que la de los datos de entrada $\mathbf{x}$ pero la diferencia radica en que podremos representar los datos $\mathbf{y}$ en $d-1$ dimensiones sin pérdida de información, siempre que encontremos los vectores $\mathbf{u_1} \cdots \mathbf{u_{n-1}}$ y esto los conseguimos gracias al *Análisis de Componentes Principales* (PCA). El proceso de *PCA* se describe en 5.2.3, al desarrollar el bloque funcional que realiza el Análisis de Componentes Principales.

El proceso de análisis de componentes principales nos devolverá los datos $\mathbf{y}$ en la forma

$$\mathbf{y} = \sum_{i=1}^{n-1} t_i \mathbf{o_i} \tag{5.23}$$

es decir, como una combinación lineal de vectores ortonormales en este caso los vectores $\mathbf{o_i}$ (el cálculo de estos vectores se estudiará en la siguiente sección), y una serie de coeficientes $t_i$. A partir de aquí consideramos los datos $\mathbf{y}$ de dimensión $d-1$ como si fueran los datos de entrada originales y sobre estos calculamos de nuevo el discriminante de Fisher obteniendo en este caso $\mathbf{w_2}$ que se corresponde con $\mathbf{w_2} = \mathbf{S_{W_2}}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$ calculado sobre los datos $\mathbf{y}$. Este nuevo discriminante debe aportarnos información extra para la discriminación.

Podremos aplicar este proceso sucesivamente y calcular $d-1$ discriminantes de Fisher suponiendo $d$ como la dimensión de los datos de entrada de origen.

### Coste computacional

Para la implementación del proceso de proyección se propone una solución de proyección bloque. Este método de proyección bloque se basa en el siguiente desarrollo, el cual posteriormente es implementado en *Matlab*.

Para proyectar los $k$ ejemplos de $d$-dimensiones $\mathbf{x}_1 \dots \mathbf{x}_k$ al subespacio ortogonal al vector $\mathbf{w}$ también de $d$-dimensiones que obtenemos al calcular el discriminante de Fisher debemos aplicar, con $\|\mathbf{w}\| = 1$

$$\mathbf{y} = \mathbf{x} - \mathbf{w}^\mathbf{T}\mathbf{x}\mathbf{w} \tag{5.24}$$

para cada uno de los $k$ ejemplos. Para aprovechar las capacidades de *Matlab* buscamos realizar las proyecciones de cada uno de los ejemplos en una única instrucción procediendo de la siguiente manera, redefiniendo la ecuación anterior como

$$\mathbf{Y} = \mathbf{X} - (\mathbf{w}^\mathbf{T}\mathbf{X}). \times \mathbf{W} \tag{5.25}$$

siendo $\mathbf{X}$ la matriz con todo el conjunto de datos es decir

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(k)} \\ \vdots & \vdots & \vdots \\ x_d^{(1)} & \cdots & x_d^{(k)} \end{bmatrix} \tag{5.26}$$

Primero calculamos

$$\mathbf{w}^\mathbf{T}\mathbf{X} = \begin{bmatrix} w_1 & \cdots & w_d \end{bmatrix} \times \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(k)} \\ \vdots & \vdots & \vdots \\ x_d^{(1)} & \cdots & x_d^{(k)} \end{bmatrix} \tag{5.27}$$

y obtenemos

$$\begin{bmatrix} \mathbf{w}^\mathbf{T}\mathbf{x}^{(1)} & \cdots & \mathbf{w}^\mathbf{T}\mathbf{x}^{(k)} \end{bmatrix} \tag{5.28}$$

replicando el resultado anterior en $d$ filas para obtener

$$\begin{bmatrix} \mathbf{w}^\mathbf{T}\mathbf{x}^{(1)} & \cdots & \mathbf{w}^\mathbf{T}\mathbf{x}^{(k)} \\ \vdots & \vdots & \vdots \\ \mathbf{w}^\mathbf{T}\mathbf{x}^{(1)} & \cdots & \mathbf{w}^\mathbf{T}\mathbf{x}^{(k)} \end{bmatrix}_{d \times k} \tag{5.29}$$

y siendo $\mathbf{W}$ una matriz con replicas de $\mathbf{w}$

$$\begin{bmatrix} w_1 & \cdots & w_1 \\ \vdots & \vdots & \vdots \\ w_d & \cdots & w_d \end{bmatrix}_{d \times k} \tag{5.30}$$

Para finalmente restar a $\mathbf{X}$ el producto punto a punto, representado por el símbolo $.\times$, de las dos últimas matrices (5.30) y (5.29)

$$\mathbf{Y} = \mathbf{X} - \begin{bmatrix} \mathbf{w^T x}^{(1)} & \cdots & \mathbf{w^T x}^{(k)} \\ \vdots & & \vdots \\ \mathbf{w^T x}^{(1)} & \cdots & \mathbf{w^T x}^{(k)} \end{bmatrix}_{d\times k} .\times \begin{bmatrix} w_1 & \cdots & w_1 \\ \vdots & \vdots & \vdots \\ w_d & \cdots & w_d \end{bmatrix}_{d\times k} \tag{5.31}$$

El número de proyecciones que realiza este *Método Constructivo* es $d-1$ siendo $d$ el número de variables de los ejemplos, tanto a la hora de entrenar la arquitectura como al evaluarla. Por lo que la optimización de este bloque funcional es fundamental para que no se disparen los tiempos de entrenamiento y de evaluación de nuestra arquitectura.

Para comprobar que realmente se disminuye el tiempo de computo al realizar una proyección bloque en lugar de una proyección secuencial generamos el documento con la información de coste computacional mediante los siguientes comandos para el caso de una `única` proyección secuencial:

```
>>profile on;proyeccionSecuencial(w,datos);
p=profile('info');profsave(p,'proyeccionSecuencial');
```

La proyección se ha realizado sobre un conjunto de 4000 ejemplos para la proyección de 5 a 4 dimensiones y la tabla con los resultados nos dice que el tiempo de computo es de **0.359** seg.

Para el caso de proyección bloque se ejecutan los siguiente comandos

```
>>profile on;proyeccionBloque(w,datos);
p=profile('info');profsave(p,'proyeccionBloque');
```

Se observa que el tiempo de cómputo para la proyección de 5 a 4-dimensiones con los mismos 4000 ejemplos de entrada pasa a ser de **0.016** seg.

El alto tiempo de cómputo es debido a que realizábamos la proyección ejemplo a ejemplo con lo que no utilizábamos el potencial de *Matlab* en el manejo de matrices de datos.

Considerando además que estos resultados son sólo para una única proyección cuando se realizan $d-1$ llamadas a esta función a la hora de entrenar la arquitectura, y otras

$d-1$ llamadas a la hora de evaluar la red, la reducción de tiempo de computo es muy grande.

Estos procedimientos se han ejecutado en un ordenador personal Pentium IV 3 GHz y con un 1 GB de memoria RAM.

### 5.2.3. PCA

El ***Análisis de Componentes Principales*** (PCA), generalmente es utilizado como herramienta de análisis para reducir la dimensión del espacio de los datos de entrada, evitando así el problema conocido como *The curse of dimensionality* [Bellmann, 1961], este término se refiere al hecho de que en el caso de problemas de alta dimensionalidad la red utiliza gran parte de sus recursos para modelar zonas del espacio de entrada irrelevantes. Este problema también se puede observar desde otro punto de vista. Para el caso de un problema de una única dimensión con un conjunto de datos relativamente pequeño podríamos encontrar el patrón o patrones para la clasificación pero en un problema de alta dimensionalidad necesitaríamos una enorme cantidad de datos para que nuestra máquina pudiera crear patrones que cubran todo el espacio de entrada. No obstante, en la práctica, el conjunto de datos de entrada va a estar limitado, por lo que nos vemos obligados a reducir la dimensión del espacio de entrada descartando las variables irrelevantes, lo que también ayuda a centrar los recursos de la red en la información relevante.

Dentro de las técnicas de reducción de la dimensionalidad podríamos haber optado simplemente por la selección de un conjunto menor de las variables de entrada, pero ¿Cuáles serían esas variables?. PCA nos ayuda a elegir esas variables. Con el Análisis de Componentes Principales se persigue obtener una nueva representación de los datos de entrada de la manera más exacta posible en el sentido de reducción del error cuadrático medio en un menor número de dimensiones (variables de entrada).

PCA podría ser considerada una técnica de aprendizaje no-supervisado ya que manipulamos los datos de entrada sin tener en cuenta el objetivo de los mismos (la clase a la que pertenecen).

**Desarrollo.** En nuestro proyecto utilizamos PCA como herramienta de representación de variables. Intentando que esta representación no conlleve una pérdida de información que nos sería útil para la clasificación de los datos.

Los datos a la entrada de este bloque funcional presentan $d$ dimensiones pero después de este bloque la datos tendrán $d-1$ dimensiones sin pérdida de información.

Como hemos dicho PCA intenta representar un vector $\mathbf{x}$ de $d$-dimensiones en un menor número de dimensiones por ejemplo $d'$-dimensiones, con $d' < d$, mediante una transformación lineal de las variables. Sabemos que un vector $\mathbf{x}$ se puede representar como una combinación lineal de $d$ vectores ortonormales $\mathbf{u_i}$ de la siguiente manera:

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{u_i} \tag{5.32}$$

Los coeficientes $z_i$ se obtienen de:

$$z_i = \mathbf{u}_i^T \mathbf{x} \tag{5.33}$$

Ahora introducimos la reducción de la dimensión representando un vector $\tilde{\mathbf{x}}$ que es una aproximación del vector $\mathbf{x}$ original, como:

$$\tilde{\mathbf{x}} = \sum_{i=1}^{d'} z_i \mathbf{u_i} + \sum_{i=d'+1}^{d} b_i \mathbf{u_i} \tag{5.34}$$

PCA persigue conseguir la mejor representación $\tilde{\mathbf{x}}$ para todo el conjunto de datos de entrada de manera que minimice el error cuadrático medio para todos los datos, pero no busca conservar la información de clasificación como hace `Fisher`, sino mantener la estructura de los datos. El error que cometemos en la aproximación para una muestra $\mathbf{x}^{(k)}$ viene dado por

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \tilde{\mathbf{x}}^{(k)} = \sum_{i=d'+1}^{d} (z_i^{(k)} - b_i)\mathbf{u_i} \tag{5.35}$$

Por lo tanto, si queremos minimizar el error cuadrático, pero para todo el conjuntos de datos, la función a minimizar pasa a ser

$$E = \frac{1}{2} \sum_{k=1}^{K} \sum_{i=d'+1}^{d} (z_i^{(k)} - b_i)^2 \tag{5.36}$$

Buscamos los valores $b_i$ que minimizan la función anterior, por lo que derivamos la función anterior en función de $b_i$ obteniendo

$$b_i = \frac{1}{K}\sum_{k=1}^{K} z_i^{(k)} = \mathbf{u}_i^T \mathbf{m} \tag{5.37}$$

siendo $\mathbf{m}$ el vector de medias para todos los datos

$$\mathbf{m} = \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}^{(k)}, \tag{5.38}$$

podemos reescribir el error cuadrático a partir de (5.33) y (5.37) como

$$E = \frac{1}{2}\sum_{i=d'+1}^{d} \mathbf{u}_i^T \Sigma \mathbf{u}_i \tag{5.39}$$

Donde $\Sigma$ representa la matriz de covarianzas del conjunto de datos $\{\mathbf{x}^{(k)}\}$

$$\Sigma = \sum_{k=1}^{n} (\mathbf{x}^{(k)} - \mathbf{m})(\mathbf{x}^{(k)} - \mathbf{m})^t \tag{5.40}$$

La minimización de $E$ en función de los valores de los vectores $\mathbf{u_i}$ sujeto a la siguiente restricción $\|\mathbf{u_i}\| = 1$ se resuelve utilizando el método de los multiplicadores de Lagrange (descrito en [Bishop, 1995] Appendix E).

Esta minimización nos indica que los vectores $\mathbf{u_i}$ que minimizan el error se corresponden con los autovectores de la matriz de covarianzas.

$$\Sigma \mathbf{u_i} = \lambda \mathbf{u_i} \tag{5.41}$$

Por tanto, para obtener la mejor representación de los datos en $d'$ dimensiones debemos quedarnos con los $d'$- autovectores asociados a los $d'$- autovalores de mayor valor de la matriz de covarianzas. Al ser la matriz de covarianzas real y simétrica sus autovectores son ortogonales.

También se obtiene que el valor del error cuadrático para esos vectores $\mathbf{u}_i$ es

$$E = \frac{1}{2}\sum_{i=d'+1}^{d} \lambda_i \tag{5.42}$$

por lo que podríamos evaluar el valor del error cometido en la aproximación.

Y con lo visto aquí nos basta con que el conjunto de vectores (de datos de entrada) tenga media 0, si no simplemente les restaríamos la media del conjunto. Calculamos la matriz de covarianzas $\Sigma$ descrita en (5.40) y por último seleccionamos los $d'$ autovectores asociados a los $d'$ mayores autovalores de la matriz de covarianzas. Es decir,seleccionar

$$\mathbf{\Lambda} = (\lambda_{\mathbf{1}}, \cdots, \lambda_{\mathbf{d'}}) \tag{5.43}$$

Luego únicamente quedaría realizar la proyección de los datos de entrada $\mathbf{x}^{(k)}$ con los autovectores elegidos.

Ahora, centrándonos en nuestro proyecto, y considerando la proyección de los datos de entrada $\mathbf{x}^{(k)}$ que se realiza en el bloque funcional ***proyección*** antes de realizar PCA, se observa que al calcular la matriz de covarianzas uno de los autovalores de la matriz de covarianzas tiene valor 0, es decir, esta variable no aporta ninguna información de representación. Este valor 0 se debe a la proyección previa que se realiza en el bloque funcional anterior. El autovector asociado a este autovalor $\lambda_w$ se corresponde con el vector $\mathbf{w}$ obtenido por el Discriminante Lineal de Fisher y a partir del cual realizamos la proyección al subespacio ortogonal. Teniendo en cuenta esto y recordando que podíamos evaluar el error cometido que realiza PCA con (5.42) y considerando que solo reducimos una dimensión con esta arquitectura no vamos a cometer error en esta aproximación quedando demostrado por

$$E = \sum_{i=d'+1}^{d} \lambda_i = \sum_{i=d'+1}^{d-1} \lambda_i + \lambda_w \tag{5.44}$$

y como $\lambda_w = 0$

$$E = \sum_{i=d'+1}^{d} \lambda_i = \sum_{i=d'+1}^{d-1} \lambda_i \tag{5.45}$$

Por lo tanto, queda demostrado que en nuestro proyecto PCA es utilizado para eliminar variables que no aportan información útil para la clasificación. Y que la información de clasificación se obtiene mediante los DLF, a su vez PCA permite aplicar sucesivos DLF's a los datos.

### 5.2.4. Bloques opcionales

Los tres bloques funcionales presentados hasta ahora ***Fisher, proyección, PCA*** constituyen los elementos fundamentales de esta `Arquitectura_2`, los bloques que presentamos a continuación son opcionales mediante una serie de parámetros y han sido introducidos con el objetivo de mejorar las prestaciones de la arquitectura.

1. Normalización del *Discriminante de Fisher* $\mathbf{w}$ obteniendo $\|\mathbf{w}\| = 1$

2. Normalización de datos proyectados

   Realizamos un escalado de los datos para que los datos queden distribuidos en el rango $[-1, 1]$, lo conseguimos mediante la siguiente expresión, aplicada sobre cada una de las muestras del conjunto.

   $$x^k = \frac{2(x^k - min)}{(max - min)} - 1 \tag{5.46}$$

   donde $max$ es el valor máximo del conjunto de datos que se presenta, es decir, $max = max\left\{x^k\right\}_{k=1}^{K}$ y donde $min = min\left\{x^k\right\}_{k=1}^{K}$ se corresponde con el valor mínimo. Esta expresión ha sido obtenida de la ayuda de *MATLAB*, se corresponde con la función `premnmx`

3. Función de saturación

   Aplicamos una función no lineal a los datos de la forma

   $$x = \tanh(x) \tag{5.47}$$

   obteniendo valores en el rango $[-1, 1]$.

### 5.2.5. Cálculo de la salida global

Una vez hemos obtenido la transformación de las variables con el método constructivo realizaremos una combinación lineal de las mismas mediante un *SLP* (Ver 3.2.5). Este SLP consigue integrar de forma natural la inclusión de los sucesivos discriminantes dada su actualización muestra a muestra. Por lo que la arquitectura denominada **Arquitectura_2** presenta la forma que muestra la siguiente imagen.

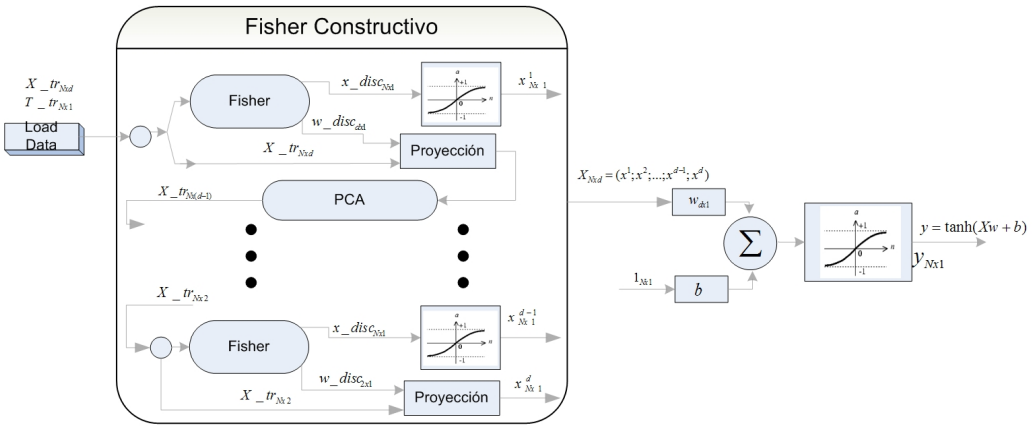



Figura 5.5: Esquema final Arquitectura_2

## 5.3. Comités

En esta sección comentamos el desarrollo de dos sistemas de múltiples redes que se pueden calificar como *comités* según lo visto en 4.2.1.

Dentro de los comités hemos utilizado dos técnicas que ya presentamos en 4.2 como son el *Bagging* y el *AdaBoost*.

### AdaBoost

Para el desarrollo de esta arquitectura, que denominamos `AdaFisher`, nos hemos basado en las ideas explicadas en el apartado *Boosting*, en las que se explicaba una de las técnicas más utilizadas y populares para la construcción de conjuntos de múltiples redes el *Boosting*, en concreto, una versión denominada *Adaboost*.

Con esta arquitectura intentamos aplicar la idea de *Boosting* a la estructura que denominamos `Arquitectura_1`.

La forma de la arquitectura global a partir de redes como las de la sección 5.1 se muestra en la figura 5.6 teniendo en cuenta que la arquitectura base es de la forma que muestra la Figura 5.1.

La figura 5.6 muestra un esquema del resultado de aplicar el algoritmo adaboost

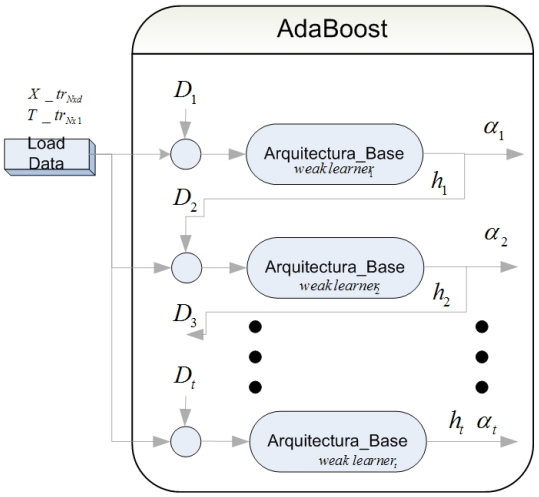



Figura 5.6: Esquema entrenamiento Adaboost

durante el entrenamiento. La figura 5.7 muestra la arquitectura final que se utilizará para la clasificación de las muestras de test.

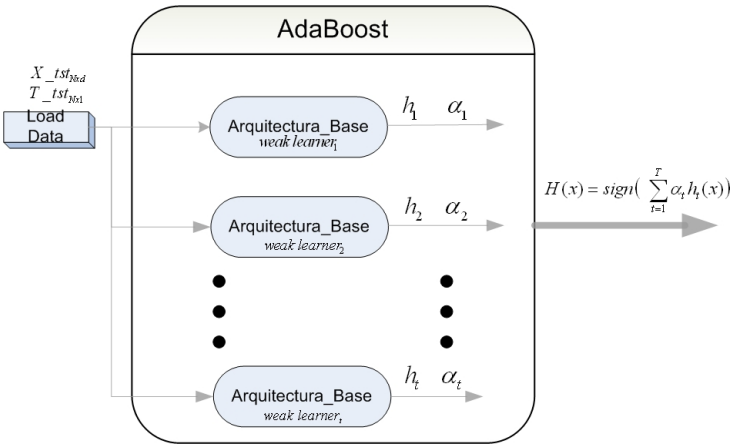



Figura 5.7: Esquema evaluación Adaboost

Con el algoritmo de boosting obtenemos los aprendedores débiles que son máquinas que tiene una tasa de acierto como mínimo superior al 50 %. También obtenemos los parámetros $\alpha$ que representan la importancia de la salida que proporciona esa máquina. A la hora de clasificar las muestras de test no se utilizan en ningún momento las dis-

tribuciones $D$ (ver página 4.2.1), éstas solo se utilizan a la hora de entrenar cada una de los aprendedores.

## Bagging

Esta técnica puede aplicarse tanto a las estructuras vistas en 5.1 como a las comentadas en 5.2. La siguiente figura muestra la forma que presenta una red de este tipo, en la que denominamos *arquitectura base* a un componente que puede ser un componente `Arquitectura_1` o `Arquitectura_2`.

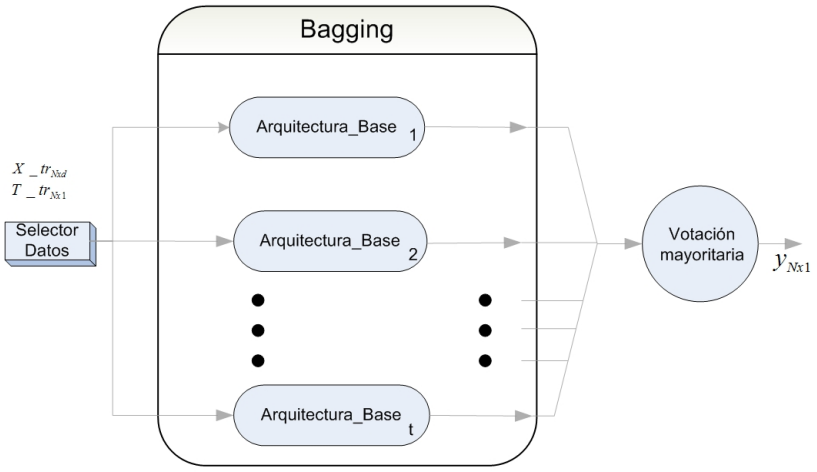



Figura 5.8: Esquema Bagging

En la imagen aparece una función con la etiqueta **Votación mayoritaria**. Esta función de salida lo que hace es sumar para cada muestra la salida que resulta en cada uno de los componentes que puede ser 1 o $-1$ y devuelve el signo de esta suma, que será 1 o $-1$ siempre que el número de componentes sea impar.

En la imagen también aparece lo que denominamos **Selector Datos** que se encarga de realizar el muestreo con reemplazamiento para formar el conjunto de datos con el que será entrenado el componente del sistema.

## 5.4. Fisher-Modular

Como ya explicamos en el apartado 4.2.2 los sistemas modulares son sistemas de múltiples redes en los que cada componente intentar resolver un subproblema o módulo del problema general. Esta arquitectura se presenta en este proyecto únicamente como solución a problemas bidimensionales.

Para esta arquitectura que denominamos *Fisher-Modular* lo que buscamos es separar el conjunto total de datos de entrenamiento en subconjuntos disjuntos. Posteriormente obtenemos un componente por cada uno de estos subconjuntos de datos, en nuestro caso estos componentes se corresponden con un Discriminante Lineal de Fisher (DLF).

De la formación de los subconjuntos va a depender el resultado global de esta arquitectura. Esta separación en subconjuntos debe permitir que la componente que se construya, a partir de los datos de un determinado subconjunto, esté más especializada que una componente global. Aunque también debemos tener en cuenta que esta especialización puede llevar asociada una pérdida de generalización. La siguiente Figura 5.9 muestra los componentes que hemos utilizado para formar esta arquitectura.

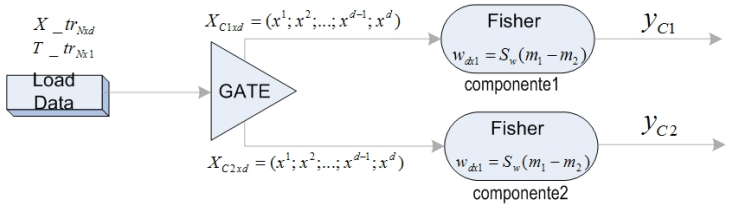



Figura 5.9: Arquitectura Fisher-Modular

El componente denominado **gate** en la Figura 5.9 es el encargado de realizar la separación del conjunto de datos inicial $X_{Nxd}$ en los dos subconjuntos $X_{C1xd}$ y $X_{C2xd}$, los cuales no comparten ningún elemento en común.

Este componente queda definido por un vector cuya orientación define el límite que conforma los subconjuntos. Para el cálculo de este vector podemos optar por utilizar un Discriminante Lineal de Fisher (DLF) o por un Análisis de Componentes Principales (PCA).

Una vez definidos los dos subconjuntos entrenamos cada componente con uno de estos subconjuntos de datos, quedando completamente definida la arquitectura. A la hora de clasificar las muestras de test, el gate decidirá para cada muestra de test, a partir del límite obtenido en el entrenamiento, cual de los componentes de la arquitectura será el encargado de clasificar esa muestra.

### 5.4.1. Obtención del *gate*

Como comentamos anteriormente para el cálculo del gate hemos optado por dos métodos: mediante un DLF o mediante PCA que detallamos a continuación.

- DLF.

  definiendo $\mathbf{z} \perp \mathbf{w}$ y sabiendo que $\mathbf{w} \propto \mathbf{S_W}^{-1} (\mathbf{m_1} - \mathbf{m_2})$

  $$\text{si } g(\mathbf{x_k}) = \mathbf{z^T x_k} \geq 0 \rightarrow \mathbf{x_k} \in C_1$$
  $$\text{si } g(\mathbf{x_k}) = \mathbf{z^T x_k} < 0 \rightarrow \mathbf{x_k} \in C_2$$

  Con $C_1$ y $C_2$ cada uno de los subconjuntos de datos que se forman. De este modo la dirección que limita los dos subconjuntos es la misma dirección que la del vector $\mathbf{w}$. Esta arquitectura la denominamos **Fisher Modular - DLF**

- PCA.

  en el caso de utilizar el Análisis de Componentes Principales el vector que limitará los dos subconjuntos va a ser alguno de los autovectores de la matriz de covarianzas de los datos del conjunto de entrenamiento. En el caso de problemas de 2 variables podremos optar por el autovector asociado al menor o al mayor de los autovalores asociados, obteniendo con ello distintos resultados. Con lo que surgen 2 arquitecturas que denominamos:

  - **Fisher Modular - PCA↑** en la que el límite queda definido por el autovector asociado al mayor (↑) autovalor.

- **Fisher Modular - PCA↓** en la que el límite queda definido por el autovector asociado al menor (↓) autovalor.

En el caso de utilizar un DLF los vectores **z** y **w** definen el **gate**, para el caso de utilizar PCA nos quedaríamos con los autovectores de la matriz de covarianzas para definir el **gate**. Posteriormente se procede a entrenar los otros dos componentes del sistema, mediante el cálculo del DLF para cada unos de los subconjuntos de datos $X_{C1xd}$ y $X_{C2xd}$.

### 5.4.2. Clasificación de las muestras

Para obtener la clasificación para cada una de las muestras se procede de la siguiente manera.

1. Se pasa la muestra por el **gate**, el cual decide de manera unívoca que componente será el encargado de clasificar la muestra.
2. El componente al que se asigna la muestra clasifica la muestra en función de su entrenamiento previo.
3. Sucesivamente se procede de forma secuencial con cada muestra del conjunto.

De esta manera para cada muestra se obtiene una única salida, procedente de uno de los dos componentes de clasificación que componen este sistema modular.

Capítulo 6

# Resultados

En este capítulo se mostrarán los resultados obtenidos por las distintas arquitecturas junto con el tiempo de cómputo para cada una de ellas. Como medida de la eficiencia computacional de las distintas arquitecturas utilizamos el tiempo de cómputo, es decir, el tiempo que emplea el ordenador en realizar una determinada tarea en lugar del coste computacional. El tiempo de cómputo puede depender del ordenador donde se realizan las medidas aunque el coste computacional sea constante. En nuestro caso todas las medidas se han realizado en el mismo ordenador y realizando varias iteraciones para calcular un promedio. Además, junto con cada arquitectura, se discutirán los resultados obtenidos.

En este capítulo también detallaremos los conjuntos de datos utilizados para evaluar las distintas arquitecturas; divididos en dos subconjuntos, los conjuntos de datos *sintéticos* y los conjuntos de datos aportados por el departamento, *reales*.

Por último se realizará un resumen de los resultados obtenidos para el conjunto del trabajo y se extraerán las conclusiones.

## 6.1. Descripción de problemas *reales*

Una vez presentadas las arquitecturas sobre las que realizaremos los test, presentamos en este apartado los problemas sobre los que se realiza la evaluación de las arquitecturas propuestas. En la tabla 6.1 se muestran el número de muestras de entrenamiento y de test y el número de variables de los datos (dimensión) de entrada para cada unos de los problemas presentados.

| Problema | **Muestras entrenamiento** | **Muestras test** | **Dimensión** |
|---|---|---|---|
| **Kwok** | 500 | 10200 | 2 |
| **Phoneme** | 4323 | 1081 | 5 |
| **Ripley** | 250 | 1000 | 2 |
| **Spam** | 3681 | 920 | 57 |

Tabla 6.1: Caracterísiticas de los conjuntos de datos *reales*

## 6.2. Arquitectura_1

Como ya vimos en la sección 5.1 la función discriminante de Fisher, la cual constituye la `Arquitectura_1 - Fisher` y sus variantes, presenta la misma forma que la función discriminante $g(\mathbf{x})$ para el caso de dos clases con idéntica matriz de covarianzas $\mathbf{\Sigma} = \mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$ con densidades de probabilidad gaussianas,

$$g(\mathbf{x}) = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \ln \frac{P(C_1)}{P(C_2)}$$

Para corroborar esto generamos una primera serie de problema sintéticos que presentamos a continuación.

### Problemas sintéticos - Descripción

A continuación comentaremos las características con las que ha sido generado cada uno de los conjuntos de datos.

1. **Sintético-1**

   - Hipótesis de normalidad.
   - Hipótesis de equicovarianza.
   - Probabilidades a priori iguales.

2. **Sintético-2**

   - Hipótesis de normalidad.
   - Hipótesis de equicovarianza.
   - Probabilidades a priori diferentes

   ejemplo:

   $$\mathbf{\Sigma} = \begin{pmatrix} 1{,}5 & 0 \\ 0 & 1{,}2 \end{pmatrix} \quad P(C_1) = 0{,}7 \quad P(C_2) = 0{,}3$$

3. **Sintético-3**

- Hipótesis de normalidad.
- Diferente matriz de covarianza para cada clase.
- Probabilidades a priori diferentes

ejemplo:

$$\mathbf{\Sigma}_1 = \begin{pmatrix} 1{,}5 & 0 \\ 0 & 1{,}2 \end{pmatrix} \quad \mathbf{\Sigma}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1{,}3 \end{pmatrix} \quad P(C_1) = 0{,}7 \quad P(C_2) = 0{,}3$$

Para todos estos conjuntos de datos sintéticos el número de patrones de entrenamientos es 10000 y el número de patrones de test es de 2000. El número de variables de estos ejemplos es 2 para todos los problemas.

### 6.2.1. Resultados

En las siguientes tablas (6.2) y (6.3) se pueden ver cuales han sido las tasas de error para las distintas variantes de esta arquitectura. Los resultados de esta arquitectura son deterministas por lo que no es necesario hacer un promediado de los resultados, la tasa de error es evaluada sobre el conjunto de test y estos valores están representados en %.

| | Error de clasificación en % | | |
|---|---|---|---|
| Arquitectura | **sintetico_1** | **sintetico_2** | **sintetico_3** |
| **Arquitectura_1 - Fisher (y0)** | **17.2** | 14.4 | 12.4 |
| **Arquitectura_1 - Fisher (y0')** | **17.2** | **11.6** | 13.6 |
| **Arquitectura_1 - Fisher (y0")** | **17.2** | **11.6** | 11.6 |
| **Función discriminante óptima** | **17.2** | **11.6** | **11.2** |
| **Single Layer Network** | 17.39 | 11.71 | 11.41 |

Tabla 6.2: Evaluación Arquitectura_1 - Problemas Sintéticos

| | Error de clasificación en % | | | |
|---|---|---|---|---|
| Arquitectura | **Kwok** | **Phoneme** | **Ripley** | **Spam** |
| **Arquitectura_1 - Fisher (y0)** | 22.13 | 23.77 | 10.8 | 10.65 |
| **Arquitectura_1 - Fisher (y0')** | 21.39 | 23.60 | 10.8 | 13.36 |
| **Arquitectura_1 - Fisher (y0")** | 20.66 | **22.75** | **10.7** | 10.20 |
| **Single Layer Network** | **20.17** | 23.10 | 11.17 | **8.82** |

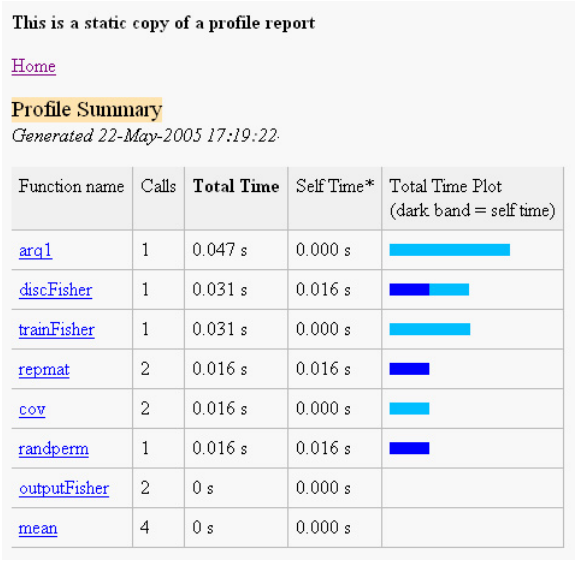
Tabla 6.3: Evaluación Arquitectura_1 - Problemas *reales*

### 6.2.2. Tiempo de cómputo

Como ya mencionamos en 5.2.2 al realizar una optimización del proceso de proyección, nos vamos a valer de una herramienta *MATLAB* para mostrar lo tiempos de cómputo de esta arquitectura de una manera completa.

Mostraremos unas figuras obtenidas mediante el uso de la función `profile` de MATLAB en las que mostraremos el tiempo de cómputo de entrenamiento más evaluación para cada una de estas arquitecturas, para un problema genérico. Y por último mostraremos una tabla conjunta con los tiempos de cómputo para cada arquitectura en cada uno de los problemas propuestos.

El tiempo de cómputo de las arquitecturas esta fijado, salvo para el caso del SLP que depende del número de épocas de entrenamiento. Sin embargo, este número de épocas lo vamos a fijar a un valor de 80 épocas, ya que un número mayor no implica una reducción del error comparable al aumento en el tiempo de cómputo.



**This is a static copy of a profile report**

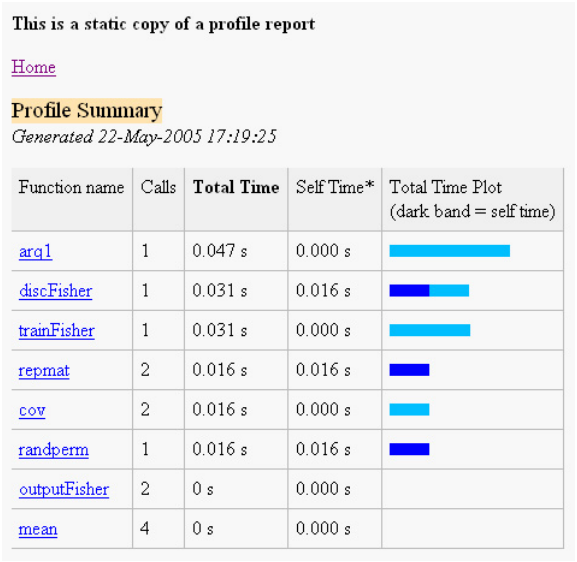
Home

**Profile Summary**

*Generated 22-May-2005 17:19:22*

| Function name | Calls | **Total Time** | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| arq1 | 1 | 0.047 s | 0.000 s | |
| discFisher | 1 | 0.031 s | 0.016 s | |
| trainFisher | 1 | 0.031 s | 0.000 s | |
| repmat | 2 | 0.016 s | 0.016 s | |
| cov | 2 | 0.016 s | 0.000 s | |
| randperm | 1 | 0.016 s | 0.016 s | |
| outputFisher | 2 | 0 s | 0.000 s | |
| mean | 4 | 0 s | 0.000 s | |

Figura 6.1: Tiempo de cómputo arq1 - y0, ejemplo



**This is a static copy of a profile report**

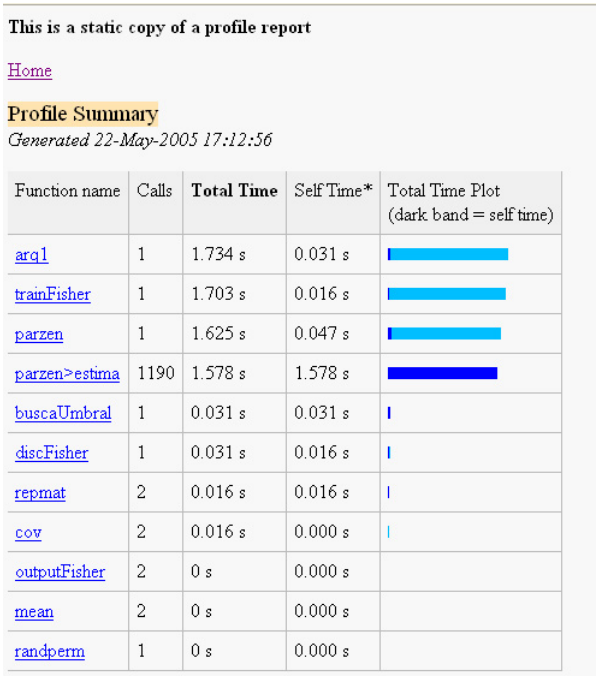
Home

**Profile Summary**

*Generated 22-May-2005 17:19:25*

| Function name | Calls | **Total Time** | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| arq1 | 1 | 0.047 s | 0.000 s | |
| discFisher | 1 | 0.031 s | 0.016 s | |
| trainFisher | 1 | 0.031 s | 0.000 s | |
| repmat | 2 | 0.016 s | 0.016 s | |
| cov | 2 | 0.016 s | 0.000 s | |
| randperm | 1 | 0.016 s | 0.016 s | |
| outputFisher | 2 | 0 s | 0.000 s | |
| mean | 4 | 0 s | 0.000 s | |

Figura 6.2: Tiempo de cómputo arq1 - y0', ejemplo



**This is a static copy of a profile report**

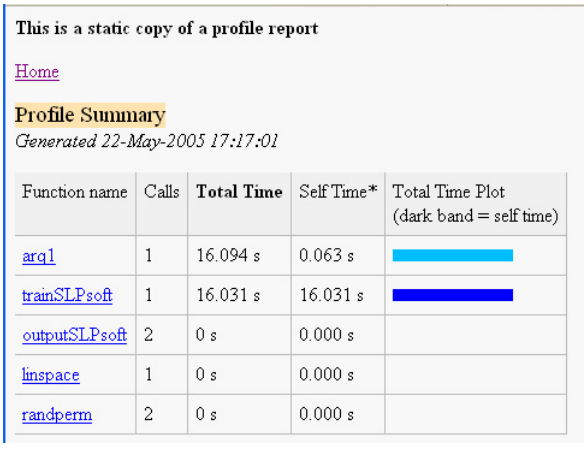
Home

**Profile Summary**

*Generated 22-May-2005 17:12:56*

| Function name | Calls | **Total Time** | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| arq1 | 1 | 1.734 s | 0.031 s | |
| trainFisher | 1 | 1.703 s | 0.016 s | |
| parzen | 1 | 1.625 s | 0.047 s | |
| parzen>estima | 1190 | 1.578 s | 1.578 s | |
| buscaUmbral | 1 | 0.031 s | 0.031 s | |
| discFisher | 1 | 0.031 s | 0.016 s | |
| repmat | 2 | 0.016 s | 0.016 s | |
| cov | 2 | 0.016 s | 0.000 s | |
| outputFisher | 2 | 0 s | 0.000 s | |
| mean | 2 | 0 s | 0.000 s | |
| randperm | 1 | 0 s | 0.000 s | |

Figura 6.3: Tiempo de cómputo arq1 - y0", ejemplo



**This is a static copy of a profile report**

Home

**Profile Summary**

*Generated 22-May-2005 17:17:01*

| Function name | Calls | **Total Time** | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| arq1 | 1 | 16.094 s | 0.063 s | |
| trainSLPsoft | 1 | 16.031 s | 16.031 s | |
| outputSLPsoft | 2 | 0 s | 0.000 s | |
| linspace | 1 | 0 s | 0.000 s | |
| randperm | 2 | 0 s | 0.000 s | |

Figura 6.4: Tiempo de cómputo SLP, ejemplo

Las siguientes tablas muestran el tiempo de cómputo para las distintas etapas, en cada una de las arquitecturas y para cada uno de los problemas, medido en segundos. El valor $<$ 0,000 indica que el tiempo de evaluación es próximo a 0 segundos y que es despreciable respecto al tiempo de entrenamiento.

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.031 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0')** | 0.031 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0")** | 0.209 | $<$ 0.000 |
| **Single Layer Network** | 1.480 | $<$ 0.000 |

Tabla 6.4: Tiempo de cómputo - problema kwok

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.047 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0')** | 0.047 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0")** | 1.292 | $<$ 0.000 |
| **Single Layer Network** | 12.48 | $<$ 0.000 |

Tabla 6.5: Tiempo de cómputo - problema phoneme

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.031 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0')** | 0.031 | $<$ 0.000 |
| **Arquitectura_1 - Fisher (y0")** | 0.172 | $<$ 0.000 |
| **Single Layer Network** | 0.750 | $<$ 0.000 |

Tabla 6.6: Tiempo de cómputo - problema ripley

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.078 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0')** | 0.078 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0'')** | 1.117 | $< 0.000$ |
| **Single Layer Network** | 12.123 | $< 0.000$ |

Tabla 6.7: Tiempo de cómputo - problema spam

### 6.2.3. Conclusiones

**Problemas sintéticos.** Como cabía esperar la tabla (6.2) nos muestra que el discriminante de Fisher es óptimo para el caso de funciones de densidad gaussianas e hipótesis de equicovarianza. También vemos que la estimación de las densidades de probabilidad de los datos proyectados no conlleva un empeoramiento de los resultados aunque sí en cuanto al tiempo de entrenamiento. Además, cuando la frontera óptima es cuadrática, como ocurre con el problema *sintetico_3*, sigue obteniendo unos resultados aceptables. Por tanto, en estos problemas nuestra arquitectura obtiene mejores resultados que un SLP y con un tiempo de entrenamiento claramente inferior.

**Problemas *reales*.** Como vemos en la tabla (6.3) la `Arquitectura_1 - Fisher y0''` obtiene en todos los casos mejores resultados aunque eso sí, empleando un tiempo de entrenamiento mayor. Comparando nuestra arquitectura con el SLP no aparecen resultados claros a favor de una u otra arquitectura salvo que se considere el tiempo de cómputo, donde el SLP emplea mucho más tiempo. Este mayor tiempo de cómputo es debido al modo de entrenamiento del SLP.

**Tiempo de computo.** Como vimos en las anteriores figuras las arquitecturas que realizan el cálculo del discriminante de Fisher invierten un menor tiempo en la obtención del valor de los pesos en comparación con un SLP. Aunque al realizar el cálculo de las densidades de probabilidad condicionadas en `Arquitectura_1 - Fisher y0''` el tiempo de cómputo aumenta considerablemente aún empleamos un 90 % menos de tiempo que en el caso de un SLP y obteniendo unos resultados semejantes. El mayor coste computacional

de un SLP respecto al cálculo del DLF es debido a la forma de entrenamiento del SLP, que se realiza por descenso de gradiente estocástico: muestra a muestra. Además, una vez se presentan todas las muestras, lo que constituye una época, se altera el orden de presentación de las mismas al SLP para cada una de estas épocas; lo que influye también en el mayor coste computacional. Sin embargo, a diferencia de la `Arquitectura_1` el SLP integra el cálculo del umbral o sesgo de la decisión. Esta forma de entrenamiento es la que mejores resultados proporciona para el caso de un SLP. El cálculo del DLF es un cálculo en bloque cuyo tiempo de cómputo depende de forma significativa del número de variables de los datos que forman el problema y no tanto del número de ejemplos. También es posible realizar un entrenamiento bloque en el caso de un SLP, esta opción se ha desechado ya que al realizar pruebas con un entrenamiento bloque se obtenía una tasa de error en promedio superior a la tasa de error de un DLF o un SLP secuencial. El tiempo de cómputo de un SLP bloque era similar al cálculo del DLF, pero dada su elevada tasa de error se desestimó su utilización en este proyecto.

El tiempo de cálculo de las densidades de probabilidad mediante ventanas de *Parzen*, que es utilizado para obtener el umbral de decisión en una de las variantes de la `Arquitectura_1` depende del número de ejemplos del problema, al ser calculadas sobre variables unidimensionales. Este cálculo del umbral permite mejorar las capacidades del DLF como clasificador, ya que este no incluye el cálculo del sesgo o umbral como en el caso de un SLP. En el caso de problemas de baja dimensionalidad (*kwok* y *ripley*) el cálculo del umbral permite una mayor precisión en la clasificación ya que se puede asemejar a una dimensión extra sobre la que clasificar las muestras. En este trabajo se observa sobre todo en la tasa de error del problema *kwok* que se reduce al introducir el cálculo del umbral mediante ventanas de Parzen (Arquitectura_1 - Fisher (y0")).

## 6.3. Arquitectura_2

Como ya hemos comentado con esta arquitectura queremos aumentar las prestaciones de la anterior aumentando la complejidad. Calcularemos sucesivos discriminantes de Fisher para posteriormente realizar una combinación lineal de los mismos. La siguiente imagen nos muestra el desarrollo de este método constructivo.

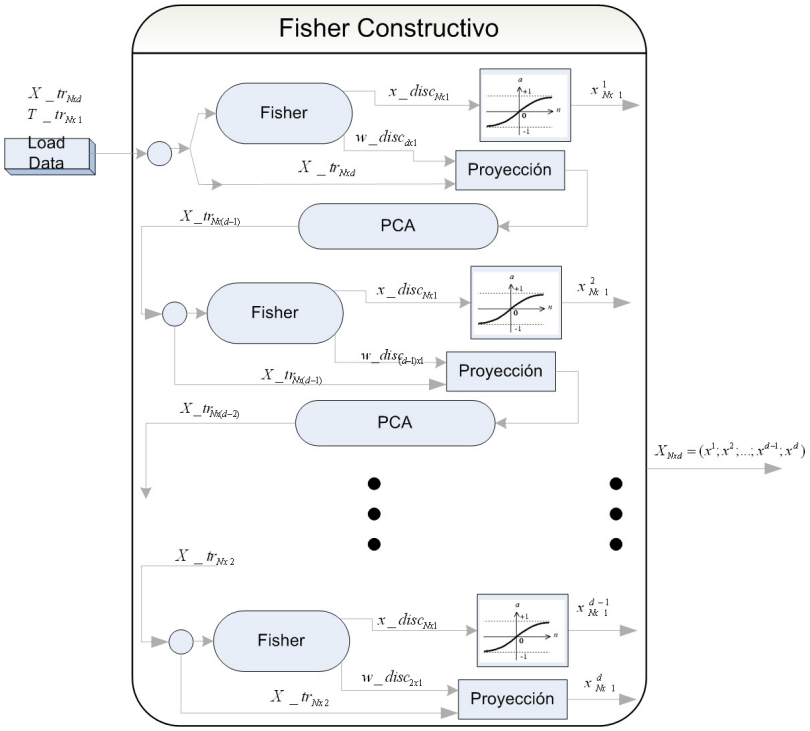



Figura 6.5: Método constructivo

La combinación lineal de los sucesivos discriminantes obtenidos se realiza mediante un SLP. El esquema completo de la `Arquitectura_2` se presenta en la Figura 6.6

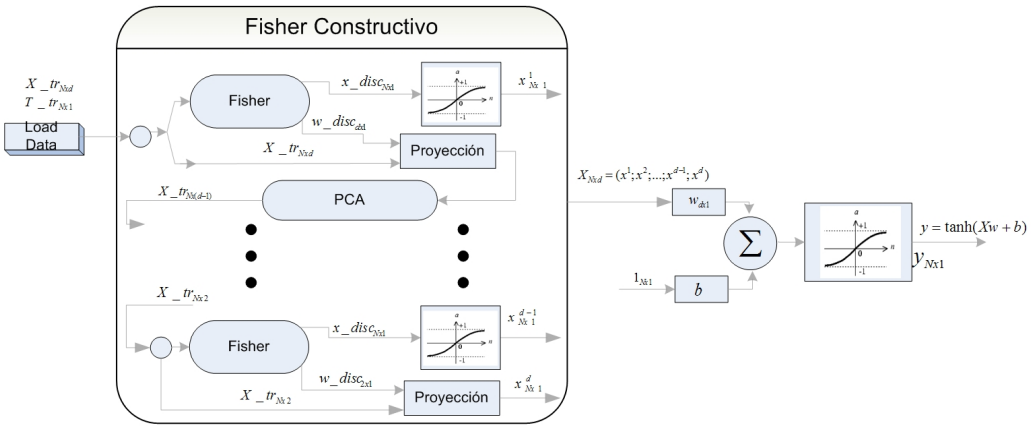



Figura 6.6: Esquema completo Arquitectura_2

### 6.3.1. Resultados

En las tablas de resultados se muestra el porcentaje de muestras de test mal clasificadas junto con el valor que toman las distintas opciones del método constructivo presentadas en 5.2.4 como bloques opcionales.

#### Resultados Arquitectura_2

Conjunto de datos ***Kwok***

| | **norm** | ’0’ | | | | ’1’ | | | |
|---|---|---|---|---|---|---|---|---|---|
| **n variables** | **linealidad** | ’yes’ | | ’no’ | | ’yes’ | | ’no’ | |
| **n - 1 disc** | **escalado** | ’yes’ | ’no’ | ’yes’ | ’no’ | ’yes’ | ’no’ | ’yes’ | ’no’ |
| 1 | | 20.73 | 22.28 | 22.07 | 22.18 | 21.35 | 21.98 | 21.74 | 22.16 |
| 2 | | 19.82 | 21.08 | 20.15 | **19.76** | 20.05 | 20.73 | 19.92 | 21.26 |

Tabla 6.8: Evaluación Arquitectura_2 - problema kwok

Conjunto de datos ***Ripley***

| | **norm** | '0' | | | | '1' | | | |
|---|---|---|---|---|---|---|---|---|---|
| **n variables** | **linealidad** | 'yes' | | 'no' | | 'yes' | | 'no' | |
| **n - 1 disc** | **escalado** | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' |
| 1 | | 11.07 | 10.79 | 10.46 | 10.83 | **10.4** | 11.01 | 10.7 | 10.66 |
| 2 | | 10.64 | 11.48 | 10.51 | 10.76 | **10.4** | 10.64 | 10.6 | 10.55 |

Tabla 6.9: Evaluación Arquitectura_2 - problema ripley

Conjunto de datos ***Phoneme***

| | **norm** | '0' | | | | '1' | | | |
|---|---|---|---|---|---|---|---|---|---|
| **n variables** | **linealidad** | 'yes' | | 'no' | | 'yes' | | 'no' | |
| **n - 1 disc** | **escalado** | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' |
| 1 | | 23.86 | 22.66 | 26.27 | 24.05 | 24.11 | 23.49 | 25.24 | 23.77 |
| 2 | | 23.12 | 22.29 | 24.05 | 23.77 | 24.14 | 22.75 | 25.34 | 23.95 |
| 3 | | 22.75 | **22.20** | 24.69 | 23.86 | 23.68 | 22.38 | 24.60 | 22.94 |
| 4 | | 23.77 | 22.26 | 24.88 | 23.95 | 25.06 | 22.47 | 24.88 | 23.86 |
| 5 | | 23.4 | 22.27 | 24.88 | 23.95 | 24.23 | 22.38 | 25.71 | 23.77 |

Tabla 6.10: Evaluación Arquitectura_2 - problema phoneme

Conjunto de datos ***Spam***

| | **norm** | '0' | | | | '1' | | | |
|---|---|---|---|---|---|---|---|---|---|
| **n variables** | **linealidad** | 'yes' | | 'no' | | 'yes' | | 'no' | |
| **n - 1 disc** | **escalado** | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' | 'yes' | 'no' |
| 1 | | 10.97 | 10.32 | 11.08 | 10.32 | 10.21 | 10.43 | 10.32 | 10.97 |
| 2 | | 10.65 | 10.32 | 10.65 | 10.21 | 10.13 | 10.43 | 10.34 | 10.65 |
| 3 | | 10.65 | 10.54 | 10.86 | 10.32 | 10.86 | 10.32 | 10.97 | 10.82 |
| 4 | | 10.63 | 10.28 | 10.86 | 9.67 | 10.76 | 10.21 | 10.43 | 10.21 |
| 5 | | 10.63 | 10.32 | 10.86 | 9.89 | 10.58 | 10.21 | 10.54 | 10.30 |
| 10 | | 10.60 | 10.10 | 10.86 | 10.04 | 10.19 | 10.76 | 10 | 10.26 |
| 15 | | 10.65 | 10.54 | 10.86 | 9.43 | 10.26 | 10.30 | 9.89 | 9.56 |
| 20 | | 10.63 | 10.10 | 10.86 | 9.89 | 10.10 | 10.71 | 9.41 | 9.80 |
| 25 | | 10.65 | 9.78 | 10.86 | 9.47 | 9.08 | 10.39 | 9.28 | 9.23 |
| 30 | | 10.63 | 9.65 | 10.86 | 9.58 | 9.43 | 10 | 9.84 | 8.84 |
| 35 | | 10.78 | 9.06 | 10.86 | 9.36 | 9.65 | 9.5 | 8.91 | 8.86 |
| 40 | | 10.65 | 9.50 | 10.86 | 9.26 | 9.86 | 9.5 | 9.13 | 8.39 |
| 45 | | 10.65 | 9.56 | 10.86 | 8.63 | 9.65 | 8.47 | 9.10 | **7.67** |
| 50 | | 10.63 | 9.26 | 10.76 | 9.10 | 9.93 | 8.76 | 8.52 | 8.21 |
| 55 | | 10.60 | 8.58 | 10.86 | 8.73 | 9.58 | 8.10 | 8.91 | 7.91 |
| 57 | | 10.63 | 8.13 | 10.86 | 7.84 | 9.04 | 7.97 | 8.30 | 8.10 |
| Record | | 10.58 | 8.00 | 10.65 | 7.84 | 8.97 | 7.97 | 8.23 | 7.67 |

Tabla 6.11: Evaluación Arquitectura_2 - problema spam

### 6.3.2. Tiempo de cómputo

En la siguiente tabla se resume el tiempo de cómputo que conlleva la realización del método constructivo que hemos denominado `Arquitectura_2`, es decir, el tiempo de entrenamiento para cada uno de los conjuntos de datos que empleamos en la realización de este trabajo.

En la tabla 6.12 se muestran los tiempos medidos en segundos y también mostramos el porcentaje de tiempo sobre el total que conlleva la realización de las etapas más importantes del método constructivo que son: el cálculo del **Discriminante Lineal de Fisher** (DLF), la realización de la **proyección** de los datos, el **Análisis de componentes Principales** (PCA) y por último el entrenamiento del **SLP**.

| *Problema* | **Tiempo Total** | **Fisher** (%/total) | **Proyección** (%/total) | **PCA** (%/total) | **SLP** (%/total) |
|---|---|---|---|---|---|
| **Kwok** | 3.032 | 0.109 (3.6 %) | 0.016 (0.5 %) | 0.094 (3.1 %) | 2.813 (92.3 %) |
| **Phoneme** | 32.422 | 0.125 (0.4 %) | 0.016 (0.1 %) | 7.750 (24 %) | 24.531 (75.6 %) |
| **Ripley** | 1.626 | 0.094 (5.8 %) | 0.016 (1 %) | 0.094 (5.8 %) | 1.422 (87.5 %) |
| **Spam** | 355.844 | 1.219 (0.3 %) | 0.750 (0.2 %) | 331.500 (93.2 %) | 22.375 (6.3 %) |

Tabla 6.12: Tiempo de entrenamiento - Arquitectura_2

Se observa que el tiempo de entrenamiento está determinado fuertemente por dos procesos del método constructivo que son el análisis de componentes principales y por la obtención del SLP para combinar las salidas de cada discriminante lineal de Fisher.

Cabe señalar también que en los problemas con mayor número de variables phoneme y spam cobra mayor importancia el análisis de componentes principales. Ya que para construir la tabla, siendo $d$ el número de variables del problema, se realizan $d-1$-discriminantes, $d-1$-proyecciones y $d-1$-análisis de componentes principales. Obteniéndose para cada problema un único SLP cuyo tiempo de cómputo depende del número de muestras y número de variables del problema dado. Debido a su modo de aprendizaje muestra a muestra como ya comentamos en 6.2.2.

La tabla anterior medía el tiempo de construcción de la `Arquitectura_2` y la tabla 6.13 muestra el tiempo empleado en el cálculo de las salidas de esta arquitectura para las muestras de test, es decir, el tiempo de evaluación en comparación con el tiempo de

entrenamiento.

| *Problema* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Kwok** | 3.032 | 0.031 |
| **Phoneme** | 32.422 | 0.031 |
| **Ripley** | 1.626 | 0.031 |
| **Phoneme** | 355.844 | 0.328 |

Tabla 6.13: Tiempo de cómputo - Arquitectura_2

Se observa que para los tres primeros problemas el tiempo de evaluación es prácticamente despreciable. Sin embargo respecto al último problema se observa una diferencia de tiempo notable debido al gran número de variables que componen el problema.

### 6.3.3. Conclusiones

Relacionando las tablas de resultados anteriores con las que muestran los resultados obtenidos para la `Arquitectura_1` se observa que para cada uno de los problemas propuestos se obtiene un porcentaje de error inferior. Por un lado era de esperar al aumentar la complejidad, pero no se cubren las expectativas de esta arquitectura. Esta complejidad implica un mayor tiempo de cómputo que para alguna aplicación específica pudiera no ser asumible en relación a la leve mejora en el porcentaje de error de clasificación.

Lo que si se puede observar en el caso del problema *spam* es que la mejora relativa es mayor. Esto se explica entendiendo que este problema cuenta con un total de 57 variables de entrada y llevar toda la información de clasificación a una única dirección como hace el DLF no es óptimo y utilizando este método constructivo conseguimos obtener más información para la clasificación con los sucesivos discriminantes.

## 6.4. Comités

Como vimos en 5.3 con estas arquitecturas nos basamos en el principio *Divide y Vencerás*. Construimos una arquitectura más compleja mediante la combinación de arquitecturas base, que serán del tipo `Arquitectura_1` presentada en 5.1 o también del tipo `Arquitectura_2` presentada en 5.2 en el caso de utilizar *Bagging*.

### 6.4.1. Resultados y Conclusiones Bagging

La forma de obtener las distintas máquinas que compondrán el sistema de múltiples redes utilizando Bagging es presentar a cada una de ellas un conjunto diferente de datos. Este conjunto se obtiene por un muestreo con reemplazamiento en el que podemos variar el número de ejemplos que compondrán el subconjunto de datos que presentaremos a la máquina. La combinación de cada una de las máquinas se realiza por votación mayoritaria.

Como ya hemos comentado para obtener buenos resultados necesitamos que las máquinas sean diferentes para que el conjunto total pueda ser susceptible de mejorar los resultados. Pero los clasificadores que obtenemos, mediante la utilización del método de Bagging, no presentan unas fronteras de decisión complementarias por lo que no se mejoran los resultados obtenidos por una única red. En las tablas de resultados finales apartado 6.6 solo aparecen resultados para una única configuración estándar ya que no encontramos ningún patrón ni configuración destacable. Esta configuración estándar esta formada por 25 componentes entrenadas mediante conjuntos formados por un 40 % de la muestras de entrenamiento obtenidos mediante un muestreo con reemplazamiento.

### 6.4.2. Resultados y Conclusiones AdaFisher

La forma de enfatizar las muestras con mayor error, como propone el esquema Adaboost (4.2.1), puede variar de un algoritmo a otro. En el caso de las redes tipo SLP, MLP o RBF; el énfasis puede simplemente añadirse en la función de coste como un término que hará que la adaptación de los pesos varíe más o menos dependiendo de la muestra

que estemos tratando en cada instante. Esta forma de incluir el énfasis es equivalente a hacer un remuestreo.

En el caso del DLF, dada su expresión

$$\mathbf{w} \propto {\mathbf{S_W}}^{-1} (\mathbf{m}_1 - \mathbf{m}_2) \tag{6.1}$$

habrá que realizar el énfasis de forma un poco diferente para poder llevar a cabo ese remuestreo. En [Kuncheva, 2004] se propone la forma de ponderar las muestras y su efecto sobre la media y la matriz de covarianzas

$$\mathbf{m_j} = \sum_{i=1}^{N_j} D(i)\mathbf{x_j^{(i)}} \tag{6.2}$$

$$\mathbf{S_w^j} = \sum_{i=1}^{N_j} (D(i)\mathbf{x_j^{(i)}} - \mathbf{m}_j)(\mathbf{x_j^{(i)}} - \mathbf{m}_j)^T \tag{6.3}$$

donde $D(i)$ es el énfasis o la ponderación que se da a la $i$-ésima muestra.

Se han realizado varias pruebas con distintos problemas en las que no se han obtenido mejoras significativas en cuanto a la tasa de acierto. Si bien la teoría de Boosting dice que combinando aprendedores débiles, es decir, con una tasa de acierto ligeramente superior a 0.5, se pueden crear conjuntos de redes con muy buenas prestaciones. Ciertamente, multitud de artículos demuestran que los métodos de Boosting y en especial el RealAdaBoost consiguen resultados excepcionalmente buenos, comparables a las SVMs ([Schapire, 1999],[Rätsch *et al.*, a]). Sin embargo, y aunque parezca obvio decirlo, para que la combinación de clasificadores débiles mejore las prestaciones de cualquiera de ellos, es necesario que todos sean diferentes produciendo diversidad de soluciones. Esto ocurre en clasificadores lineales donde el entrenamiento se realiza muestra a muestra y por tanto la solución final depende del orden de presentación de las muestras y de la inicialización del algoritmo. En el caso del DLF, la solución no depende ni de las muestras ni de la inicialización del clasificador, dado que no es necesaria. El problema surge del hecho de que el DLF, a pesar de la función de énfasis que se aplica al cálculo de las medias y a

las matrices de covarianza, obtiene clasificadores prácticamente idénticos. Por tanto, la combinación de este conjunto de aprendedores débiles no mejora la clasificación final.

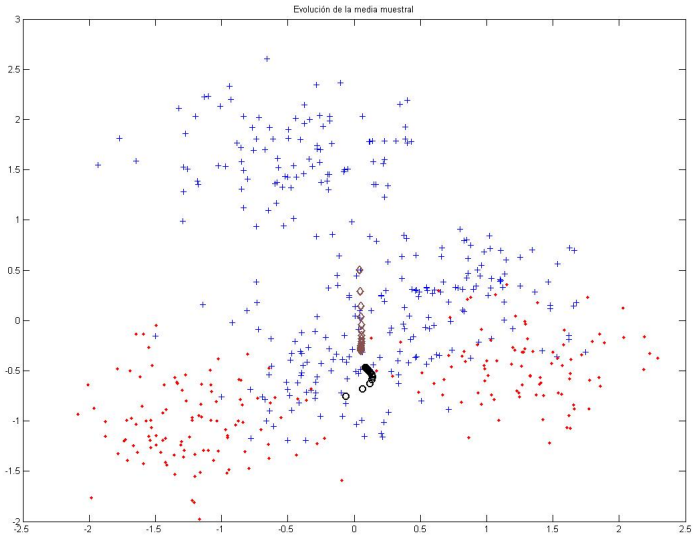



Figura 6.7: Evolución de las medias aplicando énfasis

Las figuras 6.7 y 6.8 muestran como la media muestral va variando gracias al énfasis aplicado. Sin embargo, esta variación no es lo suficientemente grande como para modificar de forma significativa el clasificador, y por tanto, no se introduce la diversidad necesaria para mejorar las prestaciones del conjunto. La figura 6.9 muestra dos aprendedores diferentes (el primero y el último), y cómo la orientación cambia de forma muy leve.

La combinación de estos aprendedores débiles para construir el sistema global no aporta mejores resultados que en el caso de utilizar un único discriminante como presentamos con la `Arquitectura_1`. Igualmente en la tabla resumen de resultados de la página 95 se incluyen los resultados correspondientes a una configuración formada por 25 aprendedores débiles entrenados utilizando la técnica de boosting.

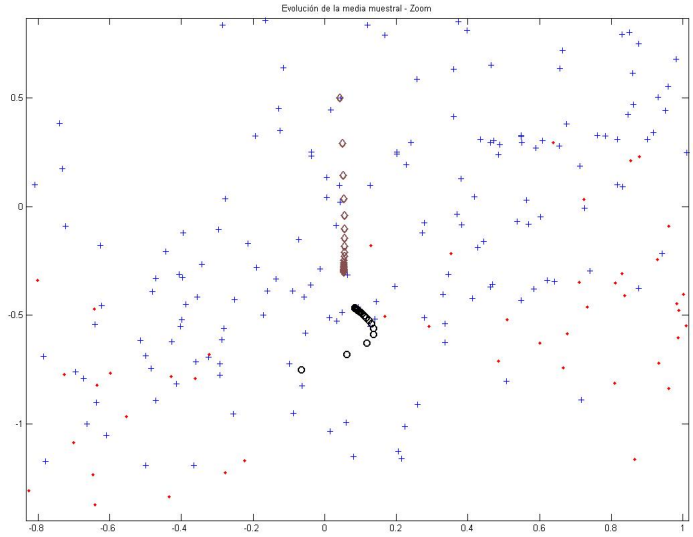



Figura 6.8: ZOOM Evolución de las medias aplicando énfasis

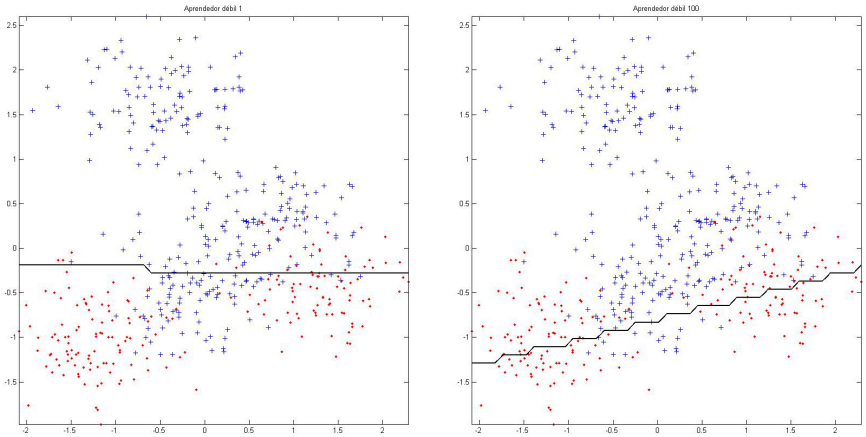



Figura 6.9: Ejemplo aprendedores débiles

## 6.5. Fisher Modular

Como vimos en 5.4 con esta arquitectura nos basamos en el principio *Divide y Vencerás*. Conformamos una arquitectura utilizando componentes especializados en clasificar un subconjunto de muestras, en lugar de utilizar un único componente que se encarge de clasificar el conjunto global de ejemplos como en el caso de la `Arquitectura_1` y de la `Arquitectura_2`.

### 6.5.1. Resultados

En la siguiente tabla se muestran los resultados obtenidos aplicando esta arquitectura. Para el caso de los problemas *kwok* y *ripley*.

**Problemas Reales**

| | Error de clasificación en % | |
|---|---|---|
| Arquitectura | **Kwok** | **Ripley** |
| **Fisher Modular - DLF** | 13.95 | **8.5** |
| **Fisher Modular - PCA↑** | 13.2 | 9.6 |
| **Fisher Modular - PCA↓** | **12.62** | 11.7 |

Tabla 6.14: Evaluación Fisher Modular - Problemas *reales*

Las siguientes figuras 6.10 y 6.11 muestran de una manera gráfica los resultados obtenidos por esta arquitectura. Las figuras representan la frontera de decisión que se define al entrenar la red y con la cual realizamos la clasificación de las muestras.

En las figuras se observan las muestras pertenecientes a una y otra clase de datos y en negro se observa la frontera de decisión que define la arquitectura. Se observa que la frontera de decisión es poco suave similar al tipo de fronteras que se consiguen con los árboles de decisión. Hay métodos para regularizar árboles como el algoritmo *random-forest*.

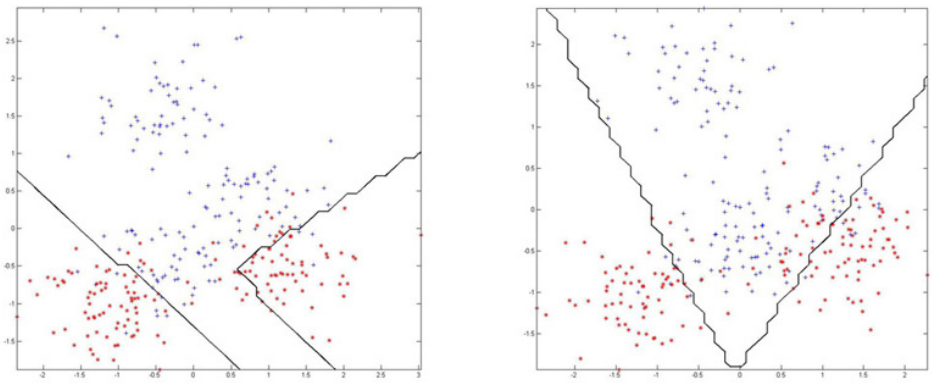


Figura 6.10: Ejemplos frontera de decisión - problema Kwok

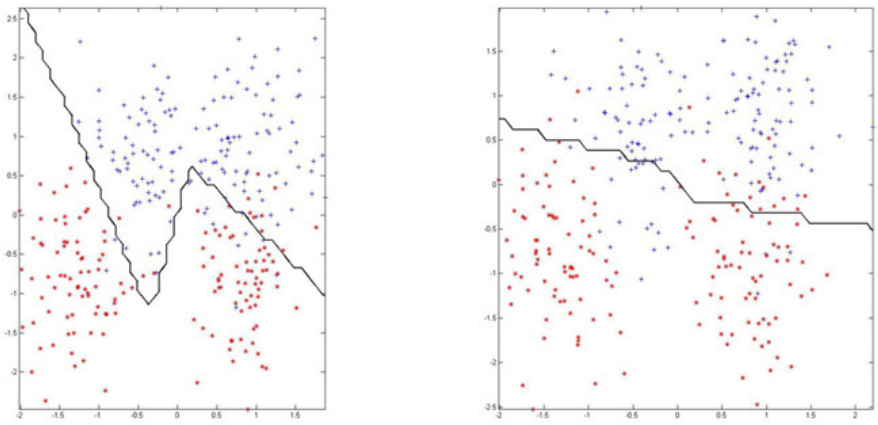


Figura 6.11: Ejemplos frontera de decisión - problema Ripley

### 6.5.2. Tiempo de Cómputo

El tiempo de entrenamiento o de parametrización de los componentes y parámetros que conforman este sistema de múltiples redes se resume en la siguiente tabla 6.15.

| *Arquitectura* | **Problema** | **Tiempo Total** | **Gate (%/total)** | **Fisher (%/total)** |
|---|---|---|---|---|
| **Fisher Modular - DLF** | Kwok | 0.344 | 0.078 (22.7 %) | 0.261 (75.9 %) |
| **Fisher Modular - DLF** | Ripley | 0.281 | 0.078 (27.7 %) | 0.198 (70.5 %) |
| **Fisher Modular - PCA** | Kwok | 0.359 | 0.027 (7.5 %) | 0.328 (91.4 %) |
| **Fisher Modular - PCA** | Ripley | 0.281 | 0.027 (9.6 %) | 0.25 (89 %) |

Tabla 6.15: Tiempo de Entrenamiento - Fisher-Modular

En la tabla anterior se observa que el tiempo de entrenamiento es sensiblemente inferior si comparamos los tiempos totales con los obtenidos por la `Arquitectura_2` (ver tabla 6.13).

La siguiente tabla nos muestra el tiempo de evaluación de este sistema de múltiples redes, el cual es prácticamente despreciable para los problemas tratados.

| *Arquitectura* | **Problema** | **Tiempo Evaluación** |
|---|---|---|
| **Fisher Modular - DLF** | Kwok | 0.016 |
| **Fisher Modular - DLF** | Ripley | 0.016 |
| **Fisher Modular - PCA** | Kwok | 0.016 |
| **Fisher Modular - PCA** | Ripley | 0.016 |

Tabla 6.16: Tiempo de Evaluación - Fisher-Modular

### 6.5.3. Conclusiones

Como vemos en la tabla 6.15 los resultados obtenidos por este sistema modular son claramente mejores que los resultados obtenidos por la `Arquitectura_1` y por la `Arquitectura_2` mostrados en 6.2 y en 6.3.1 respectivamente. Esta mejora se consigue gracias a la combinación de dos clasificadores que se complementan. Como ya comentamos en el apartado relativo a los comités (6.4), esta combinación de elementos complementarios es la que aporta la diversidad necesaria que nos faltó para las arquitecturas

previas y que permite mejorar de forma notable nuestro clasificador final.

Además los tiempos de cómputo son claramente inferiores a los de la `Arquitectura_2` y solo ligeramente superiores a los de la `Arquitectura_1`.

Sin embargo, la arquitectura que denominamos `Fisher-Modular` solo ha sido implementada para trabajar con problemas de dos dimensiones como son kwok y ripley; y no para los otros problemas presentados phoneme y spam.

En ampliaciones futuras de este trabajo se podría abordar el extender este sistemas de múltiples redes a problemas con un mayor número de variables, bien aumentando el número de gates o el número de componentes base que conformarían el sistema.

## 6.6. Resultados y Conclusiones finales

En este apartado vamos a realizar un resumen de los resultados obtenidos por las distintas arquitecturas propuestas con el objetivo de clarificar los resultados globales obtenidos y posteriormente resumimos las conclusiones obtenidas.

Se agrupan los resultados por problemas y arquitecturas y se mostrarán en tablas en las que van a aparecer los mejores resultados obtenidos por cada una de las arquitecturas propuestas.

**Resultados globales de error**

| | Error de clasificación en % | | | |
|---|---|---|---|---|
| Arquitectura | **Kwok** | **Phoneme** | **Ripley** | **Spam** |
| **Arquitectura_1 - Fisher (y0)** | 22.13 | 23.77 | 10.8 | 10.65 |
| **Arquitectura_1 - Fisher (y0')** | 21.39 | 23.60 | 10.8 | 13.36 |
| **Arquitectura_1 - Fisher (y0")** | 20.66 | 22.75 | 10.7 | 10.20 |
| **Single Layer Network** | 20.17 | 23.10 | 11.17 | 8.82 |
| **Arquitectura_2** | 19.76 | **22.2** | 10.4 | **7.67** |
| **Bagging** (25 apren - 40 %) | 20.27 | 23.31 | 10.7 | 10.52 |
| **Boosting** (25 apren) | 20.35 | 22.97 | 10.7 | 10.74 |
| **Fisher Modular - DLF** | 13.95 | - | **8.5** | - |
| **Fisher Modular - PCA** | **12.62** | - | 11.7 | - |

Tabla 6.17: Resumen de resultados de evaluación

Igualmente para los tiempos de entrenamiento y los tiempos de evaluación de los distintos problemas mostramos unas tablas resumen generales en las que se muestran tiempos para cada una de las arquitecturas para un problema dado.

**Resumen tiempos de cómputo**

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.031 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0')** | 0.031 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0")** | 0.209 | $< 0.000$ |
| **Single Layer Network** | 1.480 | $< 0.000$ |
| **Arquitectura_2** | 3.032 | 0.031 |
| **Bagging** (25 apren - 40 %) | 3.672 | 0.031 |
| **Boosting** (25 apren) | 3.328 | 0.031 |
| **Fisher Modular - DLF** | 0.344 | 0.016 |
| **Fisher Modular - PCA** | 0.359 | 0.016 |

Tabla 6.18: Resumen Tiempos de cómputo - problema kwok

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.047 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0')** | 0.047 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0")** | 1.292 | $< 0.000$ |
| **Single Layer Network** | 12.480 | $< 0.000$ |
| **Arquitectura_2** | 32.422 | 0.031 |
| **Bagging** (25 apren - 40 %) | 17.750 | 0.031 |
| **Boosting** (25 apren) | 15.250 | 0.031 |
| **Fisher Modular - DLF** | - | - |
| **Fisher Modular - PCA** | - | - |

Tabla 6.19: Resumen Tiempos de cómputo - problema phoneme

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.031 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0')** | 0.031 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0")** | 0.172 | $< 0.000$ |
| **Single Layer Network** | 0.750 | $< 0.000$ |
| **Arquitectura_2** | 1.626 | 0.031 |
| **Bagging** (25 apren - 40 %) | 2.797 | 0.031 |
| **Boosting** (25 apren) | 2.423 | 0.031 |
| **Fisher Modular - DLF** | 0.281 | 0.016 |
| **Fisher Modular - PCA** | 0.281 | 0.016 |

Tabla 6.20: Resumen Tiempos de cómputo - problema ripley

| *Arquitectura* | **Tiempo entrenamiento** | **Tiempo evaluación** |
|---|---|---|
| **Arquitectura_1 - Fisher (y0)** | 0.078 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0')** | 0.078 | $< 0.000$ |
| **Arquitectura_1 - Fisher (y0")** | 1.117 | $< 0.000$ |
| **Single Layer Network** | 12.123 | $< 0.000$ |
| **Arquitectura_2** | 355.844 | 1.219 |
| **Bagging** (25 apren - 40 %) | 23.375 | 0.047 |
| **Boosting** (25 apren) | 19.735 | 0.047 |
| **Fisher Modular - DLF** | - | - |
| **Fisher Modular - PCA** | - | - |

Tabla 6.21: Resumen Tiempos de cómputo - problema spam

Como se observa en la tabla 6.17 conforme aumenta la complejidad de la arquitectura implementada, el porcentaje de muestras de ejemplo mal clasificadas disminuye para los problemas tratados. Sin embargo, esta mejora en los porcentajes de error lleva asociado un aumento en el tiempo de cómputo, el cual se observa en las tablas 6.18 6.19 6.20 6.21, coste que en unos casos es asumible pero no en otros dependiendo de la aplicación.

Se observa en estas últimas tablas que el aumento del tiempo de cómputo es muy superior para el caso de la `Arquitectura_2`, en concreto para aquellos problemas con un gran número de variables como son los problemas *spam* y *phoneme*. Siendo tal vez un coste demasiado elevado respecto a la `Arquitectura_1` para la pequeña mejora de resultados que se obtiene. Realmente la `Arquitectura_2`, cuyo desarrollo ocupa gran parte de este proyecto, no ha cubierto las expectativas fundamentalmente en cuanto a resultados; salvo para el caso del problema *spam*, en el cual la mejora relativa respecto a la `Arquitectura_1` se debe destacar. Esta mejora es debida a la alta dimensionalidad del problema que impide que un único DLF extraiga tanta información de clasificación como obtenemos aplicando sucesivos DLFs.

El desarrollo de sistemas de múltiples redes comenzó con el desarrollo de sistemas que incluimos en las técnicas que agrupamos dentro de lo que se denominó comité y que eran *Boosting* y *Bagging*. Estos comités presentan el problema de la falta de diversidad de los componentes que los constituyen ya que no se complementan con el objetivo de mejorar los porcentajes de error al formar el sistema global. Esta falta de diversidad se achaca a la gran robustez del discriminante de Fisher frente a la selección y/o remuestreo de ejemplos y por tratarse de un sistema determinista.

Cabe destacar especialmente la mejora de resultados que acarrea el desarrollo de la arquitectura que denominamos `Fisher Modular`. Aunque este desarrollo solo ha sido utilizado para resolver dos de los problemas generales que son: *kwok* y *ripley* la gran reducción de los porcentajes de error compensa el ligero aumento en el tiempo de cómputo que con lleva el desarrollo de este sistema de múltiples redes respecto de la utilización de la primera solución obtenida denominada `Arquitectura_1`. Esta mejora la aporta la combinación de dos clasificadores que se complementan fuertemente.

En general se han cubierto los objetivos del proyecto, ya que, partiendo de un Discriminante Lineal de Fisher se desarrollo en primer lugar un clasificador con sus distintas formas de calcular el umbral de clasificación (ver 5.1). Posteriormente se han desarrollado soluciones basadas en el Discriminante Lineal de Fisher aumentando la complejidad y el número de componentes de la estructura. Evaluando también los resultados obtenidos por las distintas arquitecturas desarrolladas.

Finalmente con el desarrollo de la arquitectura denominada `Fisher Modular` se consiguen muy buenos resultados en los problemas que aborda. Estructura que básicamente consiste en el cálculo de dos Discriminantes Lineales de Fisher cada uno de los cuales se utiliza para clasificar un subconjunto de muestras del problema a abordar y cuya combinación conforma un clasificador que obtiene buenos resultados.

Como **resumen** o **conclusiones finales** a este trabajo queda claro que el Discriminante Lineal de Fisher es óptimo en problemas gaussianos, el cálculo del clasificador es un proceso determinista y no depende del orden de presentación de las muestras y presenta una gran robustez frente al remuestreo. El DLF puede ser útil en problemas que requieran la extracción de variables características orientadas a la clasificación de muestras en conjuntos de datos de alta dimensionalidad. En este tipo de conjuntos de datos no se considera relevante el cálculo del sesgo o umbral de clasificación que como hemos visto no incluye el cálculo del DLF, aunque en este mismo trabajo se presenta una solución como es el cálculo del sesgo mediante ventanas de Parzen.

Las tasas de error obtenidas por las distintas arquitecturas que se han desarrollado y evaluado en este trabajo no aportan mejoras respecto a soluciones ya existentes en otros métodos de aprendizaje lineales, tal vez por no haber llegado a encontrar un método constructivo o un sistema de múltiples redes que aproveche mejor las propiedades del DLF.

A continuación se enumeran las posibles **líneas futuras** de investigación que se podrían desarrollar tomando como base este trabajo:

- Desarrollo de otros métodos constructivos similares a los presentados en la `Arquitectura_2`.
- Estudio de la viabilidad de la arquitectura denominada `Fisher-Modular` en problemas con un mayor número de dimensiones, bien variando el número de gates y/o variando el número de DLF's utilizados.
- Utilizar como gate en el caso del `Fisher-Modular` alguno otro método que nos permita definir subconjuntos de datos con los que obtener un DLF, por ejemplo utilizando clustering.
- Añadir control sobre la selección de muestras para que no queden subconjuntos vacíos (poda del árbol).

Apéndice A

# Duración y presupuesto del proyecto

El desarrollo de este proyecto se inició en Noviembre de 2004 y termina en Junio de 2006, prolongándose por tanto durante 18 meses. El trabajo se ha realizado de forma continuada salvo en los meses de Julio, Agosto y Septiembre de 2005 y se puede dividir en varias fases, las cuales se indican a continuación.

1. **Familiarización y toma de contacto con el problema en cuestión.** Motivación y comprensión del problema, planificación personal para la realización del trabajo y repaso de los distintos tipos de redes neuronales, así como del discriminante lineal de Fisher, análisis de componentes principales, etc...

2. **Desarrollo de las arquitecturas propuestas.** Es decir, desarrollo e implementación de cada una de las arquitecturas propuestas así como las pruebas y experimentos con los problemas propuestos.

3. **Redacción de la memoria.** Familiarización con el editor de documentos basado en LaTeX, para a continuación, redactar y corregir la memoria.

En resumen el total de semanas empleadas en este trabajo es de 70 semanas con aproximadamente una media de 12 horas por semana de trabajo. Esto hace un total aproximado de 840 horas de trabajo.

También, el tutor de este proyecto fin de carrera ha dedicado una media de 90 minutos semanales al seguimiento y supervisión del mismo durante todo el desarrollo del proyecto, obteniéndose un total de 105 horas.

Además de los costes anteriormente mencionados han surgido otros costes materiales que junto con los anteriores se resumen en la tabla A.1. Se tiene además en cuenta un coeficiente de amortización de $\frac{1}{5}$ en los conceptos que lo requieren.

| **Concepto** | **Horas** | **Precio** | **Importe** |
|---|---|---|---|
| Proyectando | 840 | 30 | 25 200 |
| Tutor | 105 | 60 | 6 300 |
| Acceso a Internet | - | - | 300 |
| Ordenador Pentium IV 2,8 Ghz | 1/5 | 1 500 | 300 |
| Software de Programación Matlab 7.0 | 1/5 | 1000 | 200 |
| Sistema Operativo Win XP | 1/5 | 1 500 | 300 |
| Microsoft Office 2000 | 1/5 | 300 | 60 |
| Material de oficina y encuadernación | - | - | 200 |
| Base imponible | - | - | 32 860 |
| I.V.A. (16 %) | | | 5 250 |
| **Total** | | | 38 110 |

Tabla A.1: Presupuesto del proyecto

El presupuesto final de este proyecto asciende a 38 110 euros.

# Bibliografía

[Bellmann, 1961] Bellmann, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

[Bishop, 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Clarendon Press.

[Cascales Salinas *et al.*, 2004] Cascales Salinas, B., Lucas Saorín, P., Mira Ros, J. M., Pallarés Ruiz, A. J., & Sánchez-Pedreño Guillén, S. (2004). *El libro de LaTeX*. Pearson España.

[Cruz & Dorronsoro, 1998] Cruz, C. S. & Dorronsoro, J. R. (1998). *A Nonlinear Discriminant Algorithm for Feature Extraction and Data Classification*.

[Cuadrado, 2005] Cuadrado, L. J. D. (2005). *Técnica de enfatizado mixto para clasificación*.

[Duda *et al.*, 2001] Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification*. John Wiley And Sons, second edition.

[Fisher, 1936] Fisher, R. (1936). *The Use of Multiple Measurements in Taxonomic Problems*. Annals of Eugenics.

[Freund & Schapire, 1997] Freund, Y. & Schapire, R. E. (1997). *A decision theoretic generalization of on-line learning and an application to boosting.* Journal of Computer an System Sciences.

[Haykin, 1999] Haykin, S. (1999). *Neural Networks, A Comprehensive Foundation.* Prentice Hall International.

[Hernández Orallo *et al.*, 2004] Hernández Orallo, J., Ramírez Quintana, M. J., & Ferri Ramírez, C. (2004). *Introducción a la Mineria de Datos.* Prentice Hall.

[Hilera & Martínez, 1995] Hilera, J. R. & Martínez, V. J. (1995). *Redes Neuronales Artificiales Fundamentos, modelos y aplicaciones.* RA-MA.

[Isasi & Galván, 2004] Isasi, P. & Galván, I. M. (2004). *Redes de Neuronas Artifiacles Un Enfoque Práctico.* Prentice Hall.

[Jacobs, 1995] Jacobs, R. A. (1995). *Methods for combining experts' probability assessments.* Cambridge, MA, USA.

[Kolmogorov, 1957] Kolmogorov, A.Ñ. (1957). *On the representation of continuos functions of several variables by superposition of continuos functions of one variable and addition* ehe edition.

[Kuncheva, 2004] Kuncheva, L. I. (2004). *Combining Pattern Classifiers : Methods and Algorithms.* Wiley-Interscience.

[Rosenblatt, 1962] Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan Books.

[Rätsch *et al.*, a] Rätsch, G., Onoda, T., & Müller, K. (a). *Soft Margins for AdaBoost.*

[S. Hettich & Merz, 1998] S. Hettich, C. B. & Merz, C. (1998). *UCI Repository of machine learning databases.*

[Sanguino Botella, 1997] Sanguino Botella, J. (1997). *Iniciación a LaTeX.* Addison-Wesley.

[Schapire, 1999] Schapire, R. E. (1999). *A brief introduction to Boosting.*

[Sharkey, 1996] Sharkey, A. J. C. (1996). *On Combining Artificial Neural Nets.*

[Wand & Jones, 1995] Wand, M. P. & Jones, M. (1995). *Kernel Smoothing.* Chapman and Hall.

[Waterhouse, 1997] Waterhouse, S. (1997). *Divide and Conquer: Pattern Recognition using Mixtures of Experts.*

[Wolpert, 1992] Wolpert, D. H. (1992). *Stacked generalization.* Oxford, UK, UK.

[Xu *et al.*, 1992] Xu, L., Krzyzak, A., & Suen, C. (1992). *Methods of combining multiple classifiers and their applications to handwriting recognition.*