Facultad de Informática
UNLP

Tesina de Licenciatura en Informática

Reconocimiento de Gestos Dinámicos

Facundo Manuel Quiroga

Directora:
Prof. Laura Lanzarini

Co-director:
Prof. Leonardo Corbalán

7 de febrero de 2014

Índice general

Resumen

El objetivo de esta tesina es estudiar, desarrollar, analizar y comparar distintas técnicas de aprendizaje automático aplicables al reconocimiento automático de gestos dinámicos. Para ello, se definió un modelo de gestos a reconocer, se generó una base de datos de prueba con gestos llamadas **LNHG** [1], y se estudiaron e implementaron clasificadores basados en máquinas de vectores de soporte (SVM), redes neuronales feedfoward (FF) y redes neuronales competitivas (CPN), utilizando representaciones locales y globales para caracterizar los gestos. Además, se propone un nuevo modelo de reconocimiento de gestos, el clasificador neuronal competitivo (**CNC**).

Los gestos a reconocer son movimientos de la mano, con invariancia a la velocidad, la rotación, la escala y la traslación.

La captura de la información referida a los gestos para generar la base de datos se realizó mediante el dispositivo Kinect y su SDK correspondiente, que reconoce las partes del cuerpo y determina sus posiciones en tiempo real. Los clasificadores se entrenaron con dichos datos para poder determinar si una secuencia de posiciones de la mano es un gesto.

Se implementó una librería de clasificadores con los métodos mencionados anteriormente, junto con las transformaciones para llevar una secuencia de posiciones a una representación adecuada para el reconocimiento [2].

Se realizaron experimentos con la base de datos LNHG, compuesta de gestos que representan dígitos y letras, y con un base de datos de otro autor con gestos típicos de interacción, obteniendo resultados satisfactorios.

[1] Disponible en https://sites.google.com/site/lnhgdb/

[2] Todo el código desarrollado en esta tesina se encuentra disponible en https://github.com/facundoq/gest

Prefacio

La aparición de nuevas tecnologías en sensores y la popularidad de los dispositivos móviles ha introducido nuevas posibilidades de interacción hombre-máquina, y a su vez han generado cambios radicales en los paradigmas de las interfaces de usuario.

La evolución de las interfaces de usuario ha visto el desarrollo desde las interfaces textuales controladas por teclado a la interfaz gráfica basada en el ratón, y actualmente el auge de las pantallas táctiles; queda por ver cuál será la próxima tecnología que cambie radicalmente los patrones de interacción máquina-hombre (HCI). Por ende, el uso de gestos como método de interacción, especialmente gestos con la mano, se ha convertido en una herramienta importante en el área de HCI en los años recientes, motivando la investigación en su modelado, análisis y reconocimiento.

Un gesto puede definirse informalmente como una secuencia de movimientos o configuraciones del cuerpo cuyo objetivo es comunicar información o interactuar con el ambiente.

El reconocimiento de gestos es una tarea compleja que involucra diversos aspectos tales como el modelado y análisis del movimiento, reconocimiento de patrones, aprendizaje automático, y estudios psicolingüísticos. Tiene un gran rango de aplicaciones como:

- Ayudas para hipoacúsicos
- Facilitación la interacción con la computadora a niños pequeños
- Diseño de técnicas para reconocimiento forense
- Reconocimiento automático de lenguaje de señas
- Monitoreo y rehabilitación médica de pacientes
- Elaboración de perfiles psicológicos
- Navegación y manipulación de entornos virtuales

- Monitoreo del nivel de alerta en conductores de vehículos
- Entrenamiento de deportistas
- Elementos de control en video juegos
- Detección de sonrisas en cámaras fotográficas

El reconocimiento de gestos es un término que engloba diversos tipos de "reconocimientos" y "gestos". A grandes rasgos, se pueden distinguir **gestos corporales**, que se realizan con movimientos de todo el cuerpo, **gestos con las manos**, como un saludo, **gestos con los dedos y las manos**, como el lenguaje de señas y **gestos faciales**, como los guiños y movimientos de los labios. Otra distinción importante es entre **gestos estáticos**, comúnmente llamados *poses*, definidos por una configuración particular del cuerpo en el entorno, y **gestos dinámicos** compuestos por una serie de movimientos de ciertas partes del cuerpo.

El reconocimiento de gestos tiene una larga historia, marcada por el avance de los sensores utilizados para captar los gestos [1]. En las primeras épocas, los gestos no eran verdaderos gestos corporales sino que se realizaban indirectamente con tabletas y lápices especiales que capturaban la escritura [2, 3, 4], interfaces sensibles al toque [5] o dispositivos para apuntar [6]. Finalmente en los años 80 se comenzaron a utilizar guantes con sensores de flexión y de posición y feedback táctil [7] o joysticks [8]. En los 90 se incrementó el trabajo en identificación de gestos y acciones de las personas en imágenes y video con métodos de visión por computadora [9, 10], los cuales, han ido mejorando hasta tener hoy sistemas de seguimiento de personas y sus partes del cuerpo que son robustas y funcionan en tiempo real [11]. Los métodos de reconocimiento basados en visión no requieren en general elementos que deben llevarse en el cuerpo y por ende proveen una solución más natural y conveniente que otros sensores.

Aún con estos avances, mientras que el uso de pantallas táctiles se ha convertido en un estándar para dispositivos móviles en ciertas aplicaciones, y el reemplazo de los joysticks tradicionales por interfaces de voz y movimiento en las consolas de juegos se está consolidando, el retiro de la dupla teclado-mouse en las PCs de propósito general por interfaces más naturales basadas en gestos todavía se encuentra lejos de ser una realidad.

En este panorama, las tecnologías con más promesa para proveer una interfaz hombre-máquina completa y eficiente son el reconocimiento de voz y de gestos en tiempo real [12]. El reconocimiento del habla consta de la traducción automática de las palabras contenidas en una grabación de audio a texto. Si bien ambos han estado en activa investigación desde

hace décadas [5, 1, 13] y han encontrado nichos con gran aplicación, no alcanzan todavía un grado de desarrollo y sofisticación suficiente para emplearlos como sustituto completo de dicha dupla en el uso diario de la computadora. Estos métodos se diferencian, sin embargo, en que la investigación en reconocimiento de voz ha dispuesto de sensores adecuados desde un comienzo (micrófonos), y ha estado más enfocada en una tarea más o menos única, la de reconocer palabras en una grabación de audio. Por estos y otros motivos, se encuentra bastante cerca de dicho objetivo en términos de capacidad de reconocimiento [14, 15, 16], salvando las dificultades inherentes al reconocimiento de voz como el cansancio y sobresfuerzo de las cuerdas vocales en el uso diario, y aquellas particulares a dicha tecnología pero todavía no resueltas como la habilidad de reconocer de forma fiable varios interlocutores, hablando en distintos idiomas, en entornos no controlados [17, 18] o su aplicación de forma más amplia [19] mediante procesamiento de lenguaje natural. Otras dos tecnologías prometedoras, las interfaces musculares [20] y cerebrales [21, 22], han hecho grandes progresos pero todavía distan de proveer una alternativa usable.

Es de especial interés notar los diversos problemas de salud asociados con el sobre uso de las computadoras con entornos e interfaces tradicionales, ampliamente documentados y considerados "epidémicos" [23, 24, 25, 26, 27]. El reconocimiento de gestos, en conjunto con otras tecnologías, parece un enfoque prometedor para, además de proveer una interfaz más natural, prevenir nuevas ocurrencias de estos problemas y desarrollar medios de comunicación e interacción que puedan utilizar sin problemas los afectados [28, 29].

El reconocimiento automático de gestos puede realizarse de dos formas básicas; definiendo explícitamente cada tipo de gesto con algún lenguaje de especificación, reconociéndolos si cumplen con dicha especificación, o grabando con algún tipo de sensor la realización de los gestos y entrenando un modelo que pueda luego reconocerlos por su similitud con los gestos grabados. Este último es el enfoque de aprendizaje automático elegido en esta tesina.

El presente documento consta de dos partes: métodos y aplicaciones.

En la primera, de **métodos**, se establecen las bases teóricas de los algoritmos de aprendizaje automático, los modelos de clasificación y el modelo de gestos utilizados (capítulos 1, 2, 3 y 4).

Los algoritmos de aprendizaje automático se utilizan para generar modelos de los problemas a partir de datos de los mismos. En este caso, se generan modelos de clasificación de gestos a partir de ejemplares de gestos. El aprendizaje automático permite al usuario grabar gestos prototipos, luego entrenar un modelo de gestos en base a esos prototipos, y

finalmente realizar el reconocimiento en base a los gestos grabados. En base a ese modelo se podrían construir aplicaciones que permitan grabar los prototipos, y asignar acciones o comandos a los gestos para interactuar con una computadora o con otros fines.

El capítulo 1 presenta las bases del aprendizaje automático. Esta área estudia diversos algoritmos para generar modelos en base a datos de ejemplares del problema, planteando un enfoque inductivo para la resolución de problemas. Los conceptos desarrollados en este capítulo valen para todo tipo de algoritmos de reconocimiento de gestos inductivos, como los clasificadores que se introducen en los capítulos siguientes.

Luego, el capítulo 2 introduce un tipo de clasificador llamado Máquina de vectores de soporte (SVM). El modelo SVM refina el concepto de un hiperplano como una superficie de separación entre ejemplares de dos clases, y presenta un modelo muy simple pero potente de clasificación, con fuertes bases teóricas.

El capítulo 3, describe el modelo computacional llamado Redes Neuronales Artificiales (ANN). Las redes neuronales artificiales son un modelo de aprendizaje automático basado en el funcionamiento de las redes neuronales biológicas. Estos modelos generalmente abstraen las redes biológicas representándolas como un grafo, donde el flujo de los estímulos entre neuronas se realiza a través de las aristas, y cada nodo del grafo realiza cierto procesamiento imitando la modulación de la señales por parte de las neuronas y las sinapsis.

Para finalizar esta parte, en el capítulo 4 se realiza una breve introducción al significado de un gesto en el contexto de interacción hombre-computador, seguido por un modelado matemático de los gestos y el desarrollo de una definición conveniente de equivalencia entre los mismos para la interacción por computadora.

En la segunda parte, de **aplicaciones**, se utilizan los conceptos anteriores para generar y probar modelos de reconocimiento de gestos con la mano en espacios 3D.

El capítulo 5 describe las bases de datos utilizadas para probar los algoritmos de reconocimiento de gestos, así como las características calculadas en base a los ejemplares para implementar de forma efectiva y eficiente el modelo de gestos presentado en el capítulo 4.

En el capítulo 6 se utilizan los conceptos de los capítulos anteriores para describir el nuevo Clasificador Neuronal Competitivo (CNC) propuesto en esta tesina. Dichos conceptos se utilizan además para generar otros 4 modelos distintos de reconocimiento de gestos para comparar con el CNC.

En el capítulo 7 se describen los experimentos realizados y sus resulta-

dos.

Finalmente, en el capítulo 8 se realiza un resumen del trabajo de la tesina, y se presentan las conclusiones generales y trabajos futuros.

Cada capítulo finaliza con un resumen que detalla el contenido del mismo en unos pocos párrafos; se espera que los mismos faciliten la lectura de esta tesina.

Un comentario sobre la notación

A través del texto, se utilizarán las siguientes convenciones y notación:

- Los escalares se denotan por una letra minúscula, sin negrita o cursiva, como α, b o y_i.
- Los vectores se denotan con letras minúsculas en negrita, como $\boldsymbol{w}$, $\boldsymbol{x_i}$ o $\boldsymbol{\alpha}$.
- El elemento i de un vector $\boldsymbol{v}$ se escribe v_i. En cambio, el símbolo $\boldsymbol{v}_i$, en negrita, denota el i-ésimo vector, y no el elemento i del vector $\boldsymbol{v}$.
- Las matrices se denotan con una letra mayúscula en negrita, como $\boldsymbol{A}$ o $\boldsymbol{S}$.
- En general, los conjuntos se denotan con letras mayúsculas, como D. Conjuntos especiales, como el de los números se reales, se denotan $\mathbb{R}$ y otros, como los del dominio de un problema, se denotan como $\mathcal{P}$.
- Las funciones se definen como $f :: D \mapsto I$, donde f es una función con dominio D e imagen I.
- Dado un conjunto A, la expresión $|A|$ se refiere a la cardinalidad de A.
- Dados un vector $\boldsymbol{x}$, la expresión $\|\boldsymbol{x}\|$ se refiere a la norma de $\boldsymbol{x}$. Cuando no se indique lo contrario, la norma referida será la euclídea.
- Se definirán problemas de minimización de funciones con la siguiente notación:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{Sujeto a} \quad & \\ & P(x) \end{aligned}$$

donde f es una función escalar, P es un predicado, y x es algún objeto de un dominio D (implícito), que generalmente se deduce del contexto. Dichos problemas plantean encontrar el valor de x que minimiza f dentro de los que cumplen el predicado P. Las expresiones de maximización se interpretan de forma equivalente, donde se busca el x que maximiza $f(x)$.

Formalmente, se define el dominio restricto $D' = \{x \mid P(x) \land x \in D\}$, como el subconjunto de D que cumple P. El conjunto D^* de puntos mínimos del problema, $D^* \subset D'$, existe si $D' \neq \emptyset$, y se define como:

$$D^* = \{x^* \mid \forall x \in D' : \ f(x^*) \leq f(x)\}$$

Dado que para muchos problemas hay un sólo mínimo, y que si existen muchos generalmente cualquiera resulta igual, se hablará directamente del valor mínimo.

Para el problema de maximización, el conjunto D^* de puntos máximos se define análogamente como:

$$D^* = \{x^* \mid \forall x \in D' : \ f(x^*) \geq f(x)\}$$

- El símbolo $\leftarrow$ denota asignación, de modo que $a \leftarrow b$ se interpreta como asignar a la variable a el valor b. No obstante, en las porciones de pseudocódigo incluidas en esta tesina se utiliza el símbolo $=$ para denotar asignación, dado que es una notación más familiar en ese contexto.
- El símbolo $\equiv$ se utiliza para denotar relaciones de equivalencia.

Parte I

Métodos

1

Aprendizaje Automático

> The question of whether computers can think is like the question of whether submarines can swim.
>
> Edsger W. Dijkstra

1.1. Introducción

Un programa que aprende automáticamente es aquel que no se programa explícitamente para resolver un problema, sino que utiliza un algoritmo de aprendizaje en base a datos de instancias o ejemplares del problema para generar un modelo del mismo.

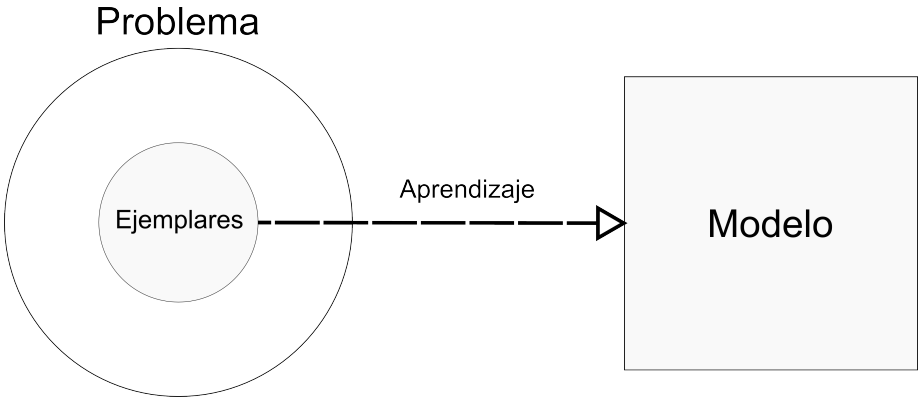


Figura 1.1: Generación de un modelo a partir de ejemplares de un problema: la esencia del aprendizaje automático.

Con ese modelo, se pueden analizar ejemplares del problema general, obteniendo alguna inferencia, con cierto grado de error.

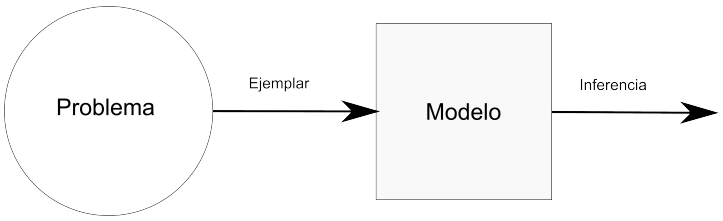


Figura 1.2: Obtención de una inferencia a partir de algún ejemplar del problema.

Siguiendo la definición más formal de Mitchell [30] donde la "experiencia" son los ejemplares del problema:

> Un programa de computadora aprende de la experiencia E respecto a una clase de tareas T y medida de error P, si su error en las tareas en T, medida por P, mejora con la experiencia E. [1]

En general, las técnicas de aprendizaje automático se aplican a problemas donde no se conoce una solución algorítmica definitiva o esta tiene características indeseables: es computacionalmente demandante, numéricamente inestable, requiere programación explícita para adaptarse a cambios, es poco robusta a fallos de sensores, o requiere intervención manual.

Una clase muy importante de problemas en donde se aplica aprendizaje automático es la de aquellos donde los humanos tienen un bajo nivel de error innato, como el reconocimiento de voz, de gestos, de objetos, etc, pero es difícil codificar reglas para realizar dichas tareas explícitamente. El aprendizaje automático es importante, entonces, porque las ocurrencias de la existencia humana acontecen en forma de patrones. La información del lenguaje, el habla, el dibujo y el entendimiento de las imágenes, todas involucran patrones.

No hay un aspecto o secuencia de aspectos que determine absolutamente y sin ambigüedad el significado de un patrón, y dicho significado varía con el contexto. Los humanos solemos ser bastante buenos en este tipo de tareas ya que tenemos un cerebro capaz de procesar la información relevante a dichas tareas de forma eficiente, adaptativa y eficaz. Si bien

[1] La P es por *performance*. En general, hablar de performance o desempeño es equivalente a hablar de error, ya que a mayor performance menor error y viceversa

esto no es algo del todo sorprendente, ya que nos interesan y nos parecen importantes justamente porque podemos hacerlos, lograr que una computadora emule ciertas capacidades de nuestro cerebro es un objetivo de gran interés para el aprendizaje automático. De hecho, se puede argumentar que, en esencia, muchos de los algoritmos de aprendizaje automático son aproximaciones matemáticas y computacionales al funcionamiento del cerebro [31, 32, 33].

1.2. Un ejemplo: Reconocimiento de Gestos

Para ilustrar la idea del aprendizaje automático, se tomará como ejemplo el problema del reconocimiento de gestos.

Supóngase que en una base de datos (BD) existe un conjunto de 20 ejemplares de gestos realizados con la mano: 10 son gestos de saludo y 10 de un movimiento de golpe. Por ahora, no es importante qué información se guarda sobre cada ejemplar de gesto o cómo se representa, pero podría ser una secuencia de posiciones de la mano, por ejemplo. Se busca desarrollar un programa que pueda, dado un nuevo ejemplar de gesto, decidir si es de la primera clase (un saludo) o de la segunda (un golpe).

En un enfoque más tradicional, se ignoraría la BD de 20 ejemplares, y se intentaría definir matemáticamente los conceptos de gesto de saludo y de golpe. Se describiría un golpe como una trayectoría de la mano bastante recta, de una longitud mayor a cierto mínimo recorrida de forma rápida y con una parada abrupta. Del mismo modo, se buscaría una definición explícita para un gesto de saludo. Luego, se escribiría un algoritmo que tome un nuevo ejemplar, realice un ajuste del mismo a los dos modelos, y decida si pertenece a alguno de los dos, o a ninguno.

Esto corresponde más al enfoque de Hoare a la computación, donde existe un programa f de reconocimiento automático, y se pueden definir tuplas de Hoare del tipo {Precondicion} *Programa* {Poscondicion} tales como:

$$\{\text{La entrada se ajusta al modelo de gesto golpe }\}\ f\ \{\text{El gesto es de golpe}\}$$

o:

$$\{\text{La entrada no se ajusta a ninguno de los modelos}\}\ f\ \{\text{El gesto no es un golpe ni un saludo}\}$$

Si bien es cierto que estas tuplas pueden ser triviales, lo importante es notar que *se puede* escribirlas porque dichos modelos de gestos se definieron *explícitamente*.

El enfoque de aprendizaje automático, en cambio, entrenaría un clasificador f que aprenda las características particulares de los ejemplares de cada clase de gestos *a partir de los ejemplares de la BD*, con algunas de las técnicas del próximo capítulo, por ejemplo. De esa manera, el modelo surge de los datos, con lo cual prescinde de la necesidad de programación explícita de cada nuevo tipo de gestos. Luego se prueba el clasificador con nuevos ejemplares de gestos; en el caso ideal, todos los nuevos ejemplares de gestos se clasificarán correctamente; en la práctica, se obtendrá un porcentaje de clasificación incorrecta que habla del **error del clasificador**.

Entonces, hablando ahora en general, el enfoque de aprendizaje automático se desliga de la noción de Hoare de computación, y modela a f como un programa o una función (matemática) que realiza una tarea de **inferencia** con cierto grado de error, de forma similar a cómo se trabaja con métodos numéricos o simulaciones. Es usual llamar a f el **modelo** del problema. Este modelo se genera en base a un conjunto de ejemplares D, que es un subconjunto del conjunto de ejemplares de un problema o **dominio** $\mathcal{P}$ (en el ejemplo, $\mathcal{P}$ es el conjunto de todos los posibles movimientos del cuerpo, a ser clasificados como gestos).

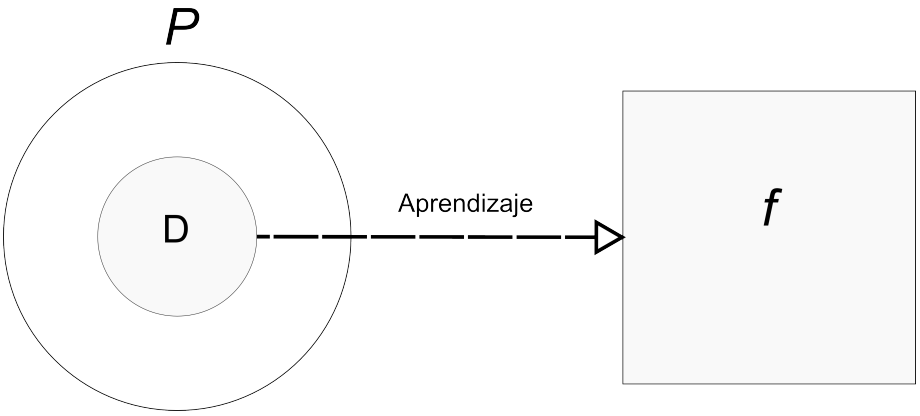


Figura 1.3: Generación de un modelo f a partir de ejemplares de un problema, D

Las distintas maneras de generar la función f dan lugar a distintos algoritmos de aprendizaje automático. En general un esquema de aprendizaje automático está definido por un **algoritmo o función de entrenamiento** g, que genera o **entrena** una función f en base al conjunto de

ejemplares D. [2]

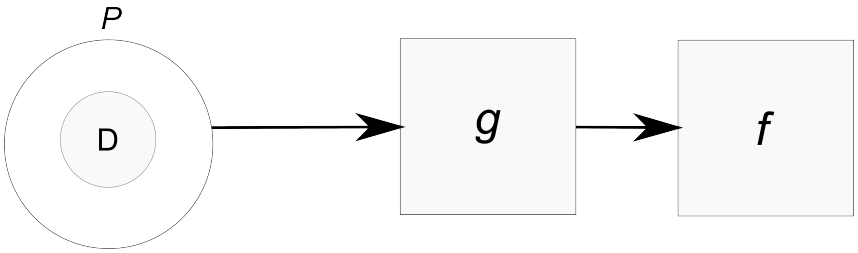


Figura 1.4: Generación de un modelo f a partir de un conjunto de ejemplares del problema, D y una función de entrenamiento, g

La elección de la función de entrenamiento g depende fuertemente del problema a resolver. Distintas funciones de aprendizaje automático g generarán distintos modelos f. Existe una gran variedad de algoritmos de entrenamiento distintos; en esta tesina se describirán algunos basados en redes neuronales y máquinas de soporte vectorial.

De todas maneras, se puede describir en forma general el objetivo de una función de entrenamiento g.

Como fue mencionado, toda función f generada por un algoritmo de aprendizaje automático g y un conjunto de datos D cometerá algún tipo de **error**, ya que el aprendizaje casi nunca es perfecto, y D representa solo una parte del dominio a modelar.

De este modo, dado un conjunto de datos D se puede definir como objetivo general de un algoritmo de entrenamiento la resolución del problema de optimización, en donde se busca una función de inferencia f que minimice el **error** del modelo, medido con el conjunto de datos D:

$$\underset{f}{\text{Minimizar}} \quad error(f, D)$$

Formalmente, $g :: [\mathcal{P}] \mapsto f$, donde P es el dominio de los ejemplares del problema, $[\mathcal{P}]$ es una lista de ejemplares de ese dominio (o sea, el conjunto D), y f es una **función de inferencia** tal que $f :: \mathcal{P} \mapsto Inferencia$. Aquí, $Inferencia$ es un conjunto de inferencias posibles a realizar sobre $\mathcal{P}$, dependiente del problema a resolver. En el ejemplo de los gestos, $Inferencia$ sería un conjunto de etiquetas para cada clase y la etiqueta que indica que el ejemplar no pertenece a ninguna clase conocida, es decir, $Inferencia = \{Saludo, Golpe, Ninguno\}$.

[2] De la misma manera en que se referencia a f como función de inferencia o modelo, se hablará de función y algoritmo de entrenamiento g de forma intercambiable

Entonces, f es una función de inferencia generada por g, una función de aprendizaje: una vez "entrenada" f con un conjunto de datos de $\mathcal{P}$, se puede utilizar para hacer inferencia sobre el dominio $\mathcal{P}$. Por otro lado, asumiendo la existencia de una "verdadera" función $f' :: \mathcal{P} \mapsto Inferencia$ que asigna correctamente una inferencia para cada ejemplar del problema, se considera que f aproxima o estima f' en base a un conjunto de datos $D \subset \mathcal{P}$ y una función de aprendizaje g.[3].

Volviendo al ejemplo, el problema de reconocimiento de gestos es un problema de **clasificación**, en donde se busca inferir la clase de los ejemplares (saludo o golpe). No obstante, un programa de aprendizaje automático se puede entrenar para aprender una función f arbitraria.

En otras palabras, la imagen de f no tiene que ser un conjunto clases a asignar a los ejemplares, como en el ejemplo; podría ser que se busque aprender la función $t(x) = x^3$, que es una función que convierte números reales a números reales (o sea, $\mathbb{R} \to \mathbb{R}$) y entonces en ese caso $Inferencia = \mathbb{R}$ y $\mathcal{P} = \mathbb{R}$.

Con este ejemplo, se espera haber establecido un contexto y una motivación para el desarrollo de las siguientes secciones. A continuación se detallan varias de las aplicaciones de aprendizaje automático, a modo de comprender los alcances de esta técnica, y luego se detalla el proceso de aprendizaje junto con sus propiedades más importantes, profundizando el esquema de generación del modelo f.

1.3. Aplicaciones

Las técnicas de aprendizaje automático se usan con éxito en una enorme variedad de problemas. En un artículo de Widrow [34] de 1994 ya se contaba con una gran cantidad de aplicaciones de las redes neuronales neuronales artificiales, uno de los temas de investigación más tradicionales de aprendizaje automático. En ellas se mencionan su uso en técnicas de telecomunicaciones para la mitigación del eco y la ecualización de la señal, de detección de eventos y disminución del error de operación en aceleradores de partículas, de detección de fraudes en operaciones bancarias y tarjetas de crédito, reconocimiento óptico de caracteres de máquina

[3]En verdad, generalmente no se puede definir a g como una verdadera función ya que el entrenamiento suele contener elementos aleatorios; es decir, un algoritmo de entrenamiento puede generar distintos clasificadores f aunque se entrene con el mismo conjunto de datos D. Se puede obviar este hecho asumiendo que en cada generación de f se utiliza un g_s de la misma familia de generadores que solo difieran en una semilla aleatoria s distinta.

y manuscritos, detección de cáncer y otras enfermedades en imágenes médicas y muestras biológicas, y enfocado automático en telescopios.

En la actualidad, la utilización de dichas técnicas no ha menguado. En el área de la industria y los servicios, se han utilizado para clasificar el tráfico en conexiones a Internet [35] y desarrollar sistemas de control integral de manufactura guiados por aprendizaje automático de reglas en base a experiencias previas [36]. También se han empleado en tareas de predicción de índices de la bolsa [37] y predicción de la disponibilidad de recursos hídricos [38]. Se utilizan en sistemas de control de manufactura, para manejar automáticamente automóviles, aviones, helicópteros y robots [39, 40, 41, 42].

El procesamiento de señales médicas en particular se ha visto muy beneficiado por el uso de aprendizaje automático debido a que este suele superar a los métodos estadísticos tradicionales. Se ha utilizado para desarrollar marcadores biológicos para el diagnóstico por computadora de cáncer [43] y para la detección de Alzheimer y Trastornos con Déficit de Atención e Hiperactividad [44, 45], respectivamente. También se han utilizado técnicas de aprendizaje automático en la evaluación de datos neurológicos para resolver diversos problemas, como el diagnóstico clínico de enfermedades, la elaboración de pronósticos y el mapeo de estructuras cerebrales y su funcionamiento. En el área de reconocimiento de patrones cerebrales, se han desarrollado verdaderos juegos mentales [46], así como interfaces para interactuar con personas en estado vegetativo [47], con posibles aplicaciones concretas debido a la comercialización masiva de dispositivos de captura.

Tienen gran utilidad en problemas de reconocimiento de voz, del hablante, de objetos en imágenes y videos y de gestos. Su efectividad en estas aplicaciones ha aumentado enormemente en los últimos años, especialmente en el reconocimiento de secuencias de acciones en videos y en etiquetado de escenas [48, 49, 50] y el reconocimiento de voz [16, 51].

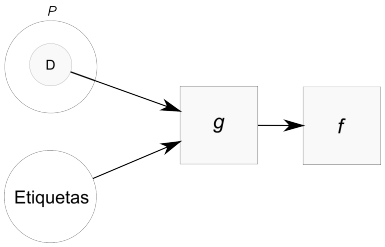
Hay que considerar que varias técnicas de aprendizaje automático como las Redes Neuronales entrenan funciones de inferencia con la propiedad de ser aproximadores universales, es decir, pueden aproximar cualquier función continua en un hipercubo m-dimensional $[0,1]^M$ con un error arbitrariamente pequeño, dado un modelo con la complejidad requerida [52], lo cual sienta bases teóricas que apoyan la experiencia de su gran aplicabilidad. Por ende se han utilizado para aproximar funciones como la Transformada Discreta de Fourier [53], implementar Análisis de Componentes Principales No-Lineal (NPCA) [54] y realizar optimización combinatoria [55].

1.4. Entrenamiento Supervisado y No Supervisado

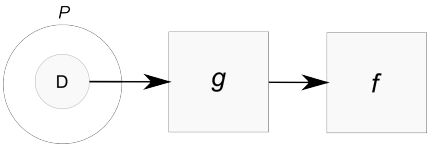
Si bien existen varias clasificaciones de los métodos y técnicas de aprendizaje automático, la más simple y quizás la más significativa distingue entre algoritmos de entrenamiento **supervisado** y **no supervisado**.

Los algoritmos supervisados son aquellos donde se conoce a priori el resultado que se busca aprender para cada ejemplar de entrenamiento del problema. En el ejemplo de los gestos, el aprendizaje es supervisado si para cada uno de los 20 gestos de la BD se conoce su clase, es decir, se sabe a priori si son saludos o golpes. El programa entonces aprende a clasificar nuevos gestos en base a etiquetas anteriores.

En el aprendizaje no supervisado no hay o se ignoran los resultados previamente conocidos, no hay etiquetas asociadas a los ejemplares. El programa aprende relaciones entre los datos, descubre las estructuras subyacentes del espacio de ejemplares $\mathcal{P}$, pero no se guía por ninguna etiqueta asociada a los ejemplares, sino que usa solamente los datos de los ejemplares mismos. En el ejemplo, sería como tener sólo los 20 gestos pero no saber (al menos a priori) de qué clase son. El programa debería aprender a distinguirlos, a reconocer que los ejemplares de los gestos de saludo tienen cierta similitud mutua, una estructura compartida; que los gestos de golpe también poseen características similares y, por último, que los gestos de saludo y de golpe son distintos. El programa, dado un nuevo gesto, debe poder entonces decir que es de un tipo o del otro (o ninguno), pero no va a tener codificadas explícitamente las etiquetas *golpe* y *saludo*.



(a) Esquema de entrenamiento de un algoritmo supervisado.



(b) Esquema de entrenamiento de un algoritmo no supervisado.

En la práctica, ambas técnicas se pueden utilizar conjuntamente, por lo que si bien un algoritmo puede ser tildado de supervisado o no, en general un sistema completo de aprendizaje automático emplea varias combinaciones de algoritmos de los dos tipos.

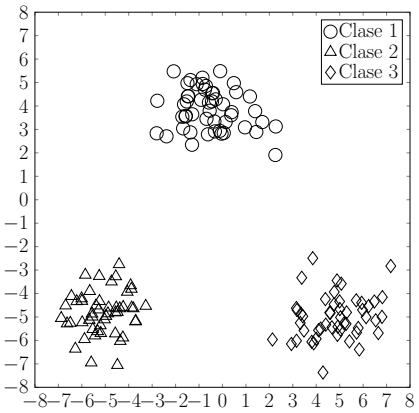
Esta tesina se enfoca en técnicas de aprendizaje automático para clasificación de patrones con el objetivo de determinar la clase a la cual pertenece un ejemplar de gesto, representado por una secuencia de posiciones en un espacio $3D$, con etiquetas de tiempo cuya clase es conocida de antemano. Por ende, a continuación se describirá formalmente el problema de la clasificación de ejemplares, con énfasis en técnicas supervisadas. Además, se introducirán diversos conceptos como generalización, separabilidad lineal, validación cruzada, sobreajuste (u *overfitting*) y regularización para establecer una base teórica común a los métodos de clasificación descriptos en los siguientes capítulos. Si bien por conveniencia dichos conceptos se tratan en el contexto de clasificación, se aplican en general a la mayoría de las técnicas de aprendizaje automático.

1.5. Clasificación

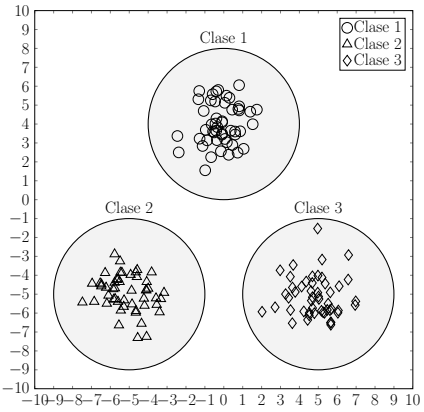
El aprendizaje automático, en general, y la clasificación automática, pueden considerarse efectivamente desde un punto de vista geométrico. En el mismo, los modelos de clasificación ven a los ejemplares a clasificar como puntos o vectores en un espacio d dimensional.

Por ejemplo, si se deben clasificar personas para saber si pertenecen a una población de riesgo de obesidad, y de ellas se conoce su edad y su peso, entonces los ejemplares son vectores de $d = 2$ componentes que existen en un espacio bi-dimensional, como $\mathbb{R}^2$. Este espacio sería el que previamente se identificó como $\mathcal{P}$. Además, como se tiene un problema de clasificación supervisada, se conoce el conjunto de clases a asignar a los ejemplares, y los del conjunto de entrenamiento D se encuentran etiquetados con la clase correspondiente.

Un algoritmo de entrenamiento genera un modelo de clasificación f en base a estos ejemplares. La hipótesis básica detrás de la mayoría de los modelos es de **localidad**, es decir, que si un ejemplar $\boldsymbol{x}$ tiene una etiqueta y, entonces los ejemplares $\boldsymbol{x}'$ cercanos a $\boldsymbol{x}$ probablemente tendrán la misma etiqueta. Al considerar todos los ejemplares, esta hipótesis da lugar a que el modelo determine distintas **regiones** de $\mathcal{P}$, que corresponden a ciertas clases.



(a) Un conjunto de datos con 3 clases.



(b) Regiones de cada clase estimadas por un clasificador entrenado con dichos datos.

Entonces, el desafío de entrenar un modelo de clasificación f es el de estimar las regiones de cada clase, en base a los ejemplares de entrenamiento. Luego de entrenado, a la hora de clasificar, f recibe un ejemplar como entrada, y da como resultado una clase estimada.

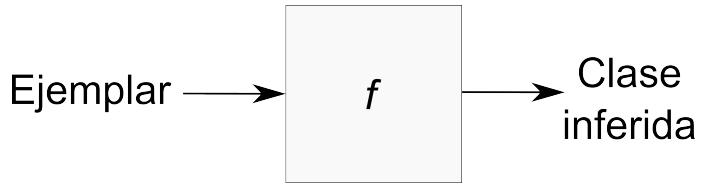


Figura 1.7: Funcionamiento de la función de clasificación o inferencia f.

Formalmente, un clasificador puede definirse como una función $f :: \mathcal{P} \mapsto C$, donde $\mathcal{P}$ es el conjunto de todos los ejemplares del problema, y $C = \{1..N\} \cup \perp$ es el conjunto de etiquetas de las N clases ($\perp$ representa la clase nula, es decir, que un ejemplar no pertenece a ninguna clase).

El clasificador f se genera en base a un conjunto ordenado D de n ejemplares $\boldsymbol{x_i}$, $D = \{\boldsymbol{x_i} \in \mathcal{P}\} \quad i = 1..n$, donde por ejemplo podría ser que $\mathcal{P} = \mathbb{R}^d$, como será en el caso de los gestos, o $\mathcal{P} = \mathbb{R}^2$, como en el de clasificar personas en base a su edad y peso. A su vez, cada ejemplar pertenece a una de las $|C|$ clases conocidas, por ende asociado a cada $\boldsymbol{x_i}$ hay una etiqueta $y_i \in C$ que indica a qué clase pertenece $\boldsymbol{x_i}$.

Volviendo al punto de vista geométrico, se puede interpretar a f como una función que particiona $\mathcal{P}$ en subconjuntos disjuntos $\mathcal{P}_i$ (posiblemente infinitos) llamados **regiones de Voronoi**, con $\cup_i \mathcal{P}_i = \mathcal{P}$, y luego asigna a cada $\mathcal{P}_i$ una clase (o ninguna).

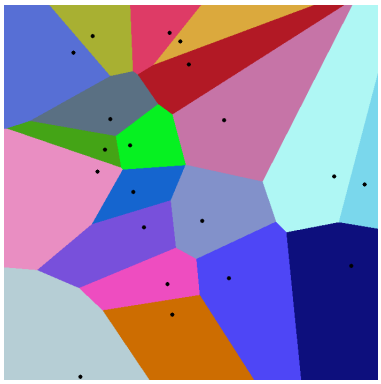


Figura 1.8: Regiones de Voronoi para un problema de clasificación con dominio $\mathcal{P} = \mathbb{R}^2$. Cada región $\mathcal{P}_i$ está representada con un color distinto. Los puntos representan a los centros de la región. Los colores indican que cada a región le corresponde una clase diferente.

Como se vio anteriormente, tal función f se genera con un algoritmo de entrenamiento en base a un conjunto de ejemplares de entrenamiento y a ciertos conocimientos específicos del dominio del problema a resolver. De forma análoga al argumento general sobre aprendizaje automático, en esencia f aproxima una función f' que codifica las clases "verdaderas" de los ejemplares de $\mathcal{P}$; es decir, f' conoce las regiones verdades de cada clase.

1.5.1. El Perceptrón

A modo de ejemplo, se puede considerar un tipo de clasificador muy simple y emblemático, el **Perceptrón** [56]. Si bien el mismo tiene origen en el campo particular de las redes neuronales dentro del aprendizaje automático, puede comprenderse solamente en términos geométricos.

El Perceptrón es arquetípico en aprendizaje automático porque es un tipo de clasificador que divide ejemplares de dos clases bajo la asunción de que existe un **hiperplano** en $\mathcal{P}$, $\boldsymbol{w}\boldsymbol{x} + b = 0$ que separa las dos clases; es decir, los ejemplares de cada clase se encuentran en lados opuestos del hiperplano [4].

El vector $\boldsymbol{w}$ es el vector normal o perpendicular al hiperplano, y determina la dirección en la que se propaga el hiperplano en cada dimensión.

El bias b permite realizar traslaciones verticales del hiperplano, de manera que no esté obligado a pasar por el origen, y por ende aumenta

[4] Un hiperplano es la generalización del concepto de punto en $\mathbb{R}^1$, de línea en $\mathbb{R}^2$, y de plano en $\mathbb{R}^3$.

su poder para dividir el espacio en dos de forma arbitraria, y así clasificar mejor.

En este caso al tener solo dos clases se asumirá que $y = \pm 1$; los ejemplares con clase positiva $+1$ se encuentran del lado a donde apunta el vector normal al hiperplano, $\boldsymbol{w} \in \mathbb{R}^d$. Los ejemplares con clase negativa -1 se encuentran del lado opuesto. [5]

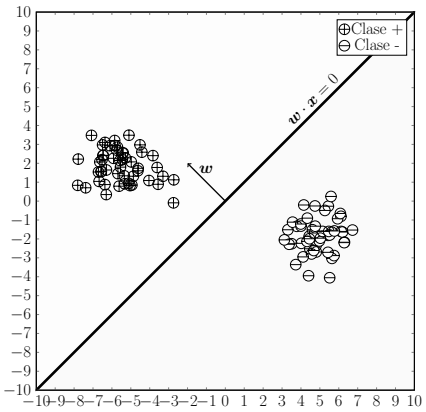


Figura 1.9: Hiperplano separador de los ejemplos de dos clases.

Entonces, un Perceptrón está dado por un par $(\boldsymbol{w}, b)$; un algoritmo de entrenamiento determina esos valores de acuerdo a un conjunto de datos $D = \{\boldsymbol{x_i}\}$ y a las clases de los ejemplares y_i.

La idea esencial del algoritmo de entrenamiento del Perceptrón es, dado un conjunto de datos D como el descripto más arriba, generar un hiperplano inicial de alguna manera y cambiarlo o moverlo (con algún criterio) hasta que clasifique bien todos los ejemplares.

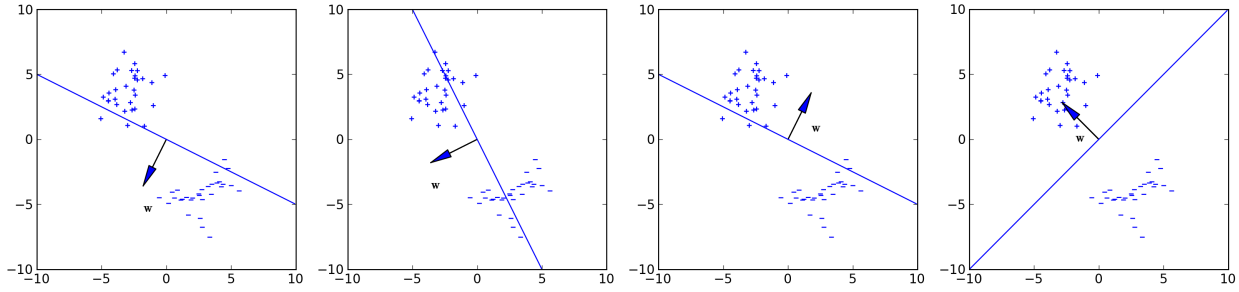


Figura 1.10: Mejora paulatina del hiperplano como separador de las clases.

[5] Los hiperplanos afines están dados por la ecuación $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$, los hiperplanos lineales por $\boldsymbol{w} \cdot \boldsymbol{x} = 0$

Para comenzar, se puede generar un hiperplano inicial donde $\boldsymbol{w}$ y b tienen valores aleatorios.

Luego, se toma un ejemplar $\boldsymbol{x_i}$ del conjunto D y se verifica si el modelo lo clasifica correctamente. Si es así, se vuelve a elegir otro ejemplar; sino, se rota el hiperplano ligeramente de manera que el ejemplar $\boldsymbol{x_i}$ quede más cerca del otro lado del hiperplano.

Este procedimiento se repite hasta que todos los ejemplares se encuentran en el lado correcto del hiperplano, es decir, sean clasificados correctamente. En pseudocódigo:

Data: Un conjunto de ejemplares $D \in \mathbb{R}^d$
Una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Result: Un hiperplano descripto por $\boldsymbol{w} \in \mathbb{R}^d$, un bias $b \in \mathbb{R}$
$\boldsymbol{w} = random(d)$;
$b = random(1)$;
while *haya ejemplares que* $\boldsymbol{w}$ *y* b *no clasifiquen correctamente* **do**
  for $\boldsymbol{x_i} \in D$ **do**
    if $\boldsymbol{w}$ *y* b *no clasifican bien a* $\boldsymbol{x_i}$ **then**
      Modificar $\boldsymbol{w}$ y b de manera que $\boldsymbol{x_i}$ esté más cerca del lado correcto ;
    end
  end
end

Algoritmo 1: Esquema del algoritmo de aprendizaje del Perceptrón

Para formalizar las nociones de "clasifica bien" y "modificar $\boldsymbol{w}$ y b", se derivará:

- Una función f de clasificación
- Un predicado p que sea cierto solo si todos los ejemplares están clasificados correctamente
- Una regla de actualización o aprendizaje para $\boldsymbol{w}$ y b, tal que su aplicación repetida logre que el hiperplano mejore su clasificación de los ejemaplres.

1.5.1.1. Derivación de f

Para facilitar la notación, se define una función h que tiene como parámetros el vector normal $\boldsymbol{w} \in \mathbb{R}^d$ y el *bias* b:

$$h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b \tag{1.1}$$

De manera que el hiperplano está definido como $h(\boldsymbol{x}) = 0$, y[6]:

- Para los ejemplares $\boldsymbol{x}$ sobre el hiperplano, vale que $h(\boldsymbol{x}) = 0$
- Para los ejemplares $\boldsymbol{x}$ del lado del hiperplano a donde apunta el vector normal, vale $h(\boldsymbol{x}) > 0$
- Para los ejemplares $\boldsymbol{x}$ del lado del hiperplano **opuesto** a donde apunta al vector normal, $h(\boldsymbol{x}) < 0$

Para clasificar un ejemplar $\boldsymbol{x}$, es necesaria entonces una función f tal que :

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{si la clase de } \boldsymbol{x} \text{ es } 1 \\ -1 & \text{si la clase de } \boldsymbol{x} \text{ es } -1 \end{cases} \tag{1.2}$$

Para realizar la transformación necesaria, se utiliza la función escalón o **step function** θ:

$$\theta(x) = \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{de lo contrario} \end{cases} \tag{1.3}$$

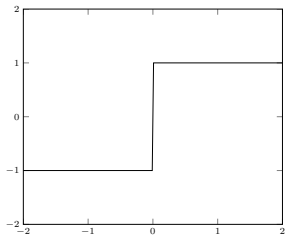


Figura 1.11: Función θ, comúnmente llamada *step function* o función escalón.

Y la función f toma la forma:

$$f(\boldsymbol{x}) = \theta(h(\boldsymbol{x}))$$

Donde h indica con su signo de qué lado del hiperplano se encuentra el ejemplar $\boldsymbol{x}$, y θ le asigna la clase correspondiente en base al signo.

[6]Por conveniencia, si bien h es una función y no una ecuación, la expresión *el hiperplano h* , en realidad hace referencia al hiperplano $h(\boldsymbol{x}) = 0$.

1.5.1.2. Predicado de clasificación correcta

Utilizando la función f, se puede expresar de forma simple que no hay ejemplares $\boldsymbol{x_i}$ clasificados incorrectamente cuando f le asigna a cada uno su clase correspondiente y_i, definiendo :

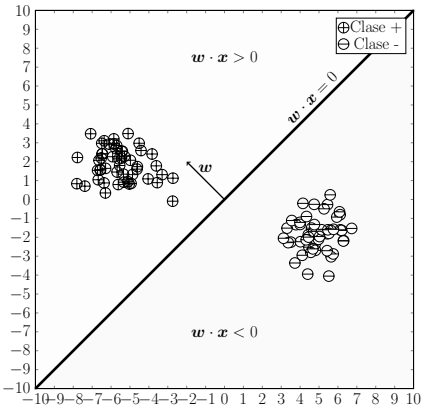
$$p \equiv \forall \boldsymbol{x_i} \in D,\ f(\boldsymbol{x_i}) = y_i \tag{1.4}$$

1.5.1.3. Regla de actualización de w y b

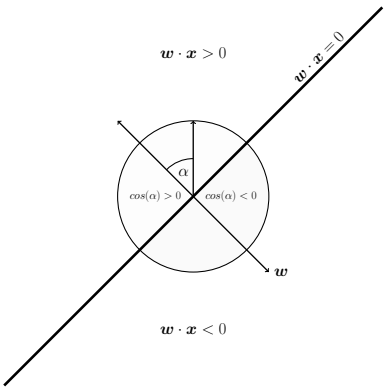
Para la modificación, se asumirá $b = 0$ momentáneamente, es decir, se asume que h es un hiperplano lineal, que pasa por el origen. El mismo está ahora dado por $h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$.

A continuación, se deducirá una regla para cambiar a $\boldsymbol{w}$ de manera que dado un $\boldsymbol{x}$ clasificado incorrectamente se pueda calcular un $\boldsymbol{w}'$ que tenga menos error que $\boldsymbol{w}$ con respecto a $\boldsymbol{x}$.

Para eso, dado que $b = 0$, si α es el "ángulo" entre $\boldsymbol{x}$ y $\boldsymbol{w}$, entonces de acuerdo con la definición geométrica del producto punto $\cdot$, $h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \|\boldsymbol{w}\|\,\|\boldsymbol{x}\|\,cos(\alpha)$. Por ende, el signo de h es el signo de $cos(\alpha)$. Si $\alpha \in (-90, 90) \rightarrow h > 0$ y si $\theta \in (90, 270) \rightarrow h < 0$.



(a) Regiones positivas y negativas para un hiperplano lineal $\boldsymbol{w} \cdot \boldsymbol{x} = 0$ en $\mathbb{R}^2$ (con $b = 0$)



(b) Perspectiva trigonométrica del signo de $\boldsymbol{w} \cdot \boldsymbol{x}$

Se puede utilizar esta relación para derivar la regla de actualización. Dado un ejemplar $\boldsymbol{x}^+$, con clase $y^+ = 1$. Si $h(\boldsymbol{x}^+) = -1$, el ejemplar está clasificado incorrectamente.

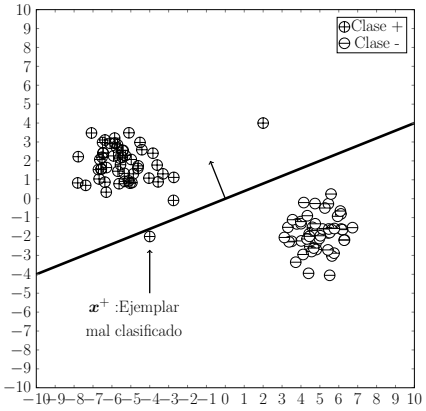


Figura 1.13: Un ejemplar x^+ mal clasificado.

De forma tentativa, se puede definir una regla simple de modificación de w para arreglar dicho error: asignar $w \leftarrow x^+$. De esta manera, claramente $w \cdot x^+ > 0$ y entonces el patrón se clasifica correctamente [7].

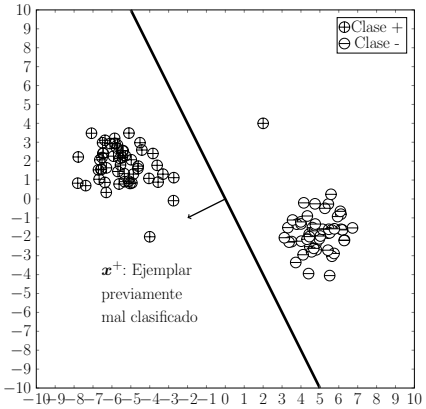


Figura 1.14: El ejemplar x^+ es clasificado correctamente, pero se ha perdido la clasificación correcta de otros ejemplares.

Esta idea también funciona con x^- tal que $y^- = -1$ y $h(x^-) = 1$, donde el problema es que x^- está del mismo lado que la normal w.

[7] Asumiendo $x_0 \neq \mathbf{0}$

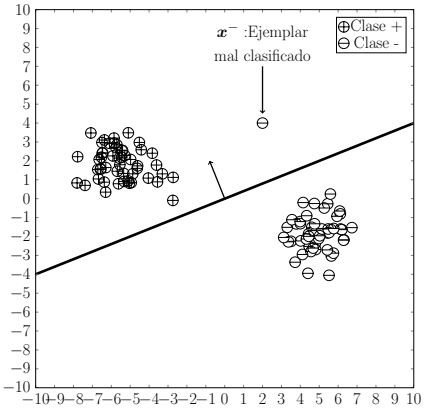


Figura 1.15: Un ejemplar x^- mal clasificado.

En este caso, se asigna $w \leftarrow -x^-$ y así $w \cdot x^- = -x^- \cdot x^- < 0$.

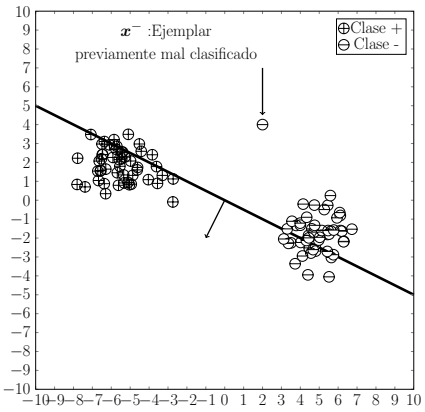


Figura 1.16: El ejemplar x^- es clasificado correctamente, pero se ha perdido la clasificación correcta de otros ejemplares.

En cierto sentido, de esta forma se elige el w óptimo para cada x por separado, ya que el ejemplar queda perpendicular al hiperplano y por ende se maximiza la distancia desde x hasta este último. Pero de esta manera para cada x se hace un cambio muy extremo. Si se repite esto para todos los ejemplares, w se irá adaptando a cada uno en particular, olvidando completamente los hiperplanos anteriores. Con esta regla de modificación solo se consigue clasificar correctamente a todos los ejemplares si por casualidad algún x es la normal de un hiperplano adecuado para ello.

Se necesita una regla que sea más conservadora con los cambios, que modifique w para que tenga menos error respecto a un x, pero que no

descarte todo el progreso hecho en las iteraciones anteriores.

El algoritmo para entrenar un Perceptrón propone corregir $\boldsymbol{w}$, dado un ejemplar mal clasificado $\boldsymbol{x}$ de clase y, con una regla del tipo:

$$\boldsymbol{w} \leftarrow \begin{cases} \boldsymbol{w} + \boldsymbol{x} & \text{si } y = 1 \\ \boldsymbol{w} - \boldsymbol{x} & \text{si } y = -1 \end{cases} \tag{1.5}$$

La idea es que en vez de reemplazar $\boldsymbol{w}$ directamente por $\boldsymbol{x}$ o $-\boldsymbol{x}$, $\boldsymbol{w}$ se convierte de a poco en los vectores $\boldsymbol{x}$. Se puede pensar en dicha conversión de la siguiente manera. Si $\boldsymbol{w}_\lambda = \boldsymbol{w} + \lambda \boldsymbol{x}^+$, a medida que $\lambda \to \infty$, $\boldsymbol{w}_\lambda$ apunta cada vez más en la dirección de $\boldsymbol{x}^+$, ya que en comparación $\|\boldsymbol{w}\| \ll \|\lambda \boldsymbol{x}^+\|$. Se habla de dirección ya que $\lambda \boldsymbol{x}^+$ representa el mismo hiperplano para todo $\lambda \in \mathbb{R}$, por ende no importa que $\|\boldsymbol{w}_\lambda\|$ también se haga arbitrariamente grande. Entonces, en cada iteración el algoritmo toma un ejemplar mal clasificado y "acerca" o "aleja" $\boldsymbol{w}$ a dicho ejemplar.

Para simplificar dicha regla de actualización eliminando los casos, se puede aprovechar el hecho de que y dice con su signo justamente si hay que sumar o restar $\boldsymbol{x}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x} \tag{1.6}$$

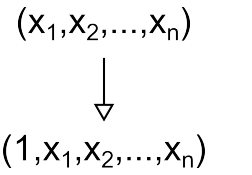
También se puede controlar la magnitud de las correcciones introduciendo una **tasa de aprendizaje** $\alpha \in \mathbb{R}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha y \boldsymbol{x} \tag{1.7}$$

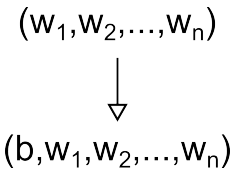
Si $\alpha \approx 0$, $\boldsymbol{w}$ se cambia poco en cada iteración y el aprendizaje es muy lento pero seguro, ya que recorre el espacio de posibles $\boldsymbol{w}$ minuciosamente. Si $\alpha \gg 0$, $\boldsymbol{w}$ se cambia mucho en cada iteración y por ende el comportamiento del entrenamiento se vuelve más errático, similar al de la regla $\boldsymbol{w} \leftarrow \boldsymbol{x}$; se debe encontrar un punto medio para dicho valor, dependiente de cada problema a resolver.

1.5.1.4. Re-introducción del bias b

Por último, para esta derivación se asumió que $b = 0$, o sea, que el hiperplano era lineal, pasando por el origen de coordenadas. Para volver a incluir el bias en esta formulación, se puede utilizar el truco de ampliar los ejemplares de entrada de vectores en $\boldsymbol{x} \in \mathbb{R}^d$ a vectores en $\boldsymbol{x}' \in \mathbb{R}^{d+1}$, donde $x_1 = 1$, y lo mismo con $\boldsymbol{w} \in \mathbb{R}^n$ a $\boldsymbol{w}' \in \mathbb{R}^{n+1}$.



(a) Conversión del ejemplar $\boldsymbol{x} \in \mathbb{R}^d$ a $\boldsymbol{x}' \in \mathbb{R}^{d+1}$, con $x_1 = 1$.



(b) Conversión del vector normal $\boldsymbol{w} \in \mathbb{R}^d$ a $\boldsymbol{w}' \in \mathbb{R}^{d+1}$, con $w_1 = b$.

De esta manera el componente $\boldsymbol{w}_1$ representa el bias b, es decir, $\boldsymbol{w}_1 = b$, y así se puede mantener la misma regla de actualización, ya que:

$$h(\boldsymbol{x}') = \boldsymbol{w}' \cdot \boldsymbol{x}' = (b, \boldsymbol{w}) \cdot (1, \boldsymbol{x}) = b + \boldsymbol{w} \cdot \boldsymbol{x} \tag{1.8}$$

En lo siguiente se hará referencia $\boldsymbol{w}'$ y $\boldsymbol{x}'$ simplemente como $\boldsymbol{w}$ y $\boldsymbol{x}$, tomando atención a que son ahora vectores de $\mathbb{R}^{d+1}$.

1.5.1.5. Algoritmo clásico del Perceptrón

El algoritmo tradicional del Perceptrón es, entonces:

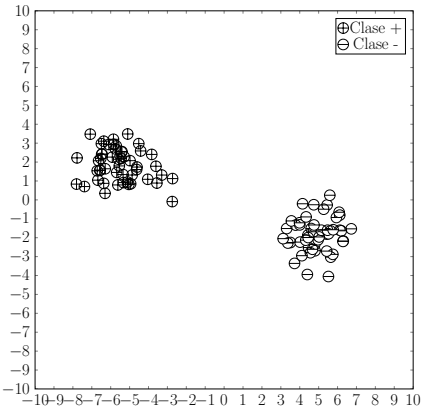
Data: Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, $D \subset \mathbb{R}^{d+1}$, donde a cada $\boldsymbol{x_i} \in \mathbb{R}^d$ original se le agregó un 1 con el objetivo descripto en el párrafo anterior
Una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Una tasa de aprendizaje α.
Result: Un hiperplano descripto por $\boldsymbol{w} \in \mathbb{R}^{d+1}$
$\boldsymbol{w} \leftarrow random(d+1)$;
while $\exists \boldsymbol{x_i} \in D : f(\boldsymbol{x_i}) \neq y_i$ **do**
  for $\boldsymbol{x_i} \in D$ **do**
    if $f(\boldsymbol{x_i}) \neq y_i$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha y_i \boldsymbol{x_i}$;
    end
  end
end

Algoritmo 1: Algoritmo de aprendizaje del Perceptrón

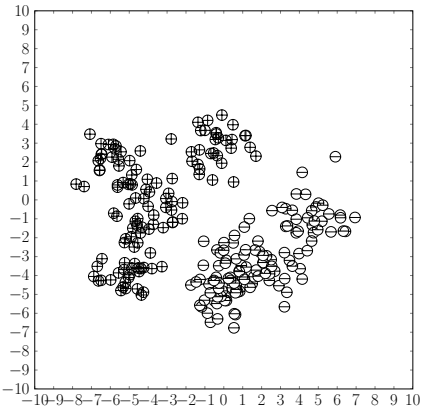
Por último, el Perceptrón es un tipo de clasificador que particiona el dominio $\mathcal{P}$ en dos; clase $+1$ y clase -1. Entonces, existen solo dos regiones de voronoi, que cubren todo el espacio. Esto tiene la consecuencia de que el Perceptrón, un clasificador lineal, solo puede indicar si un ejemplar es de una clase u otra, pero no puede indicar si el mismo no es de ninguna clase. Entonces, en ocasiones, se debe complejizar la superficie utilizada para separar las clases para poder modelar su distribución en el dominio.

1.5.1.6. Separabilidad lineal y Superficies de separación

En la formulación anterior del algoritmo del Perceptrón hay una expectativa de que, ejecutando este procedimiento una cantidad finita de pasos, se encuentre un hiperplano que divida los dos conjuntos. Según el teorema de convergencia del Perceptrón [52], esto está garantizado *siempre y cuando D* sea **linealmente separable**. Un conjunto de ejemplares pertenecientes a dos clases es linealmente separable justamente cuando existe un hiperplano $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ que puede separar los ejemplares de una clase de la otra.



(a) Dos clases linealmente separables



(b) Dos clases que no son linealmente separables

En la práctica, la gran mayoría de los dominios está compuesto por problemas cuyos ejemplares no son linealmente separables. En dichos casos existen dos alternativas. La primera, entrenar un clasificador que no utilice hiperplanos y emplee algún método de clasificación no lineal. El hiperplano es un tipo de **superficie de separación**. Un clasificador no lineal emplea una superficie de separación no-lineal, que aumenta el poder de clasificación porque resulta más flexible que un hiperplano.

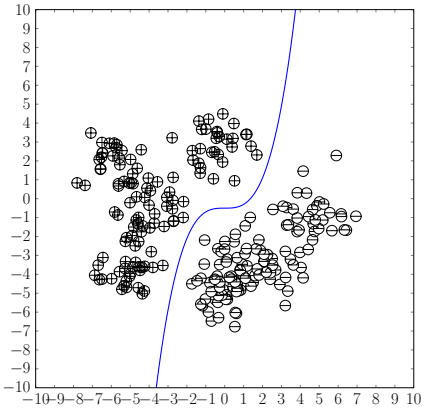
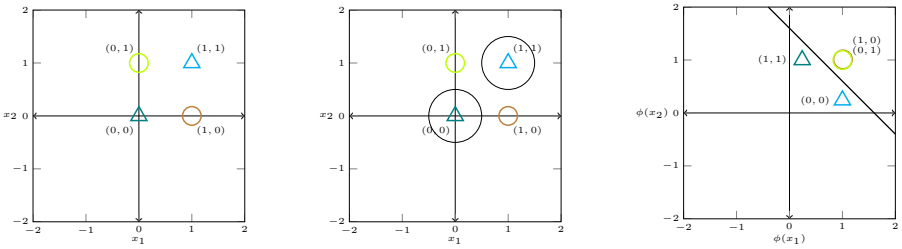


Figura 1.19: Un clasificador que utiliza una superficie de separación no lineal $y = ax^3$.

La segunda es aplicar una **transformación no-lineal** del espacio de entrada a otro espacio en donde los ejemplares si sean linealmente separables. En el problema clásico del **xor**, hay un conjunto de datos que no se puede separar con un Perceptrón. Si se aplica una transformación no lineal, a un espacio de la misma dimensión, se pueden obtener nuevas representaciones de los ejemplares de entrada que si son linealmente separables. Esto es equivalente en muchos casos a tener una superficie no-lineal de separación, pero con la simpleza de trabajar con un hiperplano en el clasificador.

Por este motivo, el concepto de separabilidad lineal sigue siendo extremadamente útil en todo tipo de problemas.

En la siguiente figura, la imagen a) muestra ejemplares de dos clases, triángulo y círculo, que no son linealmente separables ya que no existe un hiperplano que ponga de un lado los triángulos y de otro lado los círculos. La imagen b) muestra un esquema de la función de transformación del espacio descripto en a). Esta es una transformación no-lineal basada en mezclas gaussianas, con 2 centros en los elementos de la clase triángulo. La función de transformación es $\phi(\boldsymbol{x}) = (e^{-(\|\boldsymbol{x}-(1,1)\|}, e^{-(\|\boldsymbol{x}-(0,0)\|})$. Finalmente, la imagen c) muestra el espacio transformado, donde ahora el conjunto de datos es linealmente separable. Los ejemplares de la clase triángulo se transformaron en los puntos $(1, \sqrt{2})$ y $(\sqrt{2}, 1)$, y los de la clase círculo en $(1, 1)$.



(a) El problema del xor.

(b) Esquema de las funciones de transformación.

(c) Espacio transformado.

1.5.1.7. Clasificación aproximada y medidas de error

En la gran mayoría de casos, resulta imposible clasificar perfectamente un conjunto de datos. Se puede proponer entonces un algoritmo modificado de entrenamiento del perceptron que no se detenga cuando todos los ejemplares de entrenamiento están clasificados correctamente, sino cuando el error cometido por el Perceptrón es lo suficientemente bajo[8]. Para ello, se asumirá que existe una función $\rho(D, f)$ que mide el error del

[8]Se está asumiendo que el error puede llevarse a un valor menor que la tolerancia ϵ. En casos prácticos no se puede saber esto de antemano, por ende se suele agregar un contador de iteraciones t al algoritmo, estableciendo previamente una cantidad de iteraciones máximas en las que se intentará disminuir el error.

clasificador f en el conjunto de ejemplares D, de alguna manera.

Data:
Un conjunto de ejemplares $D \in \mathbb{R}^d$
Una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Una tasa de aprendizaje α.
Una tolerancia al error ϵ
Una función que mide el error de clasificación ρ
Result: Un hiperplano descripto por $\boldsymbol{w} \in \mathbb{R}^{d+1}$
$\boldsymbol{w} \leftarrow random(d+1)$;
$error \leftarrow \infty$;
while $error > \epsilon$ **do**
  for $\boldsymbol{x_i} \in D$ **do**
    if $f(\boldsymbol{x_i}) \neq y_i$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha y_i \boldsymbol{x_i}$;
    end
  end
  $error = \rho(D, f)$;
end

Algoritmo 1: Algoritmo de aprendizaje del Perceptrón con tolerancia al error.

Como caso especial, si $\epsilon = 0$, se vuelve a exigir que el entrenamiento continúe hasta que se separen perfectamente todos los ejemplares.

Se puede definir a ρ como dependiente de una función de pérdida ℓ, que mide el error para un ejemplar $\boldsymbol{x}$ en particular, en base a su clase, y, y a la salida del clasificador, $f(\boldsymbol{x})$.

Como ejemplo, la función de error ρ más general es el porcentaje de ejemplares mal clasificados. En ese caso, la función de pérdida ℓ_0 es 1 si el ejemplar está mal clasificado y 0 de lo contrario:

$$\ell_0(\boldsymbol{x}, f) = 1 - \delta(y, f(\boldsymbol{x}))$$

donde y es la clase de $\boldsymbol{x}$ y δ es el delta de Kronecker $\delta(a, b) = \begin{cases} 1 & \text{si } a = b \\ 0 & \text{de lo contrario} \end{cases}$.

En base a eso, se define la función ρ_{ℓ_0} que utiliza la función ℓ_0 para medir el error como el porcentaje de ejemplares mal clasificados:

$$\rho_{\ell_0}(D, f) = \frac{\sum\limits_{\boldsymbol{x_i} \in D} \ell_0(\boldsymbol{x_i}, f)}{|D|}$$

Existen varias funciones ρ y ℓ que se podrían definir para medir el error de clasificación, y se presentarán otras en las siguientes secciones.

1.5.2. Experimentos de clasificación

Una vez entrenado un clasificador, típicamente se quiere conocer que tan bien clasifica a los ejemplares del dominio del problema a resolver. Para evaluar un clasificador generalmente se realizan los siguientes pasos:

- Obtener los datos de los ejemplares, preprocesarlos [9], calcular las características correspondientes.
- Seleccionar los parámetros del clasificador. [10]
- Entrenar un clasificador con los ejemplares y algún algoritmo de entrenamiento.
- Probar el modelo generado obteniendo alguna medida de error (las siguientes secciones discutirán varias formas de hacer esto).

Se llamará a esta serie de pasos un **experimento**. El resultado del experimento es el modelo entrenado y una medida de error del mismo respecto al dominio $\mathcal{P}$ y la asignación de etiquetas y_i para los ejemplares.

El objetivo del experimento es el último punto, la evaluación del desempeño del modelo. Para ello, se pensará en el experimento como un experimento estadístico, que estimará el desempeño del modelo dado el dominio $\mathcal{P}$ y la asignación de etiquetas y_i.

Como en toda prueba estadística, interesará el *comportamiento promedio* del clasificador, y poder caracterizar su variabilidad.

1.5.2.1. Fuentes de variabilidad

Un experimento de clasificación tiene dos fuentes principales de variabilidad o aleatoriedad.

La primera está dada por los datos utilizados para entrenar y probar el clasificador. Distintos datos para entrenar generan modelos distintos, y por ende el error distinto. De la misma forma, usar datos distintos

[9] El preprocesamiento involucra limpiar los ejemplos, quitando casos anómalos, ruido, errores de los mismos, y en ocasiones llevar los mismos a una forma canónica o normalizarlos.

[10] Por ejemplo, si el clasificador es un perceptrón, se debe seleccionar la tasa de aprendizaje.

para probar el clasificador hará que varíe el valor del estimador calculado. En general, cuantos más datos para entrenar pueda usar el algoritmo de entrenamiento y mejor sea su calidad (representatividad de la variabilidad del dominio del problema, poco error de medición), mejor será el modelo generado, por las mismas razones que el ajuste a una curva en base a puntos de ejemplo de la misma es más preciso con más puntos.

De la misma forma, cuanto más y mejores datos se utilicen para evaluar el error del clasificador mejor será la confianza de las pruebas, por las mismas razones que una muestra grande de datos es deseable para realizar un test estadístico.

La segunda se encuentra en la generación del clasificador. Muchos algoritmos de entrenamiento utilizan el enfoque de partir de un estado generado aleatoriamente, y luego mejorarlo iterativamente durante el entrenamiento [11]. La inicialización es aleatoria, y como el estado final depende del estado inicial, también es aleatorio el error del clasificador. En otros casos, el estado inicial puede estar fijado de antemano, pero los pasos para llegar al estado final contienen elementos aleatorios.

Si bien esta fuente de aleatoriedad puede eliminarse utilizando una semilla fija en la generación de números aleatorios, dicha estrategia impediría el análisis de una serie de experimentos con herramientas estadísticas, lo cual es necesario para obtener una medida de error del clasificador en promedio realizando varios experimentos y agregando los resultados.

El concepto de experimento será útil para hacer referencia a esta serie de pasos en las siguientes secciones. El objetivo de un experimento es generar y evaluar un clasificador. En la evaluación, interesa esencialmente el comportamiento del clasificador entrenado en *nuevos* ejemplares del problema, no vistos en la etapa de entrenamiento; este es el problema de la **generalización**.

1.5.3. Generalización

Un clasificador se genera en base a un conjunto de datos de *entrenamiento*. Una vez generado el clasificador, interesa conocer su desempeño, no en el conjunto de datos de entrenamiento, sino en su dominio $\mathcal{P}$. El desempeño de un clasificador en ejemplares del problema no vistos en la etapa de entrenamiento se denomina como la **capacidad de generalización** del clasificador.

[11] El algoritmo backpropagation descripto en el capítulo 3 de redes neuronales posee esta característica.

Para conocer esta capacidad de generalización, se aplican métodos estadísticos para estimar el error del clasificador en la población de posibles ejemplares $\mathcal{P}$. Dado que es imposible probar todos los elementos de $\mathcal{P}$, se utiliza algún conjunto de datos $D \subset \mathcal{P}$ para estimar el error.

Hay distintos estimadores de error, y algunos son específicos al tipo de clasificador a utilizar. El más simple y genérico es el porcentaje de ejemplares clasificados incorrectamente $\rho_{\ell_0}(D, f)$, previamente definida para el algoritmo del perceptrón. Esta expresión del error depende del conjunto de datos D, y es un estimador del valor que realmente interesa, el error esperado de f en $\mathcal{P}$:

$$\rho_\ell(\mathcal{P}, f) = E_{\boldsymbol{x}}[\ell(\boldsymbol{x}, f)] \tag{1.9}$$

Entonces, como primera aproximación, un experimento podría obtener el conjunto de datos D, entrenar $f = g(D)$ y luego estimar su error con $\rho_\ell(D, f)$.

1.5.3.1. Validación con retención

El problema con esta estrategia es que los algoritmos de entrenamiento en general están basados, en forma implícita o explícita, en la idea de encontrar un f que justamente minimice una medida de error como $\rho_\ell(D, f)$. Entonces, si se utiliza el mismo conjunto de datos para entrenar un clasificador y estimar su error se obtendrá una estimación que subestima el error real de f en $\mathcal{P}$. Es como generar un modelo a partir de ciertos datos, y luego validar dicho modelo con esos mismos datos; claramente, si el modelo está bien generado, es improbable que tenga un alto grado de error en los datos de entrenamiento.

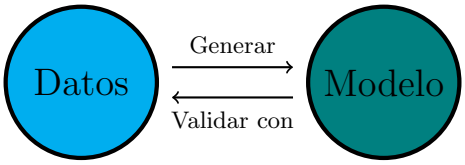


Figura 1.21: El argumento circular de generar un modelo a partir de un conjunto de datos y validarlo con el mismo conjunto de datos.

En un caso extremo, si $y(\boldsymbol{x})$ es la etiqueta correcta para $\boldsymbol{x}$, se puede definir un clasificador f_D tal que:

$$f_D(x) = \begin{cases} y(\boldsymbol{x}) & \text{si } \boldsymbol{x} \in D \\ \text{clase aleatoria} & \text{de lo contrario} \end{cases} \tag{1.10}$$

Dicha función f_D[12] tiene un error de clasificación $\rho_\ell(D, f_D) = 0$ en el conjunto D, pero clasifica de forma aleatoria cualquier otro conjunto de ejemplares. Mientras que este es un caso claramente trivial, sirve para ilustrar la importancia de no hacer estimaciones de error con los mismos datos con los cuales fue entrenado un clasificador. De todas formas, en la práctica es usual obtener funciones f que no tengan errores de clasificación en el mismo conjunto en que fueron entrenados, pero que varían ampliamente en su capacidad de clasificar otros ejemplares de $\mathcal{P}$.

Para solucionar este problema, es necesario entonces probar a f con ejemplares distintos a los utilizados para entrenar. La idea más simple es tomar a D y en lugar de utilizar *todos* los ejemplares para entrenar, dividir el conjunto en dos: D_e, el conjunto de entrenamiento, y D_p el conjunto de prueba que se reserva para hacer las pruebas de generalización, calculando $\rho_\ell(D_p, f)$, donde $f = g(D_e)$. Este método se denomina **holdout validation** (validación con retención).

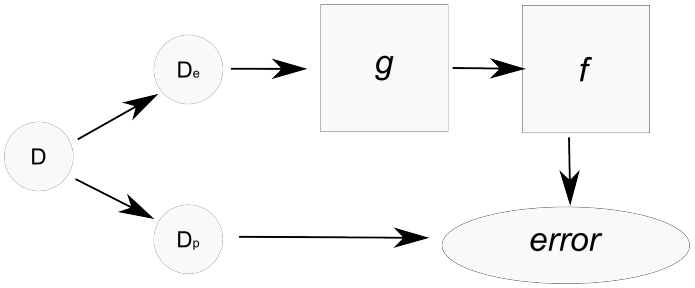


Figura 1.22: Esquema del proceso de validación con retención. El conjunto de ejemplares D se divide en dos, el de entrenamiento D_e y el de prueba D_p. Mediante la función g y D_e, se entrena el modelo f. Luego, se obtiene una medida del error de f en $\mathcal{P}$ utilizando el conjunto D_p.

Se puede considerar entonces a D_p como una muestra de $\mathcal{P}$ *limpia*, es decir, independiente de D_e y representativa de $\mathcal{P}$, que permite estimar el error esperado del clasificador entrenado f en $\mathcal{P}$, $E_{\boldsymbol{x}}(\ell(x, f))$.

Entonces, dada una medida de error ρ_ℓ, se define formalmente el concepto de un modelo bien entrenado y de generalización.

- Un modelo f bien entrenado es aquel para el cual $\rho_\ell(D_e, f) \simeq 0$.
- Un modelo f que generaliza es aquel para el cual $\rho_\ell(D_e, f) \simeq \rho_\ell(\mathcal{P}, f)$. En la práctica el segundo término es imposible de conocer, y por eso se estima como $\rho_\ell(D_e, f) \simeq \rho_\ell(D_p, f)$.

[12]Para que f_D sea realmente una función, se asume que dichas etiquetas de clase aleatorias se asignan en la definición de la función y no en su evaluación

Nuestro ejemplo de la función f_D es un caso en el cual el modelo estaría bien entrenado, ya que $\rho_\ell(D_e, f_D) = 0$, pero claramente $\rho_\ell(\mathcal{P}, f_D)$ resulta altísimo. Por ende, ambas condiciones son necesarias para un clasificador efectivo.

1.5.4. Sobre-especialización y regularización

Como ya se mencionó, hacer un experimento de clasificación involucra estimar el error de f en $\mathcal{P}$ usando D. Para eso divide D en D_p y D_e, y se mide el error en D_p. Si dicho error es aceptable, se considera que el clasificador es adecuado para la tarea. Ahora bien, si el error es muy grande puede ser que el clasificador no sea adecuado para la tarea, no se haya entrenado lo suficiente, o los datos sean insuficientes o de mala calidad. Si además se mide el error de f en D_e y también es alto, se obtiene una confirmación de ello. Pero si f se comporta bien con D_e, existe otra razón posible: el clasificador se adaptó tanto a los ejemplares de D_e que no puede generalizar cuando se presentan los de D_p.

Entonces, si un clasificador se especializa tanto en el conjunto de ejemplares con el cual fue entrenado que pierde o no adquiere la capacidad de generalizar, es decir, clasificar correctamente ejemplares no vistos, ejemplares de $\mathcal{P}$ en general, se dice que sufre del fenómeno de **overfitting** (sobre-especialización) que se da cuando un clasificador tiene un nivel de error significativamente menor en el conjunto de entrenamiento que en nuevos ejemplares del problema. Esto puede suceder cuando un clasificador se entrena de forma excesiva o utilizando métodos que no se adaptan bien al dominio del problema.

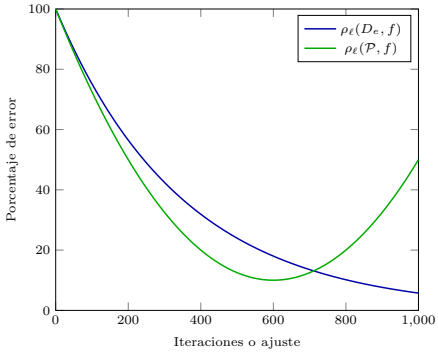
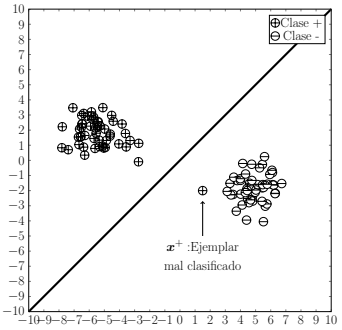
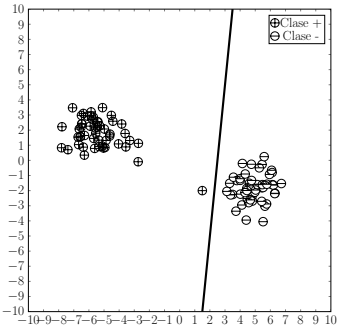


Figura 1.23: Curvas típicas de: $\rho_\ell(D_e, f)$ y $\rho_\ell(\mathcal{P}, f)$ en función de la cantidad de iteraciones de entrenamiento o fuerza del ajuste de f con D. El error se puede decrementar arbitrariamente para el conjunto de entrenamiento, pero llega un momento donde el entrenamiento extra incrementa el error sobre el dominio del problema $\mathcal{P}$.

En este sentido, si bien es importante entrenar un clasificador adecuadamente para minimizar el error en D_e, un entrenamiento que ajuste f demasiado a D_e traerá un error mayor en D_p.



(a) Conjunto de datos D con dos clases, donde hay un outlier. Si bien el hiperplano no clasifica correctamente todos los ejemplares, provee una separación coherente entre clases.



(b) Mismo D, con un hiperplano sobre-entrenado: clasifica bien todas las instancias de D pero no refleja la distribución real de las clases.

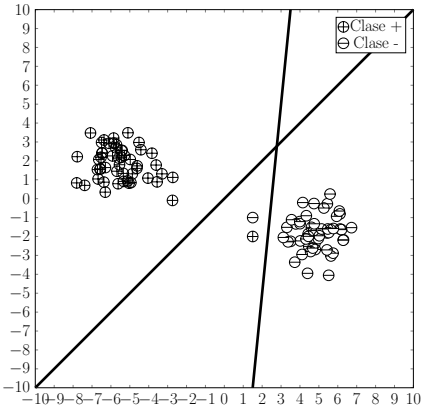
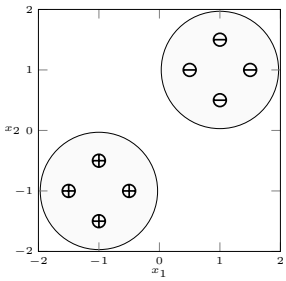
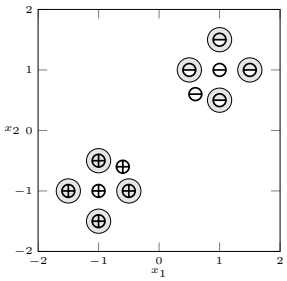


Figura 1.25: Comportamiento de ambos clasificadores ante un nuevo ejemplar.

El problema del overfitting también se da, en ocasiones, cuando la potencia del clasificador utilizado excede la necesaria para resolver un problema dado. En este caso, puede que el mismo se ajuste tan bien a los datos de entrenamiento que no pueda lidiar con desviaciones mayores de los mismos, como se muestra en la figura.



(a) Distribución real de ejemplares por clases



(b) Distribución aprendida por el clasificador. Las regiones de Voronoi se ajustan tan perfectamente a cada ejemplar que no consideran el espacio entre ejemplares de una clase como de la misma clase.

En ocasiones, lo que se necesita es limitar el poder del clasificador. En la figura anterior, se consideró el poder computacional del clasificador como bastante alto ya que debe de alguna manera recordar o incluir en su modelo todos los ejemplares usados para el entrenamiento de forma bastante específica. Si se limita la cantidad de ejemplares a recordar en el clasificador en el caso anterior, se podría obtener una clasificación mejor:

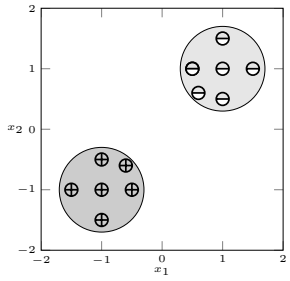


Figura 1.27: Distribución aprendida por el clasificador con capacidad limitada. Hay menos regiones de Voronoi que en el primer caso, pero son más grandes y menos específicas, obteniendo una clasificación más acertada.

En este caso se puede considerar la cantidad de ejemplares en los que basarse como un parámetro del modelo. Este método de limitación del poder computacional se conoce como un esquema de **regularización**, y sigue el principio de la navaja de Occam: si dos modelos explican un fenómeno, por principio se elige el más simple de los dos. Dicho esquema puede estar basado en la selección de algún parámetro del entrenamiento del clasificador, o puede estar incluido directamente mediante un proceso de entrenamiento que penalice clasificadores complejos en preferencia de los más simples. En el último caso, el esquema general para entrenar un clasificador podría ser:

$$\underset{f=g(D_e)}{\text{Minimizar}} \quad error(f, D_e) + complejidad(f)$$

Donde *complejidad* es alguna función que mide la complejidad de f, dependiente del algoritmo de entrenamiento del clasificador utilizado. Se verán ejemplos de esquemas de regularización cuando se desarrolle el modelo SVM y los algoritmos de entrenamiento para redes feedforward en los capítulos siguientes.

En otros casos, cambiar a un modelo de clasificador más simple que se adapte mejor al problema puede solucionar el problema.

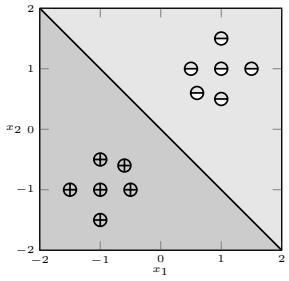


Figura 1.28: Distribución aprendida por un clasificador basado en hiperplanos.

De todas maneras, el clasificador con hiperplanos gana simplicidad porque implícitamente está haciendo una hipótesis (fuerte) de que la mitad de $\mathbb{R}^2$ pertenece a una clase, y la otra mitad a otra; es decir, para todo punto en el espacio de ejemplares, hay una clase correspondiente. El modelo de hiperesferas, por otro lado, sólo asigna una clase a los puntos cercanos a los conocidos durante el entrenamiento.

1.5.5. Modelo de clasificación con estructura probabilística

En el modelo de clasificación descripto en las secciones anteriores, se realizaron ciertas simplificaciones que no suelen ser ciertas en la práctica. Las mismas constan de asumir que:

- Las clases y son equiprobables.
- Los ejemplares $\boldsymbol{x}$ también lo son.
- A cada ejemplar corresponde una clase única.

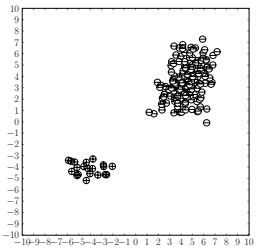
En la práctica, casi siempre hay cierto conjunto de patrones y clases más probables que otras. Además, los problemas a resolver suelen tener cierto error inherente, en el sentido de que existen ejemplares arbitrariamente parecidos $\boldsymbol{x}_0 \simeq \boldsymbol{x}_1$ con $y_0 \neq y_1$. Es decir, no siempre existe un clasificador que distinga perfectamente los ejemplares de todas las clases.

Entonces, estas tres consideraciones resultan una simplificación efectiva en ciertos casos, pero pueden traer problemas a la hora de generar un clasificador efectivo cuando las hipótesis no se cumplen. A continuación se describen tres situaciones donde se viola cada asunción:

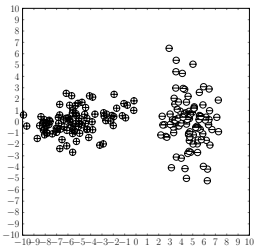
- Ciertas clases pueden ser más ocurrentes que otras. Por ejemplo, si hay que clasificar muestras de sangre para decidir si tienen o no cierta enfermedad, como la de Huntington, la clase 0, correspondiente a los sanos, en general será mucho más grande que la clase 1, correspondiente a los enfermos. Suponiendo que dichas proporciones son 99.9 % y 0.1 %, respectivamente, si un clasificador se entrena *a ciegas*, el algoritmo de aprendizaje podría determinar que con clasificar todas las muestras como sanas tendría sólo un 0.1 % de error, y entrenar un modelo que haga justamente eso. Este modelo claramente no es útil, pero así pareciera en términos del error obtenido.

- De la misma forma, ciertos ejemplares $\boldsymbol{x}$ pueden ocurrir con más frecuencia que otros; en este caso, si uno de los indicadores de la enfermedad es la aparición de una proteína de baja ocurrencia en la población, existirán menos ocurrencias de ejemplares que codifiquen dicha proteína, lo cual no implica que dicho patrón sea un outlier o sea menos importante. Por otro lado, puede ocurrir que algún ejemplar del conjunto de entrenamiento sea un outlier y no sea importante, pero se lo considere con el mismo *peso* que los otros patrones.

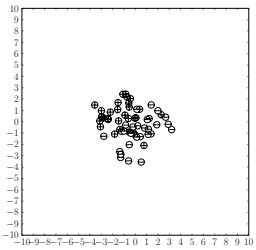
- Además, se asumió que un ejemplar $\boldsymbol{x}$ solo puede tener una clase y asociada, es decir, no puede haber ambigüedades respecto de la clase de un ejemplar. Siguiendo el mismo ejemplo, en general las características del análisis de la sangre codificadas en $\boldsymbol{x}$ no tienen información completa sobre todas las causas y los indicadores de la enfermedad. Entonces, es muy posible tener dos ejemplares, $\boldsymbol{x}_0$ y $\boldsymbol{x}_1$ tales que $\boldsymbol{x}_0 = \boldsymbol{x}_1$ pero $y_0 \neq y_1$. Es decir, al tener información incompleta, se pueden tener inconsistencias en la asignación de etiquetas *verdaderas* a los ejemplares.



(a) Desbalance en la distribución de ejemplares por clase



(b) Distribución de los ejemplares no uniforme.



(c) Ambigüedad en la asignación de clases para algunos ejemplares.

En base a estas consideraciones, se puede extender el modelo original agregando una estructura probabilística en el dominio $\mathcal{P}$ y en el espacio de clases C, de modo que ciertos ejemplares y clases sean más probables que otros, y además, haya una distribución conjunta de ejemplares y clases.

Entonces, se considera una distribución de probabilidad $P(Y)$ de las clases posibles de los patrones, C. Además, se asume una distribución $P(\mathbf{X})$ (generalmente desconocida) de los ejemplares sobre el conjunto $\mathcal{P}$. Esto da entonces una probabilidad condicional $P(Y|\mathbf{X})$ que se desea averiguar; es decir, dado un ejemplar, se desea saber la probabilidad de que pertenezca a cierta clase. De esta forma, se tienen en cuenta las tres simplificaciones mencionadas anteriormente.

Estas definiciones pueden verse en términos de un proceso generativo de ejemplares y su afiliación a ciertas clases. Para obtener un nuevo ejemplar y su clase, se selecciona de forma aleatoria un ejemplar $\boldsymbol{x} \in \mathcal{P}$ de acuerdo a $P(\mathbf{X})$. Luego, se le asigna una etiqueta y de acuerdo a $P(Y|\boldsymbol{x})$.

Desde esta perspectiva, se puede plantear el objetivo de un clasificador como encontrar $P(Y|\mathbf{X})$ a partir de los ejemplos de entrenamiento. Los mismos dan información sobre $P(Y, \mathbf{X})$; en base a eso, el algoritmo de aprendizaje estima $P(Y|\mathbf{X})$.

Se puede pensar entonces en la función f del clasificador como un funcional de la conjunción de funciones $f_c, i = 1, \ldots, C$, donde $f_c :: \mathcal{P} \mapsto [0..1]$. Cada función f_c estima $P(c|\boldsymbol{x})$, es decir, la probabilidad de que el ejemplar $\boldsymbol{x}$ pertenezca a la clase c. El funcional más simple es la función $f(\boldsymbol{x}) = \text{argmax}_c\{f_c(\boldsymbol{x})\}$, es decir, f selecciona la clase k tal que $f_k(\boldsymbol{x}) \geq f_c(\boldsymbol{x})$, $i = 1, \ldots, C$. Un clasificador ideal estimaría perfectamente $P(Y|\mathbf{X})$, de modo que $f_c = P(c|\boldsymbol{x})$.

Hay que considerar otra simplificación realizada, que consta de una distinción sutil pero importante. Hasta ahora se asumió que todos los ejemplares $\boldsymbol{x_i}$ han sido etiquetados correctamente con la clase y_i corres-

pondiente, pero volviendo al ejemplo anterior, ciertos pacientes pueden haber sido diagnosticados incorrectamente. Entonces, si y_i es simplemente la clase con la que fue etiquetado el ejemplar $\boldsymbol{x_i}$, la misma no siempre representa la clase del ejemplar, y se deberían considerar las probabilidades sobre un y_i^v que es la clase verdadera del mismo, de modo que el objetivo es entonces estimar $P(Y^v|\mathbf{X})$.

Esta es la base de lo que se conoce como **Statistical Learning Theory** (Teoría estadística del aprendizaje) [57, 58], la teoría subyacente en el modelo de **Support Vector Machine** (Máquina de Vectores de Soporte), un tipo de clasificador que se desarrollará en el capítulo 2.

1.5.6. Modelo de clasificación multiclase

Se puede extender el modelo nuevamente considerando el problema de que un ejemplar pertenezca realmente a *varias* clases. Por ejemplo, si se busca clasificar objetos detectados en imágenes para asignarles categorías, una mesa de madera podría entrar en las categorías *mesa*, de acuerdo a sus propiedades geométricas, y *objetos de madera*, en base a su textura. En el modelo de clasificación descripto en las secciones anteriores, se asume que hay una *única* clase "verdadera"y_i para asignar a cada patrón del dominio del problema, con cierta ambigüedad. Ahora se podría asumir que a cada $\boldsymbol{x}$ corresponde un vector de puntajes por clase $\boldsymbol{y} \in \mathbb{R}_p^{|C|}$, con $\mathbb{R}_p = [0..1]$ y $\mathbb{R}_p \subset \mathbb{R}$. Cada puntaje entonces determina el grado de pertenencia de un ejemplar a cada clase. Esto implica entrenar una función de clasificación $f :: \mathcal{P} \mapsto \mathbb{R}_p^{|C|}$, similar a la conjunción de las funciones f_c descriptas anteriormente, pero sin tomar el máximo, de modo que $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_C(\boldsymbol{x}))$.

Este modelo es una generalización del previo donde si antes $y_i = c$, ahora $\boldsymbol{y_i} = (0, .., 1, .., 0)$, donde el 1 está en la posición c.

Aún cuando el problema no sea de clasificación multiclase, este modelo puede ser útil ya que permite cuantificar el grado de ambigüedad en la clase asignada a un ejemplar.

1.6. Resumen

En este capítulo se brindan los conceptos básicos de aprendizaje automático para el resto de la tesina. Se introduce, mediante un ejemplo de reconocimiento de gestos, el desafío central del área, que consta del desarrollo de técnicas para generar modelos de problemas en base a instancias

o ejemplares de los mismos, un proceso conocido como el entrenamiento del modelo. A su vez, se hace referencia a la gran cantidad de aplicaciones de estas técnicas y su importancia para todas las áreas.

Luego se presenta una distinción entre el aprendizaje supervisado y el no supervisado, y en particular, un modelo supervisado de clasificación de ejemplares. Estos son de central importancia en esta tesina, ya que el reconocimiento de gestos es, principalmente, un problema de clasificación.

El algoritmo de perceptrón provee un ejemplo concreto de aprendizaje automático, y las secciones sobre experimentos, generalización, sobreentrenamiento y regularización describen conceptos claves para poder entrenar un modelo con éxito, es decir, un modelo que tenga poco error.

Finalmente, se presenta un modelo alternativo de clasificación, más general que el primero y con menos simplificaciones, que además considera la clasificación de ejemplares en múltiples categorías o clases.

Estos conceptos sientan las bases teóricas para los siguientes capítulos, y son centrales para el desarrollo de un modelo de clasificación adecuado que permita un eficaz reconocimiento de gestos. Los capítulos 2 y 3 describen modelos de clasificación y generación de características, con las cuales en el capítulo 7 se describen distintos tipos de clasificadores para el reconocimiento de gestos.

A continuación, se construye sobre dichos conceptos para ampliar el método de clasificación del perceptrón de dos maneras diferentes: Maquinas de Vectores de Soporte (capítulo 2) y Redes Neuronales (capítulo 3).

2

Maquinas de Vectores de Soporte (SVM)

> Why is it so popular? I believe there are several reasons. First of all, it is effective. It gives very stable and good results. From a theoretical point of view, it is very clear, very simple, and allows many different generalizations, called kernel machines.
>
> Vladimir N. Vapnik

2.1. Introducción

El separador de espacios por excelencia es el hiperplano. Pero dado un conjunto de datos con dos clases linealmente separables, pueden existir infinitos hiperplanos que separan los conjuntos.

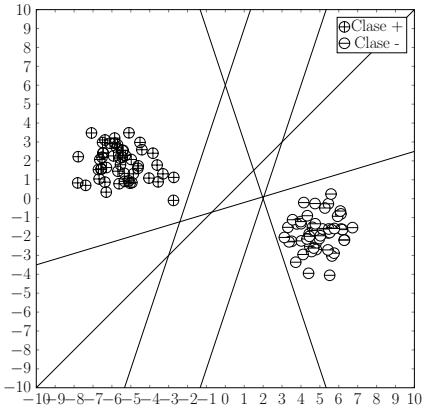


Figura 2.1: Distintos hiperplanos separadores para el mismo conjunto de datos.

Algoritmos simples como el del Perceptrón se quedan con el primero que encuentran que tenga un error lo suficientemente bajo, ya que no tienen un criterio para elegir entre todos los posibles hiperplanos que separan las clases. Si bien en la figura anterior la distribución real de los ejemplos de las clases no se conoce, y por ende no se puede asegurar cual es el hiperplano más correcto, está claro que, a priori, algunos hiperplanos capturan mejor que otros la división entre las dos clases.

Las **Máquinas de Vectores de Soporte** (SVM) extienden el perceptrón con un criterio para seleccionar entre los hiperplanos; en este modelo, se busca el *hiperplano de máximo margen*.

2.2. Clasificador de Márgen Máximo

La idea esencial, intuitiva, de las SVM es elegir el hiperplano que tenga el mayor margen M entre las dos clases. En otras palabras, la distancia de los ejemplares de entrenamiento más cercanos de cada clase al **hiperplano de máximo margen** (en adelante, simplemente el hiperplano) es la mayor posible (y es la misma para la dos clases).

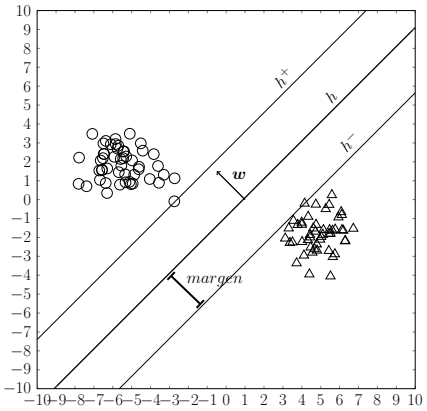


Figura 2.2: Un conjunto de datos linealmente separable, con un hiperplano de máximo margen.

Los vectores más cercanos al hiperplano se conocen como **vectores de soporte**, ya que es su ubicación exacta la que finalmente determina el hiperplano h, y los hiperplanos h^+ y h^- que lo contienen se llaman **hiperplanos de soporte**, paralelos al hiperplano h. El margen es la distancia entre un hiperplano de soporte y el hiperplano h.

La ecuación del hiperplano es:

$$h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \quad \boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^d, b \in \mathbb{R} \tag{2.1}$$

Para SVM, será conveniente codificar las clases y_i como -1 y 1 para representar de que lado del hiperplano están sus ejemplares, como con el Perceptrón.

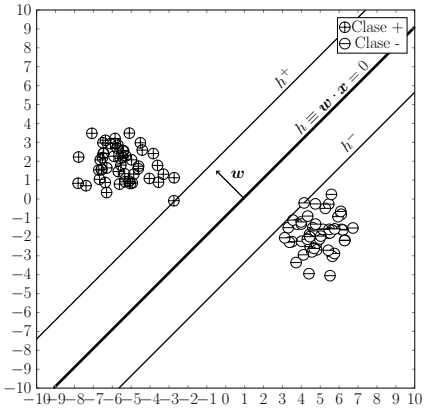


Figura 2.3: Un conjunto de datos linealmente separable, con clases $+$ y $-$. El hiperplano de máximo margen $h(\boldsymbol{x})$ que separa las clases.

Como se busca un hiperplano que clasifique correctamente a todos los ejemplares $\boldsymbol{x_i} \in D$ [1] , se pueden definir las siguientes restricciones:

$$\forall \boldsymbol{x_i} \in D \quad \begin{cases} h(\boldsymbol{x_i}) > 0 & \text{si } y_i = 1 \\ h(\boldsymbol{x_i}) < 0 & \text{si } y_i = -1 \end{cases} \tag{2.2}$$

Como el signo de y_i indica el tipo de desigualdad, se pueden combinar ambos casos como:

$$\forall \boldsymbol{x_i} \in D \quad h(\boldsymbol{x_i}) y_i > 0 \tag{2.3}$$

De manera informal, se puede especificar el problema asumiendo una función $M = margen$ que calcule el tamaño del margen entre un hiperplano y un conjunto de ejemplares D.

$$\begin{aligned} &\underset{\boldsymbol{w},b}{\text{Min}} && M = margen(\boldsymbol{w}, b, D) \\ &\text{sujeto a} && h(\boldsymbol{x_i}) y_i > 0,\ i = 1, \ldots, |D|. \end{aligned}$$

Para formalizar el problema, solo se necesita una expresión concreta de M (el tamaño del margen) en términos de $\boldsymbol{w}$ y b.

Pero antes de eso, se debe solucionar un pequeño problema técnico; la ecuación $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ está indeterminada ya que multiplicando por cualquier constante se obtienen valores de $\boldsymbol{w}$ y b distintos, pero que representan el mismo hiperplano. Esto da lugar a infinitas soluciones del problema. Entonces, se asume que se buscan los $\boldsymbol{w}$ y b con la condición de que para los vectores de soporte de cada clase, -1 y 1, $h(\boldsymbol{x}^-) = -1$ y $h(\boldsymbol{x}^+) = 1$, respectivamente. De esta manera $\boldsymbol{w}$ y b quedan completamente determinados.

Es importante notar que ahora se pide que $h(\boldsymbol{x_i}) y_i \geq 1$ en lugar de $h(\boldsymbol{x_i}) y_i > 0$, ya que los $\boldsymbol{x_i}$ más cercanos al hiperplano son los vectores de soporte, para los cuales se cumpliría entonces que $h(\boldsymbol{x_i}) y_i = 1$.

No se escribirá esta restricción explícitamente en la especificación del problema de maximización del margen ya que surgirá implícitamente en la siguiente derivación de una expresión para el margen M. Para ello,

[1] En este capítulo y el siguiente se denota D al conjunto de entrenamiento, por simplicidad. El enfoque se encuentra en el desarrollo del modelo de clasificación y su algoritmo de entrenamiento, sin considerar la utilización de métodos de evaluación como VC directamente ya que estos pertenecen a la etapa de evaluación del error del clasificador respecto a un conjunto de datos particular.

suponiendo que $\boldsymbol{x}^+$ es un vector de soporte de la clase 1, y $\boldsymbol{x}_0$ es el vector perteneciente al hiperplano ($h(\boldsymbol{x}_0) = 0$) más cercano a $\boldsymbol{x}^+$, tal que la línea que los une es perpendicular al hiperplano, o sea es paralela a $\boldsymbol{w}$.

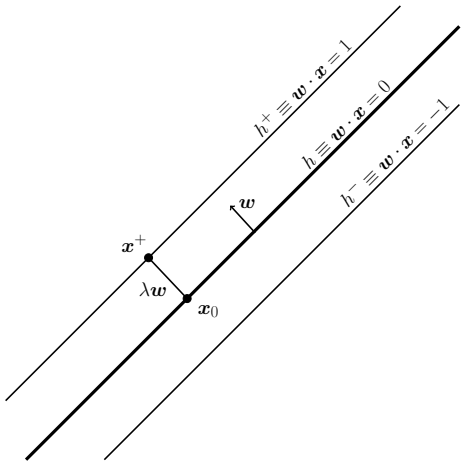


Figura 2.4: Los vectores $\boldsymbol{x}^+$ y $\boldsymbol{x}_0$, pertenecientes al hiperplano de soporte y al hiperplano de máximo margen, respectivamente. La distancia entre los vectores, o sea, la distancia entre los hiperplanos, es el tamaño del máximo margen $M = \lambda \boldsymbol{w}$.

Se puede escribir $\boldsymbol{x}^+$ como $\boldsymbol{x}^+ = \boldsymbol{x}_0 + \lambda \boldsymbol{w}$, siendo $\lambda \boldsymbol{w}$ el vector que une $\boldsymbol{x}^+$ y $\boldsymbol{x}_0$. La distancia entre $\boldsymbol{x}^+$ y $\boldsymbol{x}_0$, entre el hiperplano y el hiperplano de soporte, es $M = \|\lambda \boldsymbol{w}\|$, $\lambda \in \mathbb{R}$. Por ende:

$$
\begin{aligned}
\boldsymbol{w} \cdot \boldsymbol{x}^+ + b &= 1 \\
\boldsymbol{w} \cdot (\boldsymbol{x}_0 + \lambda \boldsymbol{w}) + b &= 1 \\
\boldsymbol{w} \cdot \boldsymbol{x}_0 + \boldsymbol{w} \cdot \lambda \boldsymbol{w} + b &= 1 \\
\boldsymbol{w} \cdot \boldsymbol{x}_0 + b + \boldsymbol{w} \cdot \lambda \boldsymbol{w} &= 1 \\
\boldsymbol{w} \cdot \lambda \boldsymbol{w} &= 1 \ (\boldsymbol{x}_0 \quad \text{pertenece al hiperplano}) \\
\lambda \|\boldsymbol{w}\|^2 &= 1 \\
\lambda \|\boldsymbol{w}\| &= \frac{1}{\|\boldsymbol{w}\|} \\
M &= \frac{1}{\|\boldsymbol{w}\|}
\end{aligned}
$$

Finalmente el problema queda planteado como:

$$\begin{array}{ll} \underset{\boldsymbol{w},b}{\text{Max}} & \dfrac{1}{\|\boldsymbol{w}\|} \\ \text{sujeto a} & h(\boldsymbol{x_i})y_i > 0,\ i = 1, \ldots, |D|. \end{array}$$

Como maximizar $\frac{1}{\|\boldsymbol{w}\|}$ es equivalente a minimizar $\frac{\|\boldsymbol{w}\|^2}{2} = \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w}$ y es más simple, se puede reformular el problema como:

$$\begin{array}{ll} \underset{\boldsymbol{w},b}{\text{Min}} & \dfrac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} \\ \text{sujeto a} & h(\boldsymbol{x_i})y_i \geq 1,\ i = 1, \ldots, |D|. \end{array}$$

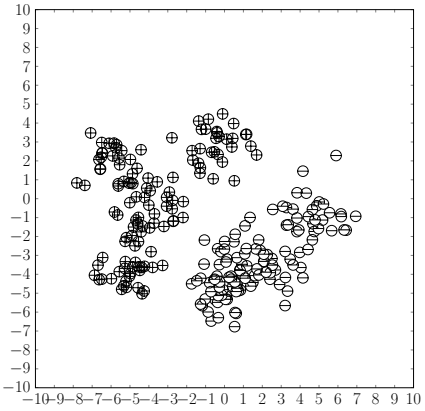
Este es un problema de optimización cuadrática ($\frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w}$), con restricciones lineales y se puede resolver con algún método de optimización cuadrática y comenzar a utilizar SVM de esa manera. Pero este no es el modelo completo de SVM, ya que es difícil de optimizar y requiere que el problema sea linealmente separable.

A continuación, se extenderá el modelo de SVM con la formulación de **Márgenes Suaves** para manejar problemas que no son linealmente separables y la presencia de ejemplares anómalos (comúnmente llamados **outliers**); se derivará la **formulación dual** que posibilita una solución más simple del problema de optimización y permite el **truco del kernel** para poder aplicar con poco costo computacional ciertos tipos de transformaciones a los ejemplares para separar mejor las clases.

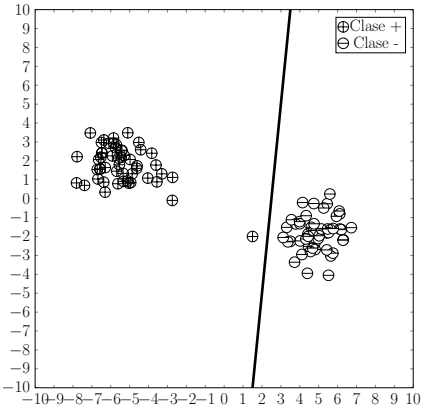
Para el lector interesado el profundizar sobre el tema, en el apéndice B.2 se describe el algoritmo de optimización **Sequential Minimal Optimization** (SMO) de Platt [59] para resolver el problema de optimización y poder entrenar una SVM.

2.3. Modelo de Márgenes Suaves

El modelo anterior asume que las clases son linealmente separables, lo cual no suele cumplirse, y además es muy sensible a la presencia de ejemplares anómalos debido a que probablemente se conviertan en vectores de soporte si se encuentran entre las dos clases pero no son representativos de ninguna.



(a) Un problema no linealmente separable



(b) Un outlier se convierte en vector de soporte y distorsiona el hiperplano de máximo margen *intuitivo*.

Una manera de lidiar con esto es mediante la extensión **Soft Margin** (de Márgenes Suaves) de SVM. Dicha extensión consiste en hacer posible que algunos de los ejemplares de entrenamiento no cumplan con la restricción $h(\boldsymbol{x_i})y_i \geq 1$ dándoles un poco de ayuda mediante una variable de holgura $s_i \geq 0$, tal que:

$$h(\boldsymbol{x_i})y_i + s_i \geq 1,\ s_i \in \mathbb{R}$$

La variable s_i "ayuda" a $h(\boldsymbol{x_i})yi$ a llegar a 1 cuando este no puede debido a elección de $\boldsymbol{w}$ y b. El problema quedaría entonces como:

$$\begin{aligned} \underset{\boldsymbol{w},b,\boldsymbol{s}}{\text{Min}} \quad & \mathcal{L}_P(\boldsymbol{w},b,\boldsymbol{s}) = \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} \\ \text{sujeto a} \quad & h(\boldsymbol{x_i})y_i + s_i \geq 1, \qquad i = 1,\ldots,|D| \\ & s_i \geq 0, \qquad i = 1,\ldots,|D| \end{aligned}$$

Ahora, si se deja así el problema, los s_i pueden tomar valores arbitrarios de manera que las restricciones se cumplirían trivialmente haciendo suficientemente grandes los s_i. De hecho, se podría definir $\boldsymbol{w} = \boldsymbol{0}$, ya que de los s_i adecuados permitirán que se cumplan las restricciones. Para evitar esto, se penaliza en la función a optimizar cada unidad de s_i que se utiliza con un costo c, dando lugar a la formulación [2]:

[2]Se eligió usar el mismo coste c para todas las restricciones; si existe información a priori del dominio del problema se puede elegir un coste distinto para cada ejemplar, por ejemplo, o para cada clase, como se suele hacer cuando las mismas están desbalanceadas

$$\begin{aligned} \underset{\boldsymbol{w},b,\boldsymbol{s}}{\text{Min}} \quad & \mathcal{L}_P(\boldsymbol{w},b,\boldsymbol{s}) = \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + c\sum_i s_i \\ \text{sujeto a} \quad & h(\boldsymbol{x_i})y_i + s_i \geq 1, \qquad i = 1,\ldots,|D| \\ & s_i \geq 0, \qquad i = 1,\ldots,|D| \end{aligned}$$

donde $\boldsymbol{s}$ es el vector de variables s_i.

Ahora se tienen dos conjuntos de restricciones: las originales y las de no-negatividad de las variables de holgura.

Dado un $\boldsymbol{w}$ y un b, es necesario que $h(\boldsymbol{x_i})y_i \geq 1$. Si esto es cierto, se puede tener $s_i = 0$ y no pagar ningún coste, ya que la restricción se cumple sola. Si no es cierto, existen 2 opciones, en términos de la función a optimizar: cambiar $\boldsymbol{w}$, lo cual puede hacer que otras restricciones no se cumplan y además puede incrementar el coste si se achica el margen, o ayudar a $h(\boldsymbol{x_i})y_i$ con s_i para que llegue a 1, y pagar el precio c. El vector $\boldsymbol{x_i}$ quedaría mal clasificado, en este último caso, pero en términos de la función a optimizar eso puede ser mejor que pagar el costo de achicar el margen.

Entonces, el proceso de optimización, además de poder cumplir las restricciones si el problema no es linealmente separable, buscará un balance entre una separación perfecta ($s_i = 0$) y el tamaño del margen ($M \gg 0$), de acuerdo al parámetro de coste c. En el caso en que $c \to \infty$, se penaliza fuertemente la ayuda de s_i, volviendo al modelo original de márgenes *duros*. Si $c \to 0$, se permite que $\boldsymbol{w}$ y b tomen cualquier valor, porque cualquier error de clasificación puede arreglarse con los s_i adecuados ya que son "baratos". En el caso extremo, $c = 0$, una solución óptima podría ser, por ejemplo, $\boldsymbol{w} = 0$, $b = 0$ y $s_i = 1$, lo cual es claramente indeseable.

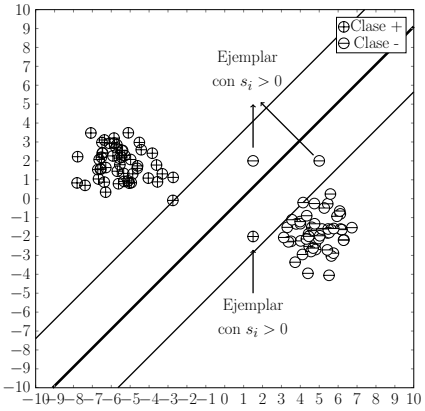
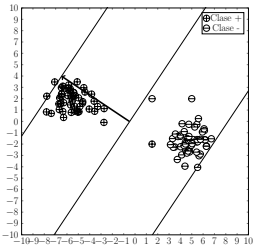
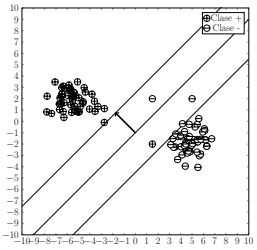


Figura 2.6: Con una elección adecuada de c, el problema de optimización converge al hiperplano natural para separar las dos clases, ignorando los outliers.

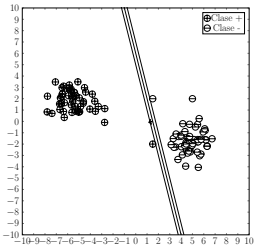
El valor de c controla el nivel de regularización de la solución, ya que ayuda a evitar que $\boldsymbol{w}$ y b se ajusten exactamente a los datos. Valores altos de c inhiben la regularización, valores muy bajos tienen el efecto contrario al deseado, permitiendo cualquier solución. Por ende, es importante determinar c cuidadosamente con el objetivo de lograr una solución balanceada.



(a) Hiperplano determinado con $c = 0.1$.



(b) Hiperplano determinado con $c = 10$.



(c) Hiperplano determinado con $c = 1000$.

Por último, la función a optimizar $\mathcal{L}_P$ es convexa, lo cual puede verse fácilmente ya que es una suma y multiplicación de funciones convexas, y su dominio es un politopo convexo ya que es la intersección de hiperplanos [60]. Esto significa que el hessiano es positivo definido, $H(\mathcal{L}_P) \succeq 0$, y que la función tiene un solo mínimo en todo su dominio. Esta simpleza de la función a optimizar contribuye a la elegancia del modelo, ya que hace más ameno el proceso de optimización.

2.4. Forma Dual

A continuación, se deriva un modelo alternativo del problema de optimización utilizando la forma dual del mismo. Dicha forma permite, en el caso de SVM, realizar la optimización con mayor eficiencia. De esta forma, dado el problema original, se puede plantear el dual, resolverlo, y luego obtener la solución del problema primal en base al dual.

Todo problema de optimización tiene un problema dual asociado. El problema dual es distinto al problema original o primal, pero están relacionados por las **condiciones de optimalidad de Karush-Kuhn-Tucker**, que se llamarán simplemente las **KKT** (ver el apéndice sobre KKT en la sección B.1 al final del capítulo para una breve referencia sobre el tema.).

El problema dual se obtiene a partir del primal introduciendo variables duales α_i por cada restricción del problema primal. Si las restricciones son desigualdades, como en el caso de SVM, las variables duales también estarán restrictas como $\alpha_i \geq 0$. Las restricciones se incorporan o codifican en la función de coste, de manera que cuando más se optimizan los términos relacionados a las restricciones en la función, más se cumplen las restricciones. Luego, se eliminan las variables primales (y con ello las restricciones del primal), obteniendo la formulación dual en términos de solamente las variables duales α_i.

En SVM, como la función a optimizar es convexa y las restricciones son afines (y representan un conjunto convexo), cumple las condiciones de dualidad fuerte, lo cual mediante las KKT implica que el valor del mínimo del primal es igual al máximo del dual. Esta relación es muy importante, ya que implica que de cierta manera, minimizar el primal es equivalente a maximizar el dual. Entonces, si se encuentra una relación entre las variables del primal y las del dual que permitan calcular las primeras a partir de las segundas, se puede plantear y resolver el dual, y luego obtener los valores de las variables del primal.

Para ello, primero se deben llevar las restricciones a la forma $r \geq 0$. Las de $s_i \geq 0$ ya se encuentren con esa forma; las otras se pueden escribir como $r_i \geq 0$ donde $r_i = h(\boldsymbol{x_i})y_i + s_i - 1$.

Luego se introducen variables duales α_i, $i = 1, \ldots, |D|$, una por cada restricción $r_i \geq 0$ y otro conjunto de variables duales u_i, $i = 1, \ldots, |D|$, una por cada restricción $s_i \geq 0$, para plantear el comúnmente llamado **lagrangiano** $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{u}) = \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + c\sum_i s_i - \sum_i \alpha_i r_i - \sum_i s_i u_i$$

donde $\boldsymbol{\alpha}$ el vector de variables α_i y $\boldsymbol{u}$ el vector de variables u_i.
El problema queda entonces como [3]:

$$\begin{aligned} &\underset{\boldsymbol{w},b,\boldsymbol{s}}{\text{Min}} \quad \{\underset{\boldsymbol{\alpha},\boldsymbol{u}}{\text{Max}} \quad \mathcal{L}(\boldsymbol{w},b,\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{u})\} \\ &\text{sujeto a} \qquad\qquad\qquad\qquad r_i \geq 0,\, s_i \geq 0 \\ &\qquad\qquad\qquad\qquad\qquad\quad \alpha_i \geq 0,\, u_i \geq 0 \end{aligned}$$

El lagrangiano captura el efecto de las restricciones pero en forma de coste, obteniendo un problema equivalente al anterior. La intuición principal detrás de esto radica en qué sucede cuando se cumplen las restricciones $r_i \geq 0$ y $s_i \geq 0$. Si $r_i \geq 0$, entonces $-\sum_i \alpha_i r_i$ es negativo, lo cual ayuda a minimizar la función de coste. Por lo tanto, para minimizar $\mathcal{L}$ desde el punto de vista de los nuevos términos en la función de coste, sería ideal que:

$$r_i \gg 0, \quad \alpha_i \gg 0$$

El mismo argumento vale para $s_i \geq 0$, $u_i \geq 0$ y el término $-c\sum_i s_i$. Por otro lado, si $r_i \geq 0$ y $s_i \geq 0$ (lo cual es algo que se necesita en la solución), los términos agregados a la función de coste nunca pueden perjudicarla; a lo sumo, no la ayudarán si $\alpha_i =$ o $u_i = 0$. Entonces, estos dos términos nunca aumentan la función de coste si las variables se mantienen dentro de las restricciones. Si no cumplen las restricciones, el coste de la función aumenta, lo cual tendrá el efecto de llevar el proceso de optimización mediante la función de coste a cumplir las mismas. En este balance hay que tener en cuenta que $\boldsymbol{w}$ y s_i tienen que ser bajos debido a los otros dos términos de la función de coste. En el mejor caso $s_i = 0$, y $h(\boldsymbol{x_i})y_i \geq 1$, con un $\boldsymbol{w}$ bajo.

Se pueden formalizar estos argumentos intuitivos mediante las KKT (ver el apéndice de la sección B.1 para un resumen), que además serán utilizadas para eliminar las variables primales y obtener el problema dual. Las KKT dicen que en el óptimo primal y dual $(\boldsymbol{w}^*, b^*, s_i^*)$ y (α_i^*, u_i^*) se cumplen estas propiedades:

- Estacionalidad

[3] Por simplicidad, se obvia el rango de la variable i ($i = 1, \ldots, |D|$) en las siguientes derivaciones

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \qquad \frac{\partial \mathcal{L}}{\partial b} = 0 \qquad \frac{\partial \mathcal{L}}{\partial s_i} = 0$$

- Feasibilidad Primal

$$r_i \geq 0 \quad s_i \geq 0$$

- Feasibilidad dual

$$\alpha_i \geq 0 \quad u_i \geq 0$$

- Holgura complementaria

$$\alpha_i r_i = 0 \quad u_i s_i = 0$$

Para justificar formalmente la noción intuitiva de que los términos agregados no cambian fundamentalmente el problema, se utiliza ahora también la propiedad de holgura complementaria; si $\alpha_i r_i = 0$ entonces $\sum_i \alpha_i r_i = 0$ en el óptimo, y lo mismo con $u_i s_i = 0$ y $\sum_i s_i u_i = 0$.

Habiendo justificado la equivalencia de los problemas, se utilizará la propiedad de estacionalidad de la solución para encontrar una relación entre las variables duales y primales y eliminar las variables primales del problema, llegando a la formulación dual. Desarrollando las derivadas parciales:

-
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{\partial \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w}}{\partial \boldsymbol{w}} - \frac{\partial \sum_i \alpha_i r_i}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x_i} = 0$$
$$\rightarrow \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i}$$

-
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \sum_i \alpha_i r_i}{\partial b} = \sum_i \alpha_i y_i = 0$$
$$\rightarrow \sum_i \alpha_i y_i = 0$$

- $$\frac{\partial \mathcal{L}}{\partial s_i} = \frac{\partial\, c\sum_i s_i}{\partial s_i} + \frac{\partial \sum_i s_i u_i}{\partial s_i} = c - \alpha_i - u_i = 0$$
$$\rightarrow \alpha_i = c - u_i, \quad i = 1, \ldots, |D|$$

Reemplazando en $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + c\sum_i s_i - \sum_i \alpha_i r_i - \sum_i s_i u_i$$

Como $c = \alpha_i + u_i$

$$c\sum_i s_i = \sum_i c s_i = \sum_i (\alpha_i + u_i) s_i = \sum_i \alpha_i s_i + \sum_i s_i u_i$$

Reemplazando $c\sum_i s_i$ en $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i s_i + \sum_i s_i u_i - \sum_i \alpha_i r_i - \sum_i s_i u_i$$
$$= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i s_i - \sum_i \alpha_i r_i$$

Como:

$$\sum_i \alpha_i r_i = \sum_i ((\boldsymbol{w}\cdot\boldsymbol{x_i} + b)\boldsymbol{y_i} + s_i - 1)\alpha_i$$
$$= \sum_i (\boldsymbol{w}\cdot\boldsymbol{x_i})\boldsymbol{y_i}\alpha_i + \sum_i \boldsymbol{y_i}\alpha_i b + \sum_i s_i\alpha_i - \sum_i \alpha_i$$

Dado que $\sum_i \boldsymbol{y_i}\alpha_i = 0$, entonces $\sum_i \boldsymbol{y_i}\alpha_i b = 0$. Entonces

$$\sum_i \alpha_i r_i = \sum_i (\boldsymbol{w}\cdot\boldsymbol{x_i})\boldsymbol{y_i}\alpha_i + \sum_i s_i\alpha_i - \sum_i \alpha_i$$

Reemplazando $\sum_i \alpha_i r_i$ en $\mathcal{L}$:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i s_i - \sum_i \alpha_i r_i \\
&= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i s_i - (\sum_i (\boldsymbol{w}\cdot\boldsymbol{x_i})\boldsymbol{y_i}\alpha_i + \sum_i s_i\alpha_i - \sum_i \alpha_i) \\
&= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i s_i - \sum_i (\boldsymbol{w}\cdot\boldsymbol{x_i})\boldsymbol{y_i}\alpha_i - \sum_i s_i\alpha_i + \sum_i \alpha_i \\
&= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \sum_i (\boldsymbol{w}\cdot\boldsymbol{x_i})\boldsymbol{y_i}\alpha_i + \sum_i \alpha_i \\
&= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \boldsymbol{w}\cdot\sum_i \alpha_i y_i \boldsymbol{x_i} + \sum_i \alpha_i
\end{aligned}$$

Como $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i} \rightarrow \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i}$:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \boldsymbol{w}\cdot\sum_i \alpha_i y_i \boldsymbol{x_i} + \sum_i \alpha_i \\
&= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i \\
&= -\frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i
\end{aligned}$$

Para finalmente librarse de la variable $\boldsymbol{w}$, se utilizará nuevamente la igualdad anterior, llegando a :

$$\begin{aligned}
\mathcal{L} &= -\frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \sum_i \alpha_i \\
&= -\frac{1}{2}(\sum_i \alpha_i y_i \boldsymbol{x_i})\cdot(\sum_j \alpha_j y_j \mathbf{x_j}) + \sum_i \alpha_i \\
&= \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j (\alpha_i y_i \boldsymbol{x_i}\cdot\alpha_j y_j \mathbf{x_j})
\end{aligned}$$

Se llamará a esta forma de la función a optimizar $\mathcal{L}_D$, por Lagrangiano del Dual. De esta manera, $\mathcal{L}$ no depende de $\boldsymbol{w}$, b o s_i, y por ende las restricciones $r_i \geq 0$ y $s_i \geq 0$ pierden efecto, obteniendo el problema equivalente:

$$\begin{aligned} \underset{\boldsymbol{\alpha},\boldsymbol{u}}{\text{Max}} \quad & \mathcal{L}_D(\boldsymbol{\alpha},\boldsymbol{u}) = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j (\alpha_i y_i \boldsymbol{x_i} \cdot \alpha_j y_j \mathbf{x_j}) \\ \text{sujeto a} \quad & \alpha_i \geq 0,\ \ u_i \geq 0,\ c = \alpha_i + u_i \end{aligned}$$

Las restricciones $\alpha_i \geq 0$, $u_i \geq 0$ y $c = \alpha_i + u_i$ pueden simplificarse a simplemente $0 \leq \alpha_i \leq c$, ya que u_i no aparece en la función de costo ni en otra restricción, obteniendo el problema final

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{Max}} \quad & \mathcal{L}_D(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j (\alpha_i y_i \boldsymbol{x_i} \cdot \alpha_j y_j \mathbf{x_j}) \\ \text{sujeto a} \quad & 0 \leq \alpha_i \leq c \end{aligned} \tag{2.4}$$

De nuevo, se puede resolver el problema dual con algún algoritmo genérico de optimización cuadrática (la función es cuadrática por $\alpha_i\alpha_j$), y averiguar el valor de b y $\boldsymbol{w}$ como:

$$\begin{aligned} \boldsymbol{w} &= \sum_i \alpha_i y_i \boldsymbol{x_i} \\ b &= 1 - (\boldsymbol{w} \cdot \boldsymbol{x}_0)\boldsymbol{y_i} \end{aligned}$$

para un vector de soporte $\boldsymbol{x}_0$ (ya que para los mismos se sabe que $\boldsymbol{w} \cdot \boldsymbol{x}_0 + b = 1$).

Para eso se debe poder determinar en base a los α_i qué ejemplares $\boldsymbol{x_i}$ son vectores de soporte. Además, resulta interesante analizar los valores de α_i ya que darán indicios de si un ejemplar $\boldsymbol{x_i}$ está bien clasificado o no, sin necesidad de calcular $\boldsymbol{w}$, b y luego la función de clasificación, como se discute en la siguiente sección. Esta ventaja es necesaria para la eficiencia de ciertos algoritmos iterativos de optimización como SMO, descripto en el apéndice B.2.

2.5. Análisis de los valores de α_i

Cuando se considera un ejemplar utilizado para encontrar un hiperplano $(\boldsymbol{w}, b)$ en el proceso de optimización y se analiza su posición en el espacio, debe ocurrir alguno de estos tres casos:

1. $h(\boldsymbol{x_i})y_i = 1$: El ejemplar está bien clasificado y es un vector de soporte

2. $h(\boldsymbol{x_i})y_i > 1$: El ejemplar está bien clasificado y no es un vector de soporte

3. $h(\boldsymbol{x_i})y_i < 1$: El ejemplar está mal clasificado

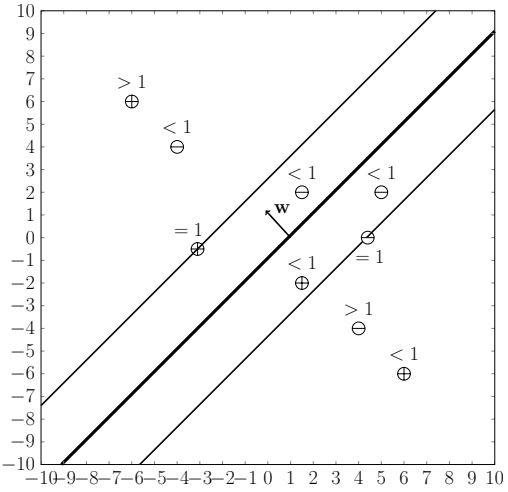


Figura 2.8: Los tres casos posibles en la posición de un ejemplar $\boldsymbol{x}$ luego de encontrar el hiperplano de máximo margen en base al valor de $h(\boldsymbol{x_i})y_i$.

Se pueden determinar parcialmente estos casos en base a los valores de α_i y las relaciones:

$$0 \leq \alpha_i \leq c \qquad \alpha_i + u_i = c \qquad \alpha_i r_i = 0 \qquad u_i s_i = 0$$

Se pueden dividir también los valores de α_i en tres casos

$$\alpha_i = 0 \qquad 0 < \alpha_i < c \qquad \alpha_i = c$$

Y relacionando los tres casos de los valores de las variables α_i con las posiciones de los ejemplares:

$$\alpha_i < c \rightarrow u_i > 0 \rightarrow s_i = 0 \rightarrow \begin{cases} \alpha_i = 0 \rightarrow r_i \geq 0 \rightarrow \text{Caso 1 o 2} \\ 0 < \alpha_i < c \rightarrow r_i = 0 \rightarrow \text{Caso 1} \end{cases}$$
$$\alpha_i = c \rightarrow u_i = 0 \rightarrow s_i \geq 0 \wedge r_i = 0 \rightarrow \text{Caso 1 o 3}$$

Pasando en limpio, se sabe que:

- $\alpha_i = 0 \rightarrow \boldsymbol{x_i}$ está bien clasificado o es un vector de soporte.
- $0 < \alpha_i < c \rightarrow \boldsymbol{x_i}$ es un vector de soporte
- $\alpha_i = c \rightarrow \boldsymbol{x_i}$ está mal clasificado o es un vector de soporte.

Entonces, si $0 < \alpha_j < c$, se puede utilizar el hecho de que $h(\mathbf{x_j})y_j = 1$ para determinar el valor de b.

2.6. El truco del Kernel

En la formulación tradicional de SVM, el modelo resultante se describe en términos de $\boldsymbol{w}$ y b, los coeficientes del hiperplano. Es un modelo lineal, y para problemas de naturaleza no-lineal requeriría una transformación del espacio de entrada que reduzca el problema al caso lineal o casi lineal. SVM ofrece la posibilidad de incorporar dicha transformación de forma implícita, dándole capacidades de clasificación no lineales.

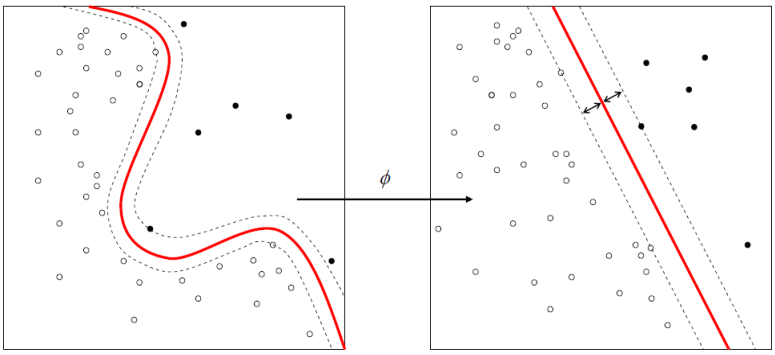


Figura 2.9: A la izquierda, un dominio donde el separador *natural* de las clases es una superficie no lineal. Transformando el dominio mediante una función ϕ se puede llevar el problema a un espacio $\mathcal{P}'$ (derecha) en donde los ejemplares sean linealmente separables, y clasificarlos utilizando el hiperplano determinado por el modelo SVM.

La idea es simple. Generalmente se utiliza una función no-lineal $\phi :: \mathcal{P} \mapsto \mathcal{P}'$ para transformar ejemplares $\boldsymbol{x}$ del dominio $\mathcal{P}$ a ejemplares $\boldsymbol{x}'$ de $\mathcal{P}'$ en donde son linealmente separables o más separables, y $\mathcal{P}'$ es un espacio con un producto interno $\langle \cdot, \cdot \rangle$ definido. Si se plantea y deriva la formulación de SVM con $\phi(\boldsymbol{x_i})$ en lugar de $\boldsymbol{x_i}$, se obtiene, con las condiciones KKT, que:

$$\boldsymbol{w} = \sum \phi(\boldsymbol{x_i})\alpha_i y_i$$

Y para clasificar un nuevo ejemplar, se calcula:

$$\begin{aligned} h'(\boldsymbol{x}) &= \boldsymbol{w} \cdot \phi(\boldsymbol{x_i}) + b \\ &= (\sum \phi(\boldsymbol{x_i})\alpha_i y_i) \cdot \phi(\boldsymbol{x}) + b \\ &= (\sum (\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x})\alpha_i y_i) + b \end{aligned}$$

Esto no difiere de la metodología genérica que funciona con cualquier método de clasificación. Como con las transformaciones no lineales antes de aplicar el perceptrón del capítulo 1, la característica ϕ busca complejizar las regiones de voronoi simples que genera el modelo lineal de un hiperplano separador.

El problema es que, en ocasiones, la transformación ϕ puede ser difícil de calcular explícitamente de forma eficiente, o hasta imposible si el $\mathcal{P}'$ es un espacio de dimensionalidad infinita.

El truco del kernel consiste en evitar calcular explícitamente dicha transformación. Para ello, se expresa el modelo de clasificación en base a un operador de similitud $K(\boldsymbol{x}_1, \boldsymbol{x}_2)$ entre ejemplares $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$ de $\mathcal{P}$. El operador de similitud se suele llamar **kernel**, y en un mismo paso realiza la transformación ϕ de los ejemplares y aplica el producto interno del espacio $\mathcal{P}'$ al par de ejemplares transformados. Entonces $K :: (\mathcal{P}, \mathcal{P}) \mapsto \mathcal{F}$, donde $\mathcal{F}$ es un campo, generalmente $\mathbb{R}$. Dada una transformación ϕ y un producto interno $\langle \cdot, \cdot \rangle$:

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2) \rangle$$

Como ejemplo, tomando un kernel polinomial $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 + b)^d$ con $d = 2$ y bias $b = 0$, y $\mathcal{P} = \mathbb{R}^2$:

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^2$$

Este kernel corresponde a la transformación y producto interno:

$$\begin{aligned} \phi(\boldsymbol{x}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \\ \langle \boldsymbol{x}, \boldsymbol{x}' \rangle &= \boldsymbol{x} \cdot \boldsymbol{x}' \end{aligned}$$

Comprobando que son iguales:

$$\begin{aligned}
\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle &= \langle \phi((x_1, x_2)), \phi((x_1', x_2')) \rangle \\
&= \left\langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \right\rangle \\
&= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \\
&= x_1^2x_1'^2 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + x_2^2x_2'^2 \\
&= x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 \\
&= (x_1x_1')^2 + 2(x_1x_1')(x_2x_2') + (x_2x_2')^2 \\
&= ((x_1, x_2) \cdot (x_1', x_2'))^2 \\
&= (\boldsymbol{x} \cdot \boldsymbol{x}')^2
\end{aligned}$$

La imagen de ϕ es $\mathbb{R}^3$, un espacio de distinta dimensión que el de origen y la transformación es no lineal, pero aplicando luego el producto interno se convierte en un simple producto punto entre los vectores elevado al cuadrado.

En la etapa de optimización, se utilizan repetidamente los valores $K(\boldsymbol{x_i}, \mathbf{x_j})$ para ejemplares de entrenamiento $\boldsymbol{x_i}$ y $\boldsymbol{x_i}$, por lo cual es usual construir una matriz simétrica $\boldsymbol{\kappa}$ donde se precalcula el kernel entre todos los pares de ejemplares tal que $\kappa_{\boldsymbol{x_i}, \mathbf{x_j}} = \kappa_{i,j} = K(\boldsymbol{x_i}, \mathbf{x_j})$. Entonces, la función a optimizar en la formulación dual queda como:

$$\mathcal{L}_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j (\alpha_i y_i \alpha_j y_j \kappa_{i,j})$$

Si se resuelve el problema dual con un K no lineal que aproveche el truco del kernel, es generalmente preferible no calcular $\boldsymbol{w}$ explícitamente, ya que de esa manera al clasificar un nuevo ejemplar con la función $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b$, se necesita calcular también a $\phi(\boldsymbol{x})$ explícitamente (lo cual puede ser imposible, como con espacios de características de dimensión infinita) . En ese caso, es mejor guardar los valores de los vectores de soporte, así como de las variables duales y el bias b, y calcular f como $f(\boldsymbol{x}) = \sum_i \alpha_i \boldsymbol{y_i} \kappa_{\boldsymbol{x_i}, \boldsymbol{x}} + b$.

2.6.0.1. Condición de Mercer

Ciertos kernels son, hasta cierto grado, de propósito general, como el **kernel gaussiano** $e^{-\frac{||x_1 - x_2||}{2\sigma^2}}$, el **kernel polinomial** $(\boldsymbol{x}_1 \cdot \boldsymbol{x}_2^t + b)^d$, y con el

kernel lineal $x_1 \cdot x_2$ se recupera la formulación original de SVM. Distintos dominios pueden requerir distintos kernels que representen medidas de similitud apropiadas. Si los ejemplares son strings, por ejemplo, se puede utilizar un kernel basado en la **distancia de Damerau-Levenshtein** [61], o uno basado en **Dynamic Time Warping** (DTW) [62] para secuencias temporales.

En general, el diseño de un kernel no parte de definir una transformación a otro dominio y un producto interno en ese dominio, sino de una medida de similitud K como la distancia de Damerau-Levenshtein. En tales casos, es necesario saber si dicha función K corresponde a una transformación a y un producto interno en algún dominio $\mathcal{P}'$.

En términos formales, se necesita saber si $\exists(\phi, \langle\cdot,\cdot\rangle)$ tal que $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2)\rangle$. Una condición necesaria y suficiente es la **condición de Mercer**:

$$\forall g \in \mathbf{L}^2 : \int_y \int_x K(x,y)g(x)g(y)dxdy$$

Donde $\mathbf{L}^2$ es el espacio de las funciones g cuadráticamente integrables, o sea, $\int_{-\infty}^{\infty} |g(x)|^2 dx < \infty$.

Los kernels K para los cuales esto se cumple se llaman **kernels de Mercer**. Dichos kernels aseguran que pueden interpretarse en términos de una transformación y un producto interno. En cierto sentido estos son los *verdaderos* kernels, y aquellas funciones K que no cumplen la condición de mercer serían simplemente medidas de similitud, pero en la práctica se suele hablar de kernels para toda función que representa una medida de similitud y kernels de Mercer para aquellos que cumplen la condición. Los kernels gaussianos, lineales y polinomiales son kernels de Mercer.

En la práctica, la condición de Mercer puede ser difícil de comprobar, ya que involucra toda $g \in \mathbf{L}^2$, y por ende se han desarrollado condiciones más simples sobre la matriz $\boldsymbol{\kappa}$ generada a partir de un conjunto de ejemplares para el uso particular de SVM y otros métodos. Una de ellas es que la matriz $\boldsymbol{\kappa}$ sea simétrica y positiva semi-definida, o sea $\kappa(\boldsymbol{x_i}, \mathbf{x_j}) = \kappa_{i,j} = \kappa_{j,i} \geq 0$ para ejemplares $\boldsymbol{x_i}, \mathbf{x_j}$ de cualquier conjunto de datos D, lo cual es mucho más fácil de verificar.

Es de notar que ciertas funciones de similitud K, aún sin cumplir la condición de Mercer, pueden ser utilizadas si la matriz $\boldsymbol{\kappa}$ para un conjunto asociada cumple las condiciones de Mercer simplificadas, y K cumple las propiedades de un producto interno. Sin esta condición, el hessiano de K puede no ser positivo semi-definido y entonces el problema

de optimización pierde su convexidad y puede tener más de una solución óptima.

2.7. Adaptación para el reconocimiento multiclase

El modelo de SVM que se ha desarrollado solo considera dos clases. Para resolver problemas multiclase, se debe buscar alguna forma de extender el algoritmo. Hay dos enfoques principales; el primero, replantear el formalismo detrás de SVM para funciones con varios resultados (o sea, $f :: \mathcal{P} \mapsto \mathbb{R}^C$). El segundo, combinar varios SVM de dos clases para formar un SVM multiclase.

Este último camino a su vez tiene dos estrategias posibles:

- La primera es entrenar $C(C-1)/2$ clasificadores SVM, donde cada clasificador distingue entre dos clases. Este enfoque tiene la ventaja de que puede distinguir entre clases individualmente y cada clasificador se entrenaría rápidamente ya que habría una cantidad mucho menor de ejemplos, pero el número de SVMs a entrenar crece cuadráticamente con la cantidad de clases y también se incrementa la necesidad computacional del algoritmo en total.

- La segunda es entrenar una SVM por clase, que distinga entre ejemplares de esa clase y el resto, teniendo de esta manera hay C clasificadores. Esta última técnica tiene ventajas y desventajas opuestas a la primera opción. Cada clasificador se asemejaría a la función f_c descripta en la sección de aprendizaje automático, y el criterio para elegir la clase correspondiente en base a las salidas de cada clasificador f_c podría ser $\text{argmax}_c\, f_c(\boldsymbol{x})$.

2.8. Resumen

En este capítulo se presentan las máquinas de vectores de soporte (SVM). Las mismas representan un modelo de clasificación, de aprendizaje de tipo supervisado, desarrollado por Vapnik y otros en los años 90, con fuertes bases en la teoría estadística del aprendizaje.

La idea esencial de las mismas es extender el algoritmo del perceptrón de manera que se selecciona el hiperplano de máximo margen entre las clases, es decir, el hiperplano que maximiza la distancia entre los ejemplares

más cercanos de cada clase al hiperplano. De esta forma, el entrenamiento del SVM consiste en resolver un problema de optimización matemática, en donde hay que maximizar dicho margen.

Además se presentan algunas extensiones para lidiar con conjuntos de datos que no se pueden separar exactamente con un hiperplano, manejar ejemplares anómalos, e incluir transformaciones no-lineales de los ejemplares de modo de ampliar el poder de clasificación de las SVM. Las pruebas del capítulo 7 hacen referencia a estas extensiones, debido a que se prueban distintas configuraciones del modelo variando los parámetros de las extensiones.

Este capítulo hace una introducción al modelo de clasificación. Como se mencionó, para entrenar dicho modelo se debe resolver un problema de optimización; para el lector interesado se presenta en el apéndice B.2 un algoritmo para tal fin. Además, en la derivación y justificación del modelo se utilizan ciertos conceptos matemáticos, de los cuales el apéndice B.1 hace una breve referencia.

Por último, el modelo SVM de clasificación ha sido ampliamente utilizado en los últimos tiempos, y por ende es un candidato ideal para comparar contra el clasificador propuesto en esta tesina, el CNC.

3

Redes neuronales

"No, I'm not interested in developing a powerful brain [..]. All I'm after is just a mediocre brain."

Alan M. Turing

3.1. Introducción

Las redes neuronales artificiales son una familia de modelos computacionales inspirados por las redes neuronales biológicas.

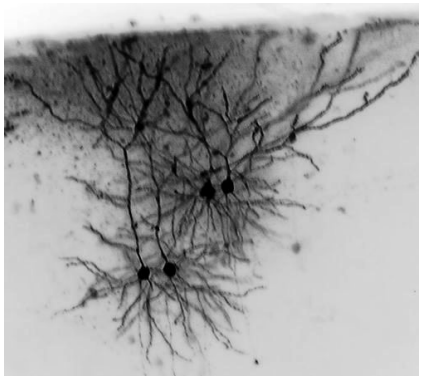


Figura 3.1: Red neuronal biológica, obtenida del área del *giro cingulado* del cerebro de un ratón.

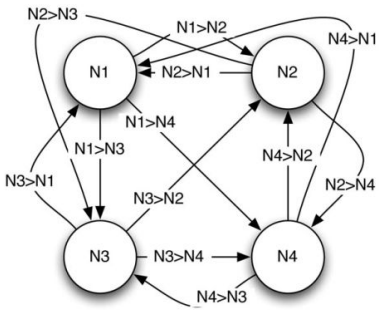


Figura 3.2: Red neuronal artificial. Los círculos representan neuronas y las flechas conexiones entre las mismas.

Desde el punto de vista fisiológico, las neuronas biológicas son células que pueden estimularse eléctricamente. Las neuronas se pueden conectar

unas a otras para formar **redes neuronales**. Los estímulos que genera una neurona son función de los estímulos actualmente recibidos, su estado interno y el tipo y contexto (presión, alargamiento, etc) de la neurona. Los estímulos son mayormente excitativos o inhibitorios.

Las neuronas son el componente principal del sistema nervioso, que incluye el cerebro, la médula espinal y los ganglios nerviosos periféricos. Se pueden distinguir entonces tres tipos de neuronas principales, en base a la función general que cumplen:

- **Sensoriales**, que responden al tacto, sonido luz y otros estímulos de los órganos sensoriales y los transmiten al cerebro y la médula espinal.
- **Motrices**, que reciben señales del cerebro o la médula espinal y estimulan partes del cuerpo (músculos y glándulas).
- **Interneuronas** que conectan unas neuronas con otras en la misma región del cerebro o la médula espinal.

Desde el punto de vista histológico, las neuronas están compuestas por el cuerpo de la célula, llamado **soma**, conectores de señales entrantes, llamadas **dendritas**, y un conector de salida llamado **axón**. Las dendritas son fibras finas y cortas que actuan como receptores de impulsos y hay una gran cantidad en cada soma. Hay un solo axón por soma, pero luego de salir de la neurona puede bifurcarse cientos de veces para conectarse con varias dendritas de otras neuronas, y puede tener hasta 1 metro de longitud en los humanos.

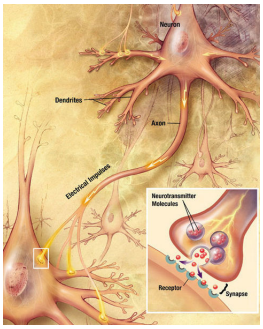


Figura 3.3: Diagrama de las partes que componen una neurona y su interconexión

La estimulación entre neuronas ocurre mediante señales electroquímicas, que se propagan a través de las **sinapsis**, que son conexiones especializadas entre neuronas donde se encuentran ramas del axón de una célula

con ramas de las dendritas de otra. La sinapsis no es solamente un simple conector, ya que también modula las señales entre neuronas.

Por supuesto, hay excepciones a estas reglas: neuronas que no tienen dendritas o axón, sinapsis que conectan un axón a otro axón o una dendrita a otra dendrita, etc, pero en general se siguen los patrones recién descriptos.

Si bien la histología de las neuronas se estudia hace tiempo y se conoce con bastante detalle, desde el punto de vista fisiológico todavía queda mucho para descubrir sobre el sistema nervioso. Hay diversos modelos sobre cómo las neuronas cómo grupo o individuos representan y procesan información, cómo influye la arquitectura de las redes en dicha función, cómo se crean y se adaptan a las necesidades del organismo o cómo influye la codificación genética en el desarrollo del cerebro; ninguno de ellos es final, en el sentido de que no existe todavía una teoría completa sobre el funcionamiento del cerebro y el sistema nervioso.

Cualquier modelo neuronal que quiera acercar su comportamiento al de los sistemas biológicos debe contar con una serie de características esenciales que son las que proporcionan a los sistemas biológicos gran parte de sus capacidades.

Una de las principales características de las Redes Neuronales biológicas es su dimensión. En el cerebro existen entre 10^{11} y 10^{14} neuronas, y cada una de ellas posee entre 1000 y 10000 conexiones con otras tantas células. Esto deja a los sistemas artificiales varios órdenes de magnitud por debajo en cuanto a conectividad.

El funcionamiento general de los sistemas nerviosos es altamente paralelo. Es este grado de paralelismo el que permite que, con una velocidad de señal relativamente baja (1 mseg) el cerebro pueda reaccionar con rapidez ante la continua necesidad de tomar decisiones, en las que existen un gran número de variables. Dichas decisiones resultan extraordinariamente complejas de plantear, y en muchas casos irresolubles en tiempo real, incluso para los mejores computadores actuales, a pesar de que estos poseen velocidades de alrededor de 6 órdenes de magnitud superiores a las redes neuronales biológicas.

En los sistemas nerviosos se realizan procesos de convergencia y divergencia de la información. Mediante la existencia de múltiples conexiones de entrada y salida para cada neurona se produce una multiplicación del uso de una determinada información (divergencia) así como una combinación de diversas señales de entrada para producir un efecto conjunto en una sola neurona (convergencia).

Las redes neuronales biológicas poseen una plasticidad intrínseca, mediante la cual están dotadas de la capacidad de aprender a partir de la

experiencia y adaptarse a nuevos entornos con extrema facilidad.

También poseen una cierta predeterminación genética, dada por la existencia de un mapa de conexiones prefijado. Esto significa, básicamente, que el diseño fundamental de la arquitectura de las distintas Redes neuronales está predeterminado genéticamente y ha sido seleccionado de entre los modelos posibles por su valor adaptativo para cada especie. Así pues, la arquitectura general de las conexiones está diseñada de antemano, si bien la potencia relativa de dichas conexiones se obtendrá a lo largo del aprendizaje. Esto no entra en absoluto en contradicción con la idea anterior de plasticidad, como podría en un principio parecer.

En general, los modelos de neuronas artificiales toman en cuenta la idea de neuronas sensoriales, interneuronas y motrices que se llamarán de **entrada**, **ocultas** y de **salida**, respectivamente. Esta correspondencia no es perfecta, ya que algunas neuronas realizan varias funciones, pero es útil en general. También utilizan el concepto de las neuronas y las sinapsis como moduladoras de las señales transmitidas. La mayoría abandona la representación temporal continua por una discreta donde las simulaciones se realizan paso a paso. En la simulación las neuronas actúan de forma sincrónica, o sea, en cada paso de simulación se evalúa la entrada y la salida de cada neurona. Algunas toman en cuenta el tiempo, aunque casi siempre de forma discreta. Además, son de naturaleza digital en lugar de analógica, debido a que se suelen simular con computadoras.

La gran mayoría de los modelos de neuronas biológicas definen su funcionamiento en términos de ecuaciones diferenciales que describen el cambio de potencial de la misma en función del tiempo y el efecto de otras neuronas, en general basados en el modelo de Huxley-Hodgkin [63, 64]. Las neuronas que modelan son analógicas, de naturaleza continua y generalmente asincrónica, y debido a que consideran el paso del tiempo utilizan el concepto de pulsos de voltaje generados por la neurona como señal. Dichos modelos luego se extienden para redes enteras, aunque son computacionalmente demandantes, y requieren la estimación de muchos parámetros para generarse [65]. Además, como se mencionó anteriormente, representan aproximaciones al funcionamiento neuronal que todavía tienen mucho por recorrer para asemejarse al comportamiento real de una red neuronal biológica.

Entonces, si bien se han entrenado redes neuronales artificiales basadas estrictamente en modelos biológicos, en la práctica los modelos artificiales realizan ciertas simplificaciones que los hacen más amenos a la simulación computacional y más efectivos para la resolución de tareas.

A continuación, se describe un modelo general de red neuronal sincrónica, de tiempo discreto, sin recurrencias ni estado permanente. Luego,

se profundiza sobre el modelo de redes neuronales **feedforward** y dos algoritmos para su entrenamiento, **backpropagation** (propagación para atrás) y **resilient backpropagation** (propagación para atrás resiliente), así como el modelo **competitivo** de redes neuronales y el algoritmo de entrenamiento de **vector quantization** (cuantización vectorial).

3.2. Modelos de redes neuronales artificiales

Una red neuronal artificial (ANN, por artificial neural network) puede representarse como un digrafo $G = (V, E)$, donde los vértices v_i representan neuronas y las aristas e_i representan conexiones entre ellas, o sea, una tripla axón, sinapsis y dendrita. Las conexiones son, en principio, arbitrarias, de acuerdo a la función que la red realiza.

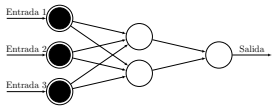
Como ya fue mencionado, suelen considerarse tres conjuntos de neuronas en la red:

- **De entrada:** Reciben los estímulos iniciales y los propagan a la red. Corresponden a las neuronas sensoriales.
- **Ocultas:** Modelan la dinámica interna (no observable) de la red. Corresponden a las interneuronas.
- **De salida:** Modelan el resultado observable de la computación de la red. Corresponden a las neuronas motrices.

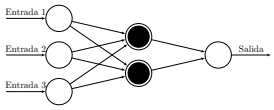
En la práctica, en las ANN se suele dar que dichos conjuntos son disjuntos, y que además las neuronas de entradas son aquellas cuyos vértices tienen grado de entrada 0 (no son estimuladas por otras neuronas) y las de salida aquellas con grado de salida 0 (no estimulan otras neuronas).

El funcionamiento general de la red es el siguiente: las neuronas de entrada son estimuladas (se asume que todas al mismo tiempo), lo cual causa que estas estimulen a sus adyacentes, y así sucesivamente, en algún orden determinado. Se tratarán solo grafos acíclicos, en donde el orden de estimulación es el topológico [1].

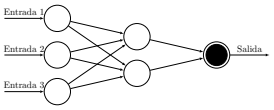
[1] Si el grafo tiene ciclos, entonces debe definirse algún orden de evaluación que rompa la dependencia circular entre los estímulos de las neuronas del ciclo, lo cual en general involucra introducir alguna noción de tiempo en la red para determinar que neurona fue estimulada antes o después.



(a) Se estimulan las neuronas de entrada



(b) Las neuronas de entrada estimulan a las ocultas, y se apagan al mismo tiempo



(c) las neuronas ocultas estimulan a las de salida, y se apagan al mismo tiempo.

Cada neurona, al recibir los estímulos de otras neuronas, calcula alguna función en base a esos estímulos, y genera otro estímulo en respuesta que transmite a sus adyacentes. Los estímulos suelen representarse con números reales, y cada neurona v_i calcula una función $o_i :: \mathbb{R}^{n_i} \mapsto \mathbb{R}$, donde n_i es el grado de entrada de la neurona v_i [2].

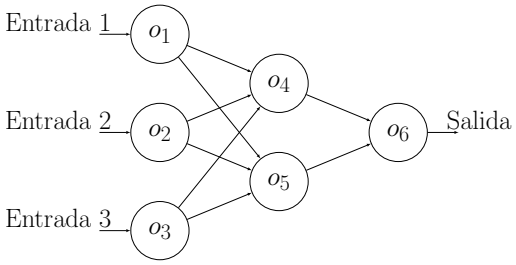


Figura 3.5: Red neuronal con funciones arbitrarias en cada neurona.

En este sentido, la red representa una combinación de funciones o_i en una función arbitraria o, donde si existen d neuronas de entrada y k de salida, $o :: \mathbb{R}^d \mapsto \mathbb{R}^k$.

Dado un problema y un conjunto de datos de entrenamiento para ese problema, un algoritmo de entrenamiento de una red neuronal debe buscar la topología de la red, las funciones de cada neurona y los parámetros de dichas funciones que resuelvan esa instancia del problema con el menor error posible.

En la práctica, muchos algoritmos de entrenamiento requieren que la topología y las funciones de cada neurona sean especificadas a priori y quedan fijas; se busca minimizar el error con respecto a los parámetros $\boldsymbol{W}$ de las funciones solamente. La elección de la topología y las funciones se realizan con información del dominio del problema y el conjunto de datos de entrenamiento disponible. Este es el caso de los algoritmos y modelos

[2]Generalizando, los estímulos podrían ser vectores de reales, enteros, cadenas de caracteres, imágenes, etc, dependiendo de la función de la red.

que serán utilizados en esta tesina.

Data:
Un conjunto de ejemplares $D \in \mathbb{R}^d$, posiblemente con una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Una topología de la red $G = (V, E)$, y una función o_i asociada a cada v_i.
Una tolerancia al error, ϵ
Un valor inicial para el conjunto de parámetros de las funciones, $\boldsymbol{W}$
Result: Un valor final del conjunto de parámetros de las funciones, $\boldsymbol{W}$
while $error(D, \boldsymbol{W}) > \epsilon$ **do**
  Modificar $\boldsymbol{W}$ para que la red tenga menor error con los ejemplares de D ;
end
Retornar $\boldsymbol{W}$;

Algoritmo 2: Esquema general de un algoritmo de entrenamiento para una red neuronal con topología y funciones fijas.

Como se puede observar, la idea es muy similar a la del entrenamiento del perceptrón o cualquier otro algoritmo de aprendizaje automático.

3.2.1. El Perceptrón: una red neuronal simple

Se puede revisitar el Perceptrón como una red neuronal simple para resolver problemas de clasificación de 2 clases; de hecho, así es como surgió el modelo en los años 60 [56].

Si $\boldsymbol{x} \in \mathbb{R}^d$, entonces se necesitan d neuronas de entrada, todas conectadas a una única neurona de salida que calcula la clase estimada del ejemplar. Esta neurona implementa la función de clasificación f del perceptrón vista en el capítulo 1.

Se denotará como $o(\boldsymbol{x})$ al resultado de la capa de salida de la red dado el ejemplar $\boldsymbol{x}$ como entrada, o simplemente $\boldsymbol{o}$ si se entiende del contexto a que $\boldsymbol{x}$ se está haciendo referencia. Cada neurona de entrada representa un componente del ejemplar $\boldsymbol{x}$, simulando la estimulación de un órgano sensorial con los valores del ejemplar.

La neurona de salida utiliza la función:

$$o(\boldsymbol{x}) = \theta_t(h(\boldsymbol{x})) \quad \text{donde:}$$
$$h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$
$$\theta_t(x) = \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{de lo contrario} \end{cases}$$

definida en el capítulo 1, representando un hiperplano separador como antes [3].

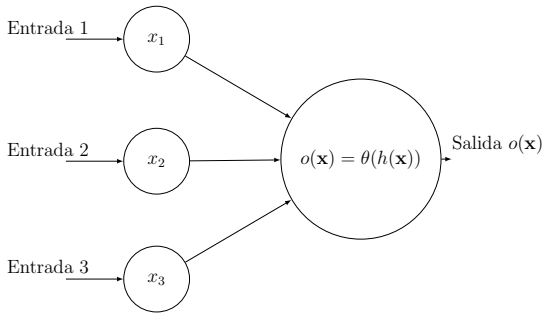


Figura 3.6: d Neuronas de entrada, una de salida, $o(\boldsymbol{x})$

Dado un ejemplar $\boldsymbol{x}$, se estimulan las neuronas de entrada con los d elementos del vector $\boldsymbol{x}$. Luego, cada neurona de entrada estimula la neurona de salida con el mismo valor con que fue estimulada [4] y la de salida entonces calcula la función $o(\boldsymbol{x})$ que representa su estímulo de salida. Como esta neurona no transmite ese estímulo a otra neurona, se asume que es la salida observable de la red.

Para entrenar esta versión del Perceptrón se puede utilizar el mismo algoritmo que el descripto en la sección 1.5.1, donde el vector normal $\boldsymbol{w}$ corresponde al parámetro de la función de la neurona de salida de la red. El conjunto de parámetros $\boldsymbol{W}$ entonces está compuesto simplemente por el vector $\boldsymbol{w}$.

El Perceptrón es una red neuronal que tiene una capacidad de separación limitada a un hiperplano. A continuación, se presenta la arquitectura feedforward de red neuronal, una extensión natural del Perceptrón. Se

[3] Recordando el truco de embeber un ejemplar $\boldsymbol{x} \in \mathbb{R}^{d-1}$ en un espacio $\mathbb{R}^d$, donde ahora $x_d = 1$, se puede tomar en cuenta el bias.

[4] Se asume que cada neurona de entrada tiene como función la identidad $id(x) = x$, de modo que no se modifican los elementos del ejemplar, una convención habitual. Algunos modelos no incluyen estas neuronas de entrada en el modelo, y asumen que la neurona de salida es estimulada directamente por algún proceso externo.

desarrollará el algoritmo tradicional de entrenamiento para esas redes, backpropagation, y una de sus variantes, de mayor efectividad, resilient backpropagation. Luego, se tratan las redes competitivas, también con su algoritmo de entrenamiento clásico. Ambos tipos de redes y algoritmos se utilizan en esta tesina para reconocer gestos.

3.3. Redes Feedforward

El modelo feedforward (FF) extiende el perceptrón introduciendo el concepto de capas de neuronas (de entrada, ocultas y de salida), funciones de cada neurona no-lineales, y la utilización de varias neuronas de salida para codificar el resultado de la red. Es uno de los modelos más populares de red neuronal, uno de los que popularizó la técnica.

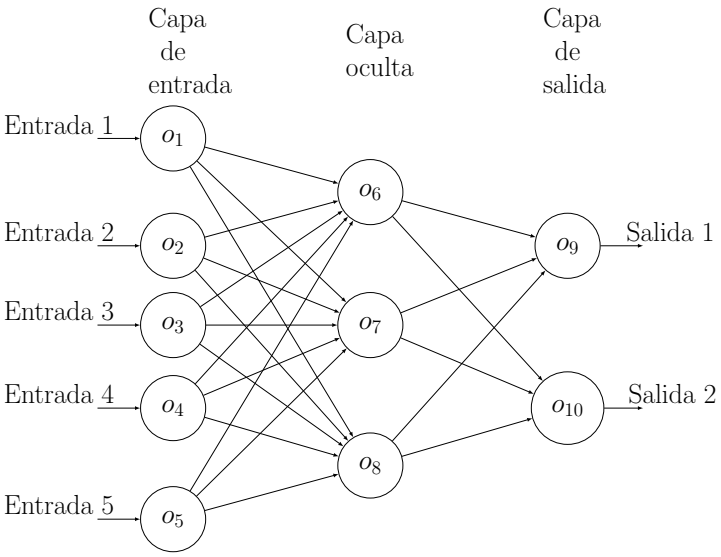
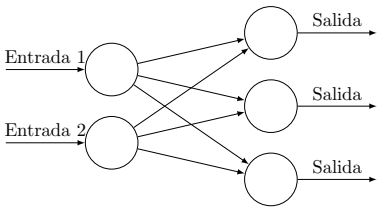


Figura 3.7: Una red feedforward con tres capas: la primera, de entrada, luego un capa oculta, y finalmente una capa de salida, con varias neuronas de salida. Las funciones de las neuronas pueden ser no-lineales.

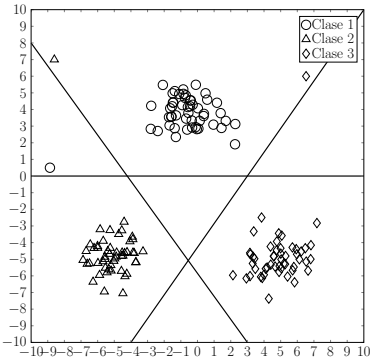
Se mostrará el modelo feedforward en tres pasos, introduciendo tres problemas que motivan cambios en la formulación del perceptrón para llegar al modelo deseado.

3.3.1. Varias clases

Supónganse que se busca clasificar C clases de ejemplares. Se puede representar la existencia de varias clases utilizando C neuronas de salida como las del perceptrón, una por cada clase. Habrá entonces C hiperplanos de separación que entrenar, cada uno con sus parámetros.



(a) Una red neuronal con 2 neuronas de entrada y 3 de salida.



(b) Ejemplares de tres clases distintas. Cada grupo de ejemplares está separado del resto por un hiperplano.

Se interpreta la salida de esta red como un vector $\boldsymbol{o}$, donde o_c es el resultado calculado para la neurona de salida c. En este caso, un vector de la clase c está bien clasificado si $o_c = 1$ y $o_j = -1, j \neq c$. Se puede entrenar esta red aplicando el algoritmo del perceptrón a cada neurona c por separado.

Se puede ver la topología de la red como dos capas de neuronas, las de entrada y las de salida. A continuación, se generaliza la noción de capas en las redes considerando el problema de la clasificación de los ejemplares de una clase c contra los del resto de las clases.

3.3.2. Capas ocultas

Supóngase que se busca clasificar ejemplares de las 3 clases:

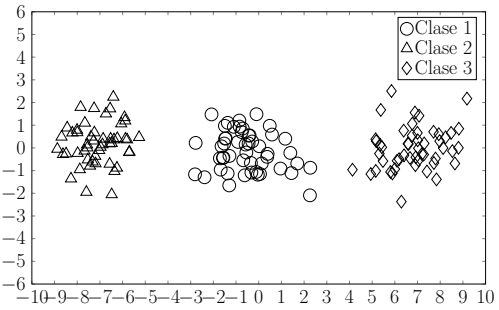


Figura 3.9: Ejemplares de tres clases distintas. La clase 2 no puede separarse del resto con un solo hiperplano.

Se ve claramente que no existe un hiperplano que pueda separar la clase 2 (triángulo) del resto de las clases.

El perceptrón multiclase no puede solucionar este problema, pero quizás realizando una transformación del espacio de ejemplares se pueda volver a un problema tratable. Es fácil observar que se puede separar a la clase 2 del resto utilizando *dos* hiperplanos, entre la clase 1 y la 2, y la 2 y la 3.

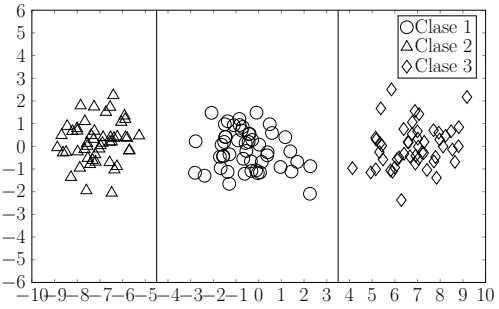


Figura 3.10: Ejemplares de tres clases distintas; los hiperplanos separan a la clase 2 del resto.

El resultado es entonces un dominio $\mathcal{P}$ particionado por regiones de voronoi ligeramente más complejas que con el simple perceptrón, y por ende más potentes para su clasificación.

Se puede implementar este concepto introduciendo una nueva capa de neuronas *ocultas* entre la capa de entrada y la de salida. El propósito de esta capa es transformar los ejemplares a una representación más simple para clasificar, manteniendo el mismo tipo y cantidad de neuronas en la capa de salida que antes.

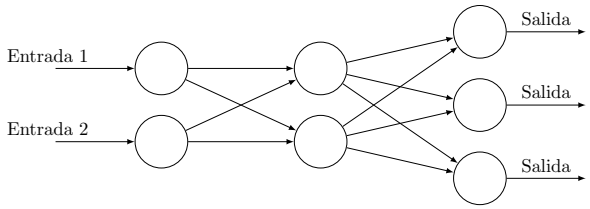


Figura 3.11: Un perceptrón multiclase con una capa oculta de dos neuronas.

En el ejemplo, se puede utilizar una capa oculta con dos neuronas que representen los hiperplanos de la figura Y.

De esta forma, asumiendo que los vectores normales a los hiperplanos están orientados hacia la derecha, los ejemplares de la clase 1 se convierten en un vector (x, y), donde $(x, y) = (-1, -1)$; para los de la clase 2, $(x, y) = (1, -1)$; y para los de la clase 3, $(x, y) = (1, 1)$.

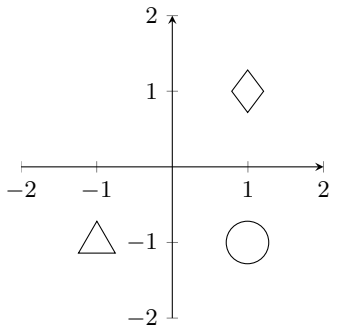


Figura 3.12: Transformación de las neuronas de entrada a las neuronas ocultas.

De esta forma, la transformación pone a los ejemplares de cada clase de forma tal que puedan ser clasificados con una capa de salida como la de antes.

En la práctica, por simplicidad se suele asignar a todas las neuronas de una misma capa la misma función, aunque con distintos parámetros. Se pueden utilizar varias capas ocultas, una a continuación de la otra, para lograr una transformación más compleja.

El entrenamiento de una red con estas características requiere minimizar respecto a los parámetros de las funciones de las neuronas ocultas y las neuronas de salida al mismo tiempo, con lo cual el algoritmo clásico de entrenamiento del perceptrón no puede aplicarse. Para entrenar esta red, se puede utilizar el algoritmo backpropagation, a describirse luego.

Antes, se introducirá la necesidad de emplear funciones no lineales en las neuronas para resolver problemas más complejos.

3.3.3. Funciones no lineales

Hasta ahora se han utilizado funciones lineales para definir superficies de separación que permiten clasificar los ejemplares.

A las funciones o_i de las neuronas se las suele definir como la composición de una **función de combinación** $h_i :: \mathbb{R}^{n_i} \mapsto \mathbb{R}$ y una **función de activación o transferencia** $\theta_i :: \mathbb{R} \mapsto \mathbb{R}$. La primera combina de alguna manera los estímulos entrantes a un solo número; la segunda aplica una transformación a ese número.

En el ejemplo del perceptrón, la función de combinación es h y la de activación θ. La función de combinación separa el espacio en dos hiperplanos; la de activación asigna la etiqueta 1 a los ejemplares que quedan a un lado del hiperplano y -1 a los otros.

Supóngase un problema de dos clases, en donde las mismas no pueden separarse fácilmente por un clasificador lineal, ni siquiera uno de dos capas.

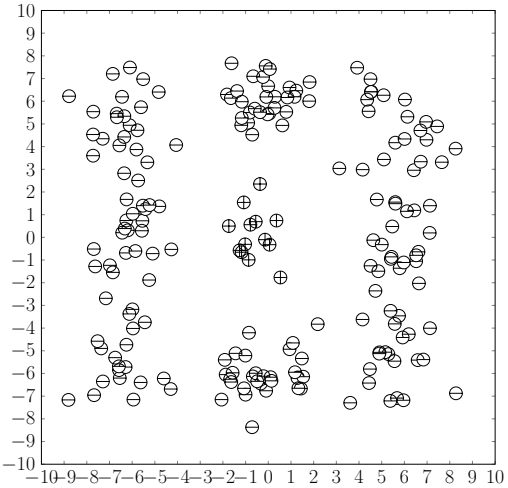


Figura 3.13: Dos clases que no son linealmente separables

Se debe buscar una superficie de separación de las clases que sea no lineal, y eso implica introducir funciones no lineales en las neuronas.

Dado que la clase 1 es un círculo limitado por la clase 2, se puede

definir una función:

$$o(\boldsymbol{x}) = \theta_t(c(\boldsymbol{x}))$$
$$c(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{c}\| - r$$

que representa la pertenencia de un ejemplar a un círculo con centro $\boldsymbol{c}$ y radio r. Optimizando el error respecto a estos parámetros, de manera que el círculo esté centrado en el centro de masa de la clase 1 y tenga un radio igual a la dispersión de los ejemplares de esa clase, se obtendrá un clasificador cuya salida es 1 para los ejemplares dentro del círculo y 0 para los otros.

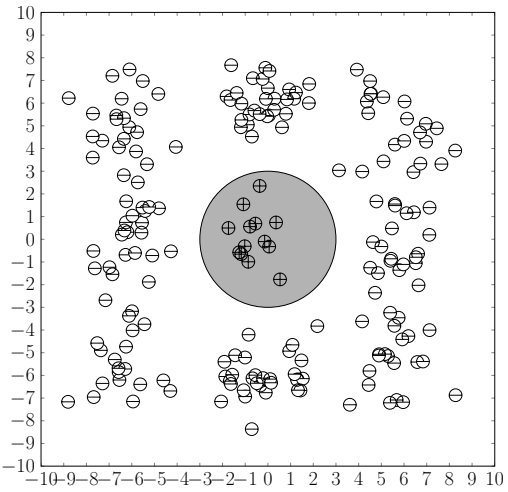


Figura 3.14: Un separador basado en hiperesferas puede separar las dos clases.

De esta manera, cambiando la función de combinación a una que mide la distancia de los ejemplares al borde del círculo, se obtuvo una transformación a un espacio donde los ejemplares son separables.

Es importante notar que en el caso de utilizar una red con una capa oculta y funciones lineales, la función total que la red computa es *no lineal*, ya que es la composición de varias funciones lineales. Agregando capas ocultas se puede complejizar la transformación de los ejemplares, haciendo aún más no lineal el comportamiento de la red.

3.3.3.1. Funciones de activación usuales

La capacidad de una red está definida en gran parte por sus funciones. En la práctica la función de combinación suele ser la lineal, ya que los

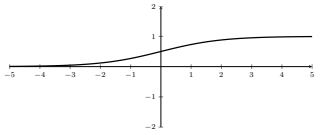
hiperplanos son buenos como separadores de primera línea, debido a que particionan el espacio *grosso modo*. En el caso más simple, se puede utilizar la función de activación θ_t, que modela un hiperplano lineal de separación. Si bien esta función es útil, tiene la desventaja de que es muy simple y no es diferenciable en el origen, requisito necesario para utilizar la mayoría de los algoritmos de entrenamiento, como backpropagation.

Dentro de las funciones no lineales, las funciones polinómicas como x^p son útiles en ciertos casos, pero tienen dos problemas principales:

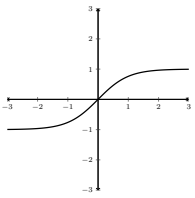
- Suelen ser poco representativas como funciones de la capa de salida, ya que su imagen es $\mathbb{R}$ y hacen difícil codificar clases en la capa de salida.
- Su utilización en las capas ocultas puede hacer que los estímulos de la red se hagan arbitrariamente grandes o pequeños, haciendo muy difícil el entrenamiento de la misma, ya que algunas neuronas tendrán tanta señal que *taparán* los estímulos de las otras.

Entonces, en la práctica se suelen utilizar funciones no lineales de activación que sean **sigmoideas**. Las funciones sigmoideas se caracterizan por tener dos asíntotas horizontales que limitan los valores de la función por arriba y por debajo, de manera que saturen en dos valores, típicamente 0 y 1 o -1 y 1 [66]. Las sigmoideas más utilizadas son la función logística S y la tangente hiperbólica *tanh*:

$$S(x) = \frac{1}{1 + e^{-x}} \qquad tanh(x) = \frac{sinh(x)}{cosh(x)}$$



(a) La función logística $S(x) = \frac{1}{1+e^{-x}}$.



(b) La tangente hiperbólica $tanh(x) = \frac{sinh(x)}{cosh(x)}$.

Las funciones S y *tanh* tienen asíntotas horizontales en 0 y 1, y -1 y 1, respectivamente. Las dos funciones están relacionadas por $tanh(x) = 2S(2x) - 1$; entonces, *tanh* es una versión escalada de S, o viceversa.

La diferencia fundamental entre las dos es que *tanh* suele funcionar mejor en las capas ocultas, ya que su imagen es $(-1, 1)$, con lo cual tiene la ventaja de cruzar el 0 y ser simétrica respecto al eje x, lo cual puede acelerar el aprendizaje ya que evita que se saturen rápidamente los valores por ser todos positivos, como en el caso de la función logística. En otras palabras, las salidas de las neuronas suman y restan, con lo cual una red aleatoria tenderá más a tener salidas centradas en 0, mientras que una red aleatoria con funciones logísticas tiende a saturarse en 1.

La logística tiene una imagen que puede ser interpretada como una confianza, haciéndola útil para la capa de salida. Se utiliza el término confianza y no probabilidad, ya que nada asegura que el valor de las neuronas de salida represente una distribución de probabilidad.

Para ello, a veces se agrega una capa de salida con funciones llamadas **softmax** que normalizan el valor de las neuronas de salida de manera que su suma sea 1 y puedan funcionar mejor como estimaciones de probabilidad. Si hay L neuronas con funciones en la capa de salida original, con valor o_i, se agregan L neuronas softmax con valor:

$$o_i^{sm} = \frac{o_i}{\sum_j o_j}$$

Las neuronas de la capa de salida original pueden tener funciones de activación logísticas u otras.

Entonces, si se utilizan funciones logísticas, por ejemplo, y hay C clases, la capa de salida tiene C elementos y la clase c se puede codificar con el vector $\boldsymbol{y}$ tal que $y_c = 1$ e $y_j = 0,\ j \neq c$. La salida de la red será ahora una función continua,

La elección de las funciones a utilizar depende fuertemente del dominio del problema particular a resolver, así como de la capacidad de cálculo disponible, y la topología elegida para la red.

3.3.3.2. El modelo feedforward

Combinando estas tres ideas, se puede definir el modelo feedforward como una red acíclica con $L + 1$ capas, donde la capa 0, la primera, es una capa de entrada, las siguientes son capas ocultas $(1, \ldots, L - 1)$, y la última, la L, es la capa de salida. Las neuronas de la capa j solo se conectan con las de la capa $j + 1$, exceptuando las de la capa de salida que no tienen conexiones salientes. Cada capa puede tener un número distinto de neuronas n_j.

Las funciones de cada neurona son una composición de una función de combinación $c_i :: \mathbb{R}^n \mapsto \mathbb{R}$ y una función de activación $\theta_i :: \mathbb{R} \mapsto \mathbb{R}$, de naturaleza lineal o no lineal. Cada función tiene parámetros que en el proceso de entrenamiento deben modificarse con el objetivo de minimizar el error en la red.

En el modelo feedforward, generalmente se realizan otras dos simplificaciones respecto del modelo general de redes neuronales:

- Se utilizará la misma familia de funciones en todas las neuronas de una misma capa, aunque cada neurona tendrá sus propios parámetros para su función.
- La función de combinación será lineal, es decir $c(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$.

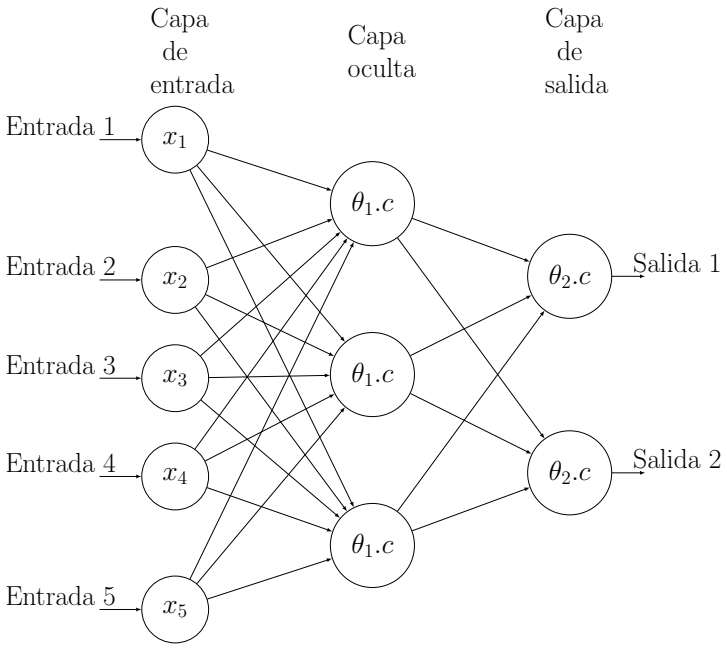


Figura 3.16: Una red feedforward con 3 capas.

Se hará referencia a la salida de la neurona i de la capa j como o_i^j. La capa de entrada será la 0, con $n_0 = d$ neuronas, y entonces los valores salida de dicha capa son $\boldsymbol{o^0} = (o_1^0, \ldots, o_d^0)$. Las neuronas de la capa de entrada no tienen ninguna función (o utilizan la función identidad), y entonces al ser estimulada con un ejemplar $\boldsymbol{x}$, los valores de sus neuronas son $\boldsymbol{o^0} = \boldsymbol{x}$.

Existe una función de activación θ^j para las neuronas de las otras capas ($j > 0$). Además, para la neurona i de la capa $j > 0$, hay un vector de coeficientes del hiperplano de la función de combinación, $\boldsymbol{w}_i^j \in \mathbb{R}^{n_{j-1}}$. Entonces, para cada capa $j > 0$ existe una matriz de coeficientes $\boldsymbol{W}^j \in \mathbb{R}^{n_j \times n_{j-1}}$, compuesta por filas $\boldsymbol{w}_1^j, \ldots, \boldsymbol{w}_{n_j}^j$ los cuales *pesan* las salidas de las neuronas de la capa $j-1$. Dicho de otra manera, $\boldsymbol{w}_i^j$ tiene los pesos de los arcos que llegan a la neurona i de la capa j.

De esta manera, si $\boldsymbol{w}_i^j(k)$ es el componente k del vector de pesos $\boldsymbol{w}_i^j$, la salida de la neurona i de la capa j, o_i^j, puede expresarse como:

$$\begin{aligned} o_i^j &= \theta_j(net_i^j) \\ net_i^j &= \sum_{k=1}^{n_{j-1}} o_k^{j-1} \boldsymbol{w}_i^j(k) = \boldsymbol{o^{j-1}} \cdot \boldsymbol{w}_i^j \end{aligned}$$

En esta expresión, net_i^j se conoce como la entrada neta a la neurona i de la capa j, asumiendo que el escalado de las salidas de las neuronas de la capa anterior por los pesos $\boldsymbol{w}_i^j$ se hace antes de que el estímulo llegue a la neurona, en la sinapsis.

En una red de dos capas, una de entrada y una de salida, para la neurona de salida i (capa 1), la expresión es:

$$\begin{aligned} o_i^1 &= \theta_1(net_i^1) \\ &= \theta_1(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) \\ &= \theta_1(\boldsymbol{x} \cdot \boldsymbol{w}_i^1) \end{aligned}$$

En una red de tres capas, una de entrada, una oculta y una de salida, para la neurona de salida i (capa 2), la expresión se expande a:

$$\begin{aligned} o_i^2 &= \theta_2(net_i^2) \\ &= \theta_2(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2) \\ &= \theta_2([\theta_1(net_1^1), \ldots, \theta_1(net_{n_1}^1)] \cdot \boldsymbol{w}_i^2) \\ &= \theta_2([\theta_1(\boldsymbol{o}^0 \cdot \boldsymbol{w}_1^1), \ldots, \theta_1(\boldsymbol{o}^0 \cdot \boldsymbol{w}_{n_1}^1)] \cdot \boldsymbol{w}_i^2) \\ &= \theta_2([\theta_1(\boldsymbol{x} \cdot \boldsymbol{w}_1^1), \ldots, \theta_1(\boldsymbol{x} \cdot \boldsymbol{w}_{n_1}^1)] \cdot \boldsymbol{w}_i^2) \end{aligned}$$

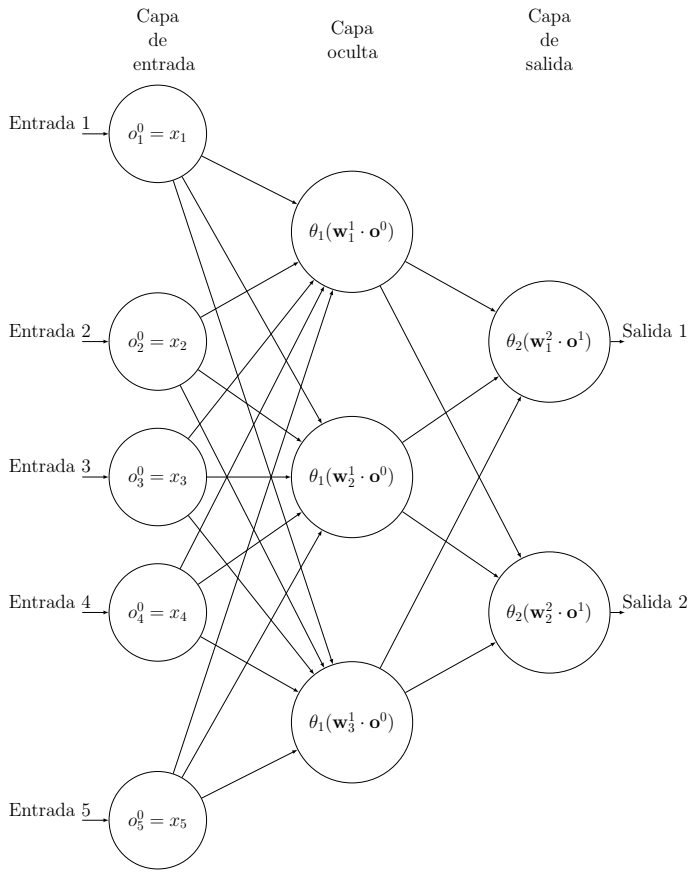


Figura 3.17: Una red feedforward con 3 capas, donde ahora se muestra la estructura recurrente de los cálculos de cada o_i^j.

Este modelo tiene, a priori, tantos parámetros como pesos $w_i^j(k)$ haya. Puede que tenga más parámetros, por algunas de las funciones θ_j. El objetivo de un algoritmo de entrenamiento será optimizar el error de la red respecto a estos parámetros.

Habiendo descripto el modelo, se explica ahora el algoritmo backpropagation para entrenar la red, y luego una de sus mejoras, el resilient backpropagation.

3.3.4. Algoritmo de entrenamiento Backpropagation

La idea esencial del backpropagation es recorrer el espacio de parámetros de la red utilizando el opuesto del gradiente del error como indicador de la dirección a tomar para minimizarlo, una técnica conocida como **des-**

censo de gradiente.

Dada una red neuronal con $L+1$ capas y un ejemplar $\boldsymbol{x_i}$, la salida de la red es $\boldsymbol{o}^L(\boldsymbol{x_i})$. Si la clase asociada al ejemplar es $\boldsymbol{y_i}$, donde ahora $\boldsymbol{y_i} = (y_1, \ldots, y_C)$ es un vector que señala la pertenencia del ejemplar a cada clase, se puede definir el error cuadrático la red sobre un conjunto de datos D como[5]:

$$E = \sum_{\boldsymbol{x_i} \in D} E(\boldsymbol{x_i})$$
$$E(\boldsymbol{x}) = \sum_{l=1}^{C} (o_l^k - y_l)^2$$

Donde en la expresión $E_{\boldsymbol{x}}$ se asume que o_l^L es la salida de la neurona l de la capa de salida para el ejemplar $\boldsymbol{x_i}$ e y_l es la salida esperada de la neurona l [6].

Entonces, dada una topología y una función de activación para cada capa, si los parámetros son $\boldsymbol{\alpha}$, el objetivo del entrenamiento es:

$$\underset{\boldsymbol{\alpha}}{\text{Min}} \quad E(\boldsymbol{\alpha}) = \sum_{\boldsymbol{x_i} \in D} E_{\boldsymbol{x_i}}$$

En este caso, si se utilizan funciones de activación sin parámetros en una red de $L+1$ capas, los parámetros estarán dados por las matrices de pesos de cada capa $\boldsymbol{\alpha} = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L\}$, y el problema definitivo es:

$$\underset{\{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L\}}{\text{Min}} \quad E(\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L) = \sum_{\boldsymbol{x_i} \in D} E_{\boldsymbol{x_i}}$$

La dirección en el espacio de parámetros en la cual E se incrementa está dada por:

$$\frac{\partial E}{\partial \boldsymbol{W}^1}, \ldots, \frac{\partial E}{\partial \boldsymbol{W}^L}$$

[5]Se incluye la constante $\frac{1}{2}$ para simplificar la derivada de $E(\boldsymbol{x})$.

[6]Si bien es poco claro que $E(\boldsymbol{x})$ depende de un ejemplar en particular viendo solamente su definición, se realiza esta simplificación ya que de lo contrario la notación debe incluir la capa y número de la neurona de cuyo valor se está hablando, y además el ejemplar que produjo el estímulo inicial para generar dicho valor.

Por ende, el opuesto de esta dirección indica la dirección en la cual E decrece, y siguiendo esta dirección se puede minimizarla. Esto induce a una regla de actualización para mejorar la solución actual del tipo:

$$\boldsymbol{W}^j \leftarrow \boldsymbol{W}^j - \alpha \frac{\partial E}{\partial \boldsymbol{W}^j}$$

Donde α es una tasa de aprendizaje que se puede utilizar para regular la magnitud de los cambios. Esta regla permite que el vector $\boldsymbol{W}^j$ se mueva en el espacio de parámetros en la dirección en que el error decrece.

El algoritmo de entrenamiento utiliza este hecho para, de forma similar al Perceptrón, modificar iterativamente los pesos de la red hasta que el error de la misma sea lo suficientemente bajo:

Data:
Una toplogía dada por $L+1$, el número de capas, y n_i la cantidad de neuronas de la capa i, con $n_L = C$.
Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, con $D \subset \mathbb{R}^d$, y clase $\boldsymbol{y_i} \in \mathbb{R}^C$.
Una tolerancia al error ϵ.
Una tasa de aprendizaje α.
Result: Pesos $\boldsymbol{W}^j$ de las neuronas de cada capa $j > 0$, optimizados para clasificar D con un error menor a ϵ

```
for j=1 to L do
    W^1 = random(n_j, n_{j-1});
end
error = ∞ ;
while error > ε do
    for j=1 to L do
        dW^j = ∂E/∂W^j;
    end
    for j=1 to L do
        W_j = W_j − α dW_j ;
    end
    error = E(W^1, ..., W^L) ;
end
Retornar W^1, ..., W^L ;
```

Algoritmo 3: Esquema del algoritmo Backpropagation para una red de tres capas.

Con la presentación de este algoritmo clásico para entrenar redes neuronales feedforward finaliza la descripción de dichas redes. Para el lector

interesado, el apéndice C presenta de forma didáctica la derivación de la forma concreta de la regla de actualización para una red neuronal feedforward genérica. Dicho apéndice también menciona algunos problemas comunes en la aplicación efectiva del Backpropagation, e introduce el Resilient Backpropagation para mitigar algunos de ellos. La aplicación del algoritmo Resilient Backpropagation fue necesaria en esta tesina para lograr un nivel de error en el reconocimiento de gestos aceptable con redes feedforward.

3.4. Redes Competitivas

Las redes neuronales competitivas (CPN) son típicamente utilizadas para realizar aprendizaje no supervisado. La no supervisión consiste en que la red descubra por sí sola características, regularidades, correlaciones o categorías en los datos de entrada, y se obtengan de forma codificada a la salida. Por ende, se puede decir que estas unidades y conexiones muestran cierto grado de auto-organización. [7]

El aprendizaje no supervisado sólo consigue resultados útiles si en los datos de entrada existe cierto tipo de redundancia. Sin redundancia sería imposible encontrar patrones o características en los datos, lo cual se asemeja necesariamente a ruido aleatorio; en este sentido, la redundancia es fuente de conocimiento. Dicho de una manera más precisa, el contenido total de información del conjunto de datos de entrada es menor que el máximo que podría ser soportado por el mismo canal; esa diferencia es la redundancia.

El tipo de patrón que una red con aprendizaje no supervisado puede aprender depende de su arquitectura. Se puede analizar el funcionamiento de una red competitiva desde varias perspectivas, mutuamente relacionadas:

- **Familiaridad:** Mediante un único valor continuo se pondera el grado de similitud entre la entrada y un valor *tipo* o media de una serie de valores presentados con anterioridad. La red, a su vez, puede ir aprendiendo que es un valor tipo.
- **Análisis de componentes principales:** La red detecta cuáles de las componentes del conjunto de entrada caracterizan en mayor grado al conjunto de datos de forma que las demás puedan eliminarse sin una significativa pérdida de información.

[7] Esta introducción está fuertemente basada en la sección 6.1 del libro *Redes Neuronales Artificiales* de Galván et al. [67].

- **Agrupamiento:** A partir de un conjunto de entrada se desea conocer si hay un conjunto de datos que están bien representados por algún elemento o elemento promedio de los ejemplares, y de qué manera se agrupan los datos a estos ejemplares representativos que ha encontrado la red. Esta sería una forma de clasificación sin conocimiento de las etiquetas de los datos.

- **Prototipado:** Igual que en el caso anterior, pero en vez de obtenerse como resultado a qué clase pertenece el dato de entrada, se obtiene un prototipo o ejemplar de la clase a la que pertenece dicho dato de entrada.

- **Codificado:** Se obtiene, a la salida de la red, una versión codificada del dato de entrada, es decir, un dato de menor dimensión que mantenga el máximo de información que le sea posible.

A veces los métodos de aprendizaje no supervisados pueden dar mejores resultados incluso en situaciones en que es posible aplicar métodos de aprendizaje supervisado.

Por ejemplo, el algoritmo backpropagation puede ser muy lento en ciertos casos, por ejemplo, si los patrones son de una dimensionalidad muy alta; una solución híbrida podría aplicar un método no supervisado para recodificar los ejemplares y luego alimentar estos ejemplares a una red feedforward entrenada con backpropagation u otra técnica.

La arquitectura de las redes con aprendizaje no supervisado suele ser bastante simple. La complejidad de las mismas radica en sus leyes de aprendizaje. Muchas de ellas tienen una sola capa, y suelen tener muchas menos neuronas que la dimensionalidad de los datos de entrada.

Además, suelen estar más cerca de las estructuras neurobiológicas que las puramente computacionales, ya que se basan en estudios cerebrales y tienden a imitar mejor sus características y comportamiento.

Los modelos de ANN con entrenamiento no supervisado más comunes quizás sean los **Mapas auto organizativos** de Kohonen [68], que no se tratarán, y las **redes competitivas** [69].

3.4.0.1. Modelo de interacción lateral

Una de las propiedades del cerebro que en general se pasa por alto en los modelos artificiales existentes es la significancia del **orden** de sus unidades de proceso. Este orden hace que unidades estructuralmente

idénticas tengan una funcionalidad diferente debido a parámetros internos que evolucionan de forma variable según sea la distribución espacial de las neuronas.

Esta propiedad *topológica* del cerebro parece ser de fundamental importancia para la representación de cierto tipo de información, en las imágenes visuales, en las abstracciones, etc. Estos mapas topológicos se encuentran presentes en la corteza cerebral y se encargan de diversas tareas de tipo sensorial y motor.

Cuando se piensa, y también en el procesado de la información del subconsciente, se realiza una compresión de la información formando representaciones reducidas con los aspectos más relevantes de la misma, sin que se produzca por ello ninguna pérdida importante de conocimiento acerca de las interrelaciones que se producen. El propósito de este procesado inteligente de la información parece ser la creación de imágenes del mundo observable, a varios niveles de abstracción.

Así pues, uno de los rasgos esenciales de las redes neuronales es el de disponer de una mecanismo capaz de extraer de manera automática las características más relevantes de un conjunto de datos, en principio arbitrario, y de ámbito heterogéneo. Corresponde a un sistema de los denominados de *clustering* capaz de adaptarse de forma automática a los estímulos recibidos y producir una clasificación en función de las características significativas, sin ningún mecanismo externo que determine el resultado.

Existen varios modelos de redes neuronales artificiales que tratan de incorporar las propiedades anteriores. Estudios realizados sobre el neocórtex cerebral indican que está formado esencialmente por capas bidimensionales de neuronas interconectadas en cada capa por conexiones laterales.

Cada neurona está conectada con otras de su entorno de manera que produce una excitación en las más próximas y una inhibición en las más alejadas. Tanto la excitación como la inhibición laterales son gradualmente más débiles a medida que se aumenta la distancia a la neurona en cuestión.

Este mecanismo hace que cuando un estímulo produce una reacción en una neurona, las neuronas de su inmediato entorno se vean influenciadas por dicha reacción, de una manera positiva las más cercanas y negativa las más alejadas.

Entonces, el aprendizaje competitivo es un tipo de aprendizaje no supervisado que sirve de base para varios modelos de redes neuronales artificiales. El objetivo de este tipo de redes es buscar algún tipo de estructura en los datos de entrada. Se trata de que los datos parecidos sean

clasificados como pertenecientes a la misma clase. En estos modelos suele haber una capa de clasificación compuesta por tantas neuronas como categorías pueda haber en los datos. Cada categoría está representada por un prototipo cuyas características son una especie de compendio de las características de los datos pertenecientes a esa misma categoría. En la capa de clasificación, cada neurona corresponde, pues, a un prototipo. El sistema debe relacionar cada neurona (prototipo) con los datos de entrada que representa. En otros términos, debe agrupar los datos de entrada en categorías, por razones de similitud, y asignar a cada categoría un prototipo que más tarde será utilizado para clasificar datos nuevos y desconocidos.

A continuación, se desarrollará el modelo más simple de redes competitivas, las redes competitivas **el ganador se lo lleva todo** o **winner-takes-all** en la cual se asume que las neuronas tienen una topología tal que la distancia entre cualquier par de neuronas es la misma [70].

3.4.1. Aprendizaje competitivo en redes "El ganador se lo lleva todo"

La esencia de las redes CPN **winner-takes-all** es tener una capa de neuronas donde las mismas compiten por parecerse a los ejemplares de entrada, y de esa manera van aprendiendo a representar de forma sucinta la variedad del dominio $\mathcal{P}$ que modelan.

La arquitectura de estas redes posee dos capas; la 1, de entrada y la 2 de salida.

Cada neurona de la capa 1 está conectada con todas las neuronas de la capa 2. Cada neurona j de la capa 2 contiene a su vez un vector de pesos $\boldsymbol{w}_j \in \mathcal{P}$ para modelar las conexiones entre la capa 1 y la 2. Luego de entrenar la red, este vector representará un ejemplar prototipo del dominio a modelar. Este vector también se conoce como el **centroide** de la neurona.

Hay dos perspectivas sobre estas redes: la dinámica, más biológicamente plausible, y la estática, más simple computacionalmente.

Perspectiva dinámica

Desde la perspectiva dinámica, se puede considerar que la capa 2, además de recibir las entradas de la capa 1, implementa el modelo de interacción lateral incluyendo conexiones inhibitorias fijas entre todas las neuronas de su capa, excepto consigo misma donde la conexión es excitatoria. Dicho de otra forma, esta arquitectura es exactamente un sistema

de interacción lateral en el que la función de interacción de cada neurona solo toma valores positivos para un radio de 0, es decir, para la propia neurona, y valores negativos constantes para el resto de las neuronas.

En este modelo, cada neurona se refuerza a sí misma, es decir, realimenta su valor de activación para compensar las inhibiciones que provienen del resto de las neuronas de su capa.

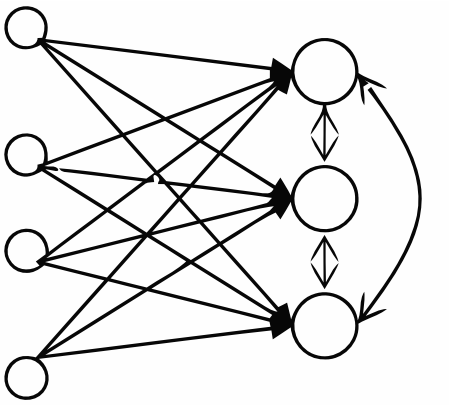


Figura 3.18: Arquitectura de una red competitiva desde una perspectiva dinámica.

Cuando se recibe una entrada en la capa 1, esta se propaga hasta la capa 2. Una vez que las neuronas de la capa 2 han sido activadas por la capa 1, ésta deja de actuar y la señal es propagada de forma asíncrona a través de la capa 2, de manera que todas las neuronas tratarán de impedir que las demás tengan un valor de activación alto, gracias a las conexiones inhibitorias con todas sus vecinas; a la vez que intentarán tener ellas mismas un valor de activación alto, gracias a las conexiones reflexivas excitativas. En algún momento la dinámica de la red hará que una neurona quede con un valor de activación alto, mientras que el resto de las neuronas son inhibidas (tienen un valor de activación nulo o cero); se dice que la primer neurona es la **neurona ganadora**. En ese punto que la capa 2 se ha **estabilizado**.

Se dice entonces que las neuronas de la capa 2 *compiten* por la entrada, de ahí el nombre del método. Sólo una consigue ganar la competición; a esa se la etiqueta con el nombre de neurona ganadora. La neurona ganadora representa al prototipo más similar que se asigna al dato de entrada.

El algoritmo que describe el funcionamiento de la red es el siguiente:

1. Se recibe el estímulo en la capa 1.

2. Se propaga la señal hasta la capa 2 y se calcula el valor de excitación para cada neurona j de esta capa, de acuerdo a los vectores de pesos internos $\boldsymbol{w}_j$.

3. Se inhiben las conexiones entre la capas 1 y 2

4. Se propaga la señal por la capa 2, calculándose los nuevos valores de excitación de las neuronas. Si la red se ha estabilizado y hay una sola neurona ganadora c, ir al paso 5. De lo contrario, volver a ejecutar el paso 4.

5. Calcular el nuevo valor para el vector $\boldsymbol{w}_c$ de la neurona ganadora c del paso 4. Re-establecer las conexiones entre las capas 1 y 2.

Perspectiva estática

Desde el punto de vista estático o computacional, se puede simplificar el modelo asumiendo que la neurona ganadora será precisamente la que al principio ha recibido la mayor excitación, ya que será la que habrá inhibido en mayor grado al resto y también se habrá reforzado a sí misma en mayor grado. Entonces, en la capa 2 no hay aprendizaje de los pesos de inhibición y excitación entre las neuronas, los mismos son fijos y todos tienen el mismo valor constante. De esta manera se evita modelar la dinámica explícitamente.

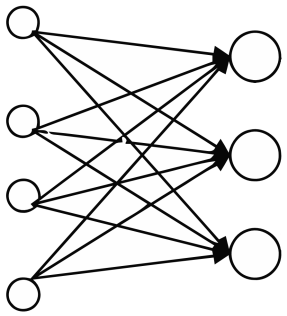


Figura 3.19: Arquitectura de una red competitiva con 4 neuronas de entrada y 2 de salida desde una perspectiva estática.

Generalizando, se puede considerar como parámetro de la red una medida de distancia entre ejemplares d, que utilizarán las neuronas para calcular la distancia entre su centroide y los ejemplares de acuerdo con alguna norma, con lo cual se puede deducir su similitud. Entonces, dado un ejemplar de entrada $\boldsymbol{x}$, se calcula la neurona ganadora c como:

$$c = \underset{j}{\operatorname{argmax}}\, g_j(\boldsymbol{x})$$

$$g_j(\boldsymbol{x}) = \begin{cases} 1 & \text{si } d(\boldsymbol{x}, \boldsymbol{w}_j) < d(\boldsymbol{x}, \boldsymbol{w}_i),\ \forall i \neq j \\ 0 & \text{de lo contrario} \end{cases}$$

La función g_j entonces es 1 si el centroide de neurona j es el de mayor similitud al ejemplar de entrada $\boldsymbol{x}$ en relación al resto de los centroides. En otras palabras, si $g_j = 1$ la neurona j es la ganadora. Como funciones de distancia, puede utilizarse la euclídea $d(x, y) = \|x - y\|_2$, la gaussiana $d(x, y) = 1 - e^{-(\|x-y\|/\sigma)}$ con alguna norma $\|\cdot\|$, o alguna otra dependiendo del dominio del problema.

Para entrenar la red, se toma un ejemplar y se calcula la neurona ganadora. Para hacer que la siguiente vez el mismo ejemplar haga activarse aún más a su prototipo relacionado, se modifica el centroide $\boldsymbol{w}$ para que sea más similar a $\boldsymbol{x}$. Este aprendizaje sólo modifica las conexiones de la neurona ganadora y por eso recibe el nombre de **winner takes all** (*el que gana se lo lleva todo*). Esto hará que en el futuro cada neurona tenga aún mayor facilidad para reconocer el estímulo que aprendió, e incluso estímulos parecidos. Entonces el cambio en $\boldsymbol{w}_j$ dado un ejemplar $\boldsymbol{x}$ será:

$$\Delta \boldsymbol{w}_j(\boldsymbol{x}) = \begin{cases} \alpha(\boldsymbol{x} - \boldsymbol{w}_j) & \text{si } g_j = 1 \\ 0 & \text{si } g_j = 0 \end{cases}$$

O de forma más succinta:

$$\Delta \boldsymbol{w}_j(\boldsymbol{x}) = \alpha g_j(\boldsymbol{x} - \boldsymbol{w}_j)$$

El aprendizaje puede realizarse durante una cantidad de iteraciones fijas, hasta que para todos los ejemplares haya una sola neurona de salida con un nivel de activación lo suficientemente grande.

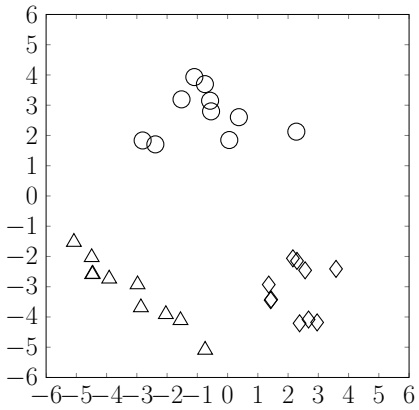
El algoritmo de aprendizaje sería entonces:
Data:
Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, $D \subset \mathbb{R}^d$
Una cantidad de iteraciones I
Una tasa de aprendizaje α
Una función de distancia d
Un tamaño de la capa de salida h
Result: Vectores de parametros $\boldsymbol{w}_j$, j=1,...,h
for $j = 1, \ldots, h$ **do**
  $\boldsymbol{w}_j = random(d)$;
end
$k = 0$;
while $k < I$ **do**
  for $\boldsymbol{x_i} \in D$ **do**
    $c = \text{argmax}_j \, g_j(\boldsymbol{x_i})$;
    $\boldsymbol{w}_c = \boldsymbol{w} + \alpha(-\boldsymbol{w}_c + \boldsymbol{x_i})$;
  end
  $k = k + 1$;
end
Retornar los $\boldsymbol{w}_j$;

Algoritmo 4: Esquema del entrenamiento de una red neuronal competitiva *winner-takes-all*.

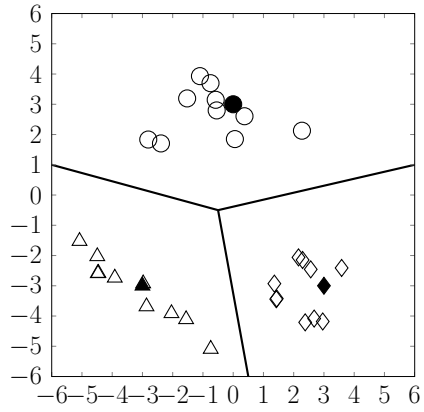
Este es el modelo al que se hará referencia al describir el clasificador neuronal competitivo desarrollado en esta tesina y descripto en el capítulo 7, y se denominará a este algorítmo como el típico para el entrenamiento de una CPN.

Geometría del clasificador y consideraciones sobre la representación

En las redes CPN, cada prototipo o centroide $\boldsymbol{w}$ tiene asignada cierta región del espacio en base a la función de similitud s:



(a) Un conjunto de ejemplares con tres grupos.



(b) Regiones de Voronoi del espacio asignadas a cada neurona por la red

En la figura b) se muestran las regiones de voronoi para una CPN de 3 neuronas de salida con una función de distancia euclídea. Cada conjunto de ejemplares agrupado se conoce como **cluster**; en la figura hay 3, y la red aprende el punto medio de cada una. Se puede considerar entonces que el algoritmo de entrenamiento genera un mapa del espacio $\mathcal{P}$, donde hay regiones significativas agrupadas.

Si la salida de la capa 2 son los g_j, de la manera que $\boldsymbol{o} = (g_1, \cdots, g_h)$, se puede ver esta salida como una codificación del ejemplar en donde se indica a qué clase pertenece mediante la neurona que es más parecida. El concepto de clase se crea en base a los patrones aprendidos y a su forma.

Se puede argumentar que con esta simple información, que puede ser de una dimensionalidad mucho menor que la del ejemplar, se logra una representación más eficiente ya que se abstraen los detalles que provienen de variaciones sin importancia en el muestreo de los ejemplares.

De esta manera se genera un espacio de características, en donde los ejemplares se representan por los vectores $\boldsymbol{o}$. Se puede relajar dicha representación si los elementos del vector $\boldsymbol{o}$ no son simplemente unos y ceros, sino alguna función continua y monótonamente creciente que indica la pertenencia del ejemplar a cada clase o *cluster*.

Si los vectores que representan los ejemplares están normalizados de manera que el mapa generado sea independiente de la norma de $\boldsymbol{x}$, se puede encontrar otra representación geométrica del aprendizaje competitivo. Como todos tienen norma 1, los ejemplares pueden visualizarse como puntos en una hiperesfera de radio 1. Si hay tres vectores de pesos $(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$, mediante el proceso de aprendizaje estos también tendrán una tendencia a ser puntos en esta esfera de radio 1.

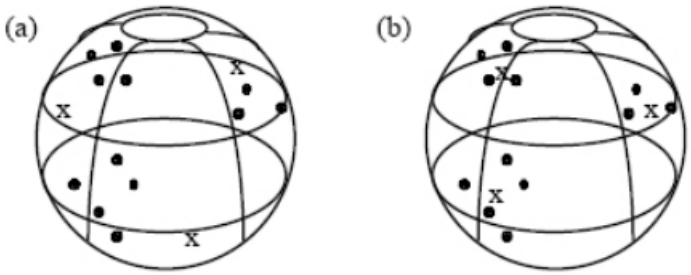


Figura 3.21: a) Los ejemplares y los 3 vectores $\boldsymbol{w}_j$ de las neuronas ocultas (X) al comienzo del algoritmo, b) Luego de entrenada la red, las neuronas $\boldsymbol{w}_j$ se posicionan en los centros de los grupos.

Por último, una de las aplicaciones más importantes del aprendizaje competitivo y sus variaciones es la **cuantización de vectores**. La idea de la cuantización de vectores es categorizar un conjunto de ejemplares de entrada en M clases, y representar cada ejemplar mediante su clase. Es clave que en lugar de utilizar los componentes de un vector, generalmente de valores continuos, se puede guardar sólo el índice de la clase. Esto puede ser utilizado, por ejemplo, tanto para almacenar información de forma comprimida como para generar características ϕ a partir de los datos.

3.5. Resumen

En este capítulo se introdujeron las redes neuronales artificiales (ANN). Se describieron brevemente las bases del sistema nervioso y el funcionamiento de las neuronas, para luego derivar un modelo computacional inspirado en los sistemas biológicos.

Las redes neuronales son aplicables al reconocimiento de gestos por dos motivos principales:

1. Son excelentes clasificadores.

2. Sirven para generar características de forma automática a partir de los ejemplares. Dichas características facilitan la clasificación de gestos.

Entonces, permiten clasificar gestos y generar características para los gestos que mejoren dicha clasificación.

Un tema principal en el área de ANN es como determinar la topología de la red y las funciones matemáticas de las neuronas para lograr que resuelvan cierto problema. Dicho proceso se denomina el *entrenamiento* de la red neuronal.

Además de introducir el concepto de ANN, los ejes de este capítulo son el modelo feedforward y dos de sus algoritmos de entrenamiento, *backpropagation* y *resilient backpropagation*, parar resolver problemas de clasificación, y el modelo de redes neuronales competitivas, para realizar aprendizaje no supervisado.

Estos dos modelos, redes feedforward y competitivas, se utilizan luego para implementar clasificadores de gestos, como se describe en el capítulo 6. En particular, se emplean redes CPN para implementar el clasificador CNC presentado en esta tesina.

4

Gestos

> "Gestures are an elaborate and secret code that is written nowhere, known to none, and understood by all."
>
> Edward Sapir

4.1. Introducción

Realizar un gesto es utilizar las distintas partes del cuerpo con el propósito de comunicar información. Mientras que es fácil entender que es un gesto intuitivamente, existen diversos tipos y en general cada tipo involucra un modelado y proceso de reconocimiento con características particulares. A continuación, se describen ciertas clasificaciones y propiedades de los gestos con el objetivo de generar criterio para elegir un modelo específico a reconocer.

El estudio de los gestos tiene una larga historia interdisciplinaria que involucra las disciplinas de la psicología, antropología, linguística, neurociencia, comunicación, actuación, danza, expresión corporal y ciencias de la computación. En la investigación de los gestos, tradicionalmente parte de las ciencias sociales, en general se considera solamente el movimiento de las manos y los brazos en el concepto de gesto y se reserva la palabra kinésica para referirse a la comunicación mediante lenguaje corporal, que involucraría los gestos, el movimiento, la postura, los movimientos de cabeza, la mirada, las expresiones faciales, etc. En la jerga informática, dichas definiciones se mezclan, y se habla de gestos refiriéndose a ambos conceptos.

Existe un gran corpus de investigación sobre la relación entre los gestos y el habla, el pensamiento y los gestos, y la clasificación de gestos [71, 72, 73, 74, 75]. Si bien no existe una taxonomía definitiva para clasificar distintos tipos de gestos, existen ciertos criterios tanto matemáticos como linguísticos para distinguirlos, y algunas clasificaciones establecidas.

4.2. Clasificaciones

4.2.1. Clasificación de McNeill

Desde el punto de vista del uso comunicacional de los gestos, es relevante la clasificación de McNeill [75, 76], una de las más conocidas, que surge de experimentos donde la premisa básica era estudiar los gestos que produce una persona que, luego de ver una película, debe contar gestualmente la historia a otra persona que no la ha visto.

En base a estas y otras experiencias, McNeill propuso la existencia de cuatro tipos principales de gestos: **icónicos**, **metafóricos**, **ilustrativos** y **deícticos**. Considera que todos son simbólicos, en el sentido en que las partes del cuerpo representan algo diferente que ellas mismas, y que se encuentran estrechamente relacionadas a los aspectos semánticos y pragmáticos del discurso hablado al que acompañan. Si bien en este sentido McNeill se enfoca en estudiar a los gestos como parte de un discurso mayor en el que participan junto al habla y otras expresiones corporales, sus distinciones no dejan de ser relevantes al uso de gestos como método de comunicación e interfaz hombre máquina en ausencia del habla, ya que en ese caso se puede argumentar que existe todavía un discurso, aquel dado por el diálogo entre el actor de los gestos y la interfaz de usuario.

Los **icónicos** son "gestos de lo concreto", representan algún tipo de objeto y dan información acerca de su tamaño, forma, orientación, relaciones espaciales, etc, exhibiendo imágenes transparentes de dicho objeto al que refieren. Tales imágenes pueden ser redundantes, es decir, co-expresivas, con el discurso: una mano girando en círculos con el dedo índice apuntando hacia abajo significa una torta en una mesa; una mano elevándose representa algo o alguien subiendo. También pueden ser complementarias, capturando un aspecto del discurso que el habla ignora, tal como cuando un narrador describe a una mujer persiguiendo a un perro fuera de su casa e indica su arma - una escoba - no en palabras, sino con movimientos amenazadores del brazo.

Los gestos icónicos tienden a predominar en las narrativas, pero los otros tres, los "gestos de lo abstracto", también suelen ocurrir en las

narraciones y predominan en otros géneros de interacción como las conversaciones y las exposiciones orales.

Los gestos **metafóricos** no son menos pictóricos que los icónicos, pero la imagen que representan es la de una abstracción. Por ejemplo, un tipo de gesto llamado "conducto metafórico" representa como algo sustancial el lenguaje, significado, conocimiento, arte u otras nociones abstractas. Tal es el caso cuando un narrador, mientras introduce su obra, levanta las manos como si estuviese sosteniendo una caja y luego las separa, como si dicha caja se rompiese o abriese revelando la historia que contenía dentro. También lo es cuando un hablante dice "tengo una pregunta", extendiendo su mano en forma de copa, como esperando recibir una respuesta en ella.

Los dos tipos de gestos restantes, ilustrativos y deícticos, no son de carácter pictoral. Los **ilustrativos** son gestos que acompañan a la comunicación verbal para matizar o recalcar lo que se dice, para suplantar una palabra en una situación difícil, etc. Se utilizan intencionadamente. Típicamente son movimientos simples en dos fases (arriba/abajo, adentro/afuera) en donde la mano se mueve de forma rítmica con el habla. Cumplen una función pragmática en un diálogo, indicando que una frase o palabra es importante, pero en general no tienen un contenido semántico de peso.

Finalmente, los gestos **deícticos** se utilizan para seleccionar o señalar objetos. Son los primeros gestos que realizan los niños para referirse a cosas en su ambiente, aunque no sólo se utilizan para señalar cosas concretas, como cuando se mueve el brazo para atrás para señalizar el pasado o se indica una dirección. En un ejemplo más elaborado, un narrador puede tomar el espacio delante suyo como el espacio de una obra en donde ubica a sus personajes, y apuntar a ellos como indicación de que las palabras que dice o acciones que realiza son obra de cierto personaje. Es interesante notar que ciertos lenguajes de señas utilizan de forma extensiva los gestos deícticos abstractos, y se presume que ahí tienen su origen.

En el contexto de HCI, los gestos más típicos han sido los deícticos ya que gran parte de la interacción es de carácter espacial, es decir, ir atrás/adelante, entrar, salir, moverse, etc. Los icónicos también son utilizados, por ejemplo, para transmitir el concepto de agrandar o achicar de la función de zoom, para representar caracteres y números gráficamente, y especialmente en juegos en donde generalmente representan acciones tales como utilizar una raqueta de tenis o dar un golpe. Los gestos ilustrativos y metafóricos no son utilizados en HCI dado que están intrínsecamente relacionados al discurso que acompañan y a relaciones sutiles con el contexto que son muy difíciles de reconocer y traducir en una interfaz.

4.2.2. Clasificación para su reconocimiento

Desde la perspectiva de las ciencias de la computación, se consideran aquellos aspectos de los gestos que impactan en el método de sensado y reconocimiento y en su utilidad como elementos de interacción en una interfaz de usuario o sistema de control. Los más importantes son:

- **Dinámicos vs Estáticos** Los gestos **dinámicos** están formados por una secuencia de movimientos de una o varias partes del cuerpo. Su reconocimiento involucra un modelado espacio-temporal de dichas partes del cuerpo a medida que se realiza el gesto. Es necesaria alguna forma de segmentación para reconocer el comienzo y el fin de los mismos.

  Los **estáticos**, usualmente llamados **poses**, se representan por una configuración específica de las posiciones de ciertas partes del cuerpo.

- **Una sola parte del cuerpo vs Varias**

  Los gestos dinámicos realizados con varias partes del cuerpo pueden tener un elemento extra de sincronización en el caso en que algunas partes tarden distinto tiempo en realizar su segmento del gesto o deban comenzar su segmento antes que otras. Si bien son más expresivos, también son más ambiguos y difíciles de realizar.

  Los estáticos con varias partes del cuerpo también deben tener en cuenta aspectos temporales para ser reconocidos ya que si bien el gesto es una configuración estática, el movimiento del cuerpo para alcanzar dicha pose es de naturaleza dinámica.

- **Faciales vs Corporales**

  Los **faciales** involucran el movimiento de los ojos, las cejas y los labios. Están presentes en sistemas de control por movimiento de los ojos, detección de sonrisa en cámaras fotográficas, etc.

  Los **corporales** involucran el movimiento de distintas partes del cuerpo y se utilizan en sistemas para análisis de movimientos de deportistas, pacientes en rehabilitación médica, entrenamiento de atletas, video juegos, entornos virtuales, etc.

  Se pueden considerar aparte también los gestos que involucran el movimiento de los **dedos de las manos** ya que requieren una precisión mayor y un modelado distinto a los gestos corporales en general.

- **2D vs 3D**

  Si bien los gestos obviamente se realizan en un espacio **3D**, por razones técnicas o de diseño de interfaz en ocasiones se pueden considerar solamente el movimiento en **dos dimensiones**, en general evitando incluir la de profundidad, ya que resulta más difícil de percibir y controlar que las otras dos.

- **Unimodal vs Multimodal**

  Los sistemas **unimodales** utilizan una sola fuente de información para realizar el reconocimiento, mientras que los **multimodales** incorporan información de distintos sensores y diferente naturaleza, como audio, imágenes RGB, imágenes de profundidad, acelerómetros, etc.

- **Dependientes vs Independientes del usuario**

  Al igual que lo que ocurre, por ejemplo, en el reconocimiento del habla, algunos sistemas crean un modelo particular de reconocimiento para cada usuario, quien debe realizar algún tipo de entrenamiento para crear nuevos gestos o adaptar los existentes teniendo en cuenta su forma particular de realizarlos. Estos sistemas se denominan **dependientes del usuario**. Tienen la ventaja de que en general proveen una performance de reconocimiento más alta y cierta capacidad de personalización al poder agregar nuevos gestos personalizados, con la desventaja de que deben ser entrenados por cada nuevo usuario. Sobre todo, permiten un reconocimiento efectivo para ciertos tipos de gestos complejos en donde hay muchos factores a tener en cuenta, como la contextura del usuario, la velocidad con que se mueve, y las particularidades del lugar donde se encuentra realizando los gestos (en el caso de utilizar una cámara como sensor, su posición respecto de la cámara, las condiciones ambientales de luz). Es apto para aplicaciones donde los gestos se utilizarán con una alta frecuencia, se necesita un gran vocabulario de gestos, y se espera y necesita una performance muy alta.

  Los sistemas **independientes del usuario** tienen un único modelo de gestos que se comparte para todos los usuarios. Este enfoque tiene la obvia ventaja de que un nuevo usuario no tiene que realizar ningún tipo de entrenamiento para utilizar el sistema, lo cual lo hace apto para interfaces en lugares públicos o de uso ocasional, como un cajero electrónico, o donde se pueden realizar pocas acciones, como en una vidriera interactiva.

En esta tesina, el foco está puesto en gestos corporales dinámicos y unimodales, dependientes del usuario, y realizados con una sola parte del cuerpo. Además, se emplea un modelo de gestos basado en el cambio de las posiciones de partes del cuerpo en un espacio 3D a través el tiempo, abstrayéndonos de la manera en que dicha información es capturada (imágenes, acelerómetros, etc).

4.3. Modelado y propiedades de los gestos dinámicos

"All models are wrong, but some are useful"
George E. P. Box

Mientras que una clasificación ciertamente reduce el conjunto de tipos gestos a reconocer, todavía se pueden distinguir diversas propiedades de los gestos dinámicos.

Desde el punto de vista del reconocimiento, se busca definir un modelo de gestos que permitirá precisar el concepto de **clases o tipos de gestos** para una aplicación particular. Por ejemplo, si el objetivo es el reconocimiento de dígitos arábigos del 0 al 9, existen 10 tipos clases distintas a reconocer.

En la etapa de reconocimiento, se asignará una clase a un nuevo ejemplar de un gesto a partir de ejemplares previamente obtenidos, que se denominan **ejemplares de entrenamiento**. Para ello, se emplean algoritmos de clasificación basados en aprendizaje automático que de alguna manera comparan el nuevo gesto con los ejemplares de entrenamiento, dando como resultado la clase inferida para el nuevo gesto (o estableciendo que no pertenece a ninguna clase conocida, si no se parece a ningún ejemplar anterior).

Se puede modelar un ejemplar de un gesto dinámico 3D unimodal c como una trayectoria. En este caso existen dos parametrizaciones posibles: una **temporal** y una por **longitud de arco**. Primero se definirá el modelo de ejemplar de gesto a reconocer como una trayectoria con parametrización temporal, con ciertas propiedades e invariancias. Luego se hará lo mismo con la parametrización por longitud de arco, y finalmente se compararán las dos. En estos modelos de gestos dichas trayectorias serán representadas por una función continua, que en secciones posteriores se adaptarán al caso discreto debido a la naturaleza discreta del proceso de captura de los sensores.

4.3.1. Modelo de gestos con parametrización temporal

Un gesto dinámico 3D unimodal es una trayectoria c tal que:

$$\begin{aligned} c &: \mathbb{R} \rightarrow \mathbb{R}^3 \\ c(t) &= (x, y, z) \qquad t = 0 \dots T \end{aligned} \tag{4.1}$$

Donde t representa el instante de tiempo en que la parte del cuerpo está en la posición $c(t)$, y T la longitud temporal del gesto. Se puede asumir c continua dado que dicha trayectoria surge del movimiento de la mano, aunque en la práctica la validez de dicha hipótesis depende del método de sensado y los algoritmos de seguimiento de posición y su habilidad de interpolación en casos de oclusión.

Esto plantea la necesidad de un criterio de equivalencia entre ejemplares, lo cuál implica responder a ciertas preguntas: ¿qué sucede si un nuevo ejemplar recorre la misma trayectoria que un gesto de entrenamiento pero a distinta velocidad? ¿si comienzan en lugares diferentes del espacio virtual donde se mueve la parte del cuerpo? ¿si son iguales, módulo una rotación en el espacio respecto a cierto eje? ¿si sólo difieren en su escala? y la más obvia ¿qué sucede si un nuevo ejemplar se realiza de forma parecida a un gesto previo, pero no exactamente de la misma forma?. Estas preguntas, y sus infinitas variaciones y combinaciones, no tienen *una* respuesta correcta, pero es necesario definir ciertas respuestas para encontrar un modelo de gesto apropiado para la aplicación objetivo. De esta manera se obtiene una definición de clase de gestos, definido como una clase de equivalencia en base a una relación de equivalencia entre ejemplares de gestos. Se definirán ciertas relaciones de equivalencia entre ejemplares y para luego seleccionar un conjunto de relaciones complementarias que definan inequívocamente el concepto de clase de gesto.

En el caso más simple, dados dos ejemplares de gestos c y c' de longitudes temporales T y T', la definición más simple de equivalencia pide que sean de la misma longitud y todas las posiciones del gesto sean las mismas.

$$c \equiv_n c' \quad \Longleftrightarrow \quad \begin{aligned} &c(t) = c'(t) \\ &t = 0 \dots T,\ T = T' \end{aligned} \tag{4.2}$$

Con esta definición, no hay dos gestos equivalentes a menos que sean exactamente el mismo, y hay tantas clases de equivalencia como trayectorias y longitudes temporales posibles. Si bien es una definición posible, es

bastante restrictiva ya que implica que un gesto debe realizarse siempre con la misma velocidad.

En primer instancia será conveniente definir una equivalencia con **invariancia a la velocidad** con la que se realiza el gesto (y su longitud). Para ello, se necesita una función v que permita convertir entre el tiempo de un gesto y el otro. El conjunto de tales funciones, $\mathbb{V}_{T,T'}$, es:

$$\mathbb{V}_{T,T'} = \left\{ v(t) \;\middle|\; \begin{array}{cc} v(0) = 0, & v(T) = T' \\ v \in \mathbb{C}, & \dfrac{\partial v}{\partial t} \geq 0 \end{array} \right\} \tag{4.3}$$

Entonces, se definen c y c' como equivalentes con invariancia a la velocidad:

$$c \equiv_v c' \iff \begin{array}{l} \exists v \in \mathbb{V}_{T,T'} \\ c(t) = c'(v(t)),\ t = 0 \dots T \end{array} \tag{4.4}$$

Es decir, c y c' recorren los mismos puntos, en el mismo orden, pero v indica la velocidad a la cual c' recorre los puntos en forma relativa a la velocidad de c.

La invariancia a la velocidad es en cierto modo ortogonal al resto de las invariancias. Se podrían definir exigiendo $T = T'$ como condición en todas ellas, o exigiendo la existencia de v, o sea, $c \equiv_v c'$. Dado que en esta tesina se busca reconocer gestos sin importar la velocidad con la que fueron realizados, se utilizará la segunda opción.

Con ella se puede una equivalencia con **invariancia a la traslación**:

$$c \equiv_t c' \iff \begin{array}{l} \exists v \in \mathbb{V}_{T,T'},\ \exists b \in \mathbb{R}^3 : \\ c(t) + b = c'(v(t)) \end{array} \tag{4.5}$$

También se puede definir **invariancia a la escala** del gesto:

$$c \equiv_s c' \iff \begin{array}{l} \exists v \in \mathbb{V}_{T,T'},\ \exists a \in \mathbb{R} \\ ac(t) = c'(v(t)) \end{array} \tag{4.6}$$

Se define la equivalencia **invariante a la posición de comienzo** para los **gestos cerrados**, que son aquellos que, idealmente, comienzan y terminan en la misma posición, es decir $c(0) = c(T)$. Esta equivalencia es útil al querer reconocer, por ejemplo, un gesto correspondiente al dígito arábigo

0, ya que se desea que el algoritmo de reconocimiento detecte dicho gesto sin que importe el lugar por donde el usuario comenzó a realizar el gesto. Esta equivalencia sólo tiene sentido para gestos cerrados ya que en ese caso hay infinitos puntos de comienzo posible, mientras que en un gesto abierto solo hay una posibilidad. Para ello, dados gestos cerrados c y c',

$$c \equiv_c c' \iff \begin{array}{l} \exists v \in \mathbb{V}_{T,T'} \exists \omega \in \mathbb{R} \\ c((t+\omega)\,\%T) = c'(v(t)) \end{array} \tag{4.7}$$

Donde % es el operador módulo. Además, se puede definir una equivalencia con **invariancia a la dirección** hacia donde se realiza la trayectoria.

$$c \equiv_c c' \iff \begin{array}{c} \exists v \in \mathbb{V}_{T,T'} \\ c(t) = c'(v(t')) \\ t' = T - t \end{array} \tag{4.8}$$

Por ejemplo, un gesto circular puede hacerse en dirección horaria o anti-horaria. Hay que tener cuidado ya que con esta invariancia ocurren ciertas equivalencias potencialmente indeseables, ya que, por ejemplo, un gesto donde se mueve la mano de izquierda a derecha es equivalente a uno donde se mueve de derecha a izquierda.

Por ese motivo se dejará esta última equivalencia definida, pero no se incluirá en el modelo final de gesto.

Por último, se busca una equivalencia con **invariancia a la rotación** con respecto a un eje determinado $\mathbf{i} \in \mathbb{R}^3$.

$$c \equiv_{\mathbf{R_i}} c' \iff \begin{array}{l} \exists v \in \mathbb{V}_{T,T'},\ \exists \mathbf{R_i} \in \mathbb{R}^{3\times 3} \\ c(t) = \mathbf{R_i} c'(v(t)) \end{array} \tag{4.9}$$

Donde $\mathbf{R_i}$ es una matriz de rotación alrededor del eje $\mathbf{i}$.

Con estas relaciones, se obtienen distintas clases de equivalencia entre ejemplares de gestos. En esta tesina, se busca una combinación de las equivalencias con invariancia a la velocidad, traslación y escala. Además, el reconocimiento será invariante a la rotación con respecto a al torso de la persona como origen de eje, donde el eje z representa posiciones atrás o adelante de dicho origen, el x a izquierda o derecha, y el y arriba o abajo.

Se ignora la invariancia a la posición de comienzo para gestos cerrados ya que se trabaja en un modelo dependiente del usuario en donde el mismo grabará sus gestos, y dicha invariancia es difícil de incluir en un

clasificador. Como se mencionó anteriormente, no se incluye la invariancia a la dirección ya que en muchos casos puede resultar contraintuitiva, y además puede agregarse muy fácilmente invirtiendo el gesto en la etapa de reconocimiento sin importar las otras características a reconocer.

De esta manera, se obtiene el siguiente modelo:

$$\begin{aligned} & & \exists v \in \mathbb{V}_{T,T'},\ \exists \mathbf{R_i} \in \mathbb{R}^{3\times 3} \\ c \equiv_{m'} c' \quad & \iff & \exists b \in \mathbb{R}^3,\ \exists a \in \mathbb{R} \\ & & c(t) = a(\mathbf{R_i} c'(v(t))) + b \end{aligned} \tag{4.10}$$

Por último, es obvio que un gesto no puede replicarse exactamente, aún con estas equivalencias. Por ende, relajando la condición de igualdad entre posiciones a una de $\epsilon -$ equivalencia:

$$\begin{aligned} & & \|a - b\| < \epsilon \\ a =_\epsilon b \quad & \iff & \text{donde } \epsilon \in \mathbb{R},\ a, b \in \mathbb{R}^3 \\ & & \text{y } \|\cdot\| \quad \text{es una norma en } \mathbb{R}^3 \end{aligned} \tag{4.11}$$

De esta manera, finalmente se obtiene la equivalencia deseada:

$$\begin{aligned} & & \exists v \in \mathbb{V}_{T,T'},\ \exists \mathbf{R_i} \in \mathbb{R}^{3\times 3} \\ c \equiv_m c' \quad & \iff & \exists b \in \mathbb{R}^3,\ \exists a \in \mathbb{R} \\ & & c(t) =_\epsilon a(\mathbf{R_i} c'(v(t))) + b \end{aligned} \tag{4.12}$$

Esta equivalencia implica que las posiciones entre los gestos son similares, luego de una misma rotación, traslación y escala de todo el gesto. Si bien esta es una definición válida, es difícil incorporar la función v en el sistema de reconocimiento de gestos ya que agrega un grado de libertad que no se contrarresta tan fácilmente como el de las otras invariancias, que pueden lograrse con transformaciones más tradicionales de la entrada.

Además, hay un compromiso inherente entre el tamaño de las clases de equivalencia gestuales y diversas propiedades de un sistema de reconocimiento como la dificultad para realizar los gestos, la facilidad para reconocer que cierta trayectoria corresponde a un gesto, y la facilidad para reconocer a qué clase de gesto pertenece cierta trayectoria. A mayor tamaño de una clase, existen más trayectorias que serán identificadas como gestos de esa clase, y por ende se ampliará la variabilidad con la que el usuario puede realizar un gesto. Pero en la práctica eso también hace que, a medida que el tamaño del vocabulario gestual aumenta, sea más difícil decidir si una trayectoria es un gesto o no, y diferenciar una de otra.

4.3.2. Modelo de gestos con parametrización por longitud de arco

La parametrización por longitud de arco obvia totalmente la dimensión temporal del gesto, y modela su posición en función a la porción de la longitud de arco total recorrida, l'. Su definición es similar a la de la parametrización temporal, reemplazando t y T por l' y L, la longitud de arco recorrida y la longitud de arco total del gesto, respectivamente:

$$\begin{aligned} c &: \mathbb{R} \to \mathbb{R}^3 \\ c(l') &= (x, y, z) \qquad l' = 0 \dots L \end{aligned} \tag{4.13}$$

De esta manera, $c(0)$ es la posición de comienzo del gesto, $c(L)$ la posición de fin, y $L = \int_0^L ||\frac{\partial c(l')}{\partial l'}||dl'$

La longitud de arco depende de la escala del gesto, y en el caso de la velocidad resulta complicado comparar gestos de distinta longitud (longitud de arco, en este caso). Por ende, se puede normalizar la longitud de arco de todos los gestos a 1 introduciendo una variable $l = l'/L$, y redefiniendo el gesto como:

$$\begin{aligned} c &: \mathbb{R} \to \mathbb{R}^3 \\ c(l) &= (x, y, z) \qquad l = 0 \dots 1 \end{aligned} \tag{4.14}$$

De esta manera, la longitud correspondiente al fin de gesto, mitad de gesto, etc, es equivalente en los distintos gestos. En este caso es posible realizar dicha normalización ya que la longitud de arco es una función de las posiciones mismas y por ende la definición es equivalente. En el caso temporal, en cambio, dicha solución sería acorde siempre y cuando el gesto se realice a velocidad constante, lo cual es una restricción extra que no es deseable agregar.

Al igual que en el caso temporal, se define una equivalencia con invariancia a la escala y traslación (y a la velocidad, trivialmente, ya que no forma parte de esta definición de gesto).

$$c \equiv_{m'} c' \iff \begin{aligned} &\exists \mathbf{R_i} \in \mathbb{R}^{3\times3},\ \exists b \in \mathbb{R}^3,\ \exists a \in \mathbb{R} \\ &c(l) = a(\mathbf{R_i} c'(l)) + b \end{aligned} \tag{4.15}$$

y nuevamente se agrega cierta tolerancia al error:

$$c \equiv_{m} c' \iff \begin{aligned} &\exists \mathbf{R_i} \in \mathbb{R}^{3\times3},\ \exists b \in \mathbb{R}^3,\ \exists a \in \mathbb{R} \\ &c(l) =_{\epsilon} a(\mathbf{R_i} c'(l)) + b \end{aligned} \tag{4.16}$$

Este es, entonces, el modelo de gestos a reconocer. Se preferirá está parametrización sobre la temporal ya que resulta más simple a la hora de definir las transformaciones del gesto para obtener características que sean invariantes a las propiedades ya mencionadas. De todas formas, se ve que son equivalentes ya que si $s(l)$ y $s'(l)$ denotan el tiempo en el cual los gestos c y c' tienen longitud de arco l, $c(s(l))$ y $c'(v(s'(l)))$ son equivalentes a $c(l)$ y $c'(l)$.

4.4. Resumen

En este capítulo se explora el concepto de gesto desde una perspectiva comunicacional y psicológica como también desde la interacción hombre-máquina y el reconocimiento automático.

Desde el primer punto de vista, se introduce la clasificación de gestos de McNeill, identificándolos como **icónicos**, **metafóricos**, **ilustrativos** y **deícticos**. Dicha consideración resulta de utilidad para decidir sobre un modelo de gestos a reconocer.

Luego, dentro del ámbito del HCI, se puede considerar una clasificación de gestos dependiendo de si son estáticos o dinámicos, con una o varias partes del cuerpo, en 2D o 3D, faciales o corporales, multimodales o unimodales y dependientes o independientes del usuario.

En esta tesina el foco está puesto en gestos corporales 3D, dinámicos y unimodales, dependientes del usuario, y realizados con una sola parte del cuerpo. En términos de la clasificación de McNeill, el modelo de reconocimiento de esta tesina se presta a ser utilizado para identificar gestos deícticos o icónicos.

Habiendo definido de forma general el tipo de gesto a reconocer, se especifica explícitamente desarrollando un modelo matemático de los gestos, en donde los mismos están formados por una trayectoria c en un espacio 3D, y se definen los conceptos de invariancia a la rotación, traslación, escala, velocidad, punto de comienzo y dirección. En base a estos conceptos se deriva la característica para clasificar gestos del capítulo 5.

Finalmente, se define también formalmente el concepto de equivalencia $\equiv_m$ entre gestos, de utilidad para basar las comparaciones entre gestos en los modelos de clasificación.

Parte II

Aplicaciones

5

Bases de datos y características para el reconocimiento de gestos

5.1. Introducción

A partir de este capítulo se presentarán las decisiones tomadas para resolver el problema de reconocimiento de gestos dinámicos. El primer paso consiste en establecer un contexto de prueba que permita comparar el método propuesto en esta tesina con otras soluciones existentes en la literatura. Sin embargo, las bases de gestos encontradas se centran más en el problema de captación del gesto que en su reconocimiento [77, 78, 79]. Por tal motivo, dichas bases suelen estar formadas por gestos muy diferentes entre si facilitando enormemente el proceso de reconocimiento.

En base a lo antes expuesto, para probar los algoritmos se generó una base de datos con gestos de números y letras utilizando un dispositivo Kinect y su SDK, que se denominó Letters and Numbers Hand Gesture (LNHG). En este capítulo se describe la manera en que fue construida dicha base y la manera en que debe ser preprocesada a fin de poder utilizarla como información de entrada a un reconocedor.

A fin de tener un punto de comparación, entre las bases de datos existentes, se seleccionó la definida en [80], que se referenciará como Celebi2013, cuyos gestos fueron reconocidos utilizando el algoritmo Dynamic Time Warping presentado en el artículo citado. Es importante remarcar que existen diferencias importantes entre los experimentos realizados en esta tesina y en el artículo [80], que serán descriptas oportunamente.

5.2. Base de datos de gestos de letras y números arábigos (LNHG)

La Letters and Numbers Hand Gesture Database (LNHG) [1], está formada por 20 ejemplares de cada uno de los 10 dígitos arábigos y las 26 letras del abecedario (sin contar la ñ), obteniendo un conjunto de 720 ejemplares en total, con 36 clases. Los gestos se realizaron con la mano izquierda, todos por la misma persona, con descansos de 10 minutos entre la grabación de cada clase de gesto. Los datos de posición de la mano fueron capturados utilizando el SDK del Kinect. La grabación fue realizada a una tasa de captura de 28fps en promedio.

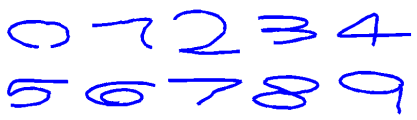


Figura 5.1: Proyección en el eje xy de ejemplares de gestos correspondientes a dígitos.

En la grabación de los distintos ejemplares de cada clase la orientación de la persona con respecto a la cámara fue la misma respecto a rotaciones del eje x y z, pero hubo variedad en la rotación del eje y (con origen en el centro de la persona). Además, los ejemplares fueron grabados comenzando desde diferentes posiciones tanto de la mano como del cuerpo, trazando cada gesto con diferentes tamaños y a distintas velocidades. La captura se realizó indicando el comienzo y fin de cada gesto mediante un pequeño pad numérico usb sostenido en la mano derecha de forma confortable, para lo cual solo era necesario presionar la tecla *Enter*.

Cada ejemplar $\boldsymbol{s_i} \in S$, donde S es la base de datos de gestos, consiste en una secuencia:

$$\boldsymbol{s_i} = \boldsymbol{s_i}[1], \boldsymbol{s_i}[2], \ldots, \boldsymbol{s_i}[n_i], \quad \boldsymbol{s_i}[j] \in \mathbb{R}^3, \quad j = 1 \ldots n_i$$

correspondiente a las posiciones de la mano en un espacio 3D, con etiquetas de tiempo:

$$T_i = t_1, \ldots, t_{n_i}, \qquad t_j \in \mathbb{R}, \qquad 0 = t_1 < t_2 < \ldots < t_{n_i}$$

[1] En https://sites.google.com/site/lnhgdb/ se encuentra una versión inicial de la base de datos que sólo contiene gestos de números arábigos

y etiquetas de clase c_i. Cada muestra $\boldsymbol{s_i}$ puede tener una cantidad distinta de posiciones n_i, dependiendo de la velocidad con la cual se ejecutó el gesto, su tamaño y la tasa de captura.

Entonces, cada gesto $\boldsymbol{s_i}$ es una versión discreta del modelo de gestos continuos descripto en el capítulo 4.

5.3. El Kinect y su SDK

5.3.1. Kinect

El Kinect es un dispositivo desarrollado por Microsoft que consiste en una cámara web junto con un sensor de profundidad. La cámara web es de características relativamente normales, capturando imágenes a una tasa máxima de 30fps y a una resolución de 640 x 480. El dispositivo posee además un acelerómetro para determinar la orientación del mismo respecto del suelo de forma automática.

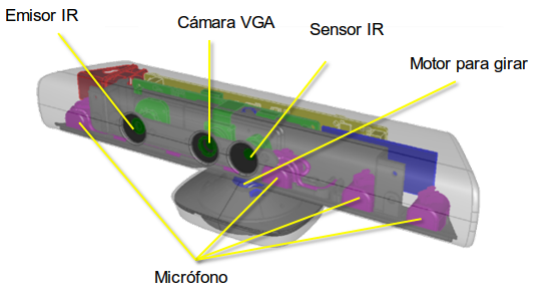


Figura 5.2: Kinect

El sensor de profundidad captura a la misma frecuencia y resolución que la cámara, y está compuesto por un láser proyector infrarrojo y un sensor CMOS; el primero proyecta continuamente varios haces de luz cuya respuesta al interactuar con el ambiente es medida por el segundo, de manera que se pueda detectar la distancia entre el dispositivo y los objetos de la sala. Esto permite distinguir objetos con mayor facilidad ya que una imagen de profundidad contiene información valiosa para ubicar objetos 3D con una imagen 2D. Esto facilita el proceso de segmentación ya que la imagen de profundidad es una buena aproximación a la profundidad o posición en el eje z de los objetos. Además, al utilizar un láser infrarrojo las condiciones de luz ambiental no afectan tanto este proceso.

Figura 5.3: Imagen capturada con el sensor de profundidad del Kinect

Acompaña al dispositivo un SDK propietario de detección de personas y las partes de sus cuerpos en tiempo real. Este SDK es desarrollado por la misma compañía que desarrolla el dispositivo, y fue usado en esta tesina sin cambios. A continuación, se describirá el funcionamiento de ambos para que se puedan comprender sus funcionalidades y limitaciones.

El SDK reconoce hasta 4 personas al mismo tiempo y monitorea a una tasa máxima de 30 fps las posiciones de las 20 articulaciones más importantes del cuerpo (solo las de la parte superior del cuerpo en modo sentado) y los ángulos entre ellas en un espacio virtual tridimensional enfrente de la cámara, y relativo a la misma.

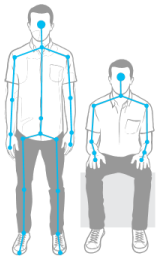


Figura 5.4: Esqueletos detectados en modo normal y modo sentado

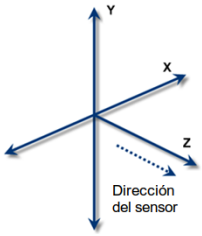


Figura 5.5: Ejes del kinect. El centro de coordenadas corresponde a la posición de la cámara

Para un correcto funcionamiento del reconocimiento, el usuario debe posicionarse a una distancia de la cámara de entre 0.8m y 4m en el modo normal, y entre 0.4m y 3m en el modo de cercanía.

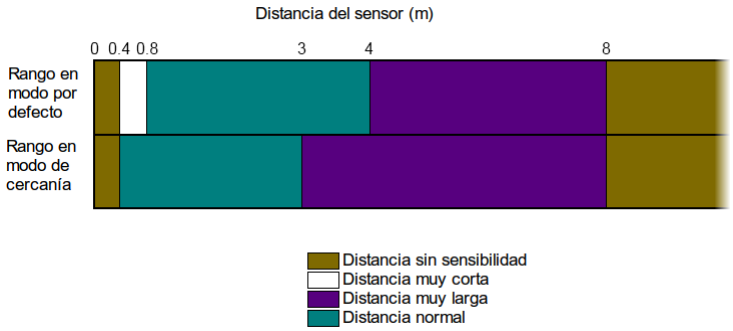


Figura 5.6: Rango del sensor

El ángulo de visión horizontal es de 57 grados y el vertical de 43 grados. Cuando la cámara funciona en el modo normal, a la distancia mínima de 0.8m, la cámara captura 87cm de forma horizontal (eje x) y 66 cm de forma vertical (eje y); a medida que la distancia a la cámara crece, aumenta también el área de captura.

5.3.2. SDK y Algoritmo de tracking del cuerpo

El reconocimiento del SDK del Kinect está orientado a movimientos «gruesos»; por ejemplo, detecta los movimientos de la mano, no de los dedos. Puede distinguir distintas personas y sus movimientos, aun cuando estén parcialmente ocultos, ya que extrapola la posición de las partes del cuerpo ocultas a partir de las visibles.

El núcleo del algoritmo de reconocimiento del Kinect trabaja sobre imágenes individuales tomadas por el sensor de profundidad, que son segmentadas etiquetando partes del cuerpo de forma probabilística; las partes del cuerpo a etiquetar son aquellas que se encuentren espacialmente cerca de las articulaciones que se quieren reconocer. Reproyectando las partes inferidas en el espacio 3D virtual, se localizan los modos espaciales de la distribución de probabilidad de cada parte del cuerpo, y así se generan varias propuestas para las ubicaciones en 3D de cada una de las articulaciones del cuerpo, cada una con cierto puntaje de confianza. La segmentación en partes del cuerpo se realiza como una clasificación por pixel para evitar la explosión combinatoria que implicaría una búsqueda sobre las distintas propuestas de posiciones de articulaciones del cuerpo; esta debilidad del método se balancea con la gran cantidad de imágenes de profundidad de entrenamiento utilizadas.

Para lograr esto, los autores del SDK en una primera etapa crearon una base de datos con imágenes de profundidad reales de personas realizando

distintos movimientos en condiciones ambientales muy diferentes, en las cuales las partes del cuerpo se etiquetaron a mano. En base a eso, generaron un modelo de movimiento humano en 3 dimensiones, mediante el cual luego generaron una cantidad enorme de datos sintéticos, con imágenes de profundidad sintéticas de humanos de distintas formas y tamaños en poses muy variadas. Entrenaron un conjunto de árboles de decisión aleatorizados (**Random Forest**) [81] que evita el sobreentrenamiento y obtiene invariancia a la traslación 3D gracias a esta gigantesca cantidad de datos. Finalmente, utilizando el algoritmo **mean-shift** [82], se infieren los modos espaciales de las distribuciones por pixel de las cuales se extraen distintas propuestas de las posiciones 3D de las articulaciones.

En todo este proceso, no se utiliza ninguna información temporal, ni datos extraídos de imágenes anteriores, lo cual lo hace más robusto y permite una re-inicialización rápida cuando una persona sale y entra al campo de visión de la cámara, o es temporalmente obstruida por otra persona u objeto [11].

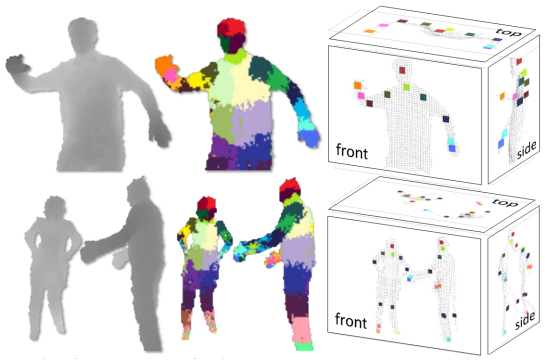


Figura 5.7: Imagen de profundidad → Partes del cuerpo → Modelos 3D propuestos

El SDK se puede utilizar con un mecanismo de polling o mediante callbacks que se realizan a la tasa máxima de 30fps mencionada anteriormente. En ambos casos, en cada poll o callback la información que se provee es la cantidad de esqueletos (personas) detectados, con un identificador que se mantiene en el tiempo, y para cada uno la posición de 20 articulaciones del cuerpo, donde para cada una se indica si esta posición ha sido detectada normalmente, ha sido inferida a través de las posiciones de otras articulaciones, o si no ha sido posible detectarla. Adicionalmente, para cada esqueleto se estima la ecuación $Ax + By + Cz + D = 0$ que describe el plano definido implícitamente por el suelo respecto a la cámara,

normalizada de manera tal que D representa la distancia entre el suelo y la cámara.

5.4. Preprocesamiento

En la etapa de preprocesamiento, las primeras y últimas tres posiciones de cada muestra se descartan porque generalmente contienen información indeseada introducida por una segmentación incorrecta del gesto y se descartan también los frames en donde el SDK no ha detectado todas las articulaciones relevantes. Luego los ejemplares se rotan, suavizan, y re-muestrean.

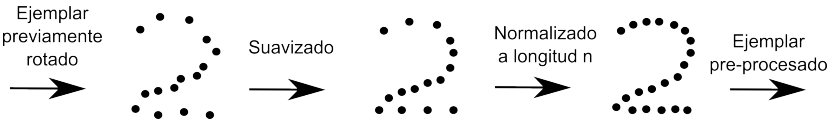


Figura 5.8: Suavizado y re-muestreo de un ejemplar previamente rotado.

5.4.1. Rotación

La rotación del usuario respecto a la cámara introduce cambios significativos en las posiciones de los puntos que determinan el gesto. Esto, si bien no modifica el proceso de captación, dificultan la etapa reconocimiento. Para evitar este inconveniente, se rota el usuario a una dirección canónica en donde la dirección a donde apunta es paralela (y opuesta) a la dirección en donde apunta la cámara. Esta rotación es solamente del plano xz, o sea, con el eje y como eje de la rotación, debido a que los otros tipos de rotaciones (con el usuario inclinado hacia los costados o hacia adelante/atrás) son poco comunes.

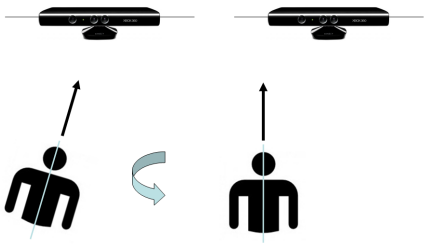


Figura 5.9: Se rota al usuario para que la dirección a la que apunta sea la opuesta a la dirección normal a la cámara, en el plano xz

A estos efectos, estableciendo la posición del centro de los hombros como origen de coordenadas, se calculan los vectores que conectan el centro de los hombros con los hombros, y se promedian en un vector $\boldsymbol{h}$ para aproximar el grado de rotación del usuario. Luego, se calcula la matriz de rotación $\boldsymbol{R}$ para llevar a dicho vector $\boldsymbol{h}$ al vector $(1, 0, 0)$, que representaría la rotación canónica del usuario. Finalmente, se aplica la matriz de rotación $\boldsymbol{R}$ a todas las posiciones del ejemplar, obteniendo una posición canónica para los mismos.

5.4.2. Suavizado

Las muestras fueron se suavizan individualmente utilizando la técnica de la media móvil aritmética con una ventana de tamaño w para quitar la información de alta frecuencia de la señal, ya que las características elegidas están basadas en la dirección entre posiciones consecutivas y pequeñas fluctuaciones en la dirección dan una información muy local como para caracterizar la forma global del gesto.

El tamaño w se mide como la distancia desde el punto a suavizar a los extremos de la ventana, de manera que en realidad se utilizan $2 * w + 1$ puntos en la ventana. Por ejemplo, si $w = 3$, entonces se utiliza una ventana de $2 * 3 + 1 = 7$ puntos para suavizar, centrada en el cuarto punto.

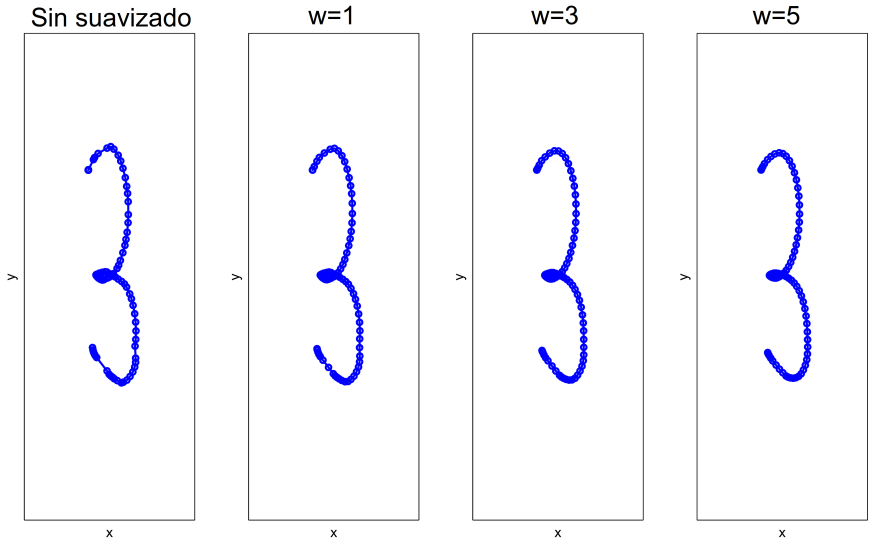


Figura 5.10: Proyección en el plano xy de un ejemplar del gesto 3, con distintos niveles de suavizado

5.4.3. Re-muestreo

Como se explicó anteriormente, los ejemplares correspondientes a los gestos capturados por el Kinect pueden tener distinta longitud. Si bien no es un inconveniente para el método propuesto en esta tesina, la mayoría de los métodos con los que se compara su performance requieren que todos los ejemplares posean la misma longitud, es decir $n = n_i, \quad i = 1, \ldots, |S|$. No solo eso, sino que debería mantenerse una cierta correspondencia semántica entre cada posición del gesto, de manera que, por ejemplo, las primeras posiciones representen la primera parte del gesto, y así sucesivamente; es decir, que no haya saltos en la representación del gesto respecto a las posiciones, que el muestreo sea uniforme. Además, es deseable obtener características invariantes a la velocidad.

Para lograr ambas cosas, cada muestra debe ser convertida en secuencia de longitud constante n utilizando interpolación cúbica con una parametrización de longitud de arco.

La parametrización de longitud de arco de cada muestra $\boldsymbol{s}_i$ de longitud original n_i da la posición de la mano $\boldsymbol{s}_i[j]$ en el espacio 3D en función de la longitud de arco recorrida desde la primera posición del gesto, $\boldsymbol{s}_i[1]$ hasta la posición $\boldsymbol{s}_i[j]$. Entonces, por cada posición $\boldsymbol{s}_i[j]$ se calcula la longitud de arco desde la primera posición, l_j:

$$l_j = \sum_{k=2}^{j} ||\boldsymbol{s}_i[k] - \boldsymbol{s}_i[k-1]||$$

donde $||\cdot||$ es la norma Euclídea. El re-muestreo se realiza en n longitudes de control distribuidos de manera uniforme a través de la longitud de arco total $L = l_q$, dada por:

$$k_j = \frac{j-1}{n-1} * L, \qquad j = 1 \ldots n$$

Por ejemplo, si se tiene un gesto con longitud de arco $L = 20$ y hay 5 longitudes de control, las mismas serán $k_1 = 0$, $k_2 = 5$, $k_3 = 10$, $k_4 = 15$, $k_5 = 20$. El proceso de re-muestreo estima las posiciones correspondientes a estas longitudes de control. Es importante notar que al estar utilizando la parametrización por longitud de arco del gesto, estas longitudes de control k_j son las longitudes de arco desde el comienzo del gesto hasta el punto.

Mediante el proceso de re-muestro, se obtiene un vector de posiciones estimadas $\boldsymbol{r}_i$ de longitud n, donde cada elemento del vector $\boldsymbol{r}_i[j]$ se estima mediante interpolación cúbica, y está dado por la fórmula:

$$\boldsymbol{r}_i[j] = \begin{cases} \boldsymbol{s}_i[1] & \text{si } j = 1 \\ \boldsymbol{s}_i[n_i] & \text{si } j = n \\ cubic(k_j, near_4(k_j)) & \text{si } j = 2 \dots n-1 \end{cases}$$

Esta fórmula, para $j = 1$ y $j = n$, indica que la primera y la última posición del gesto, respectivamente, se mantienen tal cual estaban en el ejemplar original.

Para $j = 2 \dots n-1$, indica que $\boldsymbol{r}_i[j]$ es la posición que se obtiene mediante interpolación cúbica en la longitud de control k_j, utilizando las 4 posiciones cuyas longitudes se conocen y son las más cercanas a k_j. Entonces, la función $near_4(k_j)$ retorna las longitudes asociadas a las 4 posiciones conocidas más cercanas a k_j, en orden.

La función $cubic(k_j, (d_1, d_2, d_3, d_4))$ realiza una interpolación cúbica en la longitud de control k_j, utilizando las distancias (d_1, d_2, d_3, d_4) cuyas posiciones $(\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4)$ se conocen.

La interpolación cúbica se realiza estimando los coeficientes de un polinomio p de grado 3, en base a las 4 longitudes (y posiciones conocidas asociadas) más cercanas a k_j. Luego, simplemente se evalúa p en el punto k_j, para obtener $\boldsymbol{r}_i[j]$.

Los coeficientes del polinomio b_i se estiman mediante la fórmula:

$$b_i = \prod_{j=1, j\neq i}^{4} \frac{d - d_j}{d_i - d_j}$$

Estos coeficientes sirven para formar el denominado polinomio de Lagrange:

$$p(d) = \sum_{i=1}^{4} \boldsymbol{y}_i b_i$$

El polinomio de Lagrange tiene la particularidad de que para los pares (d_k, y_k) utilizados para construirlo $p(d_k) = \boldsymbol{y}_k$, debido a que si $d = d_k$, se tiene $b_k = 1$ y $b_i = 0$, $i \neq k$.

Para ver por qué esto es así, es útil considerar el valor de la productoria b_i. Si $d = d_k$, cuando el índice de la sumatoria es k, $b_k = 1$, ya que:

$$b_k = \prod_{j=1, j\neq k}^{4} \frac{d_k - d_j}{d_k - d_j} = \prod_{j=1, j\neq k}^{4} 1 = 1$$

Por otro lado, si $d = d_k$, cuando el índice de la sumatoria es $i \neq k$, $b_i = 0$, ya que algún j tomará el valor k y como $d_k - d_j = 0$, por ende:

$$b_i = \prod_{j=1, j\neq i}^{4} \frac{d_k - d_j}{d_i - d_j} = 0,\ i \neq k$$

Entonces, al evaluar $p(d_k)$, el término $b_k = 1$, y el resto de los términos b_i son 0, por ende de la sumatoria solamente sobrevive el término k y entonces $p(d_k) = \boldsymbol{y}_k$.

La interpolación sólo es buena para las longitudes de control cercanas a las utilizadas para estimar los coeficientes del polinomio. Por ese motivo es que se calcula un polinomio distinto para cada longitud de control k_j para el cual se quiere estimar la posición correspondiente, utilizando las posiciones de las longitudes más cercanas conocidas.

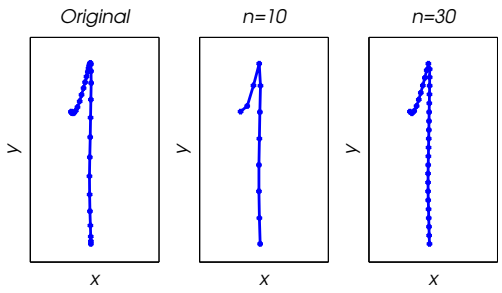


Figura 5.11: Proyección en el plano xy de un ejemplar del gesto 1, antes y después del re-muestreo con $n = 10, 30$. La mayor densidad de posiciones muestreadas, representadas con puntos, en la parte superior en relación con la parte inferior del original fue causada por una diferencia en la velocidad utilizada para realizar el gesto. Dicha diferencia fue compensada, con distintos grados de detalle de acuerdo al valor de n.

Este proceso da como resultado, para cada ejemplar $\boldsymbol{s}_i$, un vector $\boldsymbol{r}_i$ de longitud n donde se mantienen las posiciones de principio y fin del gesto, y hay un muestreo uniforme a través de la longitud de arco del gesto real que realizó el usuario.

5.5. Características

A partir de la secuencia de puntos rotada, suavizada y re-muestreada $\boldsymbol{r}_i$ se calcula la característica que se utilizará para reconocer los gestos. La

misma es el denominado **vector de primeras diferencias**; este vector contiene las direcciones entre cada par de posiciones consecutivas del gesto. Por ejemplo, el elemento 1 del vector de primeras diferencias contiene la dirección para llegar desde la posición 1 a la posición 2 del gesto.

Las direcciones de este vector a su vez se escalan a norma 1 para remover toda la información de velocidad y escala contenida en la secuencia de posiciones originales.

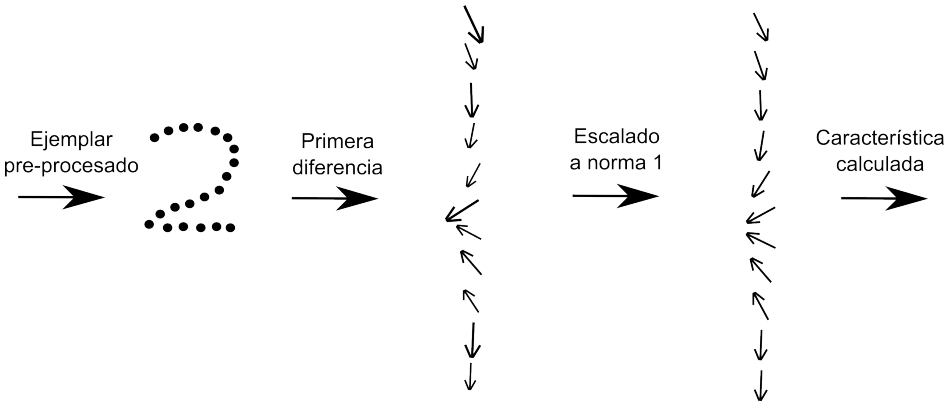


Figura 5.12: Cálculo de características a partir de un ejemplar de gesto normalizado.

El **vector de primeras diferencias normalizado** del ejemplar i, $\boldsymbol{x}_i$, está dado entonces por:

$$\boldsymbol{x}_i[j] = \frac{\boldsymbol{r}_i[j+1] - \boldsymbol{r}_i[j]}{||\boldsymbol{r}_i[j+1] - \boldsymbol{r}_i[j]||}, \qquad j = 1 \ldots n-1, \qquad \boldsymbol{x}_i[j] \in \mathbb{R}^3$$

Este vector será la característica para el gesto, y cada uno de sus elementos representa la dirección relativa entre las posiciones consecutivas del mismo. A la función característica que lo calcula se la denominará ϕ_d. El vector de primeras diferencias, sin normalizar, da una representación equivalente invariante a la traslación. Al normalizar, se remueve toda la información de velocidad y escala contenida en la norma de cada vector de dirección, tornando la característica invariante a la velocidad y escala.

Es importante notar que sin el re-muestreo esta normalización dejaría todavía una cantidad considerable de información de velocidad en la señal, debido a que la cantidad de puntos de muestreo en los segmentos (interpretando la palabra en el sentido de la longitud de arco) donde el usuario realizar el gesto a altas velocidades es más bajo que en aquellos

segmentos en donde la mano se mueve más lentamente. Es decir, a mayor velocidad, menor densidad de puntos de muestreo y viceversa.

Por último, la característica es en cierto modo equivalente a la función *ángulo-tangente* [83] para describir formas de objetos, y por eso desde esa equivalencia también se puede argumentar que cumple las propiedades de invariancia a la escala, traslación y velocidad.

5.5.1. Versiones discretas de las propiedades de los gestos

Previamente se definió un modelo de gestos donde un gesto es una trayectoria continua y hay una relación de equivalencia entre gestos con propiedades deseables. Como los ejemplares son una secuencia de posiciones discretas, se necesita una versión discreta de dichas propiedades, aplicables a la representación que se utiliza en los algoritmos de clasificación. La principal diferencia entre ambas es la función de correspondencia entre gestos, ya que de todas maneras se busca una matriz de rotación $\mathbf{R_i}$, y constantes de traslación y escala a y c. Recordando la definición para el caso continuo:

$$c \equiv_m c' \iff \begin{array}{l} \exists \mathbf{R_i} \in \mathbb{R}^{3\times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\ c(l) =_\epsilon a(\mathbf{R_i} c'(l)) + b \end{array} \tag{5.1}$$

Se busca aproximar dicha definición en el caso concreto. Para ello, el re-muestreo de un gesto da una representación de sus posiciones $\boldsymbol{r}$ que es uniforme en la longitud total de arco del gesto, y que dado un n lo suficientemente grande, aproxima con error arbitrario a la representación continua. Entonces, se define la equivalencia en el caso discreto simplemente como:

$$\boldsymbol{s} \equiv_m \boldsymbol{s}' \iff \begin{array}{l} \exists \mathbf{R_i} \in \mathbb{R}^{3\times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\ \boldsymbol{s}[j] =_\epsilon a(\mathbf{R_i} \boldsymbol{s}'[j]) + b,\ j = 1, \ldots, n \end{array} \tag{5.2}$$

Como el espacio de búsqueda de dicha transformación es muy grande, se debe encontrar una equivalencia entre características dadas por $\phi :: \mathcal{P} \mapsto \mathcal{F}$, donde $\mathcal{F}$ es el espacio de características, de manera que si $\phi(\boldsymbol{s}) =_\epsilon \phi(\boldsymbol{s}') \rightarrow \boldsymbol{x} \equiv_m \boldsymbol{x}'$. Es decir, equivalencia en el espacio de las características implica equivalencia $\equiv_m$.

En esta tesina, se llamará ϕ_d a la función característica que dado un ejemplar de gesto realiza la normalización, incluyendo rotación, suavizado

y re-muestreo, y el cálculo de la primera diferencia con norma 1; es decir, $\boldsymbol{x}_i = \phi_d(\boldsymbol{s}_i)$.

Debido a la corrección de la rotación, en donde siempre se ubica el cuerpo en cierto sentido, ϕ_d es invariante a la rotación. No es invariante a otras rotaciones, ya sean en otras direcciones o de las manos en sí.

Gracias al re-muestreo y la parametrización por longitud de arco, ϕ_d es invariante a la velocidad con la cual fue realizado el gesto, y permite que cada $\boldsymbol{x}[j]$ represente siempre la misma parte del gesto, aunque sea de forma aproximada.

El cálculo de la primera diferencia provee invariancia a la traslación, ya que no importan las posiciones absolutas sino las direcciones entre ellas. Llevar dichas diferencias a norma 1 otorga invariancia a la escala, ya que así no influye la distancia absoluta entre posiciones contiguas, sino su dirección solamente.

El vector característico $\boldsymbol{x}$ es invariante, como se deseaba, a la rotación, traslación, velocidad y escala.

Por último, se define la propiedad de invariancia al punto de comienzo para gestos cerrados discretos, para algún clasificador que pueda implementarla.

En el caso discreto, los gestos cerrados son aquellos para los cuales $\boldsymbol{s}[1] = \boldsymbol{s}[n]$. Entonces, una característica ϕ es invariante a la posición de comienzo si:

$$\phi(\boldsymbol{s}) = \phi(shift(\boldsymbol{s}, k)), \quad k = 1..n-1$$

donde

$$shift(\boldsymbol{x}, k) = (x_{(k)\,\%(n+1)}, x_{(k+1)\,\%(n+1)}, \ldots, x_{(n+k)\,\%(n+1)})$$

y % es el operador *modulo*

La característica ϕ_d que se desarrolló en la sección anterior no es invariante al punto de comienzo, debido a que claramente un *shift* de los puntos del gesto causa un *shift* en la característica calculada. El clasificador CNC que se describe en el capítulo 7 *si* provee invariancia al punto de comienzo realizando una transformación posterior a las características obtenidas con ϕ_d, a costa de perder información sobre la *secuencia* de las direcciones.

En conclusión, se ha definido una característica ϕ para una secuencia de puntos 3D $\boldsymbol{s}$, y establecido que es invariante a la rotación, traslación, velocidad y escala (o sea $\boldsymbol{x} =_\epsilon \boldsymbol{x} \rightarrow \boldsymbol{s} \equiv_m \boldsymbol{s}'$) propiedades de utilidad para el reconocimiento de gestos.

5.6. Base de datos de gestos Celebi2013

Se realizaron experimentos con la base de datos **Celebi2013** [80], también generada utilizando el Kinect [2]. En el trabajo para el cual se creó dicha base de datos, desarrollaron una versión modificada de Dynamic Type Warping que considera el movimiento de las dos manos y el de otras seis partes del cuerpo: mano izquierda, mano derecha, muñeca izquierda, muñeca derecha, codo izquierdo, codo derecho. El algoritmo DTW modificado determina automáticamente la relevancia de cada parte del cuerpo para un gesto determinado. Entonces, dicho reconocimiento tiene en cuenta varias partes del cuerpo, y no una sola.

La base datos consiste de 8 clases de gestos:

- Empuje hacia arriba con la mano izquierda
- Empuje hacia arriba con la mano derecha
- Empuje hacia abajo con la mano izquierda
- Empuje hacia abajo con la mano derecha
- Movimiento de la mano desde la parte izquierda del cuerpo hacia la derecha con la mano izquierda
- Movimiento de la mano desde la parte derecha del cuerpo hacia la izquierda con la mano derecha
- Saludo con la mano izquierda
- Saludo con la mano derecha

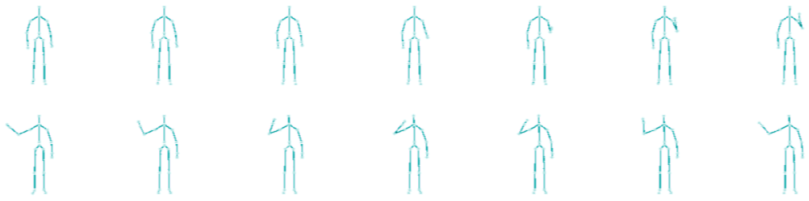


Figura 5.13: Gesto de subir la mano izquierda (arriba) y de saludo (abajo).

Para cada clase de gesto, los autores obtuvieron 8 ejemplares realizados por un usuario experto y 20 con usuarios inexpertos, dando un total de 224

[2]Visitar http://datascience.sehir.edu.tr/visapp2013/ para más información.

ejemplares de gestos. Los autores reportan un porcentaje de clasificación del 60 % para el DTW clásico y de un 97.5 % para el DTW modificado que proponen. Los resultados para los experimentos con los clasificadores desarrollados en esta tesina se presentan en el siguiente capítulo.

5.7. Resumen

En este capítulo se presentó la base de datos LNHG generada para realizar los experimentos con los clasificadores, y los detalles de cómo fue generada y el funcionamiento del dispositivo utilizado para grabar los gestos. Se describe también la base de datos Celebi2013, definida en [80], para comparar el CNC contra el método de reconocimiento descripto en dicho artículo.

Se detalló como, para cada ejemplar de gesto grabado, se realiza un preprocesamiento para mejorar la señal del gesto, y luego se calcula una característica ϕ_d a partir del mismo para facilitar su reconocimiento. La característica calculada, en base a las posiciones preprocesadas, es una lista de vectores de dirección (dx, dy, dz) entre cada posición consecutiva del gesto, donde cada vector de dirección está normalizado para que tenga norma euclídea 1.

Estos cálculos aseguran que la representación elegida para el gesto sea invariante a la rotación del usuario, a la escala, a la velocidad de realización del gesto, y a la posición en donde se realiza. Por último, se justificó la adecuación de dicha característica al modelo de gestos desarrollado en el capítulo 4.

En base a esta característica se definen los clasificadores a comparar, incluyendo el CNC, en el capítulo 6, y por ende la misma se utiliza en todos los experimentos de clasificación presentados en el capítulo 7.

6

El Clasificador Neuronal Competitivo (CNC) y otros modelos para comparar

6.1. Introducción

Este capítulo está dedicado a la construcción de clasificadores efectivos, aplicables al reconocimiento de gestos. Para ello, se hará uso de las estrategias presentadas en la parte I de este documento. En todos los casos, el gesto a reconocer será representado como se describió en el capítulo 5.

En primer lugar, se describirá el Clasificador Neuronal Competitivo (CNC) el cual constituye el aporte central de esta tesina. Se trata de un nuevo tipo de clasificador que combina un conjunto de redes neuronales competitivas con un sistema de votación para llevar a cabo el reconocimiento. Una de sus características más importantes es su capacidad de adaptarse correctamente haciendo uso de un conjunto de ejemplares de entrenamiento pequeño

Luego, se aplican los modelos SVM y Redes Neuronales Feedforward al problema de reconocimiento de gestos. Finalmente, se desarrolla también un método de reconocimiento basado en templates para resolver el mismo problema. Estas soluciones alternativas son habitualmente utilizadas en la literatura [84, 85, 86] y sirven como punto de comparación con el método CNC presentado en esta tesina.

6.2. Clasificador Neuronal Competitivo

El Clasificador Neuronal Competitivo (CNC), el aporte central de esta tesina, es una nueva estrategia de reconocimiento de gestos basada en redes neuronales competitivas, con una técnica de **bagging** que combina varios clasificadores débiles para construir uno más fuerte.

El CNC está compuesto por *r* redes neuronales, entrenadas en paralelo, que son clasificadores por sí mismas. Para clasificar un ejemplar, el CNC hace que cada red lo clasifique, y luego combina sus resultados dando lugar así a un clasificador más robusto.

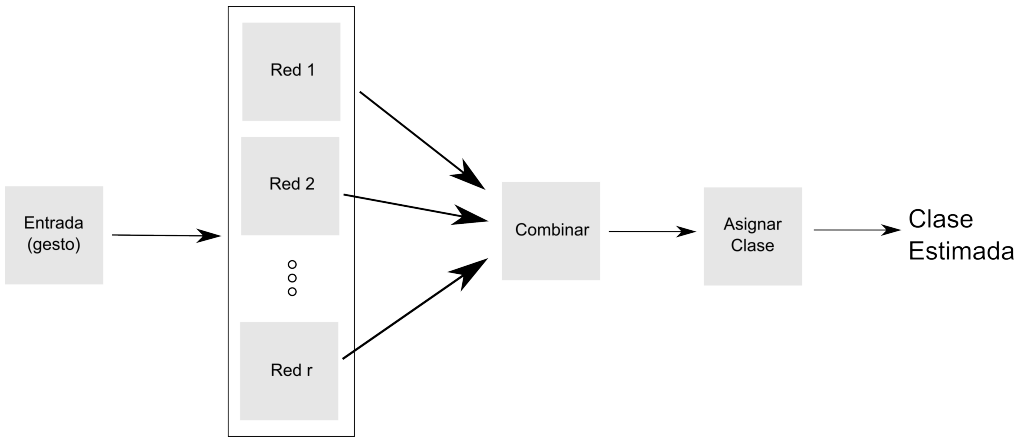


Figura 6.1: Estructura general del clasificador CNC

Cada red está formada por tres capas: una de entrada, una oculta, y una de salida. La capa 1 sólo se encarga de propagar la información de entrada, correspondiente al gesto a reconocer, hacia cada una de las neuronas de la capa oculta. Esta segunda capa genera una nueva codificación. La capa 2, la oculta, genera una nueva codificación de los ejemplares para mejorar la habilidad de reconocimiento, mediante el algoritmo de entrenamiento competitivo visto en el capítulo 3. Dicha codificación está basada solamente en las direcciones del ejemplar y no en el orden en el cual se encuentran. La capa 3, de salida, está formada por tantas neuronas competitivas como ejemplares de entrenamiento haya. Cada una de ellas tiene por objetivo medir la distancia entre el gesto ingresado a la entrada y el ejemplar de entrenamiento correspondiente.

De esta forma, la red neuronal de tres capas convierte el gesto ingresado en un vector de distancias que permite medir la distancia entre dicho gesto y cada uno de los gesto de entrenamiento. La siguiente figura ilustra el proceso de reconocimiento.

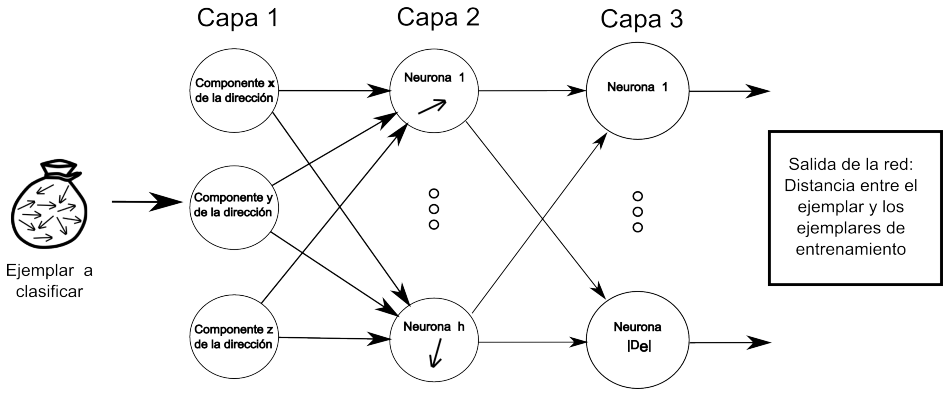


Figura 6.2: Estructura general y funcionamiento de cada red del clasificador CNC

Este proceso es realizado por cada una de las r redes que forman el clasificador CNC. Luego, se promedian los vectores de distancias generados por cada red obteniendo un puntaje de distancia que relaciona el gesto ingresado con cada ejemplar de entrenamiento. Como se ha guardado la clase de cada ejemplar de entrenamiento, se puede asignar al nuevo gesto la clase del ejemplar de entrenamiento al que menos distancia tenga.

A continuación se describe en detalle el funcionamiento de cada red, con sus capas de codificación y de salida. Luego, se dan los detalles de la estrategia de combinación de los resultados de las redes, con lo cual se concluye la descripción del clasificador.

Finalmente, se desarrolla la motivación para utilizar varias redes en la sección *Bagging* y la capacidad del CNC de clasificar gestos aunque los ejemplares no tengan todos la misma longitud o cantidad de direcciones.

6.2.1. Adaptación y funcionamiento de cada red

Como se explicó anteriormente, el clasificador CNC está formado por r redes neuronales que operan de forma totalmente independiente (ver figura 6.1). Cada una de ellas tiene por objetivo medir el parecido que existe entre el gesto ingresado y los ejemplos de entrenamiento utilizados durante el proceso de adaptación. En esta sección se describe la arquitectura de cada red, la forma de determinar los pesos de sus arcos y su funcionamiento.

A la hora de entrenar la red o clasificar un gesto, se utiliza la característica $\boldsymbol{x}$, calculada mediante el método descripto en el capítulo 5. Esta característica $\boldsymbol{x}$ fue definida como una secuencia de n direcciones d dimensionales ($d = 3$ dimensiones, para el problema de los gestos 3D).

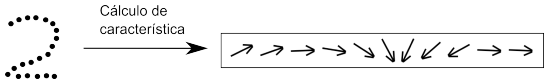


Figura 6.3: Característica $\boldsymbol{x}$ utilizada como entrada a cada red.

El clasificador no utiliza información de la secuencia de direcciones del gesto, sino que solamente se basa en cuáles son las direcciones que forman parte del mismo. Es decir, el clasificador toma el vector de direcciones de un ejemplar $\boldsymbol{x}$, y genera un conjunto de direcciones sin orden.

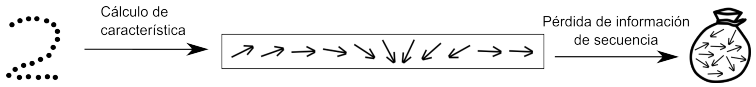


Figura 6.4: Conjunto de direcciones de la característica $\boldsymbol{x}$.

Tal como se observa en la figura 6.2 cada red está formada por tres capas. El proceso de entrenamiento se realiza en dos etapas: primero se determinan los pesos de los arcos que unen la capa de entrada con la capa oculta y luego, una vez finalizada esta etapa, se establecen los pesos de los arcos que unen la capa oculta con la de salida.

La capa oculta es una CPN formada por h neuronas competitivas y se encarga de agrupar los vectores de dirección que componen el conjunto de ejemplos de entrenamiento (la variable h es un parámetro del algoritmo y su ajuste se describe en detalle más adelante). La finalidad de este agrupamiento es reducir la diversidad de vectores de dirección presentes en los gestos de entrada. La figura 6.5 ilustra la manera de obtener la información que ingresará a la red al momento de realizar esta primera etapa del entrenamiento. Nótese que se utilizarán los vectores de dirección sin tener en cuenta el gesto al cual corresponden.

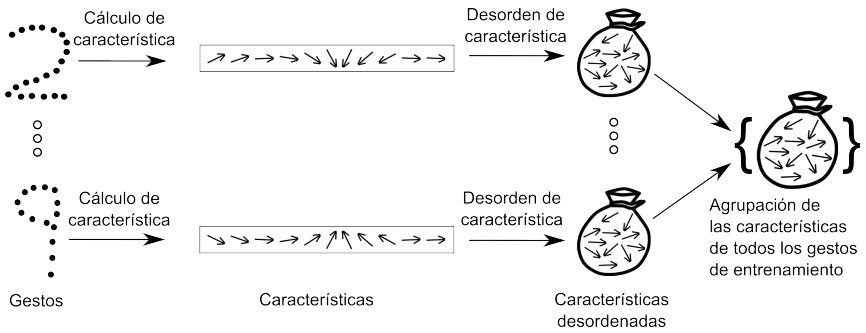


Figura 6.5: Conjunto de todas las direcciones de las características $\boldsymbol{x}$, de todos los gestos de entrenamiento.

Por lo tanto, la primera capa se encuentra formada por d neuronas, siendo $d = 3$ en el caso del reconocimiento de gestos, mientras que la capa oculta posee h neuronas competitivas. Al finalizar el entrenamiento, los pesos de los d arcos que llegan a cada una de las h neuronas competitivas permitirán identificar las h direcciones canónicas.

6.2.1.1. Capa de codificación

Como se mencionó anteriormente, la capa 2 de cada red se entrena usando como entrada todas las direcciones de *todos* los ejemplares de entrenamiento, y el algoritmo de entrenamiento de la capa 2 es el CPN desarrollado en el capítulo 3.

De esta manera, se genera un diccionario de direcciones canónicas o más ocurrentes $\boldsymbol{x}_c$, con entradas $\boldsymbol{x}_c[k] \in \mathbb{R}^3, k = 1, \ldots, h$. Cada dirección $\boldsymbol{x}_c[k]$ representa el centroide de una neurona de esta capa.

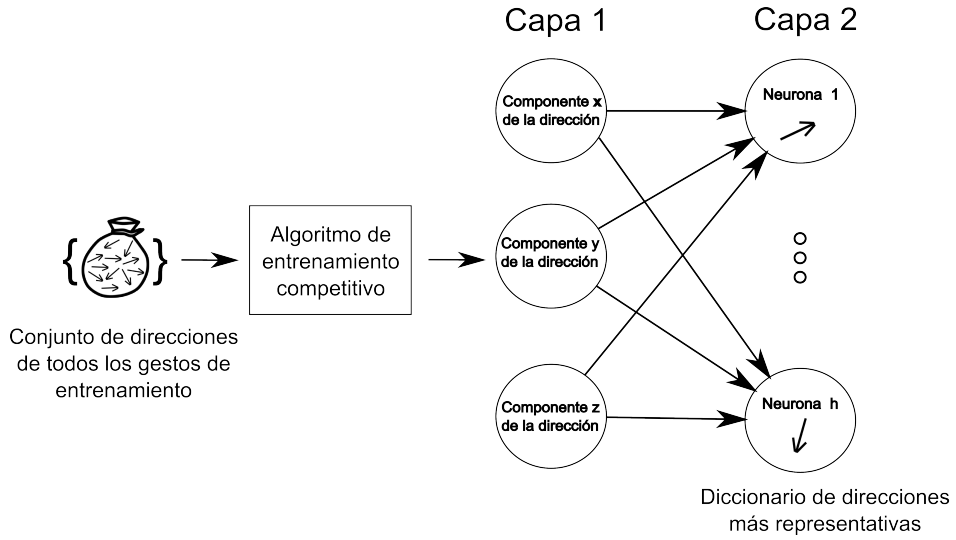


Figura 6.6: Entrenamiento de la capa 2 de una red del CNC.

El parámetro h, mencionado anteriormente, indica cuantas neuronas ocultas o direcciones canónicas se consideran. El mismo se elige de forma manual en función del problema a resolver. Cuanto más grande es h, hay más direcciones en el diccionario y por ende se puede ser más preciso, pero se utiliza más memoria y tiempo de cpu para el entrenamiento y la clasificación. Al utilizar valores de h muy altos se corre además el riesgo de overfitting, debido a que la red podría simplemente recordar todas las direcciones de todos los ejemplares en lugar capturar sólo las esenciales

o canónicas. Por otro lado, si el valor de h es muy pequeño, la red sólo considera unas pocas direcciones, y puede que no le sea posible distinguir con efectividad ciertos tipos de gestos.

La capa 2 adquiere así la habilidad de codificar ejemplares. Dada una dirección de un ejemplar cualquiera, se puede encontrar entonces la dirección canónica que más se le parece. Dada una secuencia de direcciones $\boldsymbol{x}$, se pueden encontrar las direcciones canónicas $\boldsymbol{x}_c$ que más se le parecen a cada dirección. Entonces, esto es equivalente a estimular las neuronas de la capa con las direcciones del ejemplar; las más parecidas tendrán nivel de estimulación mayor, y ganarán la competición.

Con esas direcciones canónicas $\boldsymbol{x}_c$, la capa 2 tiene la capacidad de generar un nuevo vector característico $\boldsymbol{v} \in \mathbb{R}^h$ que representa la frecuencia con que cada dirección canónica ocurre en la característica $\boldsymbol{x}$ de un gesto.

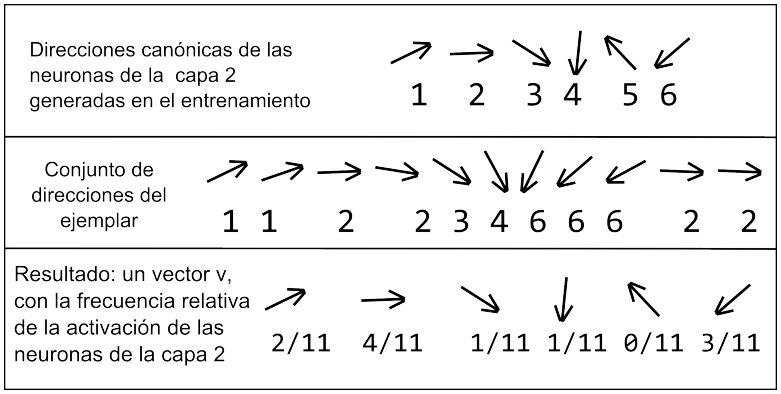


Figura 6.7: Cálculo del vector $\boldsymbol{v}$ a partir de las direcciones de un ejemplar.

Formalmente, dado un ejemplar $\boldsymbol{x}$, y una función de distancia *dist*, el vector $\boldsymbol{v}$ está dado por la fórmula:

$$\boldsymbol{v}[k] = \frac{count(k, \boldsymbol{x})}{n},\ k = 1, \ldots, h$$

$$count(k, \boldsymbol{x}) = \sum_{i=1}^{n} BMU(k, \boldsymbol{x}[i])$$

$$BMU(k, \boldsymbol{d}) = \begin{cases} 1 & \text{si } dist(\boldsymbol{x}_c[k], \boldsymbol{d}) < dist(\boldsymbol{x}_c[j], \boldsymbol{d}),\ j \neq k,\ j = 1, \ldots, h \\ 0 & \text{de lo contrario} \end{cases}$$

Donde $count(k, \boldsymbol{x})$ cuenta la cantidad de direcciones de $\boldsymbol{x}$ que están más cerca de la dirección canónica $\boldsymbol{x}_c[k]$ que a otras direcciones canónicas, y

$BMU(k, \boldsymbol{d})$ se fija si la dirección canónica k es la más cercana a la dirección $\boldsymbol{d} \in \mathbb{R}^d$.

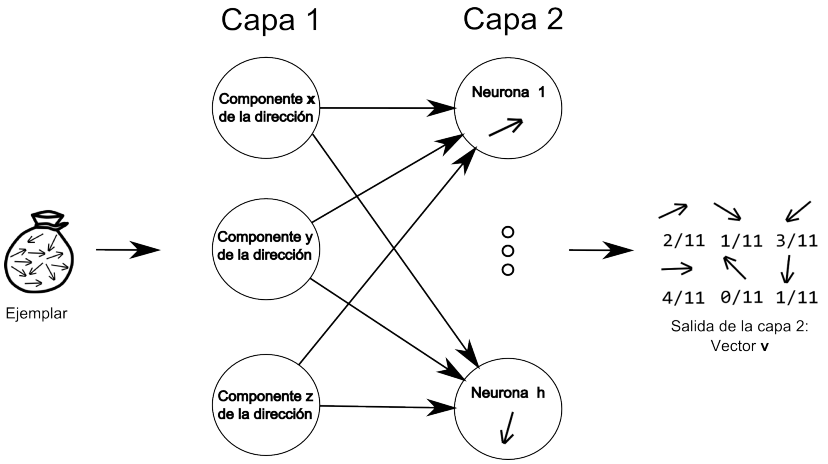


Figura 6.8: Esquema del funcionamiento de la capa 2 al codificar un ejemplar.

Es importante notar las diferencias entre el uso de esta capa en las etapas de entrenamiento y ejecución. Durante la primera, la entrada a la capa es una sola dirección 3D, y se utiliza el entrenamiento típico de CPN descripto en el capítulo 3. En la segunda, si bien la entrada a la capa sigue siendo un vector de 3 componentes, la entrada a la red es un ejemplar de gesto con varias direcciones. Entonces, en este caso se identifica el centroide más cercano a cada dirección del gesto para producir el vector v resultante.

Esta capa competitiva representa la primera parte del algoritmo, que permite codificar un ejemplar como una distribución sobre las direcciones canónicas, y representa la idea principal del CNC. Se puede pensar en esta representación como una implementación del modelo **bag-of-words** [87] para las direcciones del gesto.

A continuación, se describe el entrenamiento y el funcionamiento de la capa 3, y finalmente la función de combinación que une las salidas de cada red.

6.2.1.2. Capa de salida

La capa 3, dado un ejemplar nuevo, genera puntajes de distancia del ejemplar con respecto a cada ejemplar de entrenamiento, pero en base a la codificación que realiza la capa 2, basada en la frecuencia de activación de las neuronas correspondientes a las direcciones canónicas. Estos puntajes

serán de utilidad para que el clasificador calcule la clase estimada, ya que se conoce la clase de los ejemplares de entrenamiento.

Esta capa está formada por una red competitiva que no se entrena. Se genera tomando todos los ejemplares de *entrenamiento* $\boldsymbol{x}_i$, cuyas clases se conocen, y pasándolos por la capa 2, generando un vector de características $\boldsymbol{v}_i$ por cada ejemplar de entrenamiento $\boldsymbol{x}_i$, obteniendo entonces $|D_e|$ vectores $\boldsymbol{v}_i$, con clase asociada y_i. Esta información (los vectores $\boldsymbol{v}_i$ y las clases) se guarda para luego poder clasificar nuevos ejemplares.

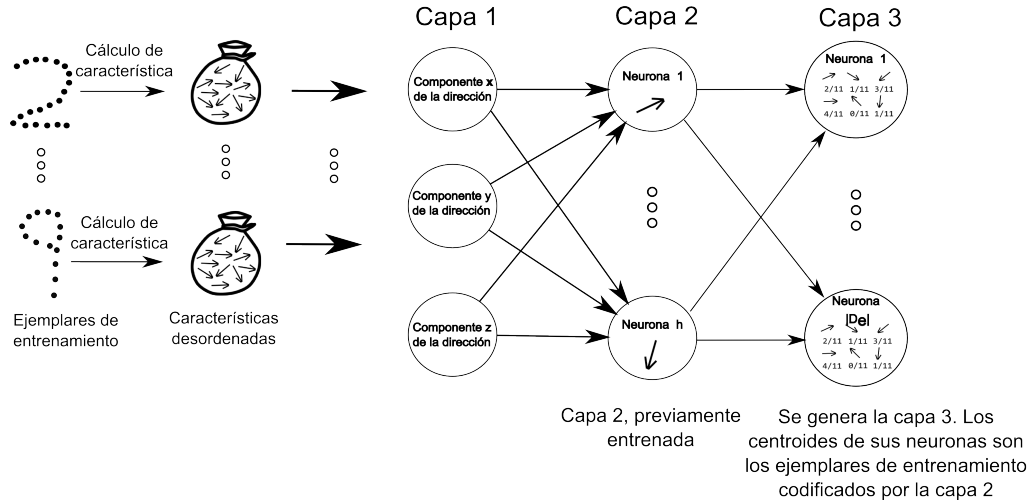


Figura 6.9: Entrenamiento de la capa 3 de una red del clasificador CNC

El funcionamiento de la capa 3 a la hora de clasificar un nuevo ejemplar es el siguiente. Dado un ejemplar $\boldsymbol{x}$ codificado por la capa 2, se obtiene un vector $\boldsymbol{v}$. Se estimula la capa 3 con $\boldsymbol{v}$ para generar la salida de la capa 3, y por ende de la red. Cada neurona $i = 1, \ldots, |D_e|$ de la capa 3 calcula una función de distancia $dist'$ entre $\boldsymbol{v}$ y el vector $\boldsymbol{v}_i$ que codifica un ejemplar del conjunto de entrenamiento, generando otro vector de distancias $\boldsymbol{w} \in \mathbb{R}^{|D_e|}$, tal que $w_i = dist'(\boldsymbol{v}_i, \boldsymbol{v})$. Esta sería la salida de la red.

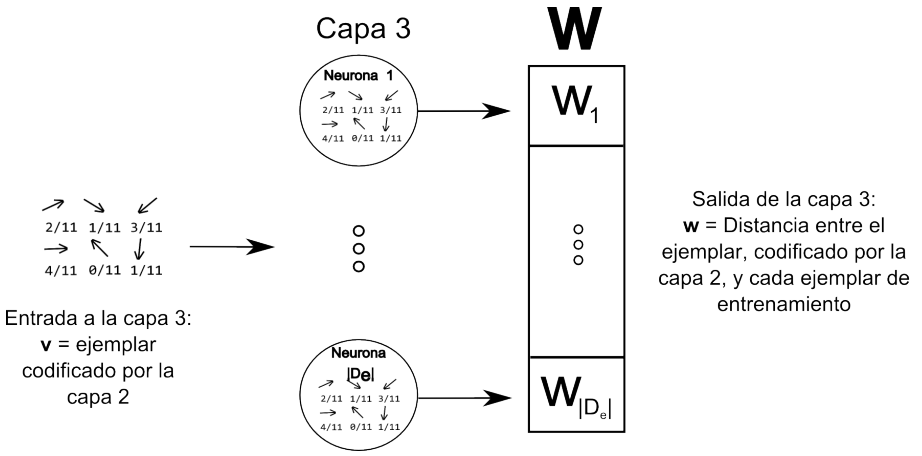


Figura 6.10: Funcionamiento de la capa 3 de una red del clasificador CNC

En resumen, el funcionamiento de la red es el siguiente. Dado un nuevo ejemplar $\boldsymbol{x} \in \mathbb{R}^{d \times n}$, se lo codifica como una distribución sobre las h direcciones canónicas utilizando la capa 2, generando un vector $\boldsymbol{v} \in \mathbb{R}^{d \times h}$.

Luego, se compara $\boldsymbol{v}$ con todos los $\boldsymbol{v}_i,\ i = 1, \ldots, |D_e|$ almacenados mediante una función de distancia $dist'$, y se genera otro vector $\boldsymbol{w} \in \mathbb{R}^{|D_e|}$, tal que $\boldsymbol{w}_i = dist'(\boldsymbol{v}_i, \boldsymbol{v})$.

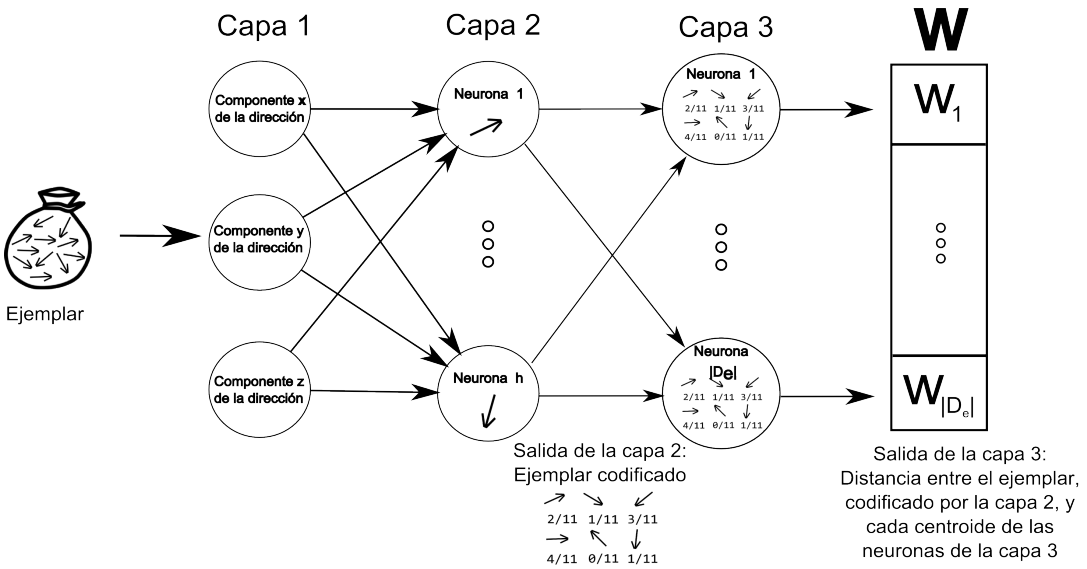


Figura 6.11: Estructura general del funcionamiento de cada red del clasificador CNC

Finalmente, el clasificador toma la salida de cada red y calcula la clase más adecuada para el ejemplar $\boldsymbol{x}$.

Es importante notar que en las primeras pruebas realizadas con el CNC para reconocer gestos con conjuntos de entrenamiento reducidos, en lugar de guardar las codificaciones de todos los gestos de entrenamiento, en la capa 3 se guardaba, para cada clase, la media de los ejemplares codificados de dicha clase. Con tan pocos ejemplares de entrenamiento, esta media representaría el centroide de un agrupamiento de los mismos.

En los experimentos (capítulo 7), dicha estructura tenía un error de clasificación mayor, y por ende se seleccionó el modelo donde se guardan todos los ejemplares.

6.2.2. Combinación de los resultados de cada red

El CNC entonces se genera entrenando en paralelo r redes. A la hora de clasificar un nuevo ejemplar $\boldsymbol{x}$, se genera la salida de cada red $\boldsymbol{w}^j$, $j = 1, \ldots, r$. Es importante notar aquí que como el orden de los ejemplares de entrenamiento es el mismo para todas las redes, la salida $\boldsymbol{w}_i^j$ se refiere, para cualquier red j, a la comparación de las direcciones canónicas de $\boldsymbol{x}$ con el ejemplar de entrenamiento $\boldsymbol{x}_i$.

La combinación de resultados de las redes consiste en tomar el promedio de todos los $\boldsymbol{w}^j$:

$$\boldsymbol{o} = \frac{\sum_{j=1}^{r} \boldsymbol{w}^j}{r}$$

De manera que el componente i de $\boldsymbol{o}$, o_i, se refiere a la distancia promedio entre las direcciones canónicas de $\boldsymbol{x}$ y las de $\boldsymbol{x}_i$ respecto a todas las redes. La clasificación se hace entonces con la función f, que elige la clase y_i del ejemplar de entrenamiento $\boldsymbol{x}_i$ más cercana a $\boldsymbol{x}$, o sea con el menor o_i:

$$\begin{aligned} f(\boldsymbol{x}) &= y_c \\ c &= \operatorname*{Min}_{i=1,\ldots,|D_e|} o_i, \end{aligned}$$

donde $\boldsymbol{o}$ es la salida del clasificador cuando se le da como entrada el vector $\boldsymbol{x}$, y o_i el elemento i-ésimo de $\boldsymbol{o}$. Por ende, la variable c representa el índice del ejemplar de entrenamiento con la menor distancia a $\boldsymbol{x}$, según la codificación de la capa oculta.

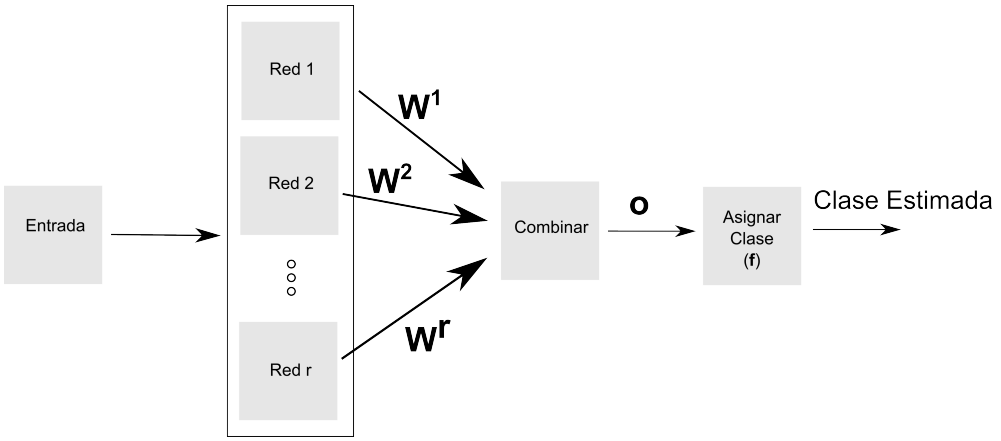


Figura 6.12: Esquema de la combinación de resultados del clasificador.

6.2.3. Bagging

El CNC utiliza la técnica de **bagging**, en la cual se entrenan múltiples clasificadores internos del mismo tipo en paralelo, y para clasificar se integran los resultados de estos clasificadores mediante un proceso de decisión para añadir robustez al proceso de reconocimiento y aumentar la tasa de aciertos [88].

La idea principal en bagging es que como cada red se entrena de forma separada, cada una tendrá un sesgo distinto, ya que el algoritmo de entrenamiento del agrupamiento competitivo comienza en un estado aleatorio diferente en cada ejecución. Si se combinan varios, es de esperar que el sesgo en promedio decrezca. Visto de otro modo, si cada red está entrenada bien pero falla en algunos ejemplares, y si aquellos en los que falla cada red son distintos, es de esperar que, dado un ejemplar nuevo $\boldsymbol{x}$, la mayoría de las redes no fallen al clasificarlo. Esto obviamente no puede aumentar la potencia del clasificador con conjuntos de datos *difíciles*, por ejemplo, con una gran cantidad de ejemplares similares con distinta clase, pero si sirve para paliar posibles entrenamientos desafortunados, y reducir en cierta medida el sesgo del clasificador.

6.2.4. Entrada al CNC

Dado que el CNC no tiene en cuenta la secuencia de direcciones del gesto, no es necesario que la longitud del vector de direcciones sea una constante n para todos los gestos, como sucede con el resto de los clasificadores.

Esto puede verse ya que la salida de la capa 2 es un vector con la frecuencia de activación de las direcciones guardadas en dicha capa, al ser estimuladas por las direcciones del gesto. Como la cantidad de neuronas de dicha capa es constante, entonces la transformación que realiza la capa 2 implementa también un esquema de normalización de la longitud de los gestos, además de codificarlos de forma distinta.

Entonces, se puede aplicar el algoritmo CNC directamente a un conjunto de ejemplares D_e, tal que cada $\boldsymbol{x_i} \in D$ tiene una longitud n_i posiblemente diferente. Para ello, se deberían aplicar todos los pasos de pre-procesamiento y cálculo de la característica excepto el de re-muestreo para hacer constante la cantidad de direcciones de la característica resultante.

6.3. Modelos de reconocimiento a comparar

6.3.1. Support Vector Machine (SVM)

Utilizando con la característica ϕ_d directamente, se puede pensar en un gesto como un punto en un espacio n-dimensional de direcciones. El clasificador toma toda la secuencia de direcciones y genera un modelo que distingue gestos.

En el caso de SVM, se asume una función de distancia definida entre puntos o gestos de dicho espacio. Dos puntos cercanos en el espacio, en base de esa medida de distancia, se consideran gestos parecidos. Dadas C clases, si se entrenan C clasificadores para que distingan los ejemplares de cada clase de los ejemplares de las otras, se encuentran C superficies que separen dichas clases; estas dependen del kernel elegido. Se utilizaron dos kernels en los experimentos: el lineal y el gaussiano.

En el caso del lineal, se busca simplemente un hiperplano separador. En el caso del kernel gaussiano, la fórmula resultante de f es:

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_i \alpha_i y_i \kappa_{\boldsymbol{x}_i, \boldsymbol{x}} + b \\ &= \sum_i \alpha_i y_i e^{-\|\boldsymbol{x_i} - \boldsymbol{x}\|/\sigma} + b \end{aligned}$$

donde σ es un parámetro de desviación estándar, $\|\cdot\|$ es la norma euclídea, y_i es la clase del gesto de entrenamiento i, y α_i es su variable dual. Entonces, se puede interpretar esta fórmula de la siguiente manera.

Cada función gaussiana es una función que calcula la similitud entre el ejemplar $\boldsymbol{x}$ y el ejemplar de entrenamiento $\boldsymbol{x_i}$, escalado por σ. En el mejor caso $\boldsymbol{x_i} = \boldsymbol{x}$, por ende $e^{-\|x_i - x\|/\sigma} = 1$ y entonces, el término i-ésimo de la sumatoria es $\alpha_i y_i$. A medida que $\boldsymbol{x}$ es cada vez más diferente de $\boldsymbol{x_i}$, $e^{-\|x_i - x\|/\sigma}$ tiende a 0, y entonces dicho término prácticamente se anula en la sumatoria.

Como $\alpha_i \geq 0$ y $y_i = \pm 1$, se puede interpretar esto como que se seleccionan los vectores del conjunto de entrenamiento más cercanos a $\boldsymbol{x}$, pesados por la distancia entre $\boldsymbol{x}$ y $\boldsymbol{x_i}$, y además α_i, y se los suma en la dirección en que apunta y_i. Si $y_i = 1$ el término i-ésimo "vota"para que el vector $\boldsymbol{x}$ sea considerado de clase 1, con una fuerza de votación $\alpha_i e^{-\|x_i - x\|/\sigma}$, y viceversa si $y_i = -1$.

Se utiliza la versión isotrópica del kernel gaussiano, $e^{-\|x_i - x\|/\sigma}$, que difiere de la versión general en que no se incluye la matriz de covarianza bajo la común hipótesis de que los ejemplares pueden tomarse distribuidos isotrópicamente en el espacio (o sea, la matriz de covarianza es la identidad).

Por último, b ajusta la línea de corte, por las mismas razones mencionadas en el capítulo 1 para agregar un bias en el perceptrón. En este caso, $\boldsymbol{x}$ y $\boldsymbol{x_i}$ son gestos; el kernel gaussiano entonces implementaría la noción del capítulo 4 de una equivalencia aproximada $=_\epsilon$ entre gestos.

6.3.2. Redes Neuronales Feedforward (FF)

Para las redes neuronales feedforward, así como para SVM, se utiliza el vector de características de cada gesto completo como si fuese un punto en un espacio n-dimensional de gestos.

De todos modos, el vector de direcciones no es simplemente un conjunto de datos, sino que tiene un orden y estructura particular, ya que representa una secuencia ordenada de direcciones, donde bajo la hipótesis de continuidad, cada dirección está correlacionada con la anterior.

Esta correlación se puede modelar utilizando los modelos de redes neuronales recurrentes, pero también utilizando el modelo **input-delay** [89], en donde se alimenta a la red con *todo* el vector de direcciones, y se espera que la red misma modele las dinámicas de las secuencias de direcciones. Este es el enfoque que aquí se toma; la red toma un vector de n direcciones, y por ende utiliza $3 \times n$ neuronas de entrada. Se utiliza una sola capa oculta con h neuronas, con h determinado experimentalmente. La capa de salida tiene tantas neuronas como clases, C; cada salida o_c representa la confianza de que el ejemplar $\boldsymbol{x}$ pertenezca a la clase c.

6.3.3. Clasificador basado en templates

Se ha desarrollado un clasificador basado en templates o ejemplares prototípicos del conjunto de entrenamiento, que clasifica cada dirección del vector de direcciones del gesto por separado y luego une dicha información para realizar una clasificación global. El mismo ha sido diseñado para usarse con conjuntos de entrenamiento de tamaño reducido.

El clasificador toma como entrada un ejemplar, que es una secuencia de longitud fija $\boldsymbol{x}$ de n direcciones, y da como resultado un vector $\boldsymbol{y}$ de C puntajes 0..1 que representan la confianza de que el ejemplar pertenezca a cada clase.

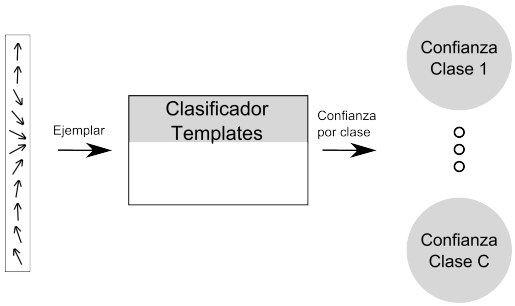


Figura 6.13: Esquema general del clasificador basado en templates.

El clasificador se construye con n clasificadores bayesianos competitivos (CBC), y cada CBC $C_j, j = 1 \ldots n$ está compuesto de un conjunto de estados $S_{j,c},\ c = 1, \ldots, C$.

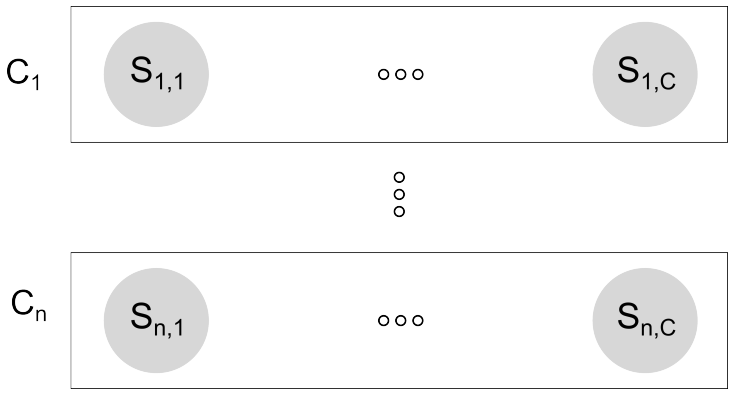


Figura 6.14: Organización interna del clasificador por templates. Se muestran los n CBC C_j, cada uno con C estados $S_{j,c}$

Hay un CBC por cada dirección del gesto; dada la dirección j del gesto, la salida de la CBC c_j es un vector de C elementos, donde cada elemento

es generado por el estado $S_{j,c}$ y representa la confianza de que la dirección j sea del gesto c, $P(S_{j,c}|\boldsymbol{x}[j])$.

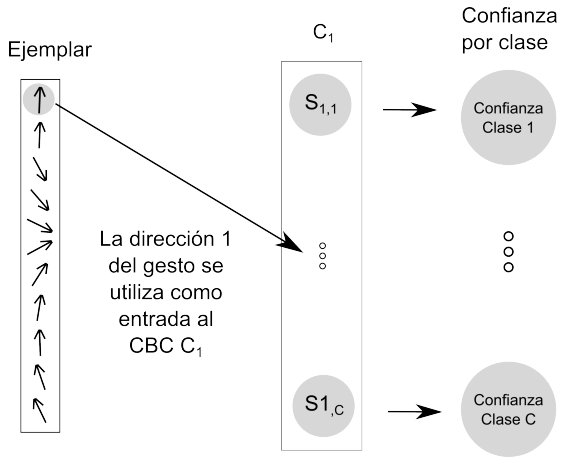


Figura 6.15: Estructura del CBC C_j, para $j = 1$, es decir, del CBC asignado a la primera dirección del gesto. Cada estado $S_{1,c}$, $c = 1, \ldots, C$, modela la emisión de posibles direcciones de cada clase para la dirección 1 de los gestos.

El clasificador, dado un gesto, alimenta cada CBC C_j con la dirección j. Luego, combina los vectores de confianza de cada C_j en una confianza global, representada por un vector $\boldsymbol{o}$ de C elementos donde $o_i = P(c|\boldsymbol{x})$.

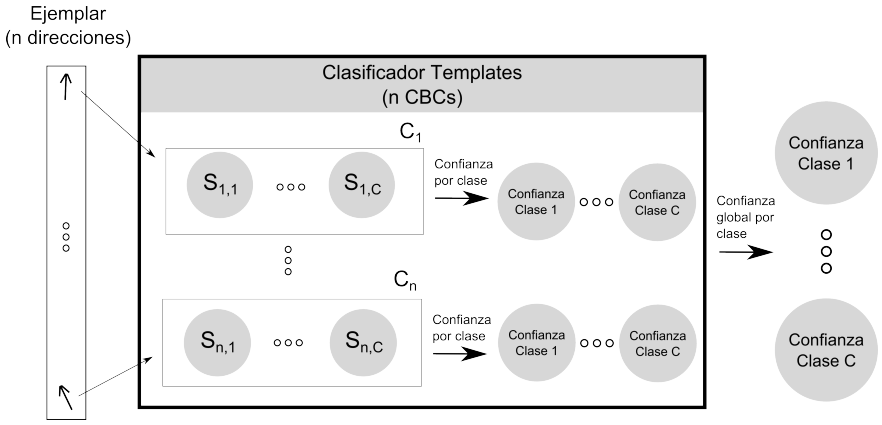


Figura 6.16: Funcionamiento general del clasificador. La secuencia de direcciones del gesto se divide. Cada dirección j, en orden, se ingresa como entrada en el CBC C_j, que genera un vector de confianza por clase. El clasificador luego combina los n resultados de los CBC para generar un vector de confianza por clase global.

Para calcular la confianza $P(c|\boldsymbol{x})$, se asume que las direcciones de los gestos se generan mediante procesos gaussianos. El estado $S_{j,c}$ modela la generación de la dirección j del un gesto, asumiendo que el mismo pertenece a la clase c. Entonces, el entrenamiento del clasificador consiste en determinar los parámetros de las distribuciones gaussianas que modelan las direcciones de los gestos. Como cada gesto puede hacerse de una manera ligeramente distinta, se considera, para cada estado $S_{j,c}$, varias distribuciones gaussianas.

Por ende, a cada estado $S_{j,c}$ del CBC j le corresponde un conjunto de funciones de distribución de probabilidad gaussianas con medias $\boldsymbol{\mu}_{j,c,k}$ y co-varianzas $\boldsymbol{\Sigma}_{j,c,k}$, $k = 1 \ldots |D_c|$, donde D_c es el conjunto de ejemplares entrenamiento de la clase c. Cada gaussiana j modela la probabilidad de una emisión de la dirección j del gesto asumiendo que el mismo pertenece a la clase c. Cada estado $S_{j,c}$ tiene una distribución por cada ejemplar de la clase c (por un total de $|D_c|$).

Los parámetros de cada función se estiman con los ejemplares del conjunto de entrenamiento como $\boldsymbol{\Sigma}_{j,c,k} = cov(\boldsymbol{I_{j,c}})$, $k = 1 \ldots |D_c|$, donde $\boldsymbol{x}_{ck}$ es el ejemplar k-ésimo de la clase c e $\boldsymbol{I_{j,c}}$ es una matriz con columnas:

$$\boldsymbol{x}_{c1}[j], \boldsymbol{x}_{c2}[j], \ldots, \boldsymbol{x}_{c|D_c|}[j]$$

Entonces $\boldsymbol{I_{j,c}}$ es la matriz que contiene la dirección j de cada ejemplar de entrenamiento de la clase c.

Se utiliza entonces cada dirección de cada ejemplar del conjunto de entrenamiento como una media del CBC, lo cual no genera un modelo computacionalmente demandante ya que se está apuntando específicamente a conjuntos de entrenamiento pequeños. Para la clasificación de un nuevo ejemplar $\boldsymbol{x}$, se calcula la confianza de la emisión de cada estado $S_{j,c}$ aplicando la regla de bayes

$$P(S_{j,c}|\boldsymbol{x}[j]) = \frac{P(\boldsymbol{x}[j]|S_{j,c})P(S_{j,c})}{P(\boldsymbol{x}[j])} = \frac{p_{j,c}P(S_{j,c})}{\sum_{k \in C} p_{j,k}P(S_{j,k})} = \frac{p_{j,c}}{\sum_{k \in C} p_{j,k}}$$

Donde:

$P(S_{j,k}) = \dfrac{1}{C}$ es igual para todas las clases k

$p_{j,k} = P(\boldsymbol{x}[j]|S_{j,c}) = max(\mathcal{N}(\boldsymbol{x}[j]; \boldsymbol{\mu}_{j,c,k}, \boldsymbol{\Sigma}_{j,c,k}))$ es el puntaje máximo entre los estados

Este último puntaje máximo modela la confianza de que la dirección j del ejemplar pertenezca a la clase c.

Nótese que se está aplicando la regla de bayes para determinar la confianza $P(S_{j,c}|\boldsymbol{x}[j])$ en cada estado y luego se elige el estado con la mayor confianza. Esta es la razón por la cual se denomina a cada C_j un clasificador bayesiano competitivo.

Finalmente, la confianza de todo el ejemplar dada la clase c se calcula combinando las salidas de cada C_j como:

$$P(c|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|c)P(c)}{\sum_{k\in C} P(\boldsymbol{x}|k)P(k)} = \frac{P(\boldsymbol{x}|c)}{\sum_{k\in C} P(\boldsymbol{x}|k)}$$

donde se define $P(k) = \frac{1}{|C|}$ y $P(\boldsymbol{x}|k) = \frac{\sum_{j=1}^{n} P(S_{j,k}|\boldsymbol{x}[j])}{n}$. Esta es la salida final del clasificador.

6.4. Resumen

En este capítulo se propuso un modelo nuevo de clasificador basado en redes neuronales competitivas, el **Clasificador Neuronal Competitivo (CNC)**, y se introdujeron otros clasificadores basados en SVM, redes neuronales feedforward, y un método de clasificación basado en templates para comparar con el CNC. Todos los modelos de clasificación utilizan la misma representación para el gesto a reconocer y buscan implementar la noción de equivalencia del capítulo 4.

El modelo CNC es un clasificador diseñado específicamente para reconocer gestos y entrenarse con pocos ejemplares de entrenamiento. El CNC utiliza un esquema de *bagging*, en donde varias instancias de un tipo de red neuronal, que se describe en el siguiente párrafo, funcionan en paralelo para clasificar de forma más efectiva. Dichas redes se entrenan por separado, pero a la hora de clasificar un nuevo ejemplar, cada red es estimulada con el mismo y genera un resultado; el CNC combina los resultados de las redes para estimar la clase del ejemplar.

Las redes del CNC se componen de tres capas: la 1, de entrada, la 2, de codificación, y la 3, de salida. La capa 2 es de tipo competitiva, y cada neurona de la misma representa una dirección; los centroides se entrenan con el algoritmo de entrenamiento CPN del capítulo 3, y representan un conjunto de direcciones *canónicas*, que son las más representativas de todas las direcciones del conjunto de entrenamiento. Luego, dado un nuevo ejemplar, la red puede codificar el mismo en base a las direcciones

canónicas. La capa 3 también es de tipo competitiva. Los centroides de las neuronas de la capa 3 se generan en base a los ejemplares del conjunto de entrenamiento, codificados de acuerdo a la capa 2; por cada ejemplar de entrenamiento, hay una neurona en la tercera capa. Dado un nuevo ejemplar, la red lo codifica con la segunda capa, y con la tercera genera un vector de distancias entre el nuevo ejemplar codificado y sus neuronas.

Al clasificar un nuevo ejemplar, el CNC hace que cada red calcule un vector de distancias, y luego genera un nuevo vector de distancias promediando los de cada red. Finalmente, como las neuronas corresponden a ejemplares de entrenamiento, y de estos se conoce la clase, se le asigna al nuevo ejemplar la clase del ejemplar de entrenamiento con menor distancia, según el vector de distancias promedio.

Los modelos de SVM, redes feedforward y de templates se aplican simplemente interpretando al vector de direcciones como un punto en el espacio n-dimensional de todos los posibles vectores de direcciones.

Para el modelo SVM, se generó un clasificador de C clases utilizando C clasificadores de 2 clases, entrenados con el algoritmo SMO. Se utiliza el separador lineal básico y opcionalmente el separador que incluye una transformación mediante un kernel gaussiano.

Para las redes neuronales feedforward, se utilizó un modelo con 3 capas, con funciones de activación *tanh*, entrenado con backpropagation y resilient backpropagation.

El método de templates guarda las características correspondientes a cada parte de los ejemplares de cada gesto de entrenamiento y a la hora de clasificar compara cada una de dichas características con las del ejemplar a clasificar por separado, mediante una función de similitud gaussiana. Luego, combina los resultados de la comparación de cada parte para estimar la probabilidad de que el ejemplar a clasificar pertenezca a cada clase.

En el capítulo siguiente, se realizan experimentos de clasificación con cada uno de estos modelos y las bases de datos presentadas en el capítulo anterior para determinar su aplicabilidad al problema del reconocimiento de gestos.

7

Experimentos y resultados

> "Torture data long enough, and it'll confess to anything"

7.1. Introducción

Este capítulo muestra los resultados de los experimentos realizados con la base de datos Letras y Números (LNHG, por Letters and Numbers Hand Gestures) y la base de datos Celebi2013. También se incluye una discusión sobre la aplicabilidad y efectividad de los métodos propuestos.

El objetivo de estos experimentos es triple: validar el modelo y la característica desarrollada como apta para clasificar gestos, encontrar el clasificador que mejor se adapte a la tarea, junto con sus parámetros ideales, y finalmente validar la hipótesis de que son aptos para generar modelos de gestos con pocas muestras de entrenamiento.

7.2. Experimentos con base de datos LNHG

A continuación, se presentan los resultados de los experimentos con cada clasificador [1].

[1] Todos los datos recabados pueden encontrarse en el sitio de la base de datos LNHG, https://sites.google.com/site/lnhgdb/, en formato csv.

Figura 7.1: Esquema de los experimentos realizados

Se compara una medida relacionada al error, el **porcentaje de clasificación correcta** ***cc*** de los métodos en la base de datos de gestos. El ***cc*** se define como el porcentaje de ejemplares que se han clasificado correctamente, de modo que si e es el error medido como el porcentaje de ejemplares clasificados incorrectamente, $cc = 1 - e$.

Se utilizan los mismos parámetros de pre-procesamiento para todos los métodos y se muestra la variación del ***cc*** en función del porcentaje de ejemplares de D utilizados como conjunto de prueba del conjunto de prueba $p = |D_p|/|D|$ y sus parámetros[2].

En todos los experimentos la clase asignada fue aquella para la cual el clasificador dio el mejor puntaje, sin importar el valor absoluto de dichos puntajes. En otras palabras, dada una muestra, los métodos calculan los puntajes por clase o_c de forma corriente, y luego la clase correspondiente se calcula como:

$$clase = \max_c o_c$$

Se realizaron dos tipos de pruebas. Las primeras tienen el objetivo de medir el ***cc*** con distintos valores para los parámetros de los algoritmos, el procesamiento de los ejemplares y el experimento. Para estas pruebas, se muestran gráficos del ***cc*** de cada clasificador en función de sus parámetros y el tamaño del vector de características, n. Además, se probó el comportamiento de cada clasificador variando el tamaño del conjunto de prueba, para verificar que el error con conjuntos de entrenamiento pequeños es comparable al obtenido con conjuntos grandes.

Las otras pruebas, las finales, se realizaron con los parámetros que mejores resultados arrojaron en las primeras y con un conjunto de entrenamiento pequeño; sólo se utilizaron 3 ejemplares de cada una de las 26 clases, es decir, $|D_e| = 3 \times 36 = 108$.

En ambos casos, se utilizó el esquema de prueba de validación cruzada aleatoria estratificada, definida en el capítulo 1. Las primeras pruebas utilizaron 30 repeticiones por cada conjunto de parámetros probado. Las

[2]O sea, también se varía el tamaño del conjunto de entrenamiento, ya que $D = D_p \cup D_e$.

finales se realizaron con 500 repeticiones, para validar con mayor certeza los resultados.

A continuación se muestran los resultados de las primeras pruebas para cada clasificador. Luego se presenta la comparación final en la tabla 7.1

7.2.1. SVM

7.2.1.1. Kernel lineal

En el caso del kernel lineal, entrenado con 3 ejemplares de cada clase, se muestra el ***cc*** en función del parámetro de regularización c y el tamaño o la cantidad de partes del vector de características n:

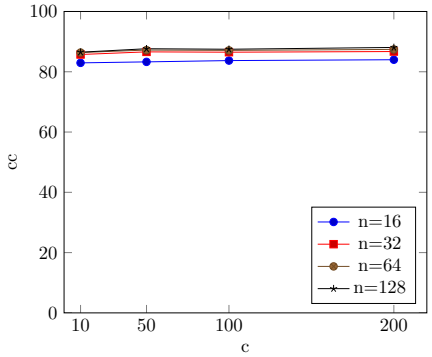


Figura 7.2: Curvas de variación del ***cc*** en función de n y c, con $p = 0.85$.

Por otro lado, el modelo es bastante estable para todo valor de c, el parámetro de regularización. Además, los ejemplares son más separables si se incrementa el tamaño del vector característico de los ejemplares, n, lo cual confirma la habilidad de clasificar patrones con alta dimensionalidad del modelo SVM.

Además, se muestra el ***cc*** en función del porcentaje de ejemplares destinados al conjunto de prueba, p.

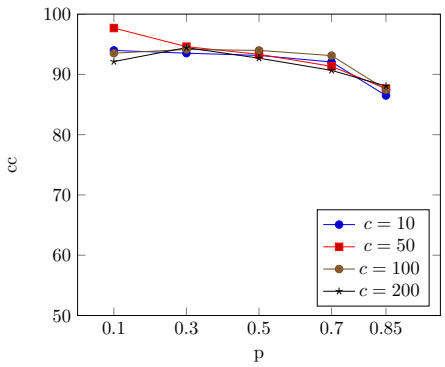
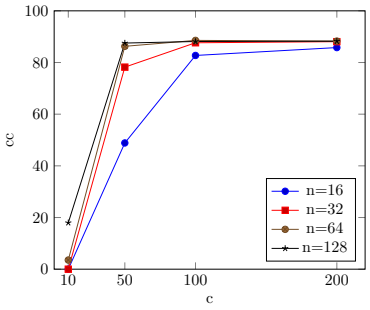


Figura 7.3: Curvas de variación del ***cc*** en función de p y c, con $n = 128$

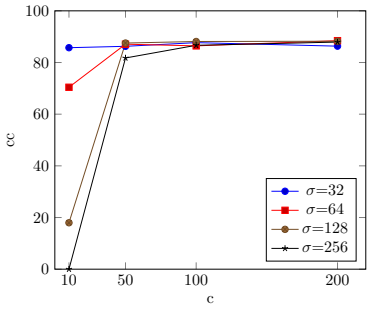
Puede observarse que el porcentaje de clasificación decae desde usar 90 % de datos para el entrenamiento (en este caso, 18 de 20 por clase), hasta usar solo un 15 % de los datos para entrenar (3 de 20 por clase), pero solo ligeramente.

7.2.1.2. Kernel Gaussiano

Para el caso del kernel gaussiano, se varió además de c y n, el parámetro de σ que controla la amplitud de la función gaussiana.



(a) Curvas de variación del ***cc*** en función de n y c.



(b) Curvas de variación del ***cc*** en función de c y σ, con $n = 128$ y $p = 0.85$.

Se observa que el parámetro de regularización $c = 10$ da malos resultados, siendo sensible a valores bajos. Los mejores resultados se obtienen con un valor alto, $c = 200$, aunque es bastante estable para $c = 50, 100$. Esta sensibilidad es más pronunciada cuando n es grande, posiblemente debido a que un valor pequeño de regularización implica más libertad para el hiperplano, y más libertad implica más formas en que la determinación del hiperplano pueda *salir mal*. A su vez, a medida que aumenta

la dimensionalidad de los datos, la cantidad de hiperplanos posibles también aumenta y por ende la combinación de ambos fenómenos puede llevar a una performance tan baja ya que es muy posible encontrar un hiperplano en altas dimensiones que maximice el margen pero no refleje la distribución de los datos.

Como antes, con los mejores valores, se evaluó la dependencia del error con el tamaño de p:

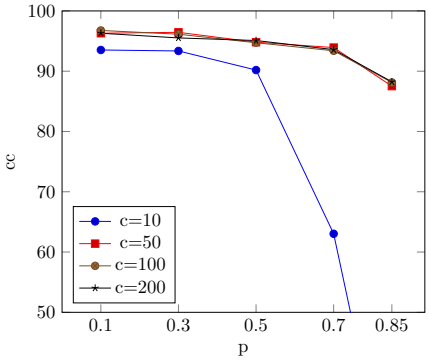


Figura 7.5: Curvas de variación del ***cc*** en función de p y c, con $n = 128$ y $\sigma = 32$.

La dependencia con la cantidad de datos es fuerte para $c = 10$, pero para los otros valores de c, ***cc*** es bastante estable. Se observa que en esos casos, ***cc*** decrece lentamente, con una baja de menos de 10 % en el peor caso.

7.2.2. Feedforward

Se encontró que la red feedforward entrenada con el algoritmo backpropagation básico da resultados de poca calidad. La red no logra en general converger a un mínimo local aceptable aún con más de 10000 iteraciones de actualización. Se estima que se debe a la dificultad del problema y su alta dimensionalidad, combinada con el algoritmo de entrenamiento simple que hace que dicho proceso se atasque muy fácilmente en mínimos locales de poca calidad.

Por otro lado, el algoritmo resilient backpropagation ha dado resultados mucho más satisfactorios. En la figura 7.6 se muestra el error del clasificador para distintos valores de h, la cantidad de neuronas ocultas, y n:

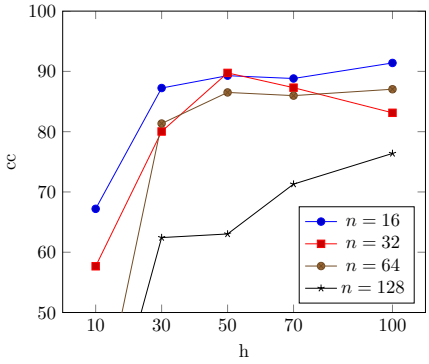


Figura 7.6: Curvas de variación del ***cc*** en función de n y h, con $p = 0.85$.

Se puede observar que, al contrario de lo que sucede con SVM, ***cc*** decrece a medida que se aumenta el tamaño de gesto, probablemente por la dificultad de entrenar el clasificador en espacios dimensionales más altos.

Por otro lado, aumentar el tamaño de h contrarresta en parte este efecto, lo cual es de esperarse ya que se necesitan más neuronas para modelar espacios de dimensionalidad más grande.

Con el mejor valor de n, 16, se muestran resultados para distintos tamaños del conjunto de prueba:

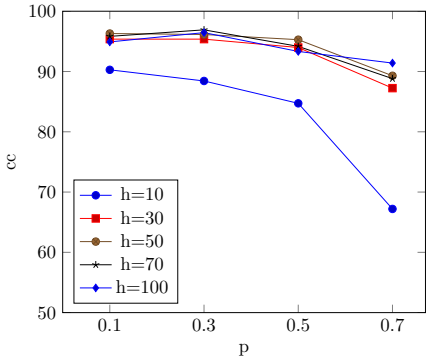


Figura 7.7: Curvas de variación del ***cc*** en función de p y h, con $n = 16$. En este caso el parámetro p no se evaluó con el valor 0.85 debido a la dificultad de entrenar la red mencionada anteriormente.

Salvo para el caso $h = 10$, el error de la red decae bastante lentamente a medida que decrece el tamaño del conjunto de entrenamiento. Si bien la red se beneficia de usar más neuronas cuando existen pocos ejemplares, esto probablemente sea porque en dichos casos se realiza un aprendizaje

más de memorización debido a la dificultad de modelar cada clase con pocos ejemplares.

Dicho beneficio desaparece a medida que se achica el conjunto de prueba, debido a que con más ejemplares de entrenamiento se puede modelar la geometría de cada clase con pocas neuronas. No obstante, el clasificador muestra buenos resultados en todos los casos con $h > 10$.

7.2.3. Templates

El método de templates no tiene parámetros para elegir, de modo que se midió directamente su performance variando n y p:

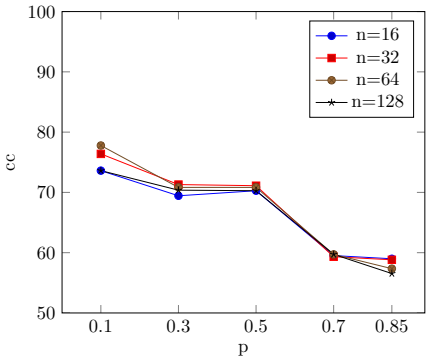


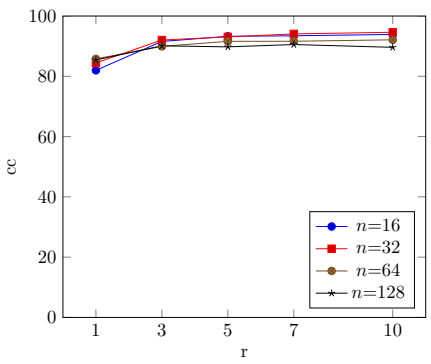
Figura 7.8: Curvas de variación del ***cc*** en función de n y p

El clasificador de templates muestra una efectividad clasificación aceptable con un conjunto de entrenamiento grande que decae grandemente con pequeños conjuntos de datos. Se entiende que dicha debilidad se debe al uso de modelos probabilísticos, ya que para una estimación efectiva de sus parámetros se requieren conjuntos de entrenamiento más grandes. Además, da mejores resultados con n grande si el conjunto de datos es grande, posiblemente debido a que puede modelar mejor el espacio altamente dimensional, mientras que para conjuntos de entrenamiento chicos, da mejores resultados con una dimensionalidad pequeña.

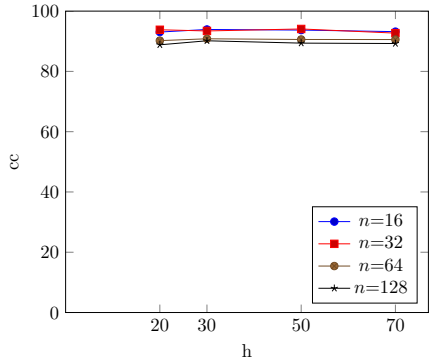
7.2.4. CNC

7.2.4.1. CNC con normalización

Para el CNC, se probaron distintas variantes de h, las neuronas ocultas, y r, la cantidad de redes, en base al tamaño n:



(a) Curvas de variación del ***cc*** en función de r y n, con $h = 30$ y $p = 0.85$



(b) Curvas de variación del ***cc*** en función de h y n, con $r = 10$ y $p = 0.85$.

En la primera figura, el ***cc*** se mantiene relativamente estable al pasar de 3 redes, pero no hay una mejora significativa al aumentar r más allá de ese valor.

En la segunda figura, se observa que el clasificador es totalmente insensible al parámetro h, probablemente debido a que la cantidad de direcciones canónicas necesaria para codificar los gestos es pequeña.

Se puede observar también su dependencia con p, ya que :

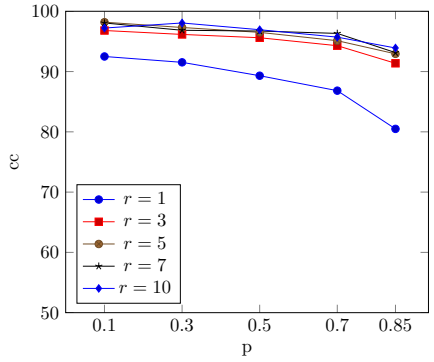


Figura 7.10: Curvas de variación del ***cc*** en función de p y r, $h = 30$, $n = 16$.

Cuando $r = 1$, ***cc*** es significativamente menor, decayendo más rápidamente que en los otros casos, pero para el resto de valores de r probados, ***cc*** se mantiene bastante estable. Usando más de 3 redes se obtienen mejoras pero no son significativas, aunque incrementan ligeramente a medida que se reduce el tamaño del conjunto de entrenamiento.

7.2.4.2. CNC sin normalización

Por otro lado, se obtuvieron resultados evitando el paso de preprocesamiento que normaliza la cantidad de elementos del vector de características a una constante n, ya que el CNC no lo requiere:

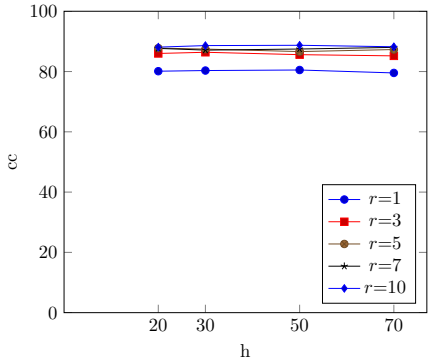


Figura 7.11: Curvas de variación del ***cc*** en función de h y r, $p = 0.85$, sin normalizar.

Y nuevamente, la dependencia con p utilizando la característica sin normalizar:

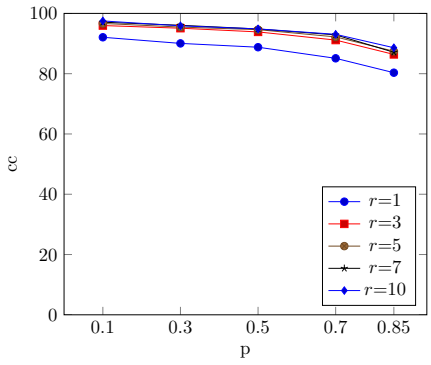


Figura 7.12: Curvas de variación del ***cc*** en función de p y r, $h = 30$, sin normalizar.

Puede observarse que el CNC es robusto respecto del tamaño de entrenamiento, de forma similar al reconocedor con SVM para ambos tipos de kernel.

En estos experimentos, sin el re-muestreo, se obtuvo un resultado comparable al que utiliza el pre-procesamiento completo, aunque ligeramente más bajo.

7.2.5. Comparación

En base a los experimentos anteriores, se determinaron los mejores valores para los parámetros de los algoritmos. En base a esos parámetros, se realizaron las pruebas finales cuyos resultados se presentan a continuación.

Como se mencionó anteriormente, para las pruebas finales por cada algoritmo se realizaron 500 experimentos utilizando validación cruzada estratificada con re-muestreo aleatorio, con 3 muestras por clase para el conjunto de entrenamiento ($3 * 36 = 108$ en total) y 17 para el conjunto de prueba (642 en total).

En el caso de la red feedforward, 2 muestras de cada clase se utilizaron como conjunto de validación, dejando solo 15 para el conjunto de prueba, debido a que los algoritmos de entrenamiento backpropagation e iRprop+ utilizan un conjunto de validación para calcular el error que determina la condición de fin, por lo cual sería muy difícil entrenar dicha red con solo 3 muestras por clase.

El clasificador CNC* es el mismo que el CNC pero sin normalizar, es decir, sin utilizar el proceso de re-muestreo en la etapa de preprocesamiento, debido a que no necesita un vector de entrada de longitud fija n, aunque en este caso los gestos no serían verdaderamente invariantes a la velocidad.

El CNC y el CNC* tienen $r = 10$ subclasificadores. Para los clasificadores FF con resilient backpropagation, CNC y CNC*, se muestran los resultados para redes con $h = 100$, $h = 50$ y $h = 50$ neuronas ocultas respectivamente, que dieron los mejores resultados en los experimentos anteriores.

Clasificador	Parámetros	Entrenamiento	cc (μ)
SVM (Lineal)	$c = 200$, $n = 128$	SMO	88.24 % (0.080)
SVM (Gaussiano)	$c = 100$, $\sigma = 64$, $n = 128$	SMO	88.56 % (0.063)
CNC	$h = 50$, $r = 10$ $n = 32$	CNC, $\alpha = 0.01$	94.08 % (0.088)
CNC*	$h = 50$, $r = 10$	CNC, $\alpha = 0.01$	88.71 % (0.092)
FF	$h = 100, n = 16$	iRprop+	91.40 % (0.158)
Template	$n = 16$	-	53.77 % (0.050)

Cuadro 7.1: Resultados de los experimentos de clasificación para la base de datos LNHG. La última columna muestra la media del porcentaje de ejemplares clasificados correctamente y la desviación estándar (en paréntesis) para cada clasificador.

Se realizaron $\binom{6}{2} = 15$ pruebas de hipótesis t-student de diferencia

entre medias, una por cada combinación de dos clasificadores distintos, para evaluar la significancia estadística de estos resultados. En todos los casos, se encontró que los mismos son significativos a un p-nivel $p < 0.01$.

Los errores de reconocimiento del CNC* respecto al CNC son sólo ligeramente inferiores, lo cual muestra, en principio, que el método podría utilizarse sin el re-muestreo en los casos donde un desempeño reducido sea preferible a un coste computacional elevado de preprocesamiento.

7.3. Experimentos con base de datos Celebi2013

Para probar los clasificadores con esta base de datos, se consideró solamente la posición de una mano (la izquierda), y los gestos realizados con dicha mano, ya que de otra manera el mismo gesto realizado con la mano izquierda y la derecha sería casi indistinguible, quedando solo 4 clases de gestos:

- Empuje hacia arriba con la mano izquierda
- Empuje hacia abajo con la mano izquierda
- Movimiento de la mano desde la parte izquierda del cuerpo hacia la derecha con la mano izquierda
- Saludo con la mano izquierda

Se realizaron todos los pasos de preprocesamiento y extracción de características descriptas anteriormente salvo el proceso de normalización que consiste en la rotación del usuario, ya que no había información de las posiciones de las otras partes del cuerpo para tal fin [3].

Se empleó la misma metodología de validación cruzada con re-muestreo aleatorio y 500 iteraciones, y se siguieron los mismos pasos de evaluación de los parámetros del modelo. Los resultados finales se presentan en la siguiente tabla:

Nuevamente, se puede demostrar que estos resultados son significativos a un p-nivel $p < 0.01$ mediante una prueba de hipótesis de diferencia entre medias con la distribución t-student.

Hay que considerar que el método desarrollado por Celebi et al. [80] utiliza información no solo de la posición de la mano, sino también de

[3]Todos los datos recabados para los experimentos realizados en esta tesina con la base de datos Celebi también pueden encontrarse en el sitio de la base de datos LNHG, https://sites.google.com/site/lnhgdb/, en formato csv.

Clasificador	Parámetros	Entrenamiento	cc (μ)
SVM (Lineal)	$c = 100, n = 16$	SMO	100 % (0.002)
SVM (Gaussiano)	$n = 16, \sigma = 64, c = 100$	SMO	100 % (0.005)
CNC	$h = 70, r = 10, n = 16$	CNC, $\alpha = 0.5$	100 % (0.015)
CNC*	$h = 70$, $p = 5$	CNC, $\alpha = 0.5$	100 % (0.022)
FF	$h = 100, n = 16$	iRprop+	100 % (0.028)
Template	$n = 16$	-	100 % (0.006)
Celebi2013		DTW adaptativo	97 %

Cuadro 7.2: Resultados de los experimentos de clasificación para la base de datos Celebi2013.

la posición del codo, brazo y hombro, lo cual pone a dicho método en una categoría ligeramente distinta de clasificador, ya que utiliza más información para clasificar. Además, el mismo utiliza los 8 gestos realizados por expertos para entrenar y los 20 gestos de peor calidad restantes como conjunto de prueba, mientras que en los experimentos (cuyos resultados se muestran en la tabla anterior) realizados en esta tesina se utilizaron solo 3 gestos de entrenamiento, y seleccionados aleatoriamente. Por otro lado, el problema es más simple para nuestros clasificadores ya que deben lidiar con 4 clases en lugar de 8.

En base a estas consideraciones, los resultados parecen ser relativamente comparables. Si bien esta situación no es ideal, es importante notar que no existen bases de datos de gestos con el formato de la LNHG (o sea, de gestos con una sola parte del cuerpo), lo cual dificulta el proceso de comparación. Por otro lado, dado el altísimo nivel de performance de clasificación, podemos asumir con confianza que la dificultad inherente del conjunto de datos no es tan alta como la de la base de datos LNHG.

7.4. Resumen

En este capítulo se comparó el método propuesto en esta tesina en el capítulo 6, denominado CNC, con otros métodos existentes, descriptos en el mismo capítulo.

Los resultados obtenidos permiten afirmar que las capacidades de clasificación del CNC y el modelo SVM son altas, casi alcanzando 90 % de ejemplares correctamente clasificados en la base de datos LNHG, cuando se utilizan conjuntos de entrenamiento pequeños, y que aumenta considerablemente con más gestos de entrenamiento. Los otros modelos funcionan relativamente bien, y algunos hasta de manera comparable al SVM y

CNC si se aumenta el tamaño del conjunto de entrenamiento. Estos resultados son bastante robustos a las variaciones de los parámetros de cada algoritmo, indicando que son aptos para el reconocimiento de gestos.

8

Conclusiones generales y trabajos futuros

El reconocimiento de gestos es un área de investigación de sumo interés en la actualidad, debido a la gran cantidad de nuevos sensores y dispositivos móviles, que trae nuevas posibilidades y desafíos a la investigación en interacción hombre-máquina y el aprendizaje automático. El aprendizaje automático aplicado al reconocimiento de gestos proporciona un enfoque flexible y dinámico para la generación de vocabularios de gestos de interacción por parte de cualquier tipo de usuario. Estos métodos pueden aplicarse para generar modelos de reconocimiento que sirvan para desarrollar programas donde el usuario interactúa con un computador mediante gestos.

En esta tesina, el autor ha presentado un nuevo método de reconocimiento de gestos invariante a la velocidad, la escala, la traslación y parcialmente invariante a la rotación. El método obtiene buenos resultados con un entrenamiento que utiliza pocas muestras, una característica necesaria para el reconocimiento de gestos dependiente del usuario, en donde el mismo tiene que generar los patrones de entrenamiento.

Se ha generado una base de datos de gestos de ejemplo para probar los métodos. Se introdujo también un modelo teórico de gestos, junto con una característica que hace más fácil su reconocimiento bajo las invariancias mencionadas.

Se han adaptado los modelos de SVM, redes neuronales feedforward y de templates al problema en base al modelo de gestos introducido, y se ha medido su performance.

Se ha propuesto el nuevo modelo de clasificación CNC, que utiliza una representación tipo bag-of-words para las direcciones de los gestos, y que, adicionalmente, es invariante al punto de comienzo del gesto.

En los experimentos, los modelos probados funcionan bien en general, y el CNC y SVM obtienen los mejores porcentajes de reconocimiento. Adicionalmente, el CNC tiene la ventaja de ser fácil para implementar, entrenar, poseer la invariancia antes mencionada, y su algoritmo de aprendizaje requiere pocas iteraciones para converger.

El CNC deriva su buen desempeño de la codificación de los gestos que realiza en la segunda capa, especialmente junto al esquema de bagging. El mismo tiene como efecto que cada ejemplar se guarde con una codificación distinta según el agrupamiento que realiza cada red. Esto le añade más robustez al proceso de estimación de la región, ya que la misma resultará ser un promedio de las regiones de voronoi estimadas por cada red. Sin este esquema, se observa que la performance se reduce significativamente, y solo se necesitan 3 redes para aumentarla a un nivel aceptable.

La estructura de la tercera capa resulta beneficiosa para el caso en donde se tienen pocos ejemplares. Una hipótesis para explicar este resultado es que al tener tan pocas muestras, es difícil para el clasificador estimar la región de Voronoi a la cual pertenecen los ejemplares de cada clase. Al basar la clasificación directamente en los ejemplares de entrenamiento, si bien no se aproxima de forma completa la región del espacio que representa cada gesto, se están estimando varios subconjuntos de dicha región. Por este motivo, si bien el clasificador es más conservador, ya que no considera relevante que un ejemplar sea un poco parecido a varios ejemplares de entrenamiento. Por ende, para aplicaciones donde el conjunto de entrenamiento es reducido, el CNC es útil donde otros enfoques como el probabilístico del modelo de templates no tienen tanto éxito. Esta funcionalidad es de gran importancia para los sistemas de reconocimiento dependientes del usuario, debido a que hacen factible que el mismo genere vocabularios con muchas clases de gestos, debiendo grabar una cantidad mínima de ejemplares de cada clase.

En trabajos futuros, el autor espera poder determinar la performance de los métodos en distintas bases de datos para determinar el alcance de su aplicabilidad. También propone mejorar y extender la base de datos actual con más ejemplos de gestos 3D para las clases existentes y nuevas clases, a fin de proveer un punto de referencia fuerte para nuevas comparaciones y análisis de performance.

Respecto al clasificador CNC, interesa determinar específicamente como interactúan las distintas partes del mismo para convertirlo en un modelo efectivo de reconocimiento de gestos. En particular, dada su estructura de tres capas, se podría probar la segunda capa, generadora de nuevas características, en conjunto con otro clasificador como SVM, posiblemente modificando o hasta quitando el esquema de bagging. A

su vez, se puede emplear la tercera capa junto a dicho esquema, con la característica original normalizada, sin la segunda capa.

Los resultados de esta investigación han sido presentados por el autor en el *Workshop de Agentes y Sistemas Inteligentes* (WASI) del *XIX Congreso Argentino de Ciencias de la Computación* (CACIC), realizado en el mes de octubre de 2013 en la Universidad CAECE de Mar del Plata, y publicados bajo el título *A novel Neural Classifier for Gesture Recognition With Small Training Sets*[90].

Apéndice A

Aprendizaje automático

A.1. Validación cruzada

En el capítulo de aprendizaje automático se planteó el problema de la medición del error de un clasificador en un conjunto de ejemplares D. En dicho capítulo, se mencionó el problema de probar un modelo con el mismo conjunto de ejemplares con el cual fue generado, y como solución tentativa se propuso dividirlo en dos, el conjunto de entrenamiento D_e y el de prueba D_p; el primero se utiliza para entrenar el clasificador, el segundo para medir su error.

Si bien disponer de un conjunto de prueba es una gran mejora sobre entrenar y estimar con D_e, plantea dos nuevas preguntas:

- ¿Qué tamaño debe tener D_p (o D_e, ya que $D_e = D - D_p$)?
- ¿Qué ejemplares deben ir en cada conjunto?

Si bien se han desarrollado algunas reglas empíricas para responder a estas preguntas, como usar el 80 % de los ejemplares para entrenar y el resto para probar, y muchas veces las respuestas a tales preguntas dependen del contexto, la calidad y la cantidad de los datos disponibles, no hay una solución perfecta al problema de estimación del error de un clasificador.

La primer pregunta depende de cual es el objetivo del experimento; en términos generales, cuanto más grande sea el tamaño de D_e mejor, ya que permitirá desarrollar un modelo más certero. Por otro lado, lo mismo sucede con el conjunto de validación, respecto a la significancia estadística en la evaluación del error del modelo.

Estas preguntas no son tan importantes si $|D|$ es muy grande (relativo al problema a resolver), ya que por motivos estadísticos es muy probable

que ambos conjuntos representen la variabilidad del dominio del problema y por ende se pueda generar un buen modelo y realizar pruebas fiables. Sin embargo, si $|D|$ es chico, al seleccionar arbitrariamente a D_e y D_p se ata el valor del estimador a una de las posibles particiones de los dos conjuntos, con lo cual se expone el experimento a realizar alguna partición desafortunada, que de resultados que no representen el comportamiento esperado del clasificador.

El problema de elegir los conjuntos de prueba y entrenamiento de forma arbitraria radica en que el error del clasificador en un experimento puede dar bien o mal de acuerdo a la división particular del conjunto de datos con el que se está trabajando. Una mala división puede llevar a asumir que el clasificador no funciona sobre el conjunto de datos, cuando en realidad si lo hace en general, y viceversa.

La idea de dividir el conjunto de datos en dos, entrenar con uno y probar con el otro es un caso especial del método de **cross validation** (validación cruzada, VC) [91], un enfoque más estructurado y justificado teóricamente para estimar el error, que provee respuestas a algunas de estas preguntas, especialmente para los casos donde $|D|$ es chico.

La premisa básica en validación cruzada es intentar responder a la segunda pregunta, la de cómo realizar la elección de los distintos elementos que van en el conjunto de prueba, haciendo varios experimentos con distintas variaciones sistemáticas de D_p y D_e. De esta manera se evita la dependencia de la estimación del error sobre la partición particular con la que se realizó la prueba.

Existen varias versiones y extensiones de validación cruzada, que se detallan a continuación.

A.1.1. VC de k-iteraciones

Volviendo al ejemplo del reconocimiento de gestos, suponiendo un problema de clasificación con 2 clases y un conjunto de 20 ejemplos, 10 de cada clase, y que se decide utilizar 14 ejemplos para entrenar y 6 para probar. Entonces, existen $\binom{20}{14} = 38760$ formas posibles para dividir D en D_e y D_p. En lugar de hacer un solo experimento con una sola de las combinaciones, se podría hacer con todas y promediar los resultados. Dado que es difícil trabajar pensando en $\binom{|D|}{|D_e|}$ particiones, se puede pensar en una formulación equivalente, la de dividir a D en k conjuntos disjuntos de prueba llamados **folds**, cada uno con $n_k = \frac{|D|}{k}$ ejemplares [1]. Por cada

[1] En esta sección se asume, por simplicidad, que $|D|$ es divisible por k

conjunto de prueba D_p, se realiza un experimento con $D_e = D - D_p$ como conjunto de entrenamiento. Hay que notar que si bien cada ejemplar solo está en D_p una sola vez, formará parte de D_e $k-1$ veces, por ende los conjuntos de entrenamiento D_e no son disjuntos.

Se realizan k pruebas o iteraciones, una por cada conjunto de prueba D_p. De esta manera existe la seguridad de que todos los ejemplares sean probados eventualmente. En el ejemplo de los gestos, si $k = 2$ se obtienen dos subconjuntos de prueba, uno con los primeros 10 ejemplares de D, el otro con los últimos 10 ejemplares, y se realizan 2 pruebas. Si $k = 5$, se realizan 5 experimentos, cada uno con 16 ejemplares de entrenamiento y 4 de prueba.

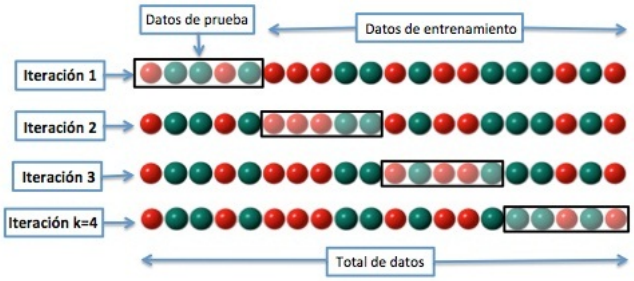


Figura A.1: Particiones posibles al usar validación cruzada con $k = 4$ y un conjunto de 20 ejemplares ($|D_p| = 5$). Cada ejemplar es utilizado una vez como dato de prueba y tres veces como dato de entrenamiento.

El estimador del error es entonces el error promedio de las k iteraciones.

Formalmente, se define D^k como el conjunto de pares (D_p, D_e) para realizar validación cruzada con k iteraciones:

$$D^k = \{(D_i, D - D_i) \mid i = 0, \ldots, k-1\} \tag{A.1}$$

Donde los conjuntos D_i están formados por n_k elementos consecutivos de D [2]., es decir:

$$D_i = \{\boldsymbol{x}_{i*n_k+1} \ldots \boldsymbol{x}_{i*n_k+n_k}, \quad i = 0, \ldots, k-1\}$$

Bajo esta definición, está claro que $D_i \cup D_j = \emptyset$ si $i \neq j$ y $\bigcup_i D_i = D$

[2]Asumiendo que el ordenamiento inicial del conjunto de ejemplares D es aleatorio para que tomar elementos contiguos no genere ningún tipo de correlación.

El estimador entonces es:

$$\rho_k = \frac{\sum_{(D_p, D_e) \in D^k} \rho(D_p, g(D_e))}{k}$$

Si bien variando k se obtiene una respuesta parcial a la pregunta de qué tamaños deben tener los conjuntos de prueba y entrenamiento, la segunda pregunta queda todavía sin una respuesta final; si se reordenan los ejemplares de D, los elementos de cada fold van a variar y por ende también el error calculado.

A medida que aumenta k aumenta el coste computacional pero también la fiabilidad de la estimación; hay que encontrar un balance entre ambas. La validación con retención es un caso especial de VC con k repeticiones con $k = 1$; aquí no hay mejora de la estimación del error pero el coste computacional se mantiene al mínimo. Una configuración popular es $k = 10$ [91, 92], ya que en general alcanza dicho balance y además es ampliamente utilizada, haciendo más fácil comparar experimentos en ciertas ocasiones . Llegando a $k = |D|$, hay otro caso especial de VC llamado **Leave-one-out CV** (VC dejando-uno-afuera).

A.1.2. VC dejando-uno-afuera

El caso particular de VC con k repeticiones con $k = |D|$ es muy interesante porque da una respuesta simple y elegante a ambas preguntas para ciertos conjuntos de trabajo.

Existen $|D|$ particiones distintas, una por cada ejemplar, por ende la complejidad de las pruebas es de orden lineal a la cantidad de ejemplares. Si hay pocos ejemplares, puede que falte significancia estadística; si hay demasiados, la demanda computacional puede ser muy elevada. Por otro lado, al aplicar un clasificador en un sistema real, típicamente se utilizan todos los datos útiles disponibles para entrenarlo (o sea $D \approx D_e$), ya que el objetivo no es medir su error sino reducirlo lo más posible. En este sentido, el estimador dejando-uno-afuera es óptimo porque refleja mejor el uso real del clasificador, aunque no necesariamente sea un mejor estimador para el error.

A.1.3. VC aleatoria

En esta variación de VC primero se elige un tamaño del conjunto de prueba n_p, y luego, cada vez que se va a realizar un experimento, se elige de forma aleatoria una partición de D con $|D_p| = n_p$. La cantidad de

experimentos a realizar puede fijarse de antemano o se puede establecer un umbral de significancia a alcanzar y se ejecutan tantos experimentos como sea necesario para alcanzarlo.

Este método tiene la ventaja de que, con las suficiente cantidad de experimentos, es el menos sesgado en cuanto a la partición de los datos [93].

A.1.4. VC estratificada

La técnica de VC estratíficada no es una variante de VC sino una técnica adicional a aplicar con las variantes anteriores.

Hay ciertas particiones para las cuales tiene más sentido hacer un experimento que otras. En el ejemplo anterior, si se elige hacer VC de k repeticiones con $k = 2$, una posible combinación elige $\frac{|D|}{k} = \frac{20}{2} = 10$ ejemplares del gesto de saludo para D_p y los 10 quedan para entrenar para D_e. Pero entonces el clasificador se entrenará con un tipo de gesto y será probado con otro, por lo cual tendrá un error muy grande ya que su conjunto de entrenamiento no era representativo de su conjunto de prueba.

Se pueden evitar estos casos de prueba degenerados exigiendo que las proporciones de clases originales, las del conjunto D, se mantengan en los conjuntos de entrenamiento y prueba D_e y D_p. La estratificación es un técnica que se aplica en conjunto con VC aleatoria o de k repeticiones, y responde parcialmente a la segunda pregunta, ya que agrega una restricción débil para decidir que ejemplares pueden ir en qué conjuntos.

A.1.5. Pruebas de hipótesis con VC

Si bien hay razones fuertes para preferir alguna combinación de estas técnicas de VC sobre el estimador más simplista $\rho(D_p, f)$, de todas maneras es necesario justificar estadísticamente la significancia de los estimadores derivados de estas técnicas. En el caso de VC esto requiere tomar ciertas precauciones ya que los distintos experimentos realizados al variar las particiones de un mismo conjunto de datos no son realmente independientes, ya que son particiones del mismo conjunto original de datos [94, 92].

Volviendo al problema de la clasificación de gestos, y olvidando el concepto de VC, se puede suponer que se dispone un conjunto de datos de 200 ejemplares de gestos, 100 de cada clase, obtenidos de forma independientemente e idénticamente distribuidos. Dividiendo este conjunto de

datos en 10 subconjuntos *disjuntos* D_i de 20 muestras (estratificadas, con 10 de cada clase cada uno) y se realizan diez experimentos, uno con cada subconjunto, dividiendo D_i en D_p yD_e en cada experimento y obteniendo 10 estimaciones de error e_i. Es importante notar que no se está empleando VC, simplemente se dividen los datos sin reutilizar ningún ejemplar, por eso los conjuntos D_i son disjuntos. Debido a eso, cada experimento es independiente del resto respecto de los datos, y por ende también lo son los e_i.

Entonces, se pueden calcular los estimadores:

$$\hat{\mu}_e = \sum_{i=1}^{10} e_i \qquad \hat{\sigma}_e = \frac{\sum_{i=1}^{10}(\hat{\mu}_e - e_i)^2}{9} \tag{A.2}$$

Ambos son estimadores insesgados y de máxima verosimilitud de μ y σ; pero en el caso de σ, este resultado depende de la hipótesis de que $Var(X+Y) = Var(X) + Var(Y)$ si X e Y no están correlacionadas, como en el caso de e_i y e_j si $i \neq j$.

Ahora, volviendo al ejemplo con 20 ejemplares de gestos, 10 de cada clase, supóngase que se generan 10 particiones utilizando VC de $k = 10$ iteraciones, y por ende se obtienen 10 estimaciones de error e_i. Éstas estimaciones de error e_i no son independientes unas de otras, porque están realizadas con (en parte) los mismos datos, y por ende están correlacionadas.

$$Var(e_i + e_j) = Var(e_i) + Var(e_j) + 2Cov(e_i, e_j) \tag{A.3}$$

$$\neq Var(e_i) + Var(e_j) \tag{A.4}$$

Por este motivo, el estimador inocente $\hat{\sigma}_e$ no es adecuado en este caso, ya que subestimará a σ y por ende cualquier test de hipótesis del error medio del clasificador que utilice σ tendrá una confianza α real menor que la calculada. Este problema no se da con el estimador de la media $\hat{\mu}_e$ ya que la linealidad en la suma de la esperanza de dos variables aleatorias no depende de su correlación.

Lo mismo sucede con VC aleatoria. Por ende, resulta necesario desarrollar pruebas de hipótesis específicas para VC, que tomen en cuenta esta dependencia indeseable entre experimentos y utilicen estimadores corregidos que compensen la estimación de la varianza. Si bien esto se encuentra fuera del alcance de esta tesina, es interesante señalar que se han desarrollado estimadores para VC aleatoria [93], y se ha probado que no existe ningún estimador insesgado para VC con k repeticiones [94], lo cual sienta al primero en bases teóricas más fuertes.

A.2. Funciones de pérdida

Las medidas de error determinan la evaluación de la generalización del modelo. Distintas medidas pueden ser apropiadas para distintos problemas o contextos.

La generalización de las medidas de error suele utilizar el concepto de **loss functions** (funciones de pérdida), ℓ [95]. Dichas funciones miden el grado de error de un clasificador dado un ejemplar, como antes. Entonces, $\ell :: (\mathcal{P}, Clasificador) \mapsto \mathbb{R}$. Las más comunes son:

$$\ell_0(\boldsymbol{x_i}, f) = \begin{cases} 1 & \text{if } \ell_1(\boldsymbol{x_i}, f) > 0 \\ 0 & \text{if } \ell_1(\boldsymbol{x_i}, f) \leq 0 \end{cases} \tag{A.5}$$

$$\ell_1(\boldsymbol{x_i}, f) = sum(abs(y_i - f(\boldsymbol{x_i}))) \tag{A.6}$$

$$\ell_2(\boldsymbol{x_i}, f) = \|y_i - f(\boldsymbol{x_i})\| \; \ell_p(\boldsymbol{x_i}, f) = \|y_i - f(\boldsymbol{x_i})\|_p \tag{A.7}$$

donde $sum(\boldsymbol{v})$ suma los elementos de un vector $\boldsymbol{v}$, $abs(\boldsymbol{v})$ devuelve un vector con los valores absolutos de $\boldsymbol{v}$, $\|\cdot\|$ es la norma euclídea de un vector, y $\|\cdot\|_p$ es la norma p.

Dada una ℓ, se puede calcular entonces el error de un clasificador f en un conjunto de ejemplares D, típicamente como:

$$\rho_\ell(D, f) = \frac{\sum_{x_i \in D} \ell(\boldsymbol{x_i}, f)}{|D|} \tag{A.8}$$

y se aplica CVcon ρ para medir el error del clasificador.

Apéndice B

SVM

B.1. Condiciones KKT, Lagrangiano y Dual de Wolfe

A continuación se describen brevemente las condiciones KKT y el problema dual [60]. Este apéndice no intenta se exhaustivo ni explicativo, sino de referencia para el capítulo sobre SVM.

Los problemas de optimización en general consisten de una función f a optimizar, un objetivo (minimizar o maximizar) y un conjunto de restricciones sobre el dominio de f que indican que soluciones son posibles.

Para cada familia de funciones y restricciones existen técnicas para encontrar soluciones, distintas en naturaleza. Algunas técnicas son de más generalidad que otras.

Por ejemplo, los problemas de minimización (optimización, en general) de funciones $f \in \mathbb{C}^1$ y restricciones del tipo $h(x) = 0,\ h \in \mathbb{C}^1$ pueden tratarse mediante el conocido método de los **multiplicadores de Lagrange**. En dicho método se busca minimizar la función $\mathcal{L}(x, \lambda) = f(x) + \lambda h(x)$, ya que se sabe que si $f(x^*)$ es el mínimo de f, entonces existe λ^* tal que $\nabla\mathcal{L} = \frac{\partial\mathcal{L}}{\partial x^*, \lambda^*} = (0, 0)$. Esta es una condición necesaria, que limita el espacio de búsqueda y simplifica la tarea de encontrar un mínimo.

Entonces, se transforma el problema a otro que es, en cierto modo, equivalente, para simplificar la tarea original de optimización.

B.1.1. Condiciones KKT

El método de los multiplicadores de Lagrange no puede lidiar con problemas con restricciones de la forma $g(x) \leq 0$. Pero se puede generalizar con el concepto de las condiciones KKT. Para un problema de

optimización:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{Min}} & f(\boldsymbol{x}) \\ \text{sujeto a} & h_i(\boldsymbol{x}) = 0,\ i = 0, \ldots, n \\ & g_i(\boldsymbol{x}) \leq 0,\ i = 0, \ldots, m \end{array}$$

Asumiendo que f, g_i y h_i son funciones suaves en un entorno del óptimo $\boldsymbol{x}^*$, y que el subconjunto del dominio definido por las restricciones no es vacío, e introduciendo un Lagrangiano modificado que tome en cuenta ambos tipos de restricciones en la función de coste:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}) + \sum_{i=1}^{n} u_i g_i(\boldsymbol{x}), \quad u_i \geq 0$$

Las condiciones KKT para un mínimo local $\boldsymbol{x}^*$ del problema son que existan constantes $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)$ y $\boldsymbol{u} = (u_1, \ldots, u_n)$, llamadas multiplicadores KKT, tales que:

- Estacionalidad

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^*} = 0 \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}} = 0 \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}} = 0$$

- Feasibilidad Primal

$$\begin{array}{l} g_i(\boldsymbol{x}^*) \leq 0,\ i = 1, \ldots, n \\ h_i(\boldsymbol{x}^*) = 0,\ i = 1, \ldots, m \end{array}$$

- Feasibilidad dual

$$u_i \geq 0,\ i = 1, \ldots, m$$

- Holgura complementaria

$$u_i g_i(\boldsymbol{x}^*) = 0,\ i = 1, \ldots, n$$

Para que dicho punto $\boldsymbol{x}^*$ satisfaga las condiciones KKT, el problema deberá también satisfacer una condición de regularidad. Existen varias condiciones de regularidad posibles; la más simple para el problema de SVM es la condición de linealidad de las restricciones, o sea;

$$g_i(\boldsymbol{x}) \text{ es afín, } i = 1, \ldots, n$$
$$h_i(\boldsymbol{x}) \text{ es afín, } i = 1, \ldots, m$$

Estas son condiciones **necesarias** para cualquier óptimo del problema $\boldsymbol{x}^*$. Son además suficientes si la función objetivo f y las funciones de restricción g_i son diferenciables y convexas y las funciones h_i son afines.

B.1.2. Problema Dual

Para todo problema de optimización de estas características existe un **problema dual** asociado. El problema dual está relacionado al problema original, que se denomina **problema primal**, por las condiciones de optimalidad KKT. Tiene la siguiente forma:

$$\begin{aligned} &\underset{\boldsymbol{\lambda},\boldsymbol{u}}{\text{Max}} && q(\boldsymbol{\lambda}, \boldsymbol{u}) \\ &\text{sujeto a} && \lambda_i \geq 0,\ i = 0, \ldots, m \\ &\text{donde} && q(\boldsymbol{\lambda}, \boldsymbol{u}) = \underset{\boldsymbol{x}}{\text{Min}} f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}) + \sum_{i=1}^{n} u_i g_i(\boldsymbol{x}) = \underset{\boldsymbol{x}}{\text{Min}} \mathcal{L} \end{aligned}$$

Los multiplicadores KKT son las variables de este problema, llamadas **variables duales**. Los problemas primales y duales están además relacionados por el teorema de **dualidad débil**, consecuencia de las KKT. El mismo dice que para todo par de soluciones feasibles $\boldsymbol{x}$ y $(\boldsymbol{\lambda}, \boldsymbol{u})$ se cumple que $f(\boldsymbol{x}) \geq q(\boldsymbol{\lambda}, \boldsymbol{u})$.

En particular, para los dos óptimos $f(\boldsymbol{x}^*)$ y $q(\boldsymbol{\lambda}^*, \boldsymbol{u}^*)$ también se cumple, por ende $f(\boldsymbol{x}^*) \geq q(\boldsymbol{\lambda}^*, \boldsymbol{u}^*)$; a la cantidad $f(\boldsymbol{x}^*) - q(\boldsymbol{\lambda}^*, \boldsymbol{u}^*)$ se la conoce como **duality gap**, o hueco dual.

Otra condición, **dualidad fuerte** relaciona aún más los problemas primales y duales; en este caso, en el óptimo $f(\boldsymbol{x}^*) = q(\boldsymbol{\lambda}^*, \boldsymbol{u}^*)$ y entonces no hay hueco dual. De esta forma, si se pueden encontrar relaciones entre las variables del dual y del primal que permiten calcular las segundas en base a las primeras, se puede resolver el problema dual, y luego calcular el valor de la solución del primal $(\boldsymbol{x}^*)$ a partir de las variables duales (u_i^* y

λ_i^*). La condición más utilizada, que aplica al problema de programación cuadrática planteado por SVM, es que el problema sea convexo en sus restricciones y función objetivo, y que cumpla la condición de Slater [60].

B.2. Algoritmo de optimización SMO

La idea esencial del algoritmo **Sequential Minimal Optimization** (SMO), en base a la formulación dual, es partir de la solución "nula"$\alpha_i = 0,\ i = 1, \ldots, |D|$, e ir eligiendo pares de variables duales $(\alpha_i, \alpha_j),\ i \neq j$, optimizando $\mathcal{L}_D$ respecto de estas dos variables, hasta que la solución se haya mejorado al punto que las condiciones KKT se cumplan, con cierta tolerancia. Siendo $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_D)$ y kkt una función que calcula cuantas variables duales violan las KKT por más que un nivel ϵ, el esquema del algoritmo es:

Data:
Un conjunto de ejemplares $D \in \mathbb{R}^d$
Una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Una tolerancia a la violación de las KKT, ϵ
Result: Un hiperplano descripto implícitamente por las variables duales $\boldsymbol{\alpha} \in \mathbb{R}^{|D|}$ y un bias $b \in \mathbb{R}$
$\boldsymbol{\alpha} = (0, \cdots, 0)$;
$b = 0$;
while $kkt(\boldsymbol{\alpha}, \epsilon) > 0$ **do**
  Elegir α_i y α_j, $i \neq j$;
  Optimizar $\mathcal{L}_D$ respecto de α_i y α_j ;
end
Retornar b y $\boldsymbol{\alpha}$

Algoritmo 5: Esquema del algoritmo SMO

Se pueden considerar tres partes esenciales del algoritmo de manera independiente:

- En una sección anterior, se analizó la relación que implican las KKT entre los valores de las variables duales ($\alpha_i = 0$, $0 < \alpha_i < c$ y $\alpha_i = c$) con el error del clasificador ($h(\boldsymbol{x_i})y_i$). Cuando dicha relación no se mantiene, se detecta una violación de las KKT. De esta forma la función kkt puede contar cuantas variables violan las KKT. La verificación del cumplimiento de las KKT es suave, es decir, se permite cierto error ϵ, típicamente 10^{-2} o 10^{-3}. Verificaciones más estrictas

tienen el efecto de retrasar demasiado la convergencia, y no son típicamente necesarias en un clasificador de patrones.

- La elección de α_i y α_j se hace prefiriendo aquellas variables duales correspondientes a vectores de soporte que no cumplan las KKT. Primero se elige α_i, y luego α_j siguiendo ciertas heurísticas.
- La optimización de f respecto a α_i y α_j puede hacerse de forma analítica, ya que se considera solo a α_i y α_j como variables y al resto de las variables duales como constantes.

Se comenzará primero con la tercera parte del algoritmo, que constituye su esencia. Luego, se lidiará con los primeros dos items que contienen mucho en común, ya que ambos tienen la tarea de ver qué variables no cumplen las KKT y por ende se tratan juntos, mezclando la elección de los α_i y α_j con la condición de fin $kkt(\boldsymbol{\alpha}, \epsilon) > 0$.

B.2.1. Optimización de $\mathcal{L}_D$ con α_i y α_j

Por simplicidad, y sin pérdida de generalidad, en esta sección se asume que las variables duales elegidas para optimizar $\mathcal{L}_D$ son α_1 y α_2.

El esquema para buscar una regla de actualización de *au* y *ad* que mejore la solución es el siguiente.

Primero, se llevará α_1 a la forma $\alpha_1 = \gamma + s\alpha_2$ (γ y s constantes) tomando en cuenta que se está optimizando solo respecto a α_1 y α_2. Utilizando dicha igualdad, se reescribirá $\mathcal{L}_D$ en una expresión G que solo dependa de α_2.

De esa manera, se puede optimizar G (y por ende, $\mathcal{L}_D$) solamente respecto a α_2, y después simplemente calcular el valor de la otra. Luego, se derivará $\frac{\partial G}{\partial \alpha_2}$ para encontrar el máximo. Finalmente, se mostrará como ajustar el α_2 y α_1 considerando las restricciones relevantes sobre α_1 y α_2 provenientes de las KKT, y se calcula el nuevo valor de α_1.

Los detalles de la derivación para encontrar la expresión G se pueden encontrar en la sección *B*.2.5 de este apéndice, y el resultado es:

$$G = \frac{1}{2}\eta\alpha_2^2 + (y_2(E_1^v - E_2^v) - \alpha_2^v\eta)\alpha_2$$

Donde α_2^v es el valor viejo de la variable α_2 y α_2 es el nuevo valor, $E_i^v = f_i^v - y_i$ el error cometido por el clasificador con los valores viejos, f_i^v es la salida para $\boldsymbol{x_i}$ con los viejos parámetros y $\eta = -\kappa_{1,1} + 2\kappa_{1,2} - \kappa_{2,2}$

es una constante. De acuerdo al apéndice de la sección $B.2.5$, la expresión G es tal que $G = \mathcal{L}_D + Constante$, y por ende es equivalente a $\mathcal{L}_D$ para optimizar. Derivando respecto a α_2 para encontrar el óptimo:

$$\frac{\partial G}{\partial \alpha_2} = \eta\alpha_2 + y_2(E_1^v - E_2^v) - \alpha_2^v\eta$$
$$\frac{\partial^2 G}{\partial^2 \alpha_2} = \eta$$

Entonces, si $\frac{\partial G}{\partial \alpha_2} = 0$, se obtiene:

$$0 = \eta\alpha_2 + y_2(E_1^v - E_2^v) - \alpha_2^v\eta$$
$$\alpha_2 = \frac{-y_2(E_1^v - E_2^v) + \alpha_2^v\eta}{\eta} \quad (\eta \neq 0)$$
$$\alpha_2 = \alpha_2^v + \frac{-y_2(E_1^v - E_2^v)}{\eta}$$
$$\alpha_2 = \alpha_2^v + \frac{y_2(E_2^v - E_1^v)}{\eta}$$
$$\alpha_2 = \alpha_2^v + \frac{y_2(E_2^v - E_1^v)}{\eta}$$

Armados con esta última ecuación para el caso $(\eta \neq 0)$, que relaciona de forma simple α_2 con α_2^v, se procede ahora a buscar una regla de actualización de las variables α_2 y α_1 para mejorar la solución actual.

B.2.2. Regla de actualización para α_1 y α_2

Debe tenerse en cuenta que las variables α_2 y α_1 no pueden tomar cualquier valor posible ya que la nueva solución debe cumplir las condiciones KKT $0 \leq \alpha_i \leq c$ y $\sum_i \alpha_i y_i = 0$. A continuación, L y H denotarán los límites inferiores y superiores que puede tomar la variable α_2, cuya existencia se asume ya que se derivarán en la sección siguiente. Estos valores límite asegurarían que la nueva solución cumpla las dos restricciones mencionadas si se elige un nuevo α_2 tal que $\alpha_2 \in [L, H]$ y se actualiza α_1 con la regla $\alpha_1 \leftarrow \sigma - s\alpha_2$,

Por otro lado, la ecuación $\alpha_2 = \alpha_2^v + \frac{y_2(E_2^v - E_1^v)}{\eta}$ sólo tiene sentido si $\eta \neq 0$. Se puede ver que $\eta \leq 0$, ya que:

$$\eta = -\kappa_{1,1} + 2\kappa_{1,2} - \kappa_{2,2} = -(\kappa_{1,1} - 2\kappa_{1,2} + \kappa_{2,2})$$
$$= -(\boldsymbol{x}_2 - \boldsymbol{x}_1)^T(\boldsymbol{x}_2 - \boldsymbol{x}_1) = -\|\boldsymbol{x}_2 - \boldsymbol{x}_1\| \leq 0$$

Por ende deben manejarse dos casos: $\eta < 0$ y $\eta = 0$.

Si $\eta = 0$, $G = y_2(E_1^v - E_2^v)\alpha_2$ es una función lineal, y entonces su máximo se encuentra en alguno de los dos límites de los valores posibles para α_2, L o H. La regla sería entonces:

$$\alpha_2 \leftarrow \underset{\alpha_2 \in \{L,H\}}{\operatorname{argmax}} y_2(E_1^v - E_2^v)\alpha_2$$

Si $\eta < 0$, G es una función cuadrática en α_2. Revisando la ecuación $\alpha_2 = \alpha_2^v + \frac{y_2(E_2^v - E_1^v)}{\eta}$, el cambio necesario para estar en el máximo respecto a α_2 sería:

$$\Delta\alpha_2 = \frac{y_2(E_2^v - E_1^v)}{\eta}$$

Entonces, se puede utilizar la regla de actualización::

$$\alpha_2 \leftarrow \alpha_2^v + \Delta\alpha_2$$

Esta regla es preliminar porque, nuevamente, falta analizar los valores límites de α_2. Asumiendo la existencia de las constantes L y H, y dada una función $restrict_{L,H}(x) = min(max(L, x), H)$ que devuelve su argumento restringido al intervalo $[L, H]$, la regla quedaría como:

$$\alpha_2 \leftarrow restrict_{L,H}(\alpha_2^v + \Delta\alpha_2)$$

En resumen, la regla final de actualización sería:

$$\alpha_2 \leftarrow \begin{cases} \operatorname{argmax}_{\alpha_2 \in \{L,H\}} y_2(E_1^v - E_2^v)\alpha_2 & \text{si } \eta = 0 \\ restrict_{L,H}(\alpha_2^v + \Delta\alpha_2) & \text{si } \eta \leq 0 \end{cases}$$

A continuación se derivan expresiones para los límites L y H, y se trata el problema de la actualización de las variables E_i al finalizar un paso de optimización exitoso. Luego, se continúa con los dos puntos restantes: la elección de α_i y α_j y la condición de fin del algoritmo.

B.2.2.1. Límites para la variable dual α_2

Cualquier nueva solución debe cumplir $0 \leq \alpha_i \leq c$, pero los límites no son simplemente $L = 0$ y $H = c$ ya que debe cumplirse $\alpha_1 + s\alpha_2 = \gamma$ también. Por ende, para elegir un nuevo valor de α_2 (y α_1, en consecuencia) se deben mantener ambas relaciones. S derivarán los límites considerando los casos $s = 1$, y $s = -1$.

Si $s = 1$, entonces $\alpha_1^v + \alpha_2^v = \gamma$. Entonces, si $\alpha_2 > \gamma$, como $0 \leq \alpha_1$, no se puede hacer negativo a α_1 para restarle a α_2 y entonces no se puede cumplir la restricción.

Por otro lado, si $\alpha_2 < \gamma - c$, entonces α_1 debería ser mayor que c para llegar a γ. Entonces:

$$s = 1 \rightarrow \begin{cases} L = max(0, \gamma - c) \\ H = min(c, \gamma) \end{cases}$$

Si $s = -1$, entonces $\alpha_1 - \alpha_2 = \gamma$. Si $\gamma \geq 0$, entonces $-\alpha_2 \leq \gamma - c$ ya que sino $\alpha_1 - \alpha_2$ no podría alcanzar a γ, debido a que $\alpha_1 \leq c$. Entonces, $c - \gamma \leq \alpha_2$. Si $\gamma < 0$, entonces como sólo α_2 es negativo, $-\alpha_2 \leq \gamma$, o sea $\alpha_2 \geq -\gamma$. Por ende:

$$s = -1 \rightarrow \begin{cases} L = max(0, -\gamma) \\ sH = min(c, c - \gamma) \end{cases}$$

Definiendo funciones *pos* y *neg* tales que:

$$\begin{aligned} pos(1) &= 1 & \qquad neg(1) &= 0 \\ pos(-1) &= 0 & \qquad neg(-1) &= 1 \end{aligned}$$

y teniendo en cuenta la relación de los segundos argumentos a *max* y *min* con el signo de s, se pueden combinar ambos casos:

$$\begin{aligned} L &= max(0, s(\gamma - pos(s)c)) \\ H &= min(c, s(\gamma + neg(s)c)) \end{aligned}$$

B.2.2.2. Actualización luego de un paso de optimización exitoso

Cuando α_1 y α_2 se cambian con $\Delta\alpha_1$ y $\Delta\alpha_2$, se deben actualizar los E_i y b para que reflejen los nuevos valores de las variables duales.

El error E_i es:

$$E_i = h(\boldsymbol{x_i}) - y_i = \boldsymbol{w} \cdot \boldsymbol{x_i} + b - y_i = \sum_j a_j \kappa_{i,j} y_j + b - y_i$$

Se podrían calcular los E_i de esta manera, pero no se conoce el nuevo valor de b. Pero si $0 < \alpha_1 < c$ o $0 < \alpha_2 < c$, esto quiere decir que dichas variables, que se llamarán variables de soporte, corresponden a vectores de soporte y por ende $E_1 = 0$ o $E_2 = 0$, respectivamente. Entonces se puede despejar b con alguna de las siguientes ecuaciones:

$$b = -\sum_j a_j \kappa_{1,j} y_j + y_1 \text{ o}$$
$$b = -\sum_j a_j \kappa_{2,j} y_j + y_2$$

Luego, se calcula E_i para todas las variables duales, utilizando la fórmula anterior o el hecho de que:

$$\Delta E_i = \Delta\alpha_1 y_1 \kappa_{1,i} + \Delta\alpha_2 y_1 \kappa_{2,i} + \Delta b$$

Si $\alpha_1 = 0$ o $\alpha_1 = c$ y $\alpha_2 = 0$ o $\alpha_2 = c$, entonces no se puede calcular de forma exacta b con α_1 o α_2. Una opción es computar b con α_1 y α_2, y promediar los valores como una forma de aproximar el nuevo b.

B.2.3. Elección de las variables α_i y α_j y pseudocódigo del algoritmo

A continuación, se presenta un pseudocódigo del algoritmo, con tres módulos principales: el procedimiento principal, la función $findAjAndOptimize(i)$ y la función $optimize(i, j)$. El primero selecciona la primer variable dual para optimizar y verifica la condición de fin del algoritmo. La segunda busca una segunda variable dual adecuada para optimizar. La tercera realiza la optimización en base a las variables seleccionadas por los procedimientos previos. En esta parte, se dejan de utilizar las variables α_1 y α_2 como las seleccionadas, y se generaliza la notación utilizando α_i y α_j, respectivamente.

B.2.3.1. Procedimiento principal

La elección de las variables duales a optimizar se realiza escogiendo primero un α_i apropiado, y luego un $\alpha_j,\ j \neq i$ en base al α_i seleccionado (siempre que sea posible). El procedimiento principal de SMO es:

Data:
Un conjunto de ejemplares $D \in \mathbb{R}^d$
Una clase $y_i \in \{-1, 1\}$ asociada a cada $\boldsymbol{x_i} \in D$
Una tolerancia a la violación de las KKT, ϵ
Result: Un hiperplano descripto implícitamente por las variables duales $\boldsymbol{\alpha} \in \mathbb{R}^{|D|}$ y un bias $b \in \mathbb{R}$

```
global α = (0, ··· , 0) [a];
global b = 0 ;
global E = (−y_1, ··· , −y_n) ;
changed = false ;
examineAll = true ;
while changed OR examineAll do
    changed= false ;
    if examineAll then
        randomOffset=random(1,|D|) ;
        for index=1 to |D| do
            i= (index + randomOffset) % |D| ;
            changed = changed OR findAjAndOptimize(i) ;
        end
    else
        randomOffset=random(1,|D|) ;
        for i=1 to |D| do
            i= (index + randomOffset) % |D| ;
            if 0 < α_i < c then
                changed = changed OR findAjAndOptimize(i) ;
            end
        end
    end
    examineAll = NOT (changed OR examineAll) ;
end
Retornar b y α ;
```

Algoritmo 6: Procedimiento principal del algoritmo SMO

[a] Las variables global están en el scope de los otros procesos, para simplificar el código.

Donde la función $findAjAndOptimize(i)$ *intenta* optimizar G utilizando la variable α_i y devuelve *true* si pudo hacerlo o $false$ de lo contrario.

El algoritmo alterna entre intentar optimizar con las variables de soporte ($0 < \alpha_i < c$) e intentar con todas. La razón por la cual se prefiere utilizar las variables de soporte es que son las que definen realmente el hiperplano, y por ende cambiarlas brinda una chance mayor de converger rápidamente.

Se alterna entre estas dos estrategias ya que llamar a $findAjAndOptimize(i)$ con una variable no implica que se optimizará la función respecto a esa variable (ver la definición de $findAjAndOptimize(i)$) y puede que no haya variables de soporte (por ejemplo, al principio cuando todas las variables duales son 0).

El recorrido por las variables duales se realiza comenzando desde una posición aleatoria del vector de variables duales $\boldsymbol{\alpha}$ para evitar sesgar al algoritmo a favor de las primeras variables.

La variable *changed* se utiliza para detectar cuando no se pudo optimizar respecto a ninguna variable dual. Esto implica, según la definición de los otros módulos, que todas las variables cumplen las *kkt* a un nivel ϵ, condición de fin del algoritmo.

B.2.3.2. Función $findAjAndOptimize(i)$

Dado el primer α_i, se busca un α_j apropiado. Antes de eso,se debe verificar que α_i no cumpla las KKT. Para ello, se define $R_i = y_i E_i$, una variable fácil de computar, y se reescriben las KKT en términos de R_i:

$$\begin{aligned} \alpha_i = 0 &\rightarrow 0 \leq R_i \\ 0 < \alpha_i < c &\rightarrow R_i \simeq 0 \\ \alpha_i = c &\rightarrow R_i \leq 0 \end{aligned}$$

que, se pueden escribir más sintéticamente como:

$$(0 < \alpha_i \wedge R_i > 0) \vee (\alpha_i < c \wedge R_i < 0) \rightarrow \alpha_i \text{ no cumple las KKT}$$

Introduciendo una tolerancia, queda como:

$$(0 < \alpha_i \wedge R_i > \epsilon) \vee (\alpha_i < c \wedge R_i < -\epsilon) \rightarrow \alpha_i \text{ no cumple las KKT}$$

Si α_i viola dichas condiciones, se procede a buscar el α_j apropiado. Se utiliza una heurística que intenta tres cosas en sucesión:

- Primero, se busca un α_j que sea una variable de soporte, y tal que $|E_i - E_j|$ sea lo más grande posible. Dado que $\Delta\alpha_j = y_j(E_2 - E_1)/\eta$, esta heurística intenta que el cambio $\Delta\alpha_j$ sea lo más grande posible para que el algoritmo converja rápidamente.
- Si el primer criterio falla, se busca un α_j que sea una variable de soporte, por las mismas razones esgrimidas para elegir α_i.
- Por último, se intenta con cualquier α_j.

De nuevo, seleccionar un α_j con estos criterio no significa que con el par (α_i, α_j) se pueda optimizar G, ya que puede suceder que $L = H$ o que $\Delta\alpha_j$ sea muy pequeño, con lo cual el algoritmo no hace realmente un paso de optimización. Por último, en esta búsqueda también se comienza siempre desde una posición aleatoria del vector para no sesgar el algoritmo.

En pseudocódigo:

```
function findAndOptimize(i) ;
R_i = y_i * E_i ;
if (0 < α_i OR R_i > 0) AND ( α_i < c OR R_i < 0) then
    return false ;
end
if ∃α_j :   0 < α_j < c then
    randomOffset=random(1,|D|) ;
    max=0 ;
    for index=1 to |D| do
        j= (index + randomOffset) % |D| ;
        if 0 < α_j < c AND |E_i − E_j| >= max then
            max = |E_i − E_j| ;
            jmax= j ;
        end
    end
    if optimize(i,jmax) then
        return true;
    end
end
randomOffset=random(1,|D|) ;
for index=1 to |D| do
    j= (index + randomOffset) % |D| ;
    if 0 < α_j < c AND optimize(i,j) then
        return true;
    end
end
randomOffset=random(1,|D|) ;
for index=1 to |D| do
    j= (index + randomOffset) % |D| ;
    if optimize(i,j) then
        return true;
    end
end
return false;
```

Función findAndOptimize(i)

B.2.4. Función $optimize(i, j)$

Finalmente, se desarrolla el proceso de optimización de $\mathcal{L}_D$ respecto a α_i y α_j. Primero, se calculan las variables relevantes: H, L, s, γ, η, $\Delta\alpha_j$, $\Delta\alpha_i$, luego el nuevo valor de las variables α_i y α_j, y finalmente el nuevo valor de b y los E_i. Antes de eso, se verifica que el cambio a realizar sea significativo, como ya fue mencionado. Para ello, se necesita que $\Delta\alpha_j/\alpha_j$ sea más grande que un valor pequeño ϵ', a definir.

function optimize(i,j) ;
Calcular H, L, s, γ, η, $\Delta\alpha_j$, $\Delta\alpha_i$ según las fórmulas dadas anteriormente ;
if *($L \geq H$ OR $\Delta\alpha_j/\alpha_j < \epsilon$)* **then**
  return false ;
end
Actualizar α_i y α_j ;
Actualizar b ;
Actualizar $\boldsymbol{E}$;
return true ;

Función optimize(i,j)

Los cálculos y actualizaciones se hacen en base a las fórmulas de las secciones anteriores, y para un pseudocódigo más detallado se puede consultar el artículo original de Platt [59].

B.2.5. Derivación de la regla de actualización de SMO

A continuación se deriva la expresión G para llegar a la regla de actualización de SMO. Primero se derivará la igualdad $\alpha_1 + \alpha_2 s = \gamma$, con γ y s constantes. Esta igualdad servirá para expresar G sólo en términos de α_2, quitando a la variable α_1 de expresión a minimizar.

Las condición de optimalidad $\frac{\partial \mathcal{L}}{\partial b} = 0$ implica $\sum_i \alpha_i y_i = 0$. Como las únicas variables son α_1 y α_2, es mejor escribir:

$$0 = \sum_{i=3}^{n} \alpha_i y_i + \alpha_1 y_1 + \alpha_2 y_2$$

$$\alpha_1 y_1 + \alpha_2 y_2 = -\sum_{i=3}^{n} \alpha_i y_i = \gamma'$$

Donde $\gamma' = --\sum_{i=3}^{n} \alpha_i y_i$ es constante y $n = |D|$. Se tomará como convención que α_1 y α_2 son los valores *nuevos* de las variables duales, y

α_1^v y α_2^v los *viejos*; para los otros α_i esta distinción no es necesaria ya que no se modifican.

Definiendo $\gamma = y_1\gamma'$ y $s = y_1y_2$, se puede simplificar dicha ecuación:

$$\begin{aligned} \alpha_1 y_1 + \alpha_2 y_2 &= \gamma' \\ \alpha_1 + \alpha_2 y_2 y_1 &= \gamma' y_1 \\ \alpha_1 + \alpha_2 s &= \gamma' y_1 \\ \alpha_1 + \alpha_2 s &= \gamma \end{aligned}$$

De esta manera se obtiene la ecuación de una recta con ordenada γ y pendiente $s = \pm 1$. Como $\alpha_1 + \alpha_2 s = \gamma$, se puede expresar α_1 en términos de α_2. A continuación, se simplificará la función $\mathcal{L}_D$ para expresarla en términos de α_1. Partiendo de:

$$\mathcal{L}_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j (\alpha_i y_i \alpha_j y_j \kappa_{i,j})$$

Separando los términos que dependen de α_1 y α_2 del resto de las variables duales:

$$
\begin{aligned}
\mathcal{L}_D &= -\frac{1}{2}(\sum_i \sum_j \alpha_i y_i \alpha_j y_j \kappa_{i,j}) + \sum_i \alpha_i \\
&= \frac{1}{2}(-\alpha_1^2 \kappa_{1,1} y_1^2 - \alpha_2^2 \kappa_{2,2} y_2^2 - 2\alpha_1 \alpha_2 \kappa_{1,2} y_1 y_2 \\
&- 2\alpha_1 y_1 \sum_{j=3}^{n} \kappa_{1,j} y_j \alpha_j - 2\alpha_2 y_2 \sum_{j=3}^{n} \kappa_{2,j} y_j \alpha_j \\
&- \sum_{i=3}^{n} \sum_{j=3}^{n} \alpha_i \alpha_j \kappa_{i,j} y_i y_j) \\
&+ \sum_{i=3}^{n} \alpha_i + \alpha_1 + \alpha_2 \\
&= -\frac{1}{2}\alpha_1^2 \kappa_{1,1} - \frac{1}{2}\alpha_2^2 \kappa_{2,2} - \alpha_1 \alpha_2 \kappa_{1,2} s \\
&- \alpha_1 y_1 \sum_{j=3}^{n} K_{1,j} y_j \alpha_j - \alpha_2 y_2 \sum_{j=3}^{n} \kappa_{2,j} y_j \alpha_j \\
&- \frac{1}{2} \sum_{i=3}^{n} \sum_{j=3}^{n} \alpha_i \alpha_j \kappa_{i,j} y_i y_j \\
&+ \sum_{i=3}^{n} \alpha_i + \alpha_1 + \alpha_2
\end{aligned}
$$

Reemplazando $(y_i)^2 = 1$ y $y_1 y_2 = s$, y dado que el máximo de $\mathcal{L}_D$ ahora no depende de los valores de $\alpha_3, \cdots, \alpha_n$, se pueden quitar los términos constantes y optimizar:

$$
\begin{aligned}
&= -\frac{1}{2}\alpha_1^2 \kappa_{1,1} - \frac{1}{2}\alpha_2^2 \kappa_{2,2} - \alpha_1 \alpha_2 s \kappa_{1,2} \\
&- \alpha_1 y_1 \sum_{j=3}^{n} \kappa_{1,j} y_j \alpha_j - \alpha_2 y_2 \sum_{j=3}^{n} \kappa_{2,j} y_j \alpha_j \\
&+ \alpha_1 + \alpha_2
\end{aligned}
$$

Ahora, reescribiendo con $\alpha_1 = \gamma - s\alpha_2$ para expresar $\mathcal{L}_D$ solamente en términos de la variable α_2:

$$
\begin{aligned}
&= -\frac{1}{2}(\gamma - s\alpha_2)^2\kappa_{1,1} - \frac{1}{2}\alpha_2^2\kappa_{2,2} - (\gamma - s\alpha_2)\alpha_2 s\kappa_{1,2} \\
&- (\gamma - s\alpha_2)y_1 \sum_{j=3}^{n} \kappa_{1,j}y_j\alpha_j - \alpha_2 y_2 \sum_{j=3}^{n} \kappa_{2,j}y_j\alpha_j \\
&+ (\gamma - s\alpha_2) + \alpha_2
\end{aligned}
$$

Sabiendo que:

$$
\begin{aligned}
s^2 &= 1 \\
(\gamma - s\alpha_2)^2 &= \gamma^2 - 2\gamma s\alpha_2 + \alpha_2^2 \\
sy_1 &= y_2
\end{aligned}
$$

Y definiendo $v_i = \sum_{j=3}^{n} \alpha_j y_j \kappa_{i,j}$, se obtiene:

$$
\begin{aligned}
&= -\frac{1}{2}\gamma^2\kappa_{1,1} + \gamma s\alpha_2\kappa_{1,1} - \frac{1}{2}\alpha_2^2\kappa_{1,1} \\
&- \frac{1}{2}\alpha_2^2\kappa_{2,2} - \alpha_2\gamma s\kappa_{1,2} + \alpha_2^2\kappa_{1,2} \\
&- \gamma y_1 v_1 + \alpha_2 y_2 v_1 - \alpha_2 y_2 v_2 \\
&+ \gamma + \alpha_2(1 - s)
\end{aligned}
$$

Descartando términos constantes $-\gamma^2\kappa_{1,1}$, $-\gamma y_1 v_1$ y γ:

$$
\begin{aligned}
&= \gamma s\alpha_2\kappa_{1,1} - \frac{1}{2}\alpha_2^2\kappa_{1,1} \\
&- \frac{1}{2}\alpha_2^2\kappa_{2,2} - \gamma s\alpha_2\kappa_{1,2} + \alpha_2^2\kappa_{1,2} \\
&+ \alpha_2 y_2 v_1 - \alpha_2 y_2 v_2 \\
&+ \alpha_2(1 - s)
\end{aligned}
$$

Sacando factor común α_2^2 y α_2:

$$
\begin{aligned}
&= \alpha_2^2(-\frac{1}{2}\kappa_{1,1} + \kappa_{1,2} - \frac{1}{2}\kappa_{2,2}) \\
&+ \alpha_2(\gamma s\kappa_{1,1} - \gamma s\kappa_{1,2} + y_2 v_1 - y_2 v_2 + (1 - s))
\end{aligned}
$$

Sea $\eta = -\kappa_{1,1} + 2\kappa_{1,2} - \kappa_{2,2}$, y agrupando y_2:

$$= \frac{1}{2}\alpha_2^2\eta + \alpha_2(\gamma s\kappa_{1,1} - \gamma s\kappa_{1,2} + y_2(v_1 - v_2) + 1 - s)$$

Para simplificar el coeficiente de α_2, se puede utiliza el hecho de que $\gamma = \alpha_1^v + s\alpha_2^v$:

$$\begin{aligned}
v_i &= \sum_{j=3}^{n} \alpha_j y_j \kappa_{i,j} = (\sum_{j=3}^{n} \alpha_j y_j \phi(\mathbf{x_j})) \cdot \phi(\boldsymbol{x_i}) \\
&= (\sum_{j} \alpha_j y_j \phi(\mathbf{x_j}) - \alpha_1^v y_1 \phi(\boldsymbol{x}_1) - \alpha_2^v y_2 \phi(\boldsymbol{x}_2)) \cdot \phi(\boldsymbol{x_i}) \\
&= (\boldsymbol{w} - \alpha_1^v y_1 \phi(\boldsymbol{x}_1) - \alpha_2^v y_2 \phi(\boldsymbol{x}_2)) \cdot \phi(\boldsymbol{x_i}) \\
&= \boldsymbol{w} \cdot \phi(\boldsymbol{x_i}) - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i} \\
&= (\boldsymbol{w} \cdot \phi(\boldsymbol{x_i}) + b^v) - b^v - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i} \\
&= f_i^v - b^v - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i}
\end{aligned}$$

Donde f_i^v es la salida para $\boldsymbol{x_i}$ bajo el $\boldsymbol{w}^v$ correspondiente a los viejos parámetros. Entonces, el coeficiente de α_2 es:

$$
\begin{aligned}
&\gamma s\kappa_{1,1} - \gamma s\kappa_{1,2} + y_2(v_1 v_2) + 1 - s \\
&= (\alpha_1^v + s\alpha_2^v)s\kappa_{1,1} - (\alpha_1^v + s\alpha_2^v)s\kappa_{1,2} + y_2(v_1 - v_2) + 1 - s \\
&= \alpha_1^v s\kappa_{1,1} + \alpha_2^v \kappa_{1,1} - \alpha_1^v s\kappa_{1,2} - \alpha_2^v \kappa_{1,2} \\
&+ y_2(v_1 - v_2) + 1 - s \\
&= \alpha_1^v s\kappa_{1,1} + \alpha_2^v \kappa_{1,1} - \alpha_1^v s\kappa_{1,2} - \alpha_2^v \kappa_{1,2} \\
&+ y_2(f_1^v - b^v - \alpha_1^v y_1 \kappa_{1,1} - \alpha_2^v y_2 \kappa_{1,2} - f_2^v + b^v + \alpha_1^v y_1 \kappa_{1,2} + \alpha_2^v y_2 \kappa_{2,2}) + 1 - s \\
&= \alpha_1^v s\kappa_{1,1} + \alpha_2^v \kappa_{1,1} - \alpha_1^v s\kappa_{1,2} - \alpha_2^v \kappa_{1,2} \\
&+ y_2(f_1^v - \alpha_1^v y_1 \kappa_{1,1} - \alpha_2^v y_2 \kappa_{1,2} - f_2^v + \alpha_1^v y_1 \kappa_{1,2} + \alpha_2^v y_2 \kappa_{2,2}) + 1 - s \\
&= \alpha_1^v s\kappa_{1,1} + \alpha_2^v \kappa_{1,1} - \alpha_1^v s\kappa_{1,2} - \alpha_2^v \kappa_{1,2} \\
&+ y_2(f_1^v - f_2^v) - \alpha_1^v s\kappa_{1,1} - \alpha_2^v \kappa_{1,2} + \alpha_1^v s\kappa_{1,2} + \alpha_2^v \kappa_{2,2} + 1 - s \\
&= \alpha_1^v s\kappa_{1,1} - \alpha_1^v s\kappa_{1,2} + \alpha_1^v s\kappa_{1,2} - \alpha_1^v s\kappa_{1,1} \\
&+ \alpha_2^v \kappa_{1,1} - \alpha_2^v \kappa_{1,2} - \alpha_2^v \kappa_{1,2} + \alpha_2^v \kappa_{2,2} \\
&+ y_2(f_1^v - f_2^v) + 1 - s \\
&= \alpha_2^v(\kappa_{1,1} - 2\kappa_{1,2} + \kappa_{2,2}) \\
&+ y_2(f_1^v - f_2^v) + 1 - s \\
&= y_2(f_1^v - f_2^v) + 1 - s - \alpha_2^v \eta \\
&= y_2(f_1^v - f_2^v) + y_2 y_2 - y_1 y_2 - \alpha_2^v \eta \\
&= y_2(f_1^v - f_2^v + y_2 - y_1) - \alpha_2^v \eta \\
&= y_2((f_1^v - y_1) - (f_2^v - y_2)) - \alpha_2^v \eta
\end{aligned}
$$

Siendo $E_i^v = f_i^v - y_i$ el error cometido por el clasificador con los valores viejos, el coeficiente de α_2 es entonces:

$$= y_2(E_1^v - E_2^v) - \alpha_2^v \eta$$

Entonces, la expresión a optimizar, G, queda como:

$$G = \frac{1}{2}\eta\alpha_2^2 + (y_2(E_1^v - E_2^v) - \alpha_2^v \eta)\alpha_2$$

Apéndice C

Redes neuronales

C.1. Backpropagation

En esta sección se desarrollarán las derivadas del error para lograr una expresión concreta de la regla de actualización de los pesos de una red con backpropagation.

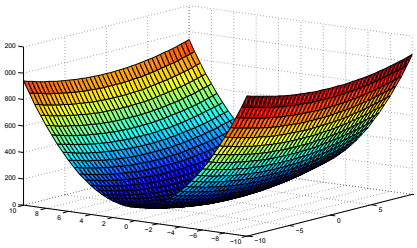
Dado que el desarrollo de las $\frac{\partial E}{\partial \boldsymbol{W}^j}$ suele traer confusión debido a la repetida aplicación de la regla de la cadena para volver para atrás por las capas de la red, se llegará a una fórmula general en tres pasos: primero, con un simple ejemplo de una red de dos capas con neuronas lineales; luego con una red de tres capas, de gran uso en la práctica, y luego de forma genérica para una red de $L+1$ capas.

C.1.1. Backpropagation con una red de dos capas y neuronas lineales

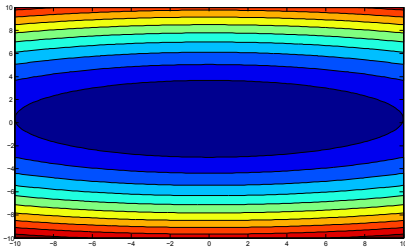
C.1.1.1. Derivación de $\frac{\partial E}{\partial \boldsymbol{W}^j}$:

Como ejemplo, supóngase una red con dos capas, una de entrada y una de salida, con dos neuronas de entrada y una de salida cuya función de activación es la identidad $\theta(x) = x$. Se asume también un conjunto de datos D con un solo ejemplar, $\boldsymbol{x} = (1,3)$ y salida esperada $(0,1)$. Entonces, la función error está definida en base a los dos pesos de la red, $\boldsymbol{w}_1^1(1), \boldsymbol{w}_1^1(2)$ que se llamarán w_1 y w_2 por simplicidad. La función de error para esta red es:

$$E(w_1, w_2) = (x_1 w_1 - \boldsymbol{y}_1)^2 + (x_2 w_2 - \boldsymbol{y}_2)^2$$
$$E(w_1, w_2) = (1w_1 - 0)^2 + (3w_2 - 1)^2$$



(a) Superficie de la función de error $E(w_1, w_2)$.



(b) Curvas de nivel de la función de error $E(w_1, w_2)$.

El opuesto del gradiente de la función indica la dirección del espacio de parámetros que minimiza E:

$$-\nabla E = -(\frac{\partial E}{\partial w1}, \frac{\partial E}{\partial w2}) = -((x_1 w_1 - \boldsymbol{y}_1)x_1 + (x_2 w_2 - \boldsymbol{y}_2)x_2)$$
$$= -(\frac{\partial E}{\partial w1}, \frac{\partial E}{\partial w2}) = -((w_1 - 0) + (3w_2 - 1)3)$$

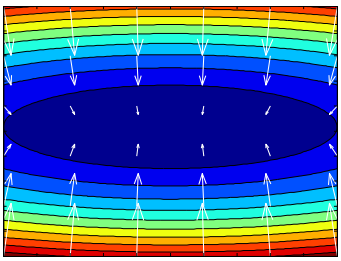


Figura C.2: Opuesto del gradiente de la función de error $E(w_1, w_2)$.

C.1.1.2. Algoritmo:

Siguiendo el ejemplo, la esencia del algoritmo backpropagation consiste en, dado una posición inicial en el espacio de parámetros, w_1^i y w_2^i, moverse de a pequeños pasos en la dirección que indica el gradiente, hasta llegar a una posición del espacio de parámetros en donde el error sea lo

suficientemente bajo, es decir, un mínimo local de la función que sea lo suficientemente bajo.

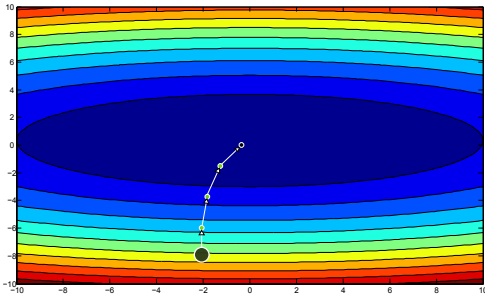


Figura C.3: Camino desde una posición inicial w_1^i y w_2^i hasta una posición con poco error en el espacio de parámetros.

Para el ejemplo, el algoritmo sería:
Data:
Un ejemplar $\boldsymbol{x} = (1, 3) \in \mathbb{R}^2$, con clase $\boldsymbol{y} = (0, 1)$
Una tolerancia al error ϵ
Una tasa de aprendizaje α
Result: Un vector de parámetros $\boldsymbol{w} = (w_1, w_2)$ optimizado para clasificar $\boldsymbol{x}$ con un error menor a ϵ
$\boldsymbol{w} = random(2)$;
$error = \infty$;
while $error > \epsilon$ **do**
  $w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$;
  $w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2}$;
  $error = E(w_1, w_2)$;
end
Retornar $\boldsymbol{w}$;

Algoritmo 7: Esquema del algoritmo Backpropagation para el problema de ejemplo.

Las líneas $w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$ del algoritmo implementan el movimiento en el espacio de parámetros en la dirección en que se minimiza el error.

La tasa de aprendizaje, α, es positiva y suele tener valores cercanos a 0, como 0.1, y se incluye para lograr que la magnitud de cada movimiento en el espacio de parámetros sea pequeña, para evitar que el algoritmo pase de largo un mínimo de la función.

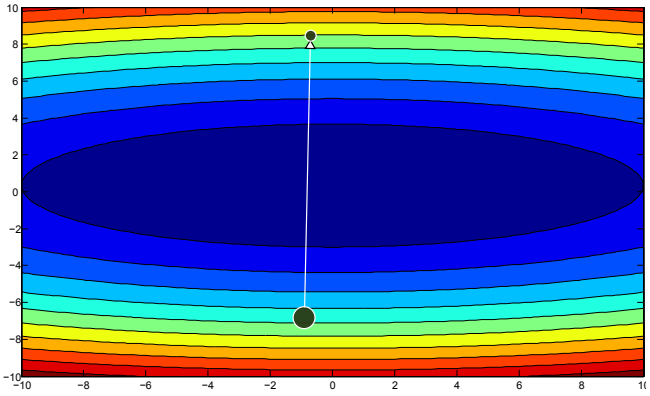


Figura C.4: Un paso con magnitud demasiado grande en la minimización de una función puede hacer que el algoritmo pase de largo un mínimo local.

En este caso, la superficie del error es convexa y por ende tiene un sólo mínimo, por lo cual es de esperar que se obtenga dicho mínimo si la tasa de aprendizaje es lo suficientemente baja, pero esto no es cierto del caso general en una red con varias capas, neuronas y funciónes de activación arbitrarias.

Generalizando este algoritmo para un conjunto de ejemplares D, se puede escribir el opuesto del gradiente del error de un ejemplar $\boldsymbol{x}$ con clase $\boldsymbol{y}$ como:

$$\begin{aligned} -\nabla E(\boldsymbol{x}) &= -(\frac{\partial E(\boldsymbol{x})}{\partial w1}, \frac{\partial E(\boldsymbol{x})}{\partial w2}) \\ &= -((x_1 w_1 - y_1)x_1, (x_2 w_2 - y_2)x_2) \end{aligned}$$

Data:
Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, con $D \subset \mathbb{R}^2$, y clase $\boldsymbol{y_i} \in \mathbb{R}^2$
Una tolerancia al error ϵ
Una tasa de aprendizaje α
Result: Un vector de parámetros $\boldsymbol{w} = (w_1, w_2)$ optimizado para clasificar D con un error menor a ϵ

$\boldsymbol{w} = random(2)$;
$error = \infty$;
while $error > \epsilon$ **do**
  $dw_1 = 0$;
  $dw_2 = 0$;
  for $\boldsymbol{x} \in D$ **do**
    $dw_1 = dw_1 + (x_1 w_1 - y_1)x_1$;
    $dw_2 = dw_2 + (x_2 w_2 - y_2)x_2$;
  end
  $w_1 = w_1 - \alpha dw_1$;
  $w_2 = w_2 - \alpha dw_2$;
  $error = E(w_1, w_2)$;
end
Retornar $\boldsymbol{w}$;

Algoritmo 8: Esquema del algoritmo Backpropagation para el problema de ejemplo con un conjunto de datos arbitrario D.

Esta forma de entrenar la red, acumulando en dw_i las derivadas de los errores para todos los patrones, se conoce como **entrenamiento batch** o por épocas. También se podrían ir actualizando los w_i cada vez que se calcula la derivada del error para cada $\boldsymbol{x}$, por ejemplo, eligiendo los $\boldsymbol{x}$ en orden aleatorio del conjunto de datos (de manera similar a como funciona el algoritmo de entrenamiento del perceptrón del capítulo 1). Este tipo de entrenamiento se conoce como **stochastic** o estocástico. El entrenamiento batch es más fácil de analizar desde el punto de vista teórico, y por ende existen resultados que proveen ciertas cotas de tiempo o error para su entrenamiento, pero en la práctica el entrenamiento estocástico puede ser más veloz y menos susceptible a mínimos locales, especialmente para conjuntos de datos muy grandes. La razón detrás de esto es que si bien calcular el gradiente del error en base a un sólo ejemplar es una estimación del mismo, posiblemente con bastante ruido: si el conjunto de ejemplares es muy grande, existirá mucha redundancia y por ende varios gradientes serán similares, con lo cual no tiene sentido calcular el gradiente sobre *todos* los ejemplares; el ruido puede ser beneficioso ya que le otorga cierto grado de exploración aleatoria al algoritmo [66]. Entonces, en la práctica,

se puede utilizar algún término medio, como mini-batches, que calculan el gradiente con un pequeños subconjuntos de los ejemplares, que van cambiando iteración a iteración. Por simplicidad se utiliza el entrenamiento batch para la descripción de los algoritmos, pero pueden adaptarse fácilmente al caso estocástico.

Continuando con la generalización, se puede formular este algoritmo para una red neuronal feedforward de 3 capas y una función de activación arbitraria y $\mathbb{C}^1$.

C.1.2. Derivación de $\frac{\partial E}{\partial W^j}$ con una red de tres capas

C.1.2.1. Derivación de $\frac{\partial E}{\partial W^j}$:

Las redes con tres capas son populares debido a que se ha probado que son aproximadoras universales, es decir, pueden aproximar cualquier función continua $f :: \mathbb{R}^n \mapsto \mathbb{R}^m$ con un grado de error arbitrariamente bajo, dada la cantidad suficiente de neuronas ocultas y funciones de activación apropiadas [52]. Además, dan buenos resultados en la práctica, y son ampliamente utilizadas.

Para ellas, se tienen solo dos capas con pesos para optimizar, la 1, oculta, y la 2, de salida. Entonces, se tendrán que derivar sólo dos reglas de actualización, una para los pesos de cada capa.

Comenzando con la más simple, que involucra la derivada de la salida o_m^2 respecto a un peso $w_i^2(p)$ para un patrón $\boldsymbol{x}$ [1]. Como el vector de pesos $\boldsymbol{w}_i^2$ no interviene en el cálculo de o_m^2 para $i \neq m$:

$$\frac{\partial o_m^2}{\partial w_i^2(p)} = \begin{cases} \frac{\partial o_i^2}{\partial w_i^2(p)} & \text{si } i = m \\ 0 & \text{de lo contrario} \end{cases}$$

Si $i = m$, continua la derivación:

[1] De nuevo, se omitió el $\boldsymbol{x}$ en $o_m^2(\boldsymbol{x})$ por simplicidad de notación.

$$\begin{aligned}
\frac{\partial o_i^2}{\partial w_i^2(p)} &= \frac{\partial \theta_2(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2)}{\partial w_i^2(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2) \frac{\partial \boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2}{\partial w_i^2(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2) \frac{\partial \sum_{l=1}^{n_1} o_l^1 w_i^2(l)}{\partial w_i^2(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2) \frac{\partial (o_p^1 w_i^2(p))}{\partial w_i^2(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_i^2) o_p^1
\end{aligned}$$

Entonces, la derivada del error para la neurona m de la capa de salida y el patrón $\boldsymbol{x}$ respecto a $w_i^2(p)$ es:

$$\begin{aligned}
\frac{\partial E_m(\boldsymbol{x})}{\partial w_i^2(p)} &= \frac{\partial (\frac{1}{2}(o_m^2 - y_m)^2)}{\partial w_i^2(p)} \\
&= (o_m^2 - y_m) \frac{\partial o_m^2}{\partial w_i^2(p)} \\
&= \begin{cases} (o_i^2 - y_i) \frac{\partial o_i^2}{\partial w_i^2(p)} & \text{si } i = m \\ 0 & \text{de lo contrario} \end{cases}
\end{aligned}$$

Y la derivada total del error para la capa de salida dado un patrón $\boldsymbol{x}$:

$$\frac{\partial E(\boldsymbol{x})}{\partial w_i^2(p)} = \sum_{m=1}^{n_2} \frac{\partial E_m(\boldsymbol{x})}{\partial w_i^2(p)} = \frac{\partial E_i(\boldsymbol{x})}{\partial w_i^2(p)}$$

Para los pesos de la capa 1, se obtiene la derivada de la salida o_m^2 respecto a un peso $w_i^1(p)$ para un patrón $\boldsymbol{x}$:

$$\begin{aligned}
\frac{\partial o_m^2}{\partial w_i^1(p)} &= \frac{\partial \theta_2(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2)}{\partial w_i^1(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) \frac{\partial \boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2}{\partial w_i^1(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) \frac{\partial \sum_{l=1}^{n_1} o_l^1 w_m^2(l)}{\partial w_i^1(p)} \\
&= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) \sum_{l=1}^{n_1} w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)}
\end{aligned}$$

Expandiendo $\frac{\partial o_l^1}{\partial w_i^1(p)}$, de forma análoga a como se expandió $\frac{\partial o_m^2}{\partial w_i^2(p)}$:

$$\frac{\partial o_l^1}{\partial w_i^1(p)} = \begin{cases} \frac{\partial o_i^1}{\partial w_i^1(p)} & \text{si } i = l \\ 0 & \text{de lo contrario} \end{cases}$$

Y entonces si $i = l$:

$$\begin{aligned}
&= \frac{\partial \theta_1(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1)}{\partial w_i^1(p)} \\
&= \theta_1'(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) \frac{\partial \boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1}{\partial w_i^1(p)} \\
&= \theta_1'(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) \frac{\partial \sum_{q=1}^{n_1} o_q^0 w_i^1(q)}{\partial w_i^1(p)} \\
&= \theta_1'(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) o_p^0 \\
&= \theta_1'(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) x_p
\end{aligned}$$

Es importante notar la similaridad entre las expresiones $\frac{\partial o_i^1}{\partial w_i^1(p)}$ y $\frac{\partial o_i^2}{\partial w_i^2(p)}$, ya que de esa manera se puede luego generalizar la regla de actualización a $L+1$ capas.

Volviendo al caso de 3 capas, la expresión para $\frac{\partial o_m^2}{\partial w_i^1(p)}$ es entonces:

$$\begin{aligned}\frac{\partial o_m^2}{\partial w_i^1(p)} &= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) \sum_{l=1}^{n_1} w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)} \\ &= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)} \\ &= \theta_2'(\boldsymbol{o}^1 \cdot \boldsymbol{w}_m^2) w_m^2(l) \theta_1'(\boldsymbol{o}^0 \cdot \boldsymbol{w}_i^1) x_p\end{aligned}$$

La derivada del error para la neurona de salida m respecto al peso $w_i^1(p)$ es entonces:

$$\begin{aligned}\frac{\partial E_m(\boldsymbol{x})}{\partial w_i^1(p)} &= \frac{\partial(\frac{1}{2}(o_m^2 - y_m)^2)}{\partial w_i^1(p)} \\ &= (o_m^2 - y_m) \frac{\partial o_m^2}{\partial w_i^1(p)}\end{aligned}$$

Y la derivada total del error para la capa de salida dado un patrón $\boldsymbol{x}$:

$$\frac{\partial E(\boldsymbol{x})}{\partial w_i^1(p)} = \sum_{m=1}^{n_2} \frac{\partial E_m(\boldsymbol{x})}{\partial w_i^1(p)}$$

Finalmente, las reglas de actualización para los pesos de las neuronas de las capas 1 y 2, respectivamente, son:

$$\begin{aligned}w_i^1(p) &\leftarrow w_i^1(p) - \alpha \sum_{\boldsymbol{x} \in D}^{max} \frac{\partial E(\boldsymbol{x})}{\partial w_i^1(p)} \\ w_i^2(p) &\leftarrow w_2^1(p) - \alpha \sum_{\boldsymbol{x} \in D} \frac{\partial E(\boldsymbol{x})}{\partial w_2^1(p)}\end{aligned}$$

C.1.2.2. Algoritmo

Para el algoritmo, se utilizarán como variables las matrices de pesos de cada capa llamándolas $\boldsymbol{W}^1$ y $\boldsymbol{W}^2$, y matrices de derivadas de esos pesos $\boldsymbol{dW}^1$ y $\boldsymbol{dW}^2$, del mismo tamaño, y se asume que $\frac{\partial E}{\partial \boldsymbol{W}^1}$ calcula una matriz de derivadas del error respecto a cada peso de la capa 1 y para

todos los ejemplares, y lo mismo con $\frac{\partial E}{\partial \boldsymbol{W}^2}$. Entonces, $\boldsymbol{W}^j(i,p) = \boldsymbol{w}_i^j(p)$ y $\frac{\partial E}{\partial \boldsymbol{W}^j(i,p)} = \frac{\partial E}{\partial \boldsymbol{w}_i^j(p)}$.

Modificando el algoritmo visto antes para actualizar ahora los pesos de cada capa, se obtiene:

Data:
Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, con $D \subset \mathbb{R}^d$, y clase $\boldsymbol{y_i} \in \mathbb{R}^{n_2}$.
Una tolerancia al error ϵ.
Una tasa de aprendizaje α.
Una cantidad de neuronas ocultas n_1.
Result: Pesos $\boldsymbol{W}^1$ y $\boldsymbol{W}^2$ de cada neurona oculta $\boldsymbol{w}_i^1$ y de salida $\boldsymbol{w}_i^2$, optimizados para clasificar D con un error menor a ϵ

$\boldsymbol{W}^1 = random(n_1, d)$;
$\boldsymbol{W}^2 = random(n_2, n_1)$;
$error = \infty$;
while $error > \epsilon$ **do**
  $\boldsymbol{dW}^1 = \frac{\partial E}{\partial \boldsymbol{W}^1}$;
  $\boldsymbol{dW}^2 = \frac{\partial E}{\partial \boldsymbol{W}^2}$;
  $\boldsymbol{W}_1 = \boldsymbol{W}_1 - \alpha \boldsymbol{dW}_1$;
  $\boldsymbol{W}_2 = \boldsymbol{W}_2 - \alpha \boldsymbol{dW}_2$;
  $error = E(\boldsymbol{W}^1, \boldsymbol{W}^2)$;
end
Retornar $\boldsymbol{W}^1, \boldsymbol{W}^2$;

Algoritmo 9: Esquema del algoritmo Backpropagation para una red de tres capas.

C.1.3. Derivación de $\frac{\partial E}{\partial \boldsymbol{W}^j}$ para cualquier topología de red

Se puede escribir la forma genérica de la derivada de la salida de cada neurona i' de cada capa j' y para un ejemplar particular $\boldsymbol{x}$, respecto a un peso $w_i^j(p)$, $j' >= j$ [2]:

[2]El peso $w_i^j(p)$ es el componente p del vector de pesos $\boldsymbol{w}_i^j \in \mathbb{R}^{n_{j-1}}$ para la conexiones de la capa $j-1$ a la neurona i de la capa j.

$$\begin{aligned}
\frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} &= \frac{\partial \theta_{j'}(net_{i'}^{j'})}{\partial w_i^j(p)} \\
&= \frac{\partial \theta_{j'}(net_{i'}^{j'})}{\partial net_{i'}^{j'}} \frac{\partial net_{i'}^{j'}}{\partial w_i^j(p)} \\
&= \theta'_{j'}(net_{i'}^{j'}) \frac{\partial net_{i'}^{j'}}{\partial w_i^j(p)} \\
&= \theta'_{j'}(net_{i'}^{j'}) \frac{\partial \boldsymbol{w}_{i'}^{j'} \cdot \boldsymbol{o}^{j'-1}}{\partial w_i^j(p)} \\
&= \theta'_{j'}(net_{i'}^{j'}) \sum_{l=1}^{n_{(j'-1)}} \frac{\partial (w_{i'}^{j'}(l) o_l^{j'-1})}{\partial w_i^j(p)}
\end{aligned}$$

Esta ecuación tiene resolución directa si el peso $w_i^j(p)$ corresponde a una neurona de la misma capa que la salida $o_{i'}^{j'}$. Entonces, si $i' = i$ y $j' = j$:

$$\frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} = \theta'_{j'}(net_{i'}^{j'}) \frac{\partial w_{i'}^{j'}(p) o_{p'}^{j'-1}}{\partial w_{i'}^{j'}(p)} = \theta'_{j'}(net_{i'}^{j'}) o_{p'}^{j'-1}$$

Si $j' = j$ pero $i' \neq i$, el peso es de otra neurona de la misma capa, y por ende no influye en la salida $o_{i'}^{j'}$:

$$\frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} = \theta'_{j'}(net_{i'}^{j'}) 0 = 0$$

Por último, si $j' > j$, quiere decir que el peso corresponde a una neurona de capas anteriores, y entonces ese peso afecta todas las entradas de la neurona actual. Por ende, se debe utilizar la regla de la cadena con todas estas entradas; como estas entradas son salidas de otras neuronas, se puede utilizar la regla recursiva:

$$\frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} = \theta'_{j'}(net_{i'}^{j'}) \sum_{l=1}^{n_{(j'-1)}} w_{i'}^{j'}(l) \frac{\partial o_l^{j'-1}}{\partial w_i^j(p)}$$

Y entonces la derivada del error para la neurona de salida o_i^k, dado un ejemplar $\boldsymbol{x}$, es:

$$\frac{\partial E_i(\boldsymbol{x})}{\partial w_i^j(p)} = \frac{\partial(\frac{1}{2}(o_{i'}^p - y_{i'})^2)}{\partial w_i^j(p)} = (o_{i'}^{j'} - y_{i'}) \frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)}$$

Y la derivada del error para la capa de salida respecto a $w_i^j(p)$:

$$\frac{\partial E(\boldsymbol{x})}{\partial w_i^j(p)} = \sum_{i=1}^{n_k} \frac{\partial E_i(\boldsymbol{x})}{\partial w_i^j(p)}$$

Donde como ahora hay una fórmula genérica para $\frac{\partial o_{i'}^{j'}}{\partial w_i^j(k)}$ se puede expandirla de acuerdo a la topología de la red.

Entonces, existe una manera de calcular la derivada del error para cualquier peso, $\frac{\partial E}{\partial w_i^j(p)} = \sum_{\boldsymbol{x_i} \in D} \frac{\partial E(\boldsymbol{x_i})}{\partial w_i^j(p)}$, y se puede aplicar la regla de actualización $w_i^j(p) \leftarrow w_i^j(p) - \alpha \frac{\partial E}{\partial w_i^j(p)}$.

C.1.3.1. Algoritmo

Finalmente, se repite aquí el algoritmo clásico de entrenamiento del backpropagation, ya presentado en el capítulo 3:

Data:
Una toplogía dada por $L+1$, el número de capas, y n_i la cantidad de neuronas de la capa i, con $n_L = C$.
Un conjunto de ejemplares $\boldsymbol{x_i} \in D$, con $D \subset \mathbb{R}^d$, y clase $\boldsymbol{y_i} \in \mathbb{R}^C$.
Una tolerancia al error ϵ.
Una tasa de aprendizaje α.
Result: Pesos $\boldsymbol{W}^j$ de las neuronas de cada capa $j > 0$, optimizados para clasificar D con un error menor a ϵ
for *j=1 to L* **do**
  $\boldsymbol{W}^1 = random(n_j, n_{j-1})$;
end
$error = \infty$;
while $error > \epsilon$ **do**
  for *j=1 to L* **do**
    $\boldsymbol{dW}^j = \frac{\partial E}{\partial \boldsymbol{W}^j}$;
  end
  for *j=1 to L* **do**
    $\boldsymbol{W}_j = \boldsymbol{W}_j - \alpha \boldsymbol{dW}_j$;
  end
  $error = E(\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L)$;
end
Retornar $\boldsymbol{W}^1, \ldots, W^L$;

Algoritmo 10: Esquema del algoritmo Backpropagation para una red de tres capas.

C.2. Consideraciones para aplicar Backpropagation

C.2.1. Overfitting

Las redes neuronales son propensas a overfitting como cualquier modelo de aprendizaje automático. Hay dos ejes principales a trabajar para evitarlo.

El primero, es utilizar un **conjunto de validación** para evaluar la con-

dición de corte del algoritmo. En la línea del algoritmo:

$$error = E(\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L)$$

En esta condición se asume que el error se calcula sobre los patrones del conjunto de entrenamiento D_e. Una mejora posible es subdividir el conjunto de entrenamiento D_e en dos; el conjunto de entrenamiento real D'_e, y otro de validación, D_v, de forma similar a como se divide el conjunto de datos D en D_e y D_v. El conjunto de validación se utilizar para calcular el error de la red neuronal en cada iteración. Esta es una forma de regularización, cuya justificación es que dado que se entrena el modelo con D'_e, el error calculado sobre D_v es un mejor estimador del error total sobre el dominio del problema $\mathcal{P}$ ya que el modelo está sesgado por D'_e pero no por D_v (de todas maneras, luego de la primer iteración, como la verificación del error se realiza sobre D_v hay un pequeño sesgo, pero mucho menor que el original).

El segundo es restringir el número de neuronas de las capas ocultas, y la cantidad de capas. A medida que aumenta la cantidad de neuronas ocultas de una red, aumenta su poder de aproximación. Si bien a priori esto parece algo bueno, salvo por el coste computacional adicional, en realidad resulta perjudicial ya que de esa manera se logra que el modelo imite tan bien a los datos que no pueda generalizar.

En términos de una red de dos capas con funciones de activación identidad, esto se asemeja a utilizar tantos hiperplanos para separar el conjunto original que prácticamente se podría clasificar cualquier cosa sin ninguna importancia por la coherencia espacial que tenga la superficie de separación resultante.

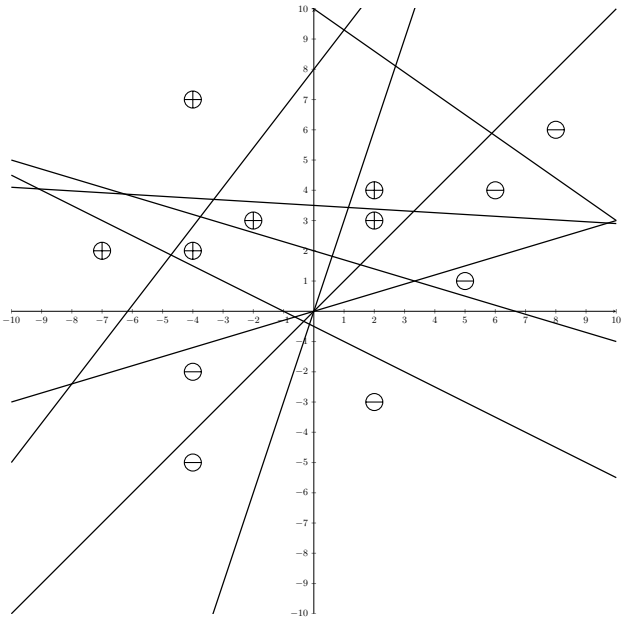


Figura C.5: Hiperplanos definidos por las neuronas ocultas de una red con demasiadas neuronas para la dificultad inherente del problema.

Por otro lado, utilizar menos hiperplanos de separación que los requeridos por el problema puede tener el efecto de no poder modelarlo adecuadamente por falta de poder de aproximación.

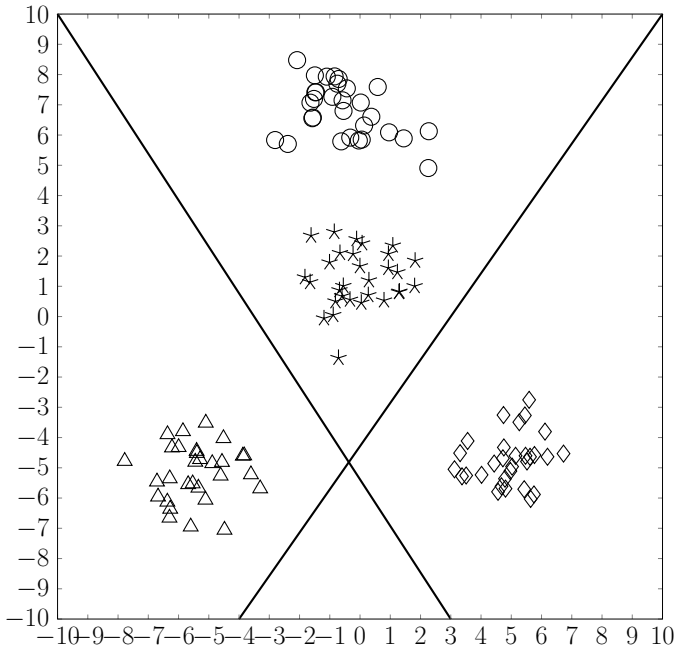


Figura C.6: Hiperplanos definidos por las neuronas ocultas de una red con pocas neuronas para la dificultad inherente del problema.

En la práctica, deben probarse distintos valores para la cantidad de neuronas ocultas de cada capa, con el objetivo de obtener un término medio entre modelización adecuada y overfitting. Dichos valores dependerán del problema y las funciones de las neuronas.

C.2.2. Mínimos locales

Como en la mayoría de los problemas de optimización, un problema ocurrente en el entrenamiento de las redes neuronales es que la función de error suele no suele ser convexa y entonces no hay garantía de que la dirección que indica el gradiente conduzca a un mínimo global del error.

Dependiendo de la función de activación, el error puede ser una función de muy poca suavidad y por ende tener una gran cantidad de mínimos locales. Algunos de estos mínimos locales representan soluciones que si bien no son las mejores, resultan aceptables para el conjunto de datos de entrenamiento. Cuando un mínimo no es una solución aceptable, se considera un lugar a evitar ya que el algoritmo backpropagation puede quedarse atascado[3].

[3]Se llama *paso* o *movimiento* en el espacio de parámetros a un cambio de los pesos w

En esencia, el problema de los mínimos locales es el de poder determinar una dirección óptima a moverse para evitar los mínimos locales con un error arriba del umbral tolerable, y una magnitud óptima para ese movimiento utilizando información local, de manera que se llegue a un mínimo global.

Por un lado, es importante poder determinar la magnitud óptima para actualizar un peso en la dirección en que indica el gradiente, de manera que se llegue a un mínimo lo más rápido posible. Por otro lado, es importante que se pueda salir de un mínimo local si el error que tiene esa posición es mayor que el umbral tolerable.

Existen varias técnicas para estimar la magnitud del cambio de los parámetros, y para recorrer el espacio de parámetros de forma más eficiente. Salvo la primera, la idea de estas técnicas es asegurarse que el movimiento en el espacio de parámetros sea más certero, a costa de utilizar más tiempo de ejecución o memoria para calcular la dirección y magnitud de ese movimiento [66, 52]:

- **Varias inicializaciones aleatorias**: Se corre el algoritmo varias veces, cada vez comenzando desde una posición inicial aleatoria distinta, entrenando varias redes, y quedándose con la mejor en el proceso. Se espera que alguna de las redes logre llegar a un mínimo aceptable.

- **Tasas de aprendizaje no constantes:** Es deseable que la magnitud del cambio de las derivadas varie con el tiempo, de manera que al principio la magnitud sea grande y se explore el espacio de manera agresiva, y a medida que pasen las iteraciones la magnitud sea más chica, haciendo menos cambios, y buscando optimizar el error en una región más pequeña.

- **Tasas de aprendizaje individuales:** Las tasas de aprendizaje puede variar para cada peso w, por ejemplo para hacer más agresivo la actualización del un peso w cuando $\frac{\partial E}{\partial w}$ es muy grande, y más chico cuando hay poco error respecto a ese peso. A su vez, se puede aumentar la granularidad haciéndolas variar para cada capa, o grupo arbitrario de neuronas. En general, es deseable que las tasas de las capas más bajas sean más altas que las finales, ya que sino el cambio en los primeros pesos afecta *poco* la salida de la red.

---

de la red, siguiendo la forma de la regla de actualización de backpropagation, debido a que esta regla se puede interpretar como *caminar* en el espacio de parámetros en alguna dirección.

- **Métodos de segundo orden:** Además de calcular el gradiente del error, se puede calcular o aproximar el Hessiano $\frac{\partial^2 E}{\partial^2 \boldsymbol{W}}$ para obtener información de segundo orden sobre la dirección del error. La esencia de la idea es que si la función está acelerando, es probable que sea mejor hacer un cambio pequeño, ya que el gradiente está creciendo y dentro de poco esa dirección puede hacer crecer el error. Por el contrario, dado un hessiano negativo, la función está acelerando negativamente y tiene sentido tomar pasos más grandes.

- **Momentum:** Se puede moverse en la dirección en la que apunta, no el gradiente actual, sino un promedio de los últimos u gradientes. Entonces, si en la iteración t se calcula la derivada para un peso w, $\frac{\partial E}{\partial w}^t$, se puede guardar el mismo en una tabla, y luego se puede calcular la derivada promedio de las últimas u iteraciones, $E^u = \sum_{i=t-u}^{t} \alpha_{t-i} \frac{\partial E}{\partial w}^i$, y usar este valor para actualizar w. En esta fórmula, α_{t-i} es una tasa de aprendizaje para pesar de forma distinta las derivadas anteriores, de acuerdo a su importancia. Esto evita movimientos innecesarios cuando el gradiente cambia de signo constantemente, ya que el promedio de las últimas derivadas tiende a ser más estable.

- **Obviar la magnitud del gradiente:** La dirección del gradiente indica el camino para minimizar el error. La magnitud del gradiente, en cambio, depende de la magnitud de los datos de entrenamiento, la posición en el espacio de parámetros y el tipo de función de activación, por ende no siempre representa adecuadamente la magnitud del paso óptimo a dar para moverse en el espacio de parámetros. El algoritmo resilient backpropagation, a describir luego, utiliza esta perspectiva y estima la magnitud del cambio de forma independiente de la magnitud del gradiente.

C.2.3. Algoritmo de entrenamiento Resilient Backpropagation

La idea principal del algoritmo Resilient Backpropagation, o Rprop [96] [4], como ya fue mencionado, es no utilizar la magnitud de la derivada del error para actualizar cada peso w, sino utilizar solamente la dirección que indica el signo de dicha derivada.

[4] En esta sección, se desarrolla desarrollando la variante iRprop- de Rprop, una mejora del mismo introducida en [97, 98].

En esta sección se utilizará una notación más simple para numerar los pesos, escribiendo simplemente w para denotar a un peso cualquiera, asumiendo un conjunto de pesos $\boldsymbol{W}$ de la red y alguna correspondencia entre esos pesos a las posiciones exactas en la red. Por último, se escribe w^t para denotar al valor del peso w en la iteración t.

Hay dos ideas importantes, además de la de ignorar la magnitud de la derivada: evitar cambios innecesarios cuando la derivada fluctúa, y calcular una magnitud o velocidad de actualización dinámicamente para cada peso.

- Actualización en base al signo de la derivada

  En lugar de utilizar la magnitud de la derivada del error, en cada iteración t para cada peso w^t se computa dinámicamente una velocidad Δ_w^t, y se mueve w^t en la dirección opuesta a la que indica la derivada, en un paso con magnitud Δ_w^t, generando w^{t+1} es decir:

$$\Delta w^t \leftarrow -sign(\frac{\partial E(t)}{\partial w^t})\Delta_w^t$$
$$w^{t+1} \leftarrow w^t + \Delta w^t$$

- Evitar fluctuaciones

  La segunda idea importante en el algoritmo es tratar de no hacer cambios cuando la derivada fluctúa para evitar pasos innecesarios. De esta manera se obtienen tres reglas de actualización adicionales a la anterior para ciertos casos.

  - La regla anterior de actualización solamente se aplica cuando la derivada no cambió de signo respecto a la iteración anterior, o sea $\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}} > 0$, ya que se busca un incremento continuo en una sola dirección.
  - Si cambia el signo de la derivada, $\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}} < 0$, y aumenta el error, $E(t) > E(t-1)$, se asume que el último cambio de w es producto de una fluctuación de la derivada. En ese caso, se deshace ese cambio con la regla:

$$\Delta w^t \leftarrow -\Delta w^{t-1}$$
$$w^{t+1} \leftarrow w^t + \Delta w^t$$

Además, si $\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}} < 0$ se pone una variable, $reset_w$ en *true* para que en la próxima iteración se obvie la dirección de la derivada en la iteración anterior, y se mueva siempre en la nueva dirección de la derivada a la velocidad Δ_w^t.

- Si en una iteración se encuentra que $reset_w = true$, o sea, que en la iteración anterior se determinó que debería comenzar una búsqueda en una nueva dirección de acuerdo al punto anterior, se da un paso en la dirección del gradiente actual mediante la regla:

$$\begin{aligned} \Delta w^t &\leftarrow -sign(\frac{\partial E(t)}{\partial w^t})\Delta_w^t \\ w^{t+1} &\leftarrow w^t + \Delta w^t \\ reset_w &\leftarrow false \end{aligned}$$

  Las variables $reset_w$ se vuelven a poner en $false$ para continuar la búsqueda con normalidad en la próxima iteración. Además, se inician en *true* para todo w al comienzo del algoritmo.

- Si $\frac{\partial E(t)}{\partial w^t} = 0$, entonces $\Delta w^t \leftarrow 0$ y no se realiza ningún cambio para ese peso en esa iteración ya que se probablemente se encuentre en un mínimo local; de todas maneras, como la derivada depende de la salida de otras partes de la red, en la siguiente iteración la derivada puede no ser 0 aunque w no cambie. Si bien en ese caso no hay cambio, por uniformidad se puede escribir la regla de actualización como:

$$\begin{aligned} \Delta w^t &\leftarrow 0 \\ w^{t+1} &\leftarrow w^t + \Delta w^t \end{aligned}$$

- Si $\frac{\partial E(t)}{\partial w^t} \neq 0$ y $\frac{\partial E(t-1)}{\partial w^{t-1}} = 0$, se realiza un paso normalmente a la velocidad Δ_w^t como antes, al igual que cuando $reset_w = true$, asumiendo una re-inicialización que requiere una nueva dirección.

$$\begin{aligned} \Delta w^t &\leftarrow 0 \\ w^{t+1} &\leftarrow w^t + \Delta w^t \end{aligned}$$

- Cálculo de velocidad independiente por peso

  La tercera idea importante del algoritmo radica en la manera en que cambia la velocidad.

  El algoritmo tiene dos parámetros de límites de velocidad, una velocidad máxima $0 < \Delta_{max}$, y una velocidad mínima $0 < \Delta_{min} < \Delta_{max}$ para que no se alcancen velocidades demasiado grandes o chicas en magnitud; inicialmente, $\Delta_w^0 = \Delta_{min}$ para todas las velocidades.

  Otros dos parámetros controlan la aceleración cuando la derivada tiene el mismo signo que en la iteración anterior o no, η^+ y η^-, respectivamente.

  La regla de actualización de la velocidad es la siguiente: si la derivada mantiene su signo entre dos iteraciones, se acelera; si cambia, se desacelera; si la derivada es o fue 0 en la iteración pasada, se mantiene la velocidad. Entonces, siendo $s = \frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}}$:

$$\Delta_w^{t+1} \leftarrow \begin{cases} max(\Delta_w^t \eta^+, \Delta_{max}) & \text{si} \quad s > 0 \\ min(\Delta_w^t \eta^-, \Delta_{min}) & \text{si} \quad s < 0 \\ \Delta_w^t & \text{si} \quad s = 0 \end{cases}$$

Juntando estas tres ideas se puede escribir la regla de actualización completa para cada peso w del algoritmo en la iteración t (recordando que para todo w, $reset_W$ se inicializa en *true* y $\Delta_w^1 = \Delta_{min}$). Esta regla calcula

los pesos w^{t+1} a partir de los datos de la iteración t y $t-1$:

for $w \in W$ **do**

$s \leftarrow sign(\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}})$;

$$\Delta_w^t \leftarrow \begin{cases} min(\Delta_w^t \eta^+, \Delta_{max}) & \text{si } s > 0 \\ max(\Delta_w^t \eta^-, \Delta_{min}) & \text{si } s < 0 \\ \Delta_w^t & \text{si } s = 0 \end{cases};$$

if $(\ \frac{\partial E(t)}{\partial w^t} \neq 0\ AND\ \frac{\partial E(t-1)}{\partial w^{t-1}} = 0\ )\ OR\ reset_w$ **then**

$\Delta w^t \leftarrow -sign(\frac{\partial E(t)}{\partial w^t})\Delta_w^t$;

$reset_w \leftarrow false$

else if $s > 0$ **then**

$\Delta w^t \leftarrow -sign(\frac{\partial E(t)}{\partial w^t})\Delta_w^t$;

else if $s < 0$ **then**

if $E(t) > E(t-1)$ **then**

$\Delta w^t \leftarrow -\Delta w^{t-1}$

end

else

$\Delta w^t \leftarrow 0$

end

$reset_w \leftarrow true$

else

{ Caso $\frac{\partial E(t)}{\partial w^t} = 0$ }

$\Delta w^t \leftarrow 0$;

end

$w^{t+1} \leftarrow w^t + \Delta w^t$;

end

Algoritmo 11: Regla de actualización de los pesos del algoritmo Rprop

Donde dada la uniformidad de las 4 reglas descriptas anteriormente, se quitó el $w^{t+1} \leftarrow w^t + \Delta w^t$ de cada condicional.

Bibliografía

[1] Brad A Myers. A brief history of human-computer interaction technology. *interactions*, 5(2):44–54, 1998.

[2] Malcolm R Davis and TO Ellis. The rand tablet: a man-machine graphical communication device. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, pages 325–331. ACM, 1964.

[3] Thomas O Ellis, John F Heafner, and William L Sibley. The grail project. *RAND*, 1969.

[4] Michael L. Coleman. Text editing on a graphic display device using hand-drawn proofreader's symbols. In *Pertinent Concepts in Computer Graphics: Proceedings of the 2nd University of Illinois Conference on Computer Graphics*, pages 282–290. ICCG, 1969.

[5] Marion Madeira. *Gesture recognition and the use of touch sensitive color displays for simple diagramming.* PhD thesis, Massachusetts Institute of Technology, 1978.

[6] Richard A. Bolt. Put-that-there: Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA, 1980. ACM.

[7] Thomas G Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. In *ACM SIGCHI Bulletin*, volume 18, pages 189–192. ACM, 1987.

[8] Randy Pausch and Ronald D Williams. Giving candy to children: user-tailored gesture input driving an articulator-based speech synthesizer. *Communications of the ACM*, 35(5):58–66, 1992.

[9] Shinichi Tamura and Shingo Kawasaki. Recognition of sign language motion images. *Pattern Recognition*, 21(4):343 – 353, 1988.

[10] Junji Yamato, Jun Ohya, and Kenichiro Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE, 1992.

[11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *In In CVPR, 2011. 3*, 2011.

[12] Jhilmil Jain, Arnold Lund, and Dennis Wixon. The future of natural user interfaces. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 211–214, New York, NY, USA, 2011. ACM.

[13] BH Juang and Lawrence R Rabiner. *Automatic speech recognition–a brief history of the technology development*. PhD thesis, University of California at Santa Barbara, 2005.

[14] MA Anusuya and Shriniwas K Katti. Speech recognition by machine, a review. *arXiv preprint arXiv:1001.2267*, 2010.

[15] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. ƶour word is my command": Google search by voice: A case study. In *Advances in Speech Recognition*, pages 61–90. Springer, 2010.

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[17] Rashmi Makhijani, Urmila Shrawankar, and Vilas M. Thakare. Opportunities & challenges in automatic speech recognition. *CoRR*, abs/1305.2846, 2013.

[18] G Heigold, V Vanhoucke, A Senior, P Nguyen, M Ranzato, M Devin, and J Dean. Multilingual acoustic models using distributed deep neural networks. In *ICASSP*. ICASSP, 2013.

[19] Li Deng and Xuedong Huang. Challenges in adopting speech recognition. *Communications of the ACM*, 47(1):69–75, 2004.

[20] Anirban Chowdhury, Rithvik Ramadas, and Sougata Karmakar. Muscle computer interface: A review. In *ICoRD'13*, pages 411–421. Springer, 2013.

[21] Mikhail A Lebedev and Miguel AL Nicolelis. Brain–machine interfaces: past, present and future. *TRENDS in Neurosciences*, 29(9):536–546, 2006.

[22] José del R Millán. Brain-computer interfaces. *Introduction to Neural Engineering for Motor Rehabilitation*, 40:237, 2013.

[23] Sara Kiesler and Tom Finholt. The mystery of rsi. *American Psychologist*, 43(12):1004, 1988.

[24] Katy Keller, Julie Corbett, and Diane Nichols. Repetitive strain injury in computer keyboard users: pathomechanics and treatment principles in individual and group intervention. *Journal of Hand Therapy*, 11(1):9–26, 1998.

[25] Rhonda Epstein, Sean Colford, Ethan Epstein, Brandon Loye, and Michael Walsh. The effects of feedback on computer workstation posture habits. *Work: A Journal of Prevention, Assessment and Rehabilitation*, 41(1):73–79, 2012.

[26] Satish Saroshe, Suraj Sirohi, Amitsingh Pawaiya, and Gunjan Taneja. Assessment of nature and type of repetitive strain injury among software professionals. *National Journal of Medical Research*, 2(4):404–406, 2012.

[27] David Coggon et al. Disabling musculoskeletal pain in working populations: Is it the job, the person, or the culture? *{PAIN}*, 154(6):856 – 863, 2013.

[28] Tin Kai Chen. *An investigation into alternative human-computer interaction in relation to ergonomics for gesture interface design*. PhD thesis, De Montfort University, 2009.

[29] Dharani Perera. Voice recognition technology for visual artists with disabilities in their upper limbs. In *Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future*, OZCHI '05, pages 1–6, Narrabundah, Australia, Australia, 2005. Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

[30] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[31] Claude E Shannon. Computers and automata. *Proceedings of the IRE*, 41(10):1234–1241, 1953.

[32] Andrew N Meltzoff, Patricia K Kuhl, Javier Movellan, and Terrence J Sejnowski. Foundations for a new science of learning. *science*, 325(5938):284–288, 2009.

[33] Rudolf Lioutikov. Machine learning and the brain. *Technological University Darmstad*, 2012.

[34] Bernard Widrow, David E Rumelhart, and Michael A Lehr. Neural networks: Applications in industry, business and science. *Communications of the ACM*, 37(3):93–105, 1994.

[35] Ruixi Yuan, Zhu Li, Xiaohong Guan, and Li Xu. An svm-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*, 12(2):149–156, 2010.

[36] Yeou-Ren Shiue, Ruey-Shiang Guh, and Ken-Chun Lee. Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach. *International Journal of Production Research*, 50(20):5887–5905, 2012.

[37] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with applications*, 19(2):125–132, 2000.

[38] Holger R Maier and Graeme C Dandy. Neural networks for the prediction and forecasting of water resources variables: a review of modelling issues and applications. *Environmental modelling & software*, 15(1):101–124, 2000.

[39] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[40] Dean A Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In *Robot learning*, pages 19–43. Springer, 1993.

[41] Mohamed Azlan Hussain. Review of the applications of neural networks in chemical process control-simulation and online implementation. *Artificial Intelligence in Engineering*, 13(1):55–68, 1999.

[42] Akbar Rahideh, AH Bajodah, and M Hasan Shaheed. Real time adaptive nonlinear model inversion control of a twin rotor mimo system using neural networks. *Engineering Applications of Artificial Intelligence*, 25(6):1289–1297, 2012.

[43] Thomas Abeel, Thibault Helleputte, Yves Van de Peer, Pierre Dupont, and Yvan Saeys. Robust biomarker identification for cancer diagnosis with ensemble feature selection methods. *Bioinformatics*, 26(3):392–398, 2010.

[44] Christoph Lehmann, Thomas Koenig, Vesna Jelic, Leslie Prichep, Roy E John, Lars-Olof Wahlund, Yadolah Dodge, and Thomas Dierks. Application and comparison of classification algorithms for recognition of alzheimer's disease in electrical brain activity (eeg). *Journal of neuroscience methods*, 161(2):342–350, 2007.

[45] Gagan S Sidhu, Nasimeh Asgarian, Russell Greiner, and Matthew RG Brown. Kernel principal component analysis for dimensionality reduction in fmri-based diagnosis of adhd. *Frontiers in systems neuroscience*, 6, 2012.

[46] Andrea Finke, Alexander Lenhardt, and Helge Ritter. The mindgame: a p300-based brain–computer interface game. *Neural Networks*, 22(9):1329–1333, 2009.

[47] Dorothée Lulé, Quentin Noirhomme, Sonja C Kleih, Camille Chatelle, Sebastian Halder, Athena Demertzi, Marie-Aurélie Bruno, Olivia Gosseries, Audrey Vanhaudenhuyse, Caroline Schnakers, et al. Probing command following in patients with disorders of consciousness using a brain–computer interface. *Clinical Neurophysiology*, 2012.

[48] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209*, 2011.

[49] Quoc V Le, Will Y Zou, Serena Y Yeung, and Andrew Y Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern*

Recognition (CVPR), 2011 IEEE Conference on, pages 3361–3368. IEEE, 2011.

[50] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE*, 2013.

[51] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. *ICASSP 2013*, 2013.

[52] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[53] Rosemarie Velik. Discrete fourier transform computation using neural networks. In *Computational Intelligence and Security, 2008. CIS'08. International Conference on*, volume 1, pages 120–123. IEEE, 2008.

[54] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[55] Kate A Smith. Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

[56] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[57] Vladimir N Vapnik. Statistical learning theory. 1998.

[58] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[59] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[60] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[61] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964.

[62] Richard Bellman and R. Kalaba. On adaptive control processes. *Automatic Control, IRE Transactions on*, 4(2):1–9, 1959.

[63] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[64] Anthony N Burkitt. A review of the integrate-and-fire neuron model: Ii. inhomogeneous synaptic input and network properties. *Biological cybernetics*, 95(2):97–112, 2006.

[65] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris Jr, et al. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398, 2007.

[66] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[67] Inés Galván León and Pedro Isasi Viñuela. *Redes Neuronales Artificales: Un enfoque práctico*. Pearson Prentice-Hall, 2004.

[68] Teuvo Kohonen. *Self-organizing maps*, volume 30. Springer, 2001.

[69] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[70] David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

[71] Nina C Capone and Karla K McGregor. Gesture development: A review for clinical and research practices. *Journal of Speech, Language and Hearing Research*, 47(1):173, 2004.

[72] Autumn B Hostetter and Martha W Alibali. Visible embodiment: Gestures as simulated action. *Psychonomic bulletin & review*, 15(3):495–514, 2008.

[73] Susan Goldin-Meadow and Sian L Beilock. Action's influence on thought: The case of gesture. *Perspectives on Psychological Science*, 5(6):664–674, 2010.

[74] Jana M Iverson and Esther Thelen. Hand, mouth and brain. the dynamic emergence of speech and gesture. *Journal of Consciousness Studies*, 6(11-12):11–12, 1999.

[75] David Mcneill. *Hand and mind: What gestures reveal about thought*. University of Chicago Press., 1992.

[76] Michael Studdert-Kennedy. A review of mcneill's hand and mind: What gestures reveal about thought. *Haskins Laboratories Status Report on Speech Research*, 1993.

[77] M. A. Moni and A. B. M. Shawkat Ali. HMM based hand gesture recognition: A review on techniques and approaches. 2009.

[78] Arpita Ray Sarkar, G. Sanyal, and S. Majumder. Article: Hand gesture recognition systems: A survey. *International Journal of Computer Applications*, 2013.

[79] Ankit Chaudhary, Jagdish Lal Raheja, Karen Das, and Sonia Raheja. Intelligent approaches to interact with machines using hand gesture recognition in natural way: a survey. *arXiv preprint arXiv:1303.2292*, 2013.

[80] Sait Celebi, Ali Selman Aydin, Talha Tarik Temiz, and Tarik Arici. Gesture Recognition Using Skeleton Data with Weighted Dynamic Time Warping. In *Computer Vision Theory and Applications*. Visapp, 2013.

[81] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[82] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[83] Volodymyr V. Kindratenko. On using functions to describe the shape. *J. Math. Imaging Vis.*, 18(3), May 2003.

[84] Liu Yun and Zhang Peng. An automatic hand gesture recognition system based on viola-jones method and svms. In *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, volume 2, pages 72–76. IEEE, 2009.

[85] O. Patsadu, C. Nukoolkit, and B. Watanapa. Human gesture recognition using kinect camera. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 28–32, 2012.

[86] Sait Celebi, Ali S Aydin, Talha T Temiz, and Tarik Arici. Gesture recognition using skeleton data with weighted dynamic time warping. *Computer Vision Theory and Applications. Visapp*, 2013.

[87] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[88] Richard Maclin and David Opitz. Popular ensemble methods: An empirical study. *arXiv preprint arXiv:1106.0257*, 2011.

[89] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[90] Facundo Quiroga and Leonardo Corbalan. A novel competitive neural classifier for gesture recognition with small training sets. In *Proceedings of the XIX Argentine Congress on Computer Science (CACIC), Inteligent Systems and Agents Workshop (WASI)*, pages 140–149. RedUNCI, 2013.

[91] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-Validation. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009.

[92] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[93] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Mach. Learn.*, 52(3):239–281, September 2003.

[94] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *The Journal of Machine Learning Research*, 5:1089–1105, 2004.

[95] Abraham Wald. Statistical decision functions. 1950.

[96] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.

[97] Christian Igel and Michael Husken. Improving the rprop learning algorithm, 2000.

[98] Christian Igel and Michael Husken. Empirical evaluation of the improved rprop learning algorithms, 2003.