

img/unlp.jpg

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

20 de noviembre de 2013

Índice general

1. Resumen	1
2. Introducción	3
I Métodos	5
3. Aprendizaje Automático	7
3.1. Un ejemplo: Reconocimiento de Gestos	7
3.2. Aplicaciones	8
3.3. Entrenamiento Supervisado y No Supervisado	9
3.4. Clasificación	9
3.4.1. El Perceptrón: un clasificador simple	9
3.4.2. Experimentos de clasificación	13
3.4.3. Generalización	14
3.4.4. Validación cruzada	15
3.4.5. Selección del modelo de clasificación (SI HAY TIEMPO)	17
3.4.6. Sobre-especialización y regularización	17
3.4.7. Modelo de clasificación con estructura probabilística	18
3.4.8. Modelo de clasificación multiclase	19
4. Maquinas de Vectores de Soporte (SVM)	21
4.1. Clasificador de Márgen Máximo	21
4.2. Modelo de Márgenes Suaves	22
4.3. Forma Dual	23
4.4. Análisis de los valores de α_i	26
4.5. El truco del Kernel	26
4.6. Algoritmo de entrenamiento SMO	27
4.6.1. Optimización de \mathcal{L}_D	28
4.6.2. Regla de actualización para α_1 y α_2	31
4.6.3. Elección de las variables α_i y α_j y pseudocódigo del algoritmo	32
4.7. Adaptación para el reconocimiento multiclase	34
4.8. SVM: Apéndice	35
4.8.1. Condiciones KKT	35
4.8.2. Problema Dual	36
5. Redes neuronales	37
5.1. Modelos de redes neuronales artificiales	38
5.1.1. El perceptrón: una red neuronal simple	39
5.2. Redes Feedforward	39
5.2.1. Varias clases	40
5.2.2. Capas ocultas	40
5.2.3. Funciones no lineales	40
5.2.4. Algoritmo de entrenamiento Backpropagation	43
5.2.5. Algoritmo de entrenamiento Resilient Backpropagation	49
5.3. Redes Competitivas	51
5.3.1. Vector Quantization	51
6. Gestos	53
6.1. Clasificaciones	53
6.1.1. Clasificación de McNeill	53
6.1.2. Clasificación de Cadoz (SI HAY TIEMPO)	54
6.1.3. Clasificación de Kendon (SI HAY TIEMPO)	54
6.1.4. Clasificación de Caridakis (SI HAY TIEMPO)	54
6.1.5. Lenguajes de señas (SI HAY TIEMPO)	54
6.1.6. Clasificación para su reconocimiento	54

6.2.	Modelado y propiedades de los gestos dinámicos	55
6.2.1.	Modelo de gestos con parametrización temporal	55
6.2.2.	Modelo de gestos con parametrización por longitud de arco	57
II	Aplicaciones	59
7.	Bases de datos y características para el reconocimiento de gestos	61
7.1.	Base de datos de letras y números arábigos	61
7.2.	El Kinect y su SDK	61
7.2.1.	Kinect	61
7.2.2.	SDK y Algoritmo de tracking del cuerpo	63
7.3.	Preprocesamiento	64
7.3.1.	Rotación	64
7.3.2.	Suavizado	65
7.3.3.	Re-muestreo	65
7.4.	Características	66
7.4.1.	Versiones discretas de las propiedades de los gestos	66
7.5.	Base de datos de gestos Visapp2013	67
8.	Experimentos y resultados	69
8.1.	Modelos de reconocimiento	69
8.1.1.	SVM	69
8.1.2.	Redes Neuronales Feedforward (FF)	69
8.1.3.	CNC: El clasificador neuronal competitivo	70
8.1.4.	Clasificador basado en templates	70
8.2.	Experimentos con base de datos de LNHGDB	70
8.3.	Experimentos con base de datos Celebi2013	70
8.4.	Conclusiones	71

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, y se estudiaron e implementaron clasificadores basados en máquinas de vectores de soporte (SVM), redes neuronales feedforward (FF) y redes neuronales competitivas (CPN), utilizando representaciones locales y globales para caracterizar los gestos.

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 ¹.

Se realizaron experimentos con la base de datos generada, compuesta de gestos que representan dígitos y letras, y con otra de gestos típicos de interacción, obteniendo resultados satisfactorios.

¹ Todo el código utilizado en esta tesina se encuentra disponible en <https://github.com/facundoq/gest>

Introducción

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 mouse, 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.

Definimos un gesto como ciertos 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 más un término global que engloba diversos tipos de “reconocimientos y “gestos”. A grandes rasgos, podemos distinguir **gestos corporales** que se realizan con movimientos de todo el cuerpo, **gestos con las manos** como el típico *swipe*, **gestos con los dedos y las manos** como el lenguaje de señas y **gestos faciales** como 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, aunque ha estado 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 se utilizaron también 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 desde entonces mejorado hasta tener hoy sistemas de **tracking** 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 touch-screens 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, encontramos que 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 sobreesfuerzo de las cuerdas vocales en el uso diario, y aquellas particulares al 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 natural [19].

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 sobreuso 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 un interfaz más natural, prevenir nuevas ocurrencias de estos problemas y proveer medios de comunicación e interacción asistivos para 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 es el enfoque de aprendizaje automático que tomamos en esta tesina.

Esta tesina consta de dos partes. 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 3, 4, 5 y 6). En la segunda, se presenta una aplicación de dichos conceptos al reconocimiento de gestos con la mano en espacios 3D en el capítulo 7, finalizando con una descripción de los experimentos realizados y sus resultados en el capítulo 8.

Parte I

Métodos

Aprendizaje Automático

The question of whether computers can think
is like the question of whether submarines can swim.

Edsger W. Dijkstra

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 patrones, instancias o ejemplares del problema para generar un modelo del mismo. Con ese modelo, se puede entonces resolver instancias del problema, con cierto grado de error.

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 tenemos un bajo nivel de error innato, como el reconocimiento de voz, de gestos, de objetos, etc, pero es difícil codificar las reglas que utilizamos para realizar dichas tareas explícitamente. El aprendizaje automático es importante, entonces, porque las ocurrencias de la existencia humana parecen ocurrir 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 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, podemos argumentar que, en esencia, muchos de los algoritmos de aprendizaje automático son aproximaciones matemáticas y computacionales a la automatización del funcionamiento del cerebro [31, 32, 33].

3.1. Un ejemplo: Reconocimiento de Gestos

Para ilustrar la idea del reconocimiento automático, veamos el caso del problema de reconocimiento de gestos.

Supongamos que en una base de datos (BD) tenemos 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. Queremos 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. Podríamos describir un golpe como una trayectoria de la mano bastante recta, de una longitud mayor a cierto mínimo recorrida de forma rápida y con una parada abrupta. 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 podríamos definir una tupla hoariana $\{ \text{La entrada se ajusta al modelo de gesto golpe} \} p \{ \text{La entrada se ajusta al modelo de gesto golpe} \}$ o $\{ \text{La entrada no se ajusta a ninguno de los modelos} \} p \{ \text{La entrada no se ajusta a ninguno de los modelos} \}$. Si bien es cierto que estas tuplas son triviales, lo importante es notar que *podemos* escribirlas porque dichos modelos se definieron explícitamente.

El enfoque de aprendizaje automático, en cambio, entrenaría un clasificador 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 probamos el clasificador con nuevos ejemplares de gestos; si el clasificador es bueno y tenemos suerte, todos los nuevos ejemplares de gestos se clasificarán correctamente; en la práctica, obtendremos un porcentaje de clasificación correcta que nos habla del error de nuestro clasificador.

¹La 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

Entonces, en el enfoque de aprendizaje automática nos desligamos la noción hoariana de computación donde dado un programa un programa $p :: Input \mapsto Output$ y una tupla $\{Pre\} p \{Pre\}$, se cumple $\forall i \in Input, Pre(i) \rightarrow Post(p(i))$, y pensamos en p como un programa que realiza una tarea con cierto grado de error, de forma similar a cómo se trabaja con métodos numéricos. De este modo, dado un conjunto de datos D podemos definir como objetivo general de un algoritmo de entrenamiento o entrenamiento la resolución del problema de optimización:

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

Las distintas maneras de generar el programa p dan lugar a distintos algoritmos de aprendizaje automático. En general p implementa de forma interna una **función de entrenamiento** $g :: \mathcal{P}(\mathcal{P}) \mapsto f$, donde P es el dominio de los ejemplares del problema, f es una **función de inferencia** tal que $f :: \mathcal{P} \mapsto Inferencia$, donde Inferencia es un conjunto de inferencias posibles a realizar sobre \mathcal{P} , dependiente del problema a resolver. En nuestro ejemplo, $Inferencia = \{Saludo, Golpe, Ninguno\}$.

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} , podemos utilizarla para hacer inferencia sobre el dominio 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, podemos considerar f aproxima o estima f' en base a un conjunto de datos $D \subset \mathcal{P}$ y una función de aprendizaje g . Es usual llamar a f el **modelo** del problema².

Es importante notar que $Inferencia$ puede ser cualquier cosa, no solamente un conjunto de clases o categorías a asignar a los ejemplares como en nuestro ejemplo; podríamos tener $\mathcal{P} = Inferencia = \mathbb{R}$ si estamos intentando aproximar una función $\mathbb{R} \rightarrow \mathbb{R}$. A continuación detallamos varias de las aplicaciones de aprendizaje automático.

3.2. Aplicaciones

El ejemplo de los gestos es de un problema de **clasificación**, pero como mencionamos anteriormente, en general un programa de aprendizaje automático se entrena para aprender una función de inferencia cuya imagen ciertamente no se encuentra restringida a clases de ejemplares.

Las técnicas de aprendizaje automático se usan con éxito en una enorme variedad de problemas; ya en 1994 se contaba con una gran cantidad de aplicaciones de redes neuronales, uno de los temas de investigación más tradicionales de aprendizaje automático, en las siguientes áreas[34]:

- Telecomunicaciones para la mitigación del eco y la equalización de la señal
- Control activo para cancelar sonido y la vibración en sistemas como aires acondicionados
- Detección de eventos y disminución del error de operación en aceleradores de partículas
- Selección de clientes para la recepción de los catálogos de una empresa
- Detección de fraudes en operaciones bancarias y tarjetas de crédito
- Reconocimiento óptico de caracteres de máquina y manuscritos
- Control de calidad en manufacturas
- Asignación de asientos en empresas de transporte
- Marketing
- Predicción y análisis financiero
- Control de la manufactura de microprocesadores
- Control de procesos químicos
- Detección de cáncer y otras enfermedades en imágenes médicas y muestras biológicas
- Identificación del origen de drogas ilícitas
- Exploración y refinación del petróleo
- Misilísticas
- Enfocado automático en telescopios
- Y muchas otras

Ampliando algunos de estos casos, 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 puede verse muy beneficiado por el uso de aprendizaje automático debido a que este suele superar a los métodos estadísticos tradicionales. Se ha utilizado para desarrollar en el uso de marcadores biológicos para la identificación por computadora de cáncer [43] y los de en 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 de dispositivos de captura en el mercado de consumo.

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].

²En 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 entrenados con el mismo conjunto de datos

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], derivar transformaciones similares al Análisis de Componentes Principales No-Lineal (NPCA) [54] y optimización combinatoria [55].

3.3. Entrenamiento Supervisado y No Supervisado

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

Los algoritmos supervisados son aquellos donde se conoce a priori un 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 ya se conoce su clase, o sea, ya se sabe si son saludos o golpes. El programa entonces aprender 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 nuestro ejemplo, sería como tener sólo los 20 gestos pero no saber (al menos a priori) de qué clase son. El programa debería a distinguirlos, a reconocer que los ejemplares de los gestos de saludo tienen cierta similitud mutua, una estructura compartida; que los egstos 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*.

En la práctica, ambas técnicas se utilizan conjuntamente, por lo que si bien podemos tildar a un algoritmo de supervisado o no, en general un sistema completo de aprendizaje automático emplea varias combinaciones de algoritmos de los dos tipos, en forma sucesiva o paralela.

En esta tesis nos enfocamos en técnicas de ML para clasificación de patrones con el objetivo de determinar la clase de gestos a la cual pertenece un ejemplar del mismo representado por una secuencia de posiciones en un espacio 3D con etiquetas de tiempo cuya clase es conocida de antemano. Por ende, a continuación describiremos formalmente el problema de la clasificación de ejemplares, enfocándonos en técnicas supervisadas. Además, introduciremos diversos conceptos como generalización, separabilidad lineal, validación cruzada, sobreajuste y regularización para establecer una base teórica común a los métodos de clasificación descritos en el siguiente capítulo. Si bien por conveniencia tratamos dichos conceptos en el contexto de clasificación, se aplican en general a la mayoría de las técnicas de aprendizaje automático.

3.4. Clasificación

Formalmente, podemos definir un clasificador 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, donde \perp representa la clase nula, es decir, que un ejemplar no pertenece a ninguna clase. Esta función 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).

(Gráfico voronoi) Regiones de voronoi en \mathbb{R}^2 para un problema de clasificación con dominio \mathcal{P} y 3 clases.

Tal función f generalmente se genera con un algoritmo de entrenamiento que en base a un conjunto de datos de entrenamiento para el clasificador y a ciertos conocimientos específicos del dominio del problema a resolver.

(Grafico datos \rightarrow regiones de voronoi) Un conjunto de datos con 3 clases y las regiones de voronoi estimadas por un clasificador entrenado con dichos datos.

Entonces, de forma análoga a nuestro argumento general sobre aprendizaje automático, en esencia f aproxima una función f' que codifica las clases “verdaderas” de los ejemplares de \mathcal{P} .

El clasificador f se genera en base a un conjunto ordenado D de n ejemplares \mathbf{x}_i , $D = \{\mathbf{x}_i \in \mathcal{P}\} \quad i = 1..n$, donde por ejemplo podríamos tener $\mathcal{P} = \mathbb{R}^d$, como será en el caso de los gestos. A su vez, cada ejemplar pertenece a una de las $|C|$ clases conocidas, por ende asociado a cada \mathbf{x}_i hay una etiqueta $y_i \in C$ que nos indica a qué clase pertenece \mathbf{x}_i .

3.4.1. El Perceptrón: un clasificador simple

A modo de ejemplo, consideremos un tipo de clasificador muy simple, 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 porque es un tipo de clasificador que divide ejemplares de dos clases bajo la asunción de que existe un **hiperplano afín** en \mathcal{P} que separa las dos clases; es decir, los ejemplares de una clase se encuentran de un lado del hiperplano, y viceversa. En este caso al tener solo dos clases consideraremos $y = \pm 1$; los positivos se encuentran del lado a donde apunta la normal al hiperplano, \mathbf{w} , y los negativos del otro lado.³

-gráfico de un conjunto de datos y un hiperplano entre medio de los dos, se ve la normal apuntando al lado de los + y del otro lado los negativos

Hiperplano separador de los ejemplos de dos clases.

³Los hiperplanos afines están dados por la ecuación $\mathbf{w} \cdot \mathbf{x} + b = 0$, los hiperplanos lineales por $\mathbf{w} \cdot \mathbf{x} = 0$

La idea esencial es, dado un conjunto de datos D como el descripto más arriba, generar un hiperplano inicial de forma aleatoria y lo movemos (con algún criterio) hasta que clasifique bien los ejemplares.

-gráficos de un conjunto de datos y un hiperplano entre medio de los dos, cada vez mejor
Mejora paulatina del hiperplano seleccionado como separador de las clases

En particular, podemos generar un hiperplano inicial cuyos coeficientes sean ceros, o con valores aleatorios cercanos a cero. Luego, tomamos un ejemplar \mathbf{x}_i del conjunto D y nos fijamos si se está clasificando bien. Si es así, no se hace nada y volvemos a elegir un ejemplar; sino, se rota el hiperplano ligeramente de manera de que el ejemplar \mathbf{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  $\mathbf{x}_i \in D$ 
Result: Un hiperplano descripto por  $\mathbf{w} \in \mathbb{R}^d$ , un bias  $b \in \mathbb{R}$ 
 $\mathbf{w} = \text{random}(d+1)$ ;
 $b = \text{random}(1)$ ;
while haya ejemplares que  $\mathbf{w}$  no clasifique correctamente do
  for  $\mathbf{x}_i \in D$  do
    if  $\mathbf{w}$  y  $b$  no clasifican bien a  $\mathbf{x}_i$  then
      Modificar  $\mathbf{w}$  y  $b$  de manera que  $\mathbf{x}_i$  esté más cerca del lado correcto ;
    end
  end
end

```

Algoritmo 1: Esquema del algoritmo de aprendizaje del Perceptrón

Formalizando las nociones de “clasifica bien” y “modificar \mathbf{w} y b ”, podemos definir un hiperplano en función de su vector normal $\mathbf{w} \in \mathbb{R}^d$ y su bias b , con ecuación:

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0 \quad (3.1)$$

Entonces, para los ejemplares \mathbf{x} sobre el hiperplano, tenemos $h(\mathbf{x}) = 0$; para los ejemplares \mathbf{x} del lado del vector normal $h(\mathbf{x}) > 0$, y para los ejemplares del lado opuesto al vector normal, $h(\mathbf{x}) < 0$. Para clasificar un ejemplar \mathbf{x} , utilizamos entonces la función:

$$f(\mathbf{x}) = \theta(\mathbf{x}) \quad \theta_t(x) = \begin{cases} 1 & \text{si } x > 0 \\ -1 & \text{de lo contrario} \end{cases} \quad (3.2)$$

Imagen: Funcion θ caption: Función θ , comúnmente llamada *step function* o función escalón.

Entonces, consideramos que no hay ejemplares clasificados incorrectamente si:

$$\forall \mathbf{x}_i \in D, \quad f(\mathbf{x}_i) = y_i \quad (3.3)$$

Para la modificación, asumamos $b = 0$ momentáneamente y veamos como podemos cambiar \mathbf{w} de manera de que dado un \mathbf{x} clasificado incorrectamente obtengamos un \mathbf{w}' que tenga menos error que \mathbf{w} con respecto a \mathbf{x} . Para eso, notemos que si $b = 0$ y α es el “ángulo” entre \mathbf{x} y \mathbf{w} , tenemos $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos(\alpha)$. Entonces, el signo de h es el signo de $\cos(\alpha)$; por ende si $\alpha \in (-90, 90) \rightarrow h > 0$ y si $\alpha \in (90, 270) \rightarrow h < 0$.

Gráfico donde se ve un hiperplano, el vector normal w y un círculo unitario alrededor de w marcando las regiones donde \cos es positivo y donde es negativo

Regiones positivas y negativas para un hiperplano con $b = 0$ en \mathbb{R}^2

Supongamos \mathbf{x}^+ , con $y_+ = 1$. Si $h(\mathbf{x}^+) = -1$, el ejemplar está clasificado incorrectamente.

Gráfico con un hiperplano con vector normal w , algunos datos bien clasificados, y un + del otro de la normal
Un ejemplar + mal clasificado.

De forma tentativa, podemos definir una regla simple de modificación de \mathbf{w} para arreglar dicho error: asignar $\mathbf{w} \leftarrow \mathbf{x}^+$. De esta manera, claramente $\mathbf{w} \cdot \mathbf{x}^+ > 0$ y entonces el patrón se clasifica correctamente⁴.

Gráfico donde se cambió el hiperplano y ahora \mathbf{x}_0 está sobre w , y bien clasificado. El resto de los patrones queda mal

El ejemplar es clasificado correctamente, pero se ha perdido la clasificación correcta de los otros ejemplares.

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

Gráfico con un hiperplano con vector normal w , algunos datos bien clasificados, y un - del lado de la normal

⁴ Asumiendo $\mathbf{x}_0 \neq \mathbf{0}$

Un ejemplar – mal clasificado.

En este caso, asignamos $\mathbf{w} \leftarrow -\mathbf{x}^-$ y así $\mathbf{w} \cdot \mathbf{x}^+ = \mathbf{x}^+ \cdot \mathbf{x}^+ < 0$.

Gráfico donde se cambio el hiperplano y ahora x_0 está sobre $-w$, y bien clasificado. El resto de los patrones queda mal

El ejemplar es clasificado correctamente, pero se ha perdido la clasificación correcta de los otros ejemplares.

En cierto sentido, de esta forma se elige el \mathbf{w} óptimo para \mathbf{x} ya que se maximiza la distancia desde el ejemplar al hiperplano, pero el cambio es muy extremo ya que ahora no se clasifican bien algunos de los otros ejemplares. Si repetimos esto para todos los ejemplares, \mathbf{w} se irá adaptando a cada uno en particular, olvidando completamente los hiperplanos anteriores. Con esta regla de modificación solo conseguiremos clasificar correctamente a todos los ejemplares si por casualidad algún \mathbf{x} es un hiperplano adecuado para ello. Necesitamos una regla que sea más conservadora con los cambios, que modifique \mathbf{w} para que tenga menos error respecto a un \mathbf{x} , pero que no descarte todo el progreso hecho en las iteraciones anteriores.

El algoritmo tradicional para entrenar un Perceptrón propone corregir \mathbf{w} , dado un ejemplar mal clasificado \mathbf{x} de clase y , con la regla:

$$\mathbf{w} \leftarrow \begin{cases} \mathbf{w} + \mathbf{x} & \text{si } y > 0 \\ \mathbf{w} - \mathbf{x} & \text{de lo contrario} \end{cases} \quad (3.4)$$

La idea es que en vez de reemplazar \mathbf{w} directamente por \mathbf{x} o $-\mathbf{x}$, \mathbf{w} se convierte de a poco en los vectores \mathbf{x} . Podemos pensar en dicha conversión de la siguiente manera. Si $\mathbf{w}_m = \mathbf{w} + m\mathbf{x}^+$, a medida que $m \rightarrow \text{inf}$, \mathbf{w}_m apunta cada vez más en la dirección de \mathbf{x}^+ , ya que en comparación $\|\mathbf{w}\| \ll \|m\mathbf{x}^+\|$. Hablamos de dirección ya que $m\mathbf{x}^+$ representa el mismo hiperplano para todo $m \in \mathbb{R}$, por ende no importa que $\|\mathbf{w}_m\|$ también se haga arbitrariamente grande. Entonces, en cada iteración el algoritmo toma un ejemplar mal clasificado y “acerca” o “aleja” \mathbf{w} a dicho ejemplar.

Podemos simplificar dicha regla de corrección eliminando los casos ya que y nos dice con su signo justamente si hay que sumar o restar \mathbf{x} :

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x} \quad (3.5)$$

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

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha y\mathbf{x} \quad (3.6)$$

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

Por último, para esta derivación asumimos que $b = 0$, o sea, que el hiperplano era lineal, pasando por el origen de coordenadas. Para volver a incluir el bias en nuestra formulación podemos utilizar el truco de ampliar los ejemplares de entrada de vectores en $\mathbf{x} \in \mathbb{R}^d$ a vectores en $\mathbf{x}' \in \mathbb{R}^{d+1}$, donde $x_0 = 1$, y lo mismo con $\mathbf{w} \in \mathbb{R}^n$ a $\mathbf{w}' \in \mathbb{R}^{n+1}$. De esta manera el componente $w_0 = b$, es decir, \mathbf{w}_0 representa el bias b , y así podemos mantener la misma regla de actualización, ya que:

$$h(\mathbf{x}') = \mathbf{w}' \cdot \mathbf{x}' = (b, \mathbf{w}) \cdot (1, \mathbf{x}) = b + \mathbf{w} \cdot \mathbf{x} \quad (3.7)$$

El algoritmo final es, entonces:

Data: Un conjunto de ejemplares $\mathbf{x}_i \in D$,

$D \subset \mathbb{R}^{d+1}$, donde a cada $\mathbf{x}_i \in \mathbb{R}^d$ original se le agregó un 1 con el objetivo descrito en el párrafo anterior

Una clase $y_i \in \{-1, 1\}$ asociada a cada $\mathbf{x}_i \in D$

Una tasa de aprendizaje α .

Result: Un hiperplano descrito por $\mathbf{w} \in \mathbb{R}^{d+1}$

$\mathbf{w} \leftarrow \text{random}(d+1)$;

while $\exists \mathbf{x}_i \in D : f(\mathbf{x}_i) \neq y_i$ **do**

for $\mathbf{x}_i \in D$ **do**

if $f(\mathbf{x}_i) \neq y_i$ **then**

$\mathbf{w} \leftarrow \mathbf{w} + \alpha y_i \mathbf{x}_i$;

end

end

end

Algoritmo 2: Algoritmo de aprendizaje del Perceptrón

Hay obviamente aquí una esperanza de que ejecutando este procedimiento, luego de 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 afín $\mathbf{w} \cdot \mathbf{x} + b = 0$ que puede separar los ejemplares de una clase de la otra.

-

gráfico con un conjunto linealmente separable y otro que no lo es

a) Un conjunto linealmente separable b) Un conjunto que no es linealmente separable

En la práctica, la gran mayoría de los dominios está compuesto por problemas cuyos conjuntos de datos ejemplares no son linealmente separables. En dichos casos encontramos 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**, de tipo lineal o afín. Un clasificador no lineal emplea una superficie de separación no-lineal, que aumenta el poder de clasificación porque resulta más flexible que un hiperplano.

-

Gráfico con una curva que clasifica los ejemplares del gráfico anterior correctamente
Un clasificador no lineal

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**, tenemos un conjunto de datos que no se puede separar con un Perceptrón. Si aplicamos una transformación no lineal, a un espacio de la misma dimensión, obtenemos 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 hiperplanos en el clasificador.

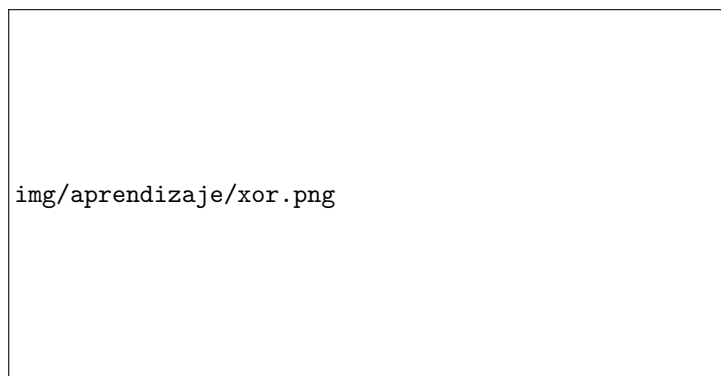


Figura 3.1: as

a) El problema del xor: un clasificador no-lineal basado en esferas ubica 4, una en cada elemento del conjunto de entrenamiento, para clasificarlos. b) El problema del xor: Se transformó el espacio descrito en a) con la función $\phi(\mathbf{x}) = (e^{-(\text{norm}(\mathbf{x} - (1,1))), e^{-(\text{norm}(\mathbf{x} - (0,0)))})$ obteniendo un conjunto de datos linealmente separable.

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

En este contexto, podemos ver el truco de embeber los ejemplares en \mathbb{R}^{d+1} como una transformación no lineal. La función h define un hiperplano afín en \mathbb{R}^d , ya que la función h es afín en \mathbb{R}^d . El término b es necesario para aumentar el poder clasificador del Perceptrón, ya que ciertos conjuntos de ejemplares no son separables con un hiperplano lineal pero si con uno afín, y si $b=0$, h define un hiperplano afín.⁵

Pero en realidad h , definida con el truco de embeber los ejemplares en \mathbb{R}^{d+1} , representa un hiperplano lineal en \mathbb{R}^{d+1} , y esta transformación no-lineal en los ejemplares a un espacio de mayor dimensionalidad permite que sean separados. Es debido a esta simple equivalencia que hablamos de separabilidad *lineal*, cuya definición involucra en realidad un hiperplano afín.

Por último, en la gran mayoría de casos, resulta imposible clasificar perfectamente un conjunto de datos. Podemos 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.

⁵Por conveniencia, cuando decimos que h define un hiperplano, nos referimos al hiperplano $h(\mathbf{x})=0$.

Data:

Un conjunto de ejemplares $D \in \mathbb{R}^d$

Una clase $y_i \in \{-1, 1\}$ asociada a cada $\mathbf{x}_i \in D$

Una tasa de aprendizaje α .

Una tolerancia al error ϵ

Una función que mide el error de clasificación ℓ

Result: Un hiperplano descrito por $\mathbf{w} \in \mathbb{R}^{d+1}$

$\mathbf{w} \leftarrow \text{random}(d+1)$;

$\text{error} \leftarrow \infty$;

while $\text{error} > \epsilon$ **do**

for $\mathbf{x}_i \in D$ **do**

if $f(\mathbf{x}_i) \neq y_i$ **then**

$\mathbf{w} \leftarrow \mathbf{w} + \alpha y_i \mathbf{x}_i$;

end

end

$\text{error} = \rho(D, f)$;

end

Algoritmo 3: Algoritmo de aprendizaje del Perceptrón con tolerancia al error.

Como ejemplo de una función de error ρ , podemos calcular el porcentaje de patrones mal clasificados:

$$\rho(D, f) = 1 - \frac{\sum_{\mathbf{x}_i \in D} \text{pos}(f(\mathbf{x}_i) y_i)}{|D|}$$

$$\text{pos}(r) = \begin{cases} 1 & \text{si } r = 0 \\ r > 0 & \text{de lo contrario} \end{cases}$$

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

Existen varias funciones ℓ que podemos definir para medir el error de clasificación, y presentaremos otras en las siguientes secciones.

3.4.2. Experimentos de clasificación

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

- Leer los datos, preprocesarlos, calcular las características correspondientes
- Seleccionar los parámetros del clasificador ⁶
- Entrenar un clasificador con los datos y algún algoritmo de entrenamiento
- Probar el modelo generado obteniendo alguna medida de error (discutiremos varias formas de hacer esto en las siguientes secciones)

Llamaremos 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 .

El objetivo del experimento es el último punto, la evaluación del desempeño del modelo. Para ello, pensaremos 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 .

Un experimento tiene dos fuentes de aleatoriedad. La primera está dada por los datos utilizados para entrenar y probar el clasificador. Distintos datos darán distinto resultado, y por ende distinto error. En general, cuantos más datos para entrenar tengamos 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 dichas cualidades son deseables en 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, como suele suceder en el caso de las redes neuronales entrenadas con el algoritmo backpropagation, y luego mejorarla iterativamente durante el entrenamiento. La inicialización es aleatoria ya que del estado inicial depende el estado final, y por ende 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 como en los algoritmos genéticos o entrenamiento estocástico.

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.

⁶Por ejemplo, si el clasificador es un perceptrón, se debe seleccionar la tasa de aprendizaje.

Entonces, definimos el concepto de experimento para referirnos a esta serie de pasos en las siguientes secciones. El objetivo de un experimento es generar y evaluar un clasificador. En la evaluación, nos 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**.

3.4.3. Generalización

Un clasificador se genera en base a un conjunto de datos de *entrenamiento*. Una vez generado el clasificador, nos 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.

Para conocer esta capacidad de generalización, aplicamos 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} , utilizamos 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 dado por:

$$\rho_\ell(D, f) = \frac{\sum_{\mathbf{x}_i \in D} \ell(\mathbf{x}_i, f)}{|D|} \quad (3.8)$$

$$\ell(\mathbf{x}_i, f) = 1 - \delta(\mathbf{y}_i, f(\mathbf{x}_i)) \quad (3.9)$$

$$\text{con el delta de Kronecker } \delta(a, b) = \begin{cases} 1 & \text{si } a = b \\ 0 & \text{de lo contrario} \end{cases}.$$

Este es un estimador del valor que realmente nos interesa, o sea el error esperado de f en \mathcal{P} :

$$\rho_\ell(\mathcal{P}, f) = E_{\mathbf{x}}[\ell(\mathbf{x}, f)] \quad (3.10)$$

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

3.4.3.1. 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 utilizamos el mismo conjunto de datos para entrenar un clasificador y estimar su error obtendremos 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.

gráfico datos \rightarrow modelo \rightarrow datos

En un caso extremo, podemos definir:

$$f_D(x) = \begin{cases} \mathbf{y} & \text{si } \mathbf{x} \in D \\ \text{clase aleatoria}^7 & \text{de lo contrario} \end{cases} \quad (3.11)$$

Dicha función 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 no es inusual 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 esto necesitamos 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, dividimos el conjunto en dos: D_e , el conjunto de entrenamiento, y D_p el conjunto de prueba que reservamos luego 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).

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

Entonces, dada una medida de error ρ_ℓ , podemos definir 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 estimamos esta definición como $\rho_\ell(D_e, f) \simeq \rho_\ell(D_p, f)$.

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

A continuación, exploramos el concepto de **validación cruzada** como una manera de estimar más fielmente $\rho_\ell(\mathcal{P}, f)$.

3.4.4. Validación cruzada

Si bien disponer de un conjunto de prueba es una gran mejora sobre 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.

La primera pregunta depende de cual es nuestro objetivo; en términos generales, cuanto más grande sea el tamaño de D_e mejor, ya que nos permitirá desarrollar un modelo más certero. Por otro lado, lo mismo sucede con el conjunto de validación, respecto a la significatividad 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, al seleccionar arbitrariamente a D_e y D_p , y especialmente si $|D|$ es muy chico, estamos atando nuestro estimador a una de las posibles particiones de los dos conjuntos lo cual puede traer problemas ya que nos exponemos a realizar alguna partición desafortunada o afortunada, que nos de resultados que no representen el comportamiento promedio del clasificador.

El problema de elegir nuestros conjuntos de prueba y entrenamiento de forma arbitraria radica en que la performance del clasificador en un experimento puede dar bien o mal de acuerdo a la división particular del conjunto de datos con el que estamos trabajando. Una mala división nos puede llevar a asumir que nuestro 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 2, entrenar con uno y probar con el otro es un caso especial del método de **cross validation** (validación cruzada, VC) [57], un enfoque más estructurado y justificado teóricamente para estimar el error, que provee respuestas a algunas de estas preguntas, especialmente en los casos donde $|D|$ es chico.

La premisa básica en validación cruzada es responder a la 2da 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 evitamos la dependencia de la estimación del error sobre la partición particular con la que realizamos la prueba.

Existen varias versiones y extensiones de validación cruzada, que detallamos a continuación.

3.4.4.1. VC de k-iteraciones

Volviendo al ejemplo del reconocimiento de gestos, supongamos que tenemos un problema de clasificación con 2 clases y un conjunto de 20 ejemplos, 10 de cada clase, y decidimos utilizar 14 ejemplos para entrenar y 6 para probar. Entonces, tenemos $\binom{20}{14} = 38760$ formas posibles para dividir D en D_e y D_p . En lugar de hacer un solo experimento con una sola de las combinaciones, podemos hacerlo con todos y promediar los resultados. Dado que es difícil trabajar pensando en $\binom{|D|}{|D_e|}$ particiones, podemos pensar en una formulación equivalente, la de dividir a D en k conjuntos

disjuntos de prueba llamados **folds**, cada uno con $n_k = \frac{|D|}{k}$ ejemplares⁸. 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, uno por cada conjunto de prueba D_p . De esta manera nos aseguramos que todos los ejemplares sean probados eventualmente. El estimador entonces es:

$$\rho_k = \frac{\sum_{(D_p, D_e) \in D^k} \rho(D_p, g(D_e))}{k} \quad (3.12)$$

$$D^k = \{(D_i, D - D_i) \mid D_i = \{\mathbf{x}_{i \cdot n_k + 1} \dots \mathbf{x}_{i \cdot n_k + n_k}\}, \quad i = 0..k-1\} \quad (3.13)$$

Donde vemos claramente que $i \neq j \rightarrow D_i \cap D_j = \emptyset$ y $\bigcup_i D_i = D$. En nuestro ejemplo, si $k=2$ tenemos dos subconjuntos de prueba, uno con los primeros 10 ejemplares de D , el otro con los últimos 10 ejemplares, y haremos 2 pruebas. Si $k=5$, haremos 5 experimentos, cada uno con 16 ejemplares de entrenamiento y 4 de prueba.

gráfico de cross validation para $k = 1, 2, 4, 5, 10$ y $n = 20$

“Particiones posibles al usar validación cruzada con $k=1, 2, 4, 5$ y 10 y un conjunto de 20 ejemplares”

Si bien variando k tenemos una respuesta parcial a nuestra pregunta de qué tamaños deben tener los conjuntos de prueba y entrenamiento, la 2da pregunta queda todavía sin respuesta; si reordenamos los ejemplares de D , los elementos de D_i 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 error pero el coste computacional se mantiene al mínimo. Una configuración popular es $k=10$ [57, 58], ya que en general alcanza dicho balance y además es ampliamente utilizada, haciendo fácil en ocasiones comparar experimentos. Llegando a $k = |D|$, encontramos otro caso especial de VC llamado **Leave-one-out CV** (VC dejando-uno-afuera).

⁸En esta sección asumimos, por simplicidad, que $|D|$ es divisible por k

3.4.4.2. 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.

Tenemos $|D|$ particiones distintas, una por cada ejemplar, por ende la complejidad de las pruebas es de orden lineal a la cantidad de ejemplares. Si tenemos pocos ejemplares, puede que nos falte significancia estadística; si tenemos 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.

3.4.4.3. VC aleatoria

En esta variación de VC se elige un tamaño del conjunto de prueba n_p , y se elige de forma aleatoria una partición de D con $|D_p| = n_p$ cada vez que se va a realizar un experimento. 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. De esta manera respondemos parcialmente a la pregunta 2; no nos da un criterio para elegir qué ejemplares van en cada conjunto, pero si como reducir al mínimo la cantidad de experimentos alcanzando un nivel de eficiencia deseado.

3.4.4.4. VC estratificada

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.

Podemos 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 una técnica que se aplica en conjunto con VC aleatoria o de k repeticiones, y responde parcialmente a la 2da pregunta, ya que nos da una restricción débil para decidir que ejemplares pueden ir en qué conjuntos.

3.4.4.5. Funciones de pérdida

Las medidas de error determinan nuestra 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 [59]. Dichas funciones miden el grado de error de un clasificador dado un ejemplar, como antes. Entonces, $\ell :: (\mathcal{P}, \text{Clasificador}) \mapsto \mathbb{R}$. Las más comunes son:

$$\ell_0(\mathbf{x}_i, f) = \begin{cases} 1 & \text{if } \ell_1(\mathbf{x}_i, f) > 0 \\ 0 & \text{if } \ell_1(\mathbf{x}_i, f) \leq 0 \end{cases} \quad (3.14)$$

$$\ell_1(\mathbf{x}_i, f) = \text{sum}(\text{abs}(y_i - f(\mathbf{x}_i))) \quad (3.15)$$

$$\ell_2(\mathbf{x}_i, f) = \|y_i - f(\mathbf{x}_i)\| \ell_p(\mathbf{x}_i, f) = \|y_i - f(\mathbf{x}_i)\|_p \quad (3.16)$$

donde $\text{sum}(\mathbf{v})$ suma los elementos de un vector \mathbf{v} , $\text{abs}(\mathbf{v})$ devuelve un vector con los valores absolutos de \mathbf{v} , $\|\cdot\|$ es la norma euclídea de un vector, y $\|\cdot\|_p$ es la norma p .

Dada una ℓ , podemos calcular entonces el error de un clasificador f en un conjunto de ejemplares D , típicamente como:

$$\rho_\ell(D, f) = \sum_{\mathbf{x}_i \in D} \ell(\mathbf{x}_i, f) \quad (3.17)$$

y podemos aplicar CV con esta medida de error.

3.4.4.6. 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 CV 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 [60, 58].

Para ilustrar con un ejemplo, volvemos al problema de la clasificación de gestos. Olvidando el concepto de VC, imaginemos tener un conjunto de datos de 200 ejemplares de gestos, 100 de cada clase, obtenidos de forma independiente e idénticamente distribuidos. Dividimos este conjunto de datos en 10 subconjuntos disjuntos D_i de 20 muestras (estratificadas, con 10 de cada clase cada uno) y realizamos diez experimentos, uno con cada subconjunto, dividiendo D_i en D_p y D_e en cada experimento y obteniendo 10 estimaciones de error e_i . Es importante notar que no estamos empleando VC, simplemente dividimos 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, podemos calcular los estimadores:

$$\hat{\mu}_e = \sum_{i=1}^{10} e_i \quad \hat{\sigma}_e = \frac{\sum_{i=1}^{10} (\hat{\mu}_e - e_i)^2}{9} \quad (3.18)$$

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, si volvemos a nuestro ejemplo con 20 ejemplares de gestos, 10 de cada clase, y generamos 10 particiones utilizando VC de $k=10$ iteraciones, también obtendremos 10 estimaciones de error e_i . Pero estas 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.

$$Var(e_i + e_j) = Var(e_i) + Var(e_j) + 2Cov(e_i, e_j) \quad (3.19)$$

$$\neq Var(e_i) + Var(e_j) \quad (3.20)$$

Por este motivo, el estimador inocente $\hat{\sigma}_e$ no es adecuado en este caso, ya que subestimaré 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 dos variables aleatorias no depende de su correlación.

Lo mismo sucede con VC aleatoria. Por ende, resulta necesario desarrollar pruebas de hipótesis específicas para CV, 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, se han desarrollado estimadores para VC aleatoria [61], y se ha probado que no existe ningún estimador insesgado para VC con k repeticiones[60], lo cual sienta al primero en bases teóricas más fuertes.

3.4.5. Selección del modelo de clasificación (SI HAY TIEMPO)

3.4.6. Sobre-especialización y regularización

Como mencionamos, hacer un experimento de clasificación involucra medir el error de f con D . Para eso utilizamos alguna variante de validación cruzada que divida D en D_p y D_e , y medimos el error en D_p . Si dicho error es aceptable, consideramos 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 medimos el error de f en D_e y también es alto, tenemos nuestra confirmación. 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, decimos 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.

gráfico error vs cantidad de entrenamiento. “curvas típicas de $\rho(D, f)$ en función de la cantidad de entrenamiento o fuerza del ajuste de f con D .”

En este sentido, vemos que 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 .

gráfico de dos clases con un outlier y un hiperplano “correcto” que obvia al outlier \rightarrow idem pero con un hiperplano overfitted \rightarrow idem pero con un nuevo ejemplar, que lo clasifica bien el 1er hip pero no el 2do “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 c) 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.

gráfico de una distribución real de clases y una muestra D superpuesta \rightarrow gráfico de una región de voronoi por cada elemento de D sin cubrir el espacio entre muestras. “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, cambiar a un modelo de clasificador que se adapte mejor al problema puede tener el mismo efecto.

gráfico de una distribución real de clases y una muestra D superpuesta \rightarrow gráfico con un hiperplano que separa las regiones de las clases, con algunos errores. “a) Distribución real de ejemplares por clases b) Distribución aprendida

por un clasificador basado en hiperplanos.”

En otras, lo que necesitamos es limitar el poder del clasificador. En la figura X – 1, consideramos 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. Por eso, muchas veces es en realidad necesario limitar el poder clasificatorio con el que se genera f para obtener mejores resultados. Por ejemplo, si limitamos la cantidad de ejemplares a recordar en el clasificador en el caso anterior, podríamos obtener una clasificación mejor como la de la figura X.

gráfico de una distribución real de clases y un muestra D superpuesta \rightarrow gráfico de varias regiones de voronoi por cada elemento de D cubriendo el espacio entre muestras. “a) Distribución real de ejemplares por clases b) Distribución aprendida por el clasificador con capacidad limitada. Hay menos regiones de Voronoi que en el caso anterior, pero son más grandes y menos específicas, obteniendo una clasificación más acertada.

En este caso podemos 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, podemos decir que nuestro esquema general para entrenar un clasificador podría ser:

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

Donde *complejidad* es alguna función que mide la complejidad de f , dependiente del algoritmo de entrenamiento del clasificador utilizado.

gráfico error vs complejidad del modelo. “curvas típicas de $\rho(D, f)$ en función de la complejidad del modelo f ”

3.4.7. Modelo de clasificación con estructura probabilística

En el modelo de clasificación descrito en las secciones anteriores, realizamos ciertas simplificaciones que no suelen ser ciertas en la práctica. Las mismas constan de asumir que las clases y son equiprobables, que los ejemplares x también lo son, y que a cada ejemplar corresponde una clase única. Los problemas a resolver suelen tener cierto error inherente, en el sentido de que podemos tener ejemplares arbitrariamente parecidos $x_0 \simeq x_1$ con $y_0 \neq y_1$. Es decir, no siempre existe un clasificador que distinga perfectamente los ejemplares de todas las clases.

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:

- Ciertas clases pueden ser más ocurrentes que otras. En muchos problemas reales esto no es cierto; por ejemplo, si tenemos 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. Supongamos 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 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, tendremos 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
- Además, asumimos que un ejemplar 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 x no tienen información completa sobre todas las causas y los indicadores de la enfermedad. Entonces, es muy posible tener dos ejemplares, x_0 y x_1 tales que $x_0 = x_1$ pero $y_0 \neq y_1$. Es decir, al tener información incompleta, podemos tener colisiones.^{en} la asignación de etiquetas “verdaderas.”^a los ejemplares.

- Un gráfico para cada ejemplo. En el primero, tenemos muchos puntos de lado - y pocos del lado +. En el segundo tenemos lo mismo, pero además hay un cúmulo grande de +, separados del anterior. En el tercero, tenemos + y - juntos Ejemplos de conjuntos de entrenamiento para los tres casos. En el primero, la cantidad de ejemplos x^- es mucho mayor que los x^+ . En el segundo, hay una distribución de los x^+ bastante asimétrica. En el tercero, tenemos ambigüedad en la asignación de clases para algunos x .

En base a los items anteriores, podemos extender el modelo original agregando una estructura probabilística en nuestro 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, consideramos una distribución de probabilidad $P(Y)$ de las clases posibles de los patrones, C . Además, asumimos una distribución $P(X)$ (generalmente desconocida) de los ejemplares sobre el conjunto \mathcal{P} . Esto nos da entonces una probabilidad condicional $P(Y|X)$ que queremos averiguar; es decir, dado un ejemplar, queremos saber la

probabilidad de que pertenezca a cierta clase. De esta forma, tenemos en cuenta las tres simplificaciones mencionadas anteriormente.

Podemos ver estas definiciones 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 $\mathbf{x} \in \mathcal{P}$ de acuerdo a $P(\mathbf{X})$. Luego, se le asigna una etiqueta y de acuerdo a $P(Y|\mathbf{x})$.

El objetivo de un clasificador es encontrar $P(Y|\mathbf{X})$ a partir de los ejemplos de entrenamiento. Estos ejemplos dan información sobre $P(Y, \mathbf{X})$; en base a eso, el algoritmo de aprendizaje estima $P(Y|\mathbf{X})$.

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

Hay que considerar otra simplificación que hemos hecho, que consta de una distinción sutil pero importante. Hasta ahora consideramos que todos los ejemplares \mathbf{x}_i han sido etiquetados correctamente con la clase y_i correspondiente, 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 \mathbf{x}_i , la misma no siempre representa la clase del ejemplar, y deberíamos considerar las probabilidades sobre un y_i^v que es la clase verdadera del mismo, de modo que nuestro 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) [62, 63], la teoría subyacente en del modelo de **Support Vector Machine** (Máquina de Vectores de Soporte), un tipo de clasificador que desarrollaremos en la siguientes secciones.

3.4.8. Modelo de clasificación multiclase

Podemos extender el modelo aún más si consideramos el problema de que un ejemplar pertenezca realmente a *varias* clases. Por ejemplo, si estamos clasificando objetos detectados en imágenes para asignarle 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 descrito en las secciones anteriores, asumimos que hay una *única* clase "verdadera" y_i para asignar a cada patrón del dominio del problema, con cierta ambigüedad. Ahora podríamos asumir que a cada \mathbf{x} corresponde un vector de puntajes por clase $\mathbf{y} \in \mathbb{R}_p^{|C|}$, 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 $f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_C(\mathbf{x}))$.

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

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

El separador de espacios por excelencia es el hiperplano. Pero dado un conjunto de datos con dos clases linealmente separables, podemos tener infinitos hiperplanos que separen los conjuntos. Algoritmos simples como el del Perceptrón se quedan con el primero que encuentran, ya que no tienen un criterio para elegir entre los hiperplanos posibles. Las **Máquinas de Vectores de Soporte (SVM)** extienden el perceptrón con un criterio tal.

Gráfico con dos conjuntos de datos + y - con distintos hiperplanos, uno "bueno" y los otros raros
Distintos hiperplanos separadores para el mismo conjunto de datos

En la figura anterior, si bien 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 intuitivamente que a priori el primer hiperplano captura mejor la división entre las dos clases.

4.1. 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).

Gráfico mostrando un hiperplano de máximo margen, los hiperplanos de soporte, M , y dos conjuntos de datos
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**. La ecuación del hiperplano es:

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0 \quad \mathbf{w}, \mathbf{x} \in \mathbb{R}^d, b \in \mathbb{R} \quad (4.1)$$

Para nuestro modelo, 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.

Como buscamos un hiperplano que clasifique correctamente a todos los ejemplares $\mathbf{x}_i \in D$, podemos definir las siguientes restricciones:

$$\forall \mathbf{x}_i \in D \quad \begin{cases} h(\mathbf{x}_i) > 0 & \text{si } y_i = 1 \\ h(\mathbf{x}_i) < 0 & \text{si } y_i = -1 \end{cases} \quad (4.2)$$

Podemos combinar ambos casos como:

$$\forall \mathbf{x}_i \in D \quad h(\mathbf{x}_i) y_i > 0 \quad (4.3)$$

De manera informal, podemos especificar el problema asumiendo una función que calcule el margen entre un hiperplano y un conjunto de ejemplares D .

$$\begin{aligned} \text{Min}_{\mathbf{w}, b} \quad & M = \text{margen}(\mathbf{w}, b, D) \\ \text{sujeto a} \quad & h(\mathbf{x}_i) y_i > 0, i = 1, \dots, |D|. \end{aligned}$$

Para formalizar el problema, solo necesitamos una expresión concreta de M , el tamaño del margen, en términos de \mathbf{w} y b .

Pero antes de eso, tenemos que solucionar un pequeño problema técnico; la ecuación $\mathbf{w} \cdot \mathbf{x} + b = 0$ está indeterminada ya que multiplicando por cualquier constante obtenemos valores de \mathbf{w} y b distintos, pero que representan el mismo hiperplano. Esto da lugar a infinitas soluciones del problema. Entonces, asumiremos que buscamos los \mathbf{w} y b con la

condición de que para los vectores de soporte de cada clase, -1 y 1 , $h(\mathbf{x}^-) = -1$ y $h(\mathbf{x}^+) = 1$, respectivamente. De esta manera \mathbf{w} y b quedan completamente determinados.

No escribiremos 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, supongamos que \mathbf{x}^+ es un vector de soporte de la clase 1, y \mathbf{x}_0 es el vector perteneciente al hiperplano ($h(\mathbf{x}_0) = 0$) más cercano a \mathbf{x}^+ , tal que la línea que los une es perpendicular al hiperplano, o sea es paralela a \mathbf{w} .

gráfico de \mathbf{x}^+ y \mathbf{x}_0 , con \mathbf{w}

Los vectores \mathbf{x}^+ y \mathbf{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 .

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

$$\mathbf{w} \cdot \mathbf{x}^+ + b = 1 \quad (4.4)$$

$$\mathbf{w} \cdot (\mathbf{x}_0 + \lambda \mathbf{w}) + b = 1 \quad (4.5)$$

$$\mathbf{w} \cdot \mathbf{x}_0 + \mathbf{w} \cdot \lambda \mathbf{w} + b = 1 \quad (4.6)$$

$$\mathbf{w} \cdot \mathbf{x}_0 + b + \mathbf{w} \cdot \lambda \mathbf{w} = 1 \quad (4.7)$$

$$\mathbf{w} \cdot \lambda \mathbf{w} = 1 \text{ (} \mathbf{x}_0 \text{ pertenece al hiperplano)} \quad (4.8)$$

$$\lambda \|\mathbf{w}\|^2 = 1 \quad (4.9)$$

$$\lambda \|\mathbf{w}\| = \frac{1}{\|\mathbf{w}\|} \quad (4.10)$$

$$M = \frac{1}{\|\mathbf{w}\|} \quad (4.11)$$

Finalmente el problema queda planteado como:

$$\text{Min}_{\mathbf{w}, b} \quad \frac{1}{\|\mathbf{w}\|}$$

$$\text{sujeto a } h(\mathbf{x}_i)y_i > 0, i = 1, \dots, |D|.$$

Como maximizar $\frac{1}{\|\mathbf{w}\|}$ es equivalente a minimizar $\|\mathbf{w}\|^2 = \mathbf{w} \cdot \mathbf{w}^t$ y es más simple, lo reescribimos como:

$$\text{Min}_{\mathbf{w}, b} \quad \mathbf{w} \cdot \mathbf{w}^t$$

$$\text{sujeto a } h(\mathbf{x}_i)y_i \geq 1, i = 1, \dots, |D|.$$

Es importante notar que ahora pedimos que $h(\mathbf{x}_i)y_i \geq 1$ en lugar de $h(\mathbf{x}_i)y_i > 0$, ya que los \mathbf{x}_i más cercanos al hiperplano son los vectores de soporte, para los cuales definimos arriba que $h(\mathbf{x}_i)y_i = 1$.

Este es un problema de optimización cuadrática ($\mathbf{w} \cdot \mathbf{w}^t$), con restricciones lineales y podríamos resolverlo 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, expandiremos el modelo de SVM con la formulación de **Márgenes Suaves** para manejar problemas que no son linealmente separables y la presencia de outliers; derivaremos la **formulación dual** que posibilita una solución más simple del problema de optimización y permite el **truco del kernel** para poder aplicar con poco coste computacional ciertos tipos de transformaciones a los ejemplares para separar mejor las clases, y finalmente desarrollaremos el algoritmo de optimización **Sequential Minimal Optimization** (SMO) de Platt [64] para resolver el problema de optimización y poder entrenar una SVM.

4.2. Modelo de Márgenes Suaves

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

- a) Gráfico de Un problema no linealmente separable b) Gráfico con dos clases y un outlier entre medio que comprime para un lado el margen de decisión

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 $h(\mathbf{x}_i)y_i \geq 1$ dándoles un poco de ayuda mediante una variable de holgura $s_i \geq 0$, tal que:

$$h(\mathbf{x}_i)y_i + s_i \geq 1, s_i \in \mathbb{R} \quad (4.12)$$

La variable s_i "ayuda" a $h(\mathbf{x}_i)y_i$ a llegar a 1 cuando este no puede debido a elección de \mathbf{w} y b . Ahora, si dejamos 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, podríamos definir $\mathbf{w} = \mathbf{0}$, ya que de los s_i adecuados permitirán que se cumplan las restricciones. Para evitar esto, penalizaremos en nuestra función a optimizar cada unidad de s_i que se utiliza con un coste c , dando lugar a la formulación ¹:

$$\begin{aligned} \text{Min}_{\mathbf{w}, b, s_i} \quad & \mathcal{L}_P(\mathbf{w}, b, s_i) = \mathbf{w} \cdot \mathbf{w}^t + c \sum_i s_i \\ \text{sujeto a} \quad & h(\mathbf{x}_i) y_i + s_i \geq 1, i = 1, \dots, |D| \\ & s_i \geq 0. \end{aligned}$$

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

Dado un \mathbf{w} y un b , necesitamos que $h(\mathbf{x}_i) y_i \geq 1$. Si esto es cierto, podemos tener $s_i = 0$ y no pagar ningún coste, ya que la restricción se cumple sola. Si no es cierto, tenemos 2 opciones, en términos de la función a optimizar: cambiar \mathbf{w} , lo cual puede hacer que otras restricciones no se cumplan y además puede incrementar el coste si se achica nuestro margen, o ayudar a $h(\mathbf{x}_i) y_i$ con s_i para que llegue a 1, y pagar el precio c . El vector \mathbf{x}_i quedaría mal clasificado, en este último caso, pero en términos de la función a optimizar eso puede ser mejor que pagar al 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), de acuerdo al parámetro de coste c . En el caso en que $c \rightarrow \infty$, penalizamos fuertemente la ayuda de s_i , volviendo al modelo original de márgenes *duros*. Si $c \rightarrow 0$, permitimos que \mathbf{w} y b tomen cualquier valor, porque cualquier error de clasificación puede arreglarse con los s_i adecuados ya que son “baratos”.

- Gráfico anterior con dos clases y algunos outliers, pero ahora con el hiperplano intuitivo, y se muestra que para los outliers $s_i > 0$.

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

Entonces, el valor de c controla el nivel de regularización de la solución, ya que ayuda a evitar que \mathbf{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.

- 4 gráficos del ejemplo anterior, con $c=0.1$, $c=1$, $c=10$, $c=100$

Hiperplanos determinados por el problema de optimización con $c = \{0, 1, 10, 100\}$

Por último, hay que destacar que 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 [67]. 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.

4.3. Forma 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 llamaremos simplemente las **KKT** (ver 4.8).

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 nuestro caso, las variables duales también estarán restringidas 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 el caso de nuestro problema, 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.

Si encontramos una relación entre las variables del primal y las del dual que nos permitan calcular las primeras a partir de las segundas, podemos plantear y resolver el dual, y finalmente obtener dos valores de las variables del primal. Esta será nuestra estrategia para obtener el dual del problema del máximo margen.

Primero, debemos llevar las restricciones a la forma $r \geq 0$. Las de $s_i \geq 0$ ya se encuentren en ese estado; las otras las escribimos como $r_i \geq 0$ donde $r_i = h(\mathbf{x}_i) y_i + s_i - 1$.

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

$$\mathcal{L}(\mathbf{w}, b, s_i, \alpha_i, u_i) = \mathbf{w} \cdot \mathbf{w}^t + c \sum_i s_i - \sum_i \alpha_i r_i - \sum_i s_i u_i$$

y nuestro objetivo queda como ²:

¹Elegimos usar el mismo coste c para todas las restricciones; si tenemos información a priori del dominio del problema podemos elegir un coste distinto para cada ejemplar, por ejemplo, o para cada clase, como se suele hacer cuando las mismas están desbalanceadas

²Por claridad, obviamos el rango de la variable i ($i = 1, \dots, |D|$) en las siguientes derivaciones

$$\text{Min}_{\mathbf{w}, b, s_i} \quad \{\text{Max}_{\alpha_i, u_i} \mathcal{L}(\mathbf{w}, b, s_i, \alpha_i, u_i)\}$$

$$\text{sujeto a} \quad \begin{aligned} r_i &\geq 0, & s_i &\geq 0 \\ \alpha_i &\geq 0, & u_i &\geq 0 \end{aligned}$$

El lagrangiano captura el efecto de las restricciones pero en forma de coste, obteniendo un problema equivalente al anterior. Podemos ver esto intuitivamente, ya que las restricciones piden $r_i \geq 0$ y $s_i \geq 0$. Si $r_i \geq 0$, entonces $-\sum_i \alpha_i r_i$ es negativo, lo cual ayuda a minimizar nuestra 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 necesitamos en la solución), los términos agregados a la función de coste nunca pueden perjudicarla; a lo sumo, no la ayudarán si $\alpha_i = 0$ o $u_i = 0$. Entonces, estos dos términos nunca aumentan la función de coste si nos mantenemos dentro de las restricciones. Si nos vamos de 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 restricciones. En este balance hay que tener en cuenta que \mathbf{w} y s_i tienen que ser bajos debido a los otros dos términos de la función de coste. Entonces, en el mejor caso $s_i = 0$, y $h(\mathbf{x}_i) y_i \geq 1$, con un \mathbf{w} bajo.

Podemos formalizar estos argumentos intuitivos mediante las KKT (ver sección 4.8 para un resumen), que además usamos para eliminar las variables primales y obtener el problema dual, que en este caso dicen:

- Estacionalidad

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0$$

$$\frac{\partial \mathcal{L}}{\partial s_i} = 0$$

- Feasibilidad Primal

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

- Feasibilidad dual

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

- Holgura complementaria

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

Las KKT dicen que en el óptimo primal y dual ($(\mathbf{w}^*, b^*, s_i^*)$ y (α_i^*, u_i^*)) se cumplen estas propiedades. Para justificar nuestra noción intuitiva de que los términos agregados no cambian fundamentalmente el problema, nos apoyamos en 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, utilizaremos la propiedad de estacionalidad de la solución para eliminar las variables duales. Desarrollando las derivadas parciales:

-

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathbf{w} \cdot \mathbf{w}^t}{\partial \mathbf{w}} - \frac{\partial \sum_i \alpha_i r_i}{\partial \mathbf{w}} = 2\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

$$\rightarrow \mathbf{w} = \frac{1}{2} \sum_i \alpha_i y_i \mathbf{x}_i$$

-

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

$$\rightarrow \sum_i \alpha_i y_i = 0$$

-

$$\frac{\partial \mathcal{L}}{\partial s_i} = \frac{\partial c \sum_i s_i}{\partial s_i} + \frac{\partial \sum_i s_i u_i}{\partial s_i} = c - \alpha_i - u_i = 0$$

$$\rightarrow \alpha_i = c - u_i, \quad i = 1, \dots, |D|$$

Reemplazando en \mathcal{L} :

$$\mathcal{L} = \mathbf{w} \cdot \mathbf{w}^t + c \sum_i s_i - \sum_i \alpha_i r_i - \sum_i s_i u_i$$

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} :

$$\begin{aligned}\mathcal{L} &= \mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i s_i + \sum_i s_i u_i - \sum_i \alpha_i r_i - \sum_i s_i u_i \\ &= \mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i s_i - \sum_i \alpha_i r_i\end{aligned}$$

Como:

$$\begin{aligned}\sum_i \alpha_i r_i &= \sum_i ((\mathbf{w} \cdot \mathbf{x}_i + b) \mathbf{y}_i + s_i - 1) \alpha_i \\ &= \sum_i (\mathbf{w} \cdot \mathbf{x}_i) \mathbf{y}_i \alpha_i + \sum_i \mathbf{y}_i \alpha_i b + \sum_i s_i \alpha_i - \sum_i \alpha_i\end{aligned}$$

Dado que $\sum_i \mathbf{y}_i \alpha_i = 0$, entonces $\sum_i \mathbf{y}_i \alpha_i b = 0$. Entonces

$$\sum_i \alpha_i r_i = \sum_i (\mathbf{w} \cdot \mathbf{x}_i) \mathbf{y}_i \alpha_i + \sum_i s_i \alpha_i - \sum_i \alpha_i$$

Reemplazando $\sum_i \alpha_i r_i$ en lag:

$$\begin{aligned}\mathcal{L} &= \mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i s_i - \sum_i \alpha_i r_i \\ &= \mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i s_i - (\sum_i (\mathbf{w} \cdot \mathbf{x}_i) \mathbf{y}_i \alpha_i + \sum_i s_i \alpha_i - \sum_i \alpha_i) \\ &= \mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i s_i - \sum_i (\mathbf{w} \cdot \mathbf{x}_i) \mathbf{y}_i \alpha_i - \sum_i s_i \alpha_i + \sum_i \alpha_i \\ &= \mathbf{w} \cdot \mathbf{w}^t - \sum_i (\mathbf{w} \cdot \mathbf{x}_i) \mathbf{y}_i \alpha_i + \sum_i \alpha_i \\ &= \mathbf{w} \cdot \mathbf{w}^t - \mathbf{w} \cdot \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i + \sum_i \alpha_i\end{aligned}$$

Como $\mathbf{w} = \frac{1}{2} \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i \rightarrow 2\mathbf{w} = \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i$:

$$\begin{aligned}\mathcal{L} &= \mathbf{w} \cdot \mathbf{w}^t - \mathbf{w} \cdot \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i + \sum_i \alpha_i \\ &= \mathbf{w} \cdot \mathbf{w}^t - \mathbf{w} \cdot 2\mathbf{w} + \sum_i \alpha_i \\ &= \mathbf{w} \cdot \mathbf{w}^t - 2\mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i \\ &= -\mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i\end{aligned}$$

Para finalmente librarse de la variable w , utilizamos nuevamente la igualdad anterior y llegamos a :

$$\begin{aligned}\mathcal{L} &= -\mathbf{w} \cdot \mathbf{w}^t + \sum_i \alpha_i \\ &= -(\sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i) \cdot (\sum_j \alpha_j \mathbf{y}_j \mathbf{x}_j) + \sum_i \alpha_i \\ &= \sum_i \alpha_i - \sum_i \sum_j (\alpha_i \mathbf{y}_i \mathbf{x}_i \cdot \alpha_j \mathbf{y}_j \mathbf{x}_j)\end{aligned}$$

Llamaremos a esta forma de la función a optimizar \mathcal{L}_D , por Lagrangiano del Dual. De esta manera, \mathcal{L} no depende de \mathbf{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:

$$\text{Max}_{\alpha, u} \quad \mathcal{L}_D(\alpha, u) = \sum_i \alpha_i - \sum_i \sum_j (\alpha_i \mathbf{y}_i \mathbf{x}_i \cdot \alpha_j \mathbf{y}_j \mathbf{x}_j)$$

sueto a $\alpha_i \geq 0, u_i \geq 0, c = \alpha_i + u_i$

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

$$\text{Max}_{\alpha} \quad \mathcal{L}_D(\alpha) = \sum_i \alpha_i - \sum_i \sum_j (\alpha_i \mathbf{y}_i \mathbf{x}_i \cdot \alpha_j \mathbf{y}_j \mathbf{x}_j) \quad (4.13)$$

sueto a $0 \leq \alpha_i \leq c$

De nuevo, podemos resolver el problema dual con algún algoritmo genérico de optimización cuadrática (la función es cuadrática por $\alpha_i \alpha_j$), calcular:

$$\mathbf{w} = \frac{1}{2} \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i$$

$$b = 1 - (\mathbf{w} \cdot \mathbf{x}_0) \mathbf{y}_i$$

para un vector de soporte \mathbf{x}_0 (ya que sabemos que $\mathbf{w} \cdot \mathbf{x}_0 + b = 1$).

Para eso debemos poder determinar en base a los α_i qué ejemplares \mathbf{x}_i son vectores de soporte. Además, resulta interesante analizar los valores de α_i ya que nos darán indicios de si un ejemplar \mathbf{x}_i está bien clasificado o no, sin necesidad de calcular \mathbf{w} , b y luego la función de clasificación. Esta ventaja es necesaria para la eficiencia de ciertos algoritmos iterativos de optimización como SMO.

Es importante notar que si resolvemos el problema dual con un kernel no lineal que aproveche el truco del kernel, es generalmente preferible no calcular \mathbf{w} explícitamente, ya que de esa manera al clasificar un nuevo ejemplar con la función $f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$, debemos calcular también a $\phi(\mathbf{x})$ explícitamente. 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 $f(\mathbf{x}) = \sum_i \alpha_i y_i \kappa_{\mathbf{x}_i, \mathbf{x}} + b$.

4.4. Análisis de los valores de α_i

Cuando consideramos un ejemplar utilizado para encontrar un hiperplano (\mathbf{w}, b) en el proceso de optimización y analizamos su posición en el espacio, debe ocurrir alguno de estos tres casos:

- $h(\mathbf{x}_i)y_i = 1$: El ejemplar está bien clasificado y es un vector de soporte
- $h(\mathbf{x}_i)y_i > 1$: El ejemplar está bien clasificado y no es un vector de soporte
- $h(\mathbf{x}_i)y_i < 1$: El ejemplar está mal clasificado

3 gráficos, con los tres casos

Los tres casos posibles en la posición de un ejemplar \mathbf{x}^+ luego de encontrar el hiperplano de máximo margen.

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

$$0 \leq \alpha_i \leq c \alpha_i + u_i = c$$

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

Dividiremos en tres casos, $\alpha_i = 0$, $0 < \alpha_i < c$ y $\alpha_i = c$

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

$$\alpha_i = c \rightarrow u_i = 0 \rightarrow s_i \geq 0 \wedge r_i = 0 \rightarrow \text{Caso 1 o 3}$$

Entonces, sabemos que:

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

Entonces, con seguridad, si $0 < \alpha_i < c$, podemos utilizar el hecho de que $h(\mathbf{x}_i)y_i = 1$ para determinar el valor de b .

4.5. El truco del Kernel

En la formulación tradicional de SVM, el modelo resultante se describe en términos de \mathbf{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.

La idea es simple. Generalmente usamos una función no-lineal $\phi: \mathcal{P} \mapsto \mathcal{P}'$ para transformar ejemplares \mathbf{x} del dominio \mathcal{P} a ejemplares \mathbf{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 planteamos y derivamos la formulación de SVM con $\phi(\mathbf{x}_i)$ en lugar de \mathbf{x}_i , obtendríamos con las condiciones KKT que:

$$\mathbf{w} = \sum \phi(\mathbf{x}_i) \alpha_i y_i$$

Y para clasificar un nuevo ejemplar, calculamos:

$$\begin{aligned} h'(\mathbf{x}) &= \mathbf{w} \cdot \phi(\mathbf{x}_i) + b \\ &= \left(\sum \phi(\mathbf{x}_i) \alpha_i y_i \right) \cdot \phi(\mathbf{x}) + b \\ &= \left(\sum (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \alpha_i y_i) \right) + b \end{aligned}$$

Esto no difiere de la metodología genérica que funciona con cualquier método de clasificación. Pero 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 $K(\mathbf{x}_1, \mathbf{x}_2)$ entre ejemplares \mathbf{x}_1 y \mathbf{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 $\langle \cdot, \cdot \rangle$:

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

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

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

Este kernel corresponde a la transformación y producto interno:

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x} \cdot \mathbf{x}'$$

Comprobando que son iguales:

$$\begin{aligned} \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle &= \langle \phi((x_1, x_2)), \phi((x'_1, x'_2)) \rangle \\ &= \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \rangle \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \\ &= x_1^2x_1'^2 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + x_2^2x_2'^2 \\ &= x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 \\ &= (x_1x_1')^2 + 2(x_1x_1')(x_2x_2') + (x_2x_2')^2 \\ &= ((x_1, x_2) \cdot (x_1', x_2'))^2 \\ &= (\mathbf{x} \cdot \mathbf{x}')^2 \end{aligned}$$

Como podemos ver, 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 $K(\mathbf{x}_i, \mathbf{x}_j)$ para ejemplares de entrenamiento \mathbf{x}_i y \mathbf{x}_j , por lo cual es usual construir una matriz simétrica \mathbf{K} donde se precalcula el kernel entre todos los pares de ejemplares tal que $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$.

Ciertos kernels son, hasta cierto grado, de propósito general, como el **kernel gaussiano** $e^{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}}$, el **kernel polinomial** $(\mathbf{x}_1 \cdot \mathbf{x}_2 + b)^d$, y con el **kernel lineal** $\mathbf{x}_1 \cdot \mathbf{x}_2$ recuperamos la formulación original de SVM. Distintos dominios pueden requerir distintos kernels que representen medidas de similitud apropiadas. Si los ejemplares son strings, por ejemplo, podemos utilizar un kernel basado en la **distancia de Damerau-Levenshtein** [65], o uno basado en **Dynamic Time Warping** (DTW) [66] 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, queremos saber si $\exists(\phi, \langle \cdot, \cdot \rangle)$ tal que $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$. Una condición necesaria y suficiente es la **condición de Mercer**:

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

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 \mathbf{K} generada a partir de un conjunto de ejemplares para el uso particular de SVM y otros métodos. Una de ellas es que la matriz \mathbf{K} sea simétrica y positiva semi-definida, o sea $K(\mathbf{x}_i, \mathbf{x}_j) = K_{i,j} = K_{j,i} \geq 0$ para ejemplares $\mathbf{x}_i, \mathbf{x}_j$ 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 \mathbf{K} 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.

4.6. Algoritmo de entrenamiento SMO

La idea esencial del algoritmo **Sequential Minimal Optimization** (SMO), en base a la formulación dual, es partir de la solución "nula" $\alpha_i = 0, i = 1, \dots, |D|$, 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 $\alpha = (\alpha_1, \dots, \alpha_D)$ 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 $\mathbf{x}_i \in D$

Una tolerancia a la violación de las KKT, ϵ

Result: Un hiperplano descrito implícitamente por las variables duales $\alpha \in \mathbb{R}^{|D|}$ y un bias $b \in \mathbb{R}$

$\alpha = (0, \dots, 0)$;

$b = 0$;

while $kkt(\alpha, \epsilon) > 0$ **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 α

Algoritmo 4: Esquema del algoritmo SMO

Podemos considerar esenciales del algoritmo de manera independiente:

- En una sección anterior, analizamos 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 ($h(\mathbf{x}_i)y_i$). Cuando dicha relación no se mantiene, detectamos 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 consideramos solo a α_i y α_j como variables y al resto de las variables duales como constantes.

Nos enfocamos primero en 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 $kkt(\alpha, \epsilon) > 0$.

4.6.1. Optimización de \mathcal{L}_D con α_i y α_j

Por simplicidad, y sin pérdida de generalidad, en esta sección asumimos que las variables duales elegidas para optimizar \mathcal{L}_D son α_1 y α_2 . Primero, llevaremos α_1 a la forma $\alpha_1 = \gamma + s\alpha_2$ (γ y s constantes) tomando en cuenta que optimizamos solo respecto a α_1 y α_2 . Reescribiremos \mathcal{L}_D en una expresión G que solo dependa de α_2 . De esa manera, podremos optimizar G (y por ende, \mathcal{L}_D) solamente respecto a α_2 , y después simplemente calcular el valor de la otra. Luego, derivamos $\frac{\partial G}{\partial \alpha_2}$ para encontrar el máximo. Finalmente, ajustamos el valor máximo considerando las restricciones relevantes sobre α_1 y α_2 provenientes de las KKT, y calculamos α_1 .

Las condiciones de optimalidad $\frac{\partial \mathcal{L}}{\partial b} = 0$ implican $\sum_i \alpha_i y_i = 0$. 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 $n = |D|$. Tomaremos 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 $s = y_1 y_2$, podemos simplificar dicha ecuación:

$$\alpha_1 y_1 + \alpha_2 y_2 = \gamma'$$

$$\alpha_1 + \alpha_2 y_2 y_1 = \gamma' y_1$$

$$\alpha_1 + \alpha_2 s = \gamma' y_1$$

$$\alpha_1 + \alpha_2 s = \gamma$$

De esta manera obtenemos la ecuación de una recta con ordenada γ y pendiente $s = \pm 1$. Como $\alpha_1 + \alpha_2 s = \gamma$, podemos expresar α_1 en términos de α_2 . Procederemos a simplificar la función \mathcal{L}_D para expresarla en términos de α_1 . Pero antes, redefiniremos \mathcal{L}_D sin cambiar su esencia para simplificar la derivación. La nueva expresión es:

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

Donde escalamos el segundo término con la constante $\frac{1}{2}$. Simplificando el nuevo \mathcal{L}_D , comenzamos separando los términos que dependen de α_1 y α_2 del resto:

$$\begin{aligned}
\mathcal{L}_D &= -\frac{1}{2} \left(\sum_i \sum_j \alpha_i y_i \alpha_j y_j \kappa_{i,j} \right) + \sum_i \alpha_i \\
&= \frac{1}{2} \left(-\alpha_1^2 \kappa_{1,1} y_1^2 - \alpha_2^2 \kappa_{2,2} y_2^2 - 2\alpha_1 \alpha_2 \kappa_{1,2} y_1 y_2 \right. \\
&\quad - 2\alpha_1 y_1 \sum_{j=3}^n \kappa_{1,j} y_j \alpha_j - 2\alpha_2 y_2 \sum_{j=3}^n \kappa_{2,j} y_j \alpha_j \\
&\quad \left. - \sum_{i=3}^n \sum_{j=3}^n \alpha_i \alpha_j \kappa_{i,j} y_i y_j \right) \\
&\quad + \sum_{i=3}^n \alpha_i + \alpha_1 + \alpha_2 \\
&= -\frac{1}{2} \alpha_1^2 \kappa_{1,1} - \frac{1}{2} \alpha_2^2 \kappa_{2,2} - \alpha_1 \alpha_2 \kappa_{1,2} s \\
&\quad - \alpha_1 y_1 \sum_{j=3}^n \kappa_{1,j} y_j \alpha_j - \alpha_2 y_2 \sum_{j=3}^n \kappa_{2,j} y_j \alpha_j \\
&\quad - \sum_{i=3}^n \sum_{j=3}^n \alpha_i \alpha_j \kappa_{i,j} y_i y_j \\
&\quad + \sum_{i=3}^n \alpha_i + \alpha_1 + \alpha_2
\end{aligned}$$

Reemplazando $(y_i)^2 = 1$ y $y_1 y_2 = s$, y dado que el máximo de \mathcal{L}_D no depende de $\alpha_3, \dots, \alpha_n$, podemos quitar los términos constantes y optimizar:

$$\begin{aligned}
&= -\frac{1}{2} \alpha_1^2 \kappa_{1,1} - \frac{1}{2} \alpha_2^2 \kappa_{2,2} - \alpha_1 \alpha_2 s \kappa_{1,2} \\
&\quad - \alpha_1 y_1 \sum_{j=3}^n \kappa_{1,j} y_j \alpha_j - \alpha_2 y_2 \sum_{j=3}^n \kappa_{2,j} y_j \alpha_j \\
&\quad + \alpha_1 + \alpha_2
\end{aligned}$$

Ahora, reescribimos con $\alpha_1 = \gamma - s\alpha_2$ para expresar solamente en términos de la variable α_2 :

$$\begin{aligned}
&= -\frac{1}{2} (\gamma - s\alpha_2)^2 \kappa_{1,1} - \frac{1}{2} \alpha_2^2 \kappa_{2,2} - (\gamma - s\alpha_2) \alpha_2 s \kappa_{1,2} \\
&\quad - (\gamma - s\alpha_2) y_1 \sum_{j=3}^n \kappa_{1,j} y_j \alpha_j - \alpha_2 y_2 \sum_{j=3}^n \kappa_{2,j} y_j \alpha_j \\
&\quad + (\gamma - s\alpha_2) + \alpha_2
\end{aligned}$$

Sabiendo que:

$$s^2 = 1$$

$$(\gamma - s\alpha_2)^2 = \gamma^2 - 2\gamma s\alpha_2 + \alpha_2^2$$

$$s y_1 = y_2$$

Y definiendo $v_i = \sum_{j=3}^n \alpha_j y_j \kappa_{i,j}$, llegamos a:

$$\begin{aligned}
&= -\frac{1}{2} \gamma^2 \kappa_{1,1} + \gamma s \alpha_2 \kappa_{1,1} - \frac{1}{2} \alpha_2^2 \kappa_{1,1} \\
&\quad - \frac{1}{2} \alpha_2^2 \kappa_{2,2} - \alpha_2 \gamma s \kappa_{1,2} + \alpha_2^2 \kappa_{1,2} \\
&\quad - \gamma y_1 v_1 + \alpha_2 y_2 v_1 - \alpha_2 y_2 v_2 \\
&\quad + \gamma + \alpha_2 (1 - s)
\end{aligned}$$

Descartando términos constantes $-\gamma^2 \kappa_{1,1}$, $-\gamma y_1 v_1$ y γ :

$$\begin{aligned}
&= \gamma s \alpha_2 \kappa_{1,1} - \frac{1}{2} \alpha_2^2 \kappa_{1,1} \\
&\quad - \frac{1}{2} \alpha_2^2 \kappa_{2,2} - \gamma s \alpha_2 \kappa_{1,2} + \alpha_2^2 \kappa_{1,2} \\
&\quad + \alpha_2 y_2 v_1 - \alpha_2 y_2 v_2 \\
&\quad + \alpha_2 (1 - s)
\end{aligned}$$

Sacando factor común α_2^2 y α_2 :

$$= \alpha_2^2 \left(-\frac{1}{2} \kappa_{1,1} + \kappa_{1,2} - \frac{1}{2} \kappa_{2,2} \right) + \alpha_2 (\gamma s \kappa_{1,1} - \gamma s \kappa_{1,2} + y_2 v_1 - y_2 v_2 + (1-s))$$

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 , consideramos que $\gamma = \alpha_1^v + s \alpha_2^v$ y:

$$\begin{aligned} v_i &= \sum_{j=3}^n \alpha_j y_j \kappa_{i,j} = \left(\sum_{j=3}^n \alpha_j y_j \mathbf{x}_j \right) \cdot \mathbf{x}_i \\ &= \left(\sum_j \alpha_j y_j \mathbf{x}_j - \alpha_1^v y_1 \mathbf{x}_1 - \alpha_2^v y_2 \mathbf{x}_2 \right) \cdot \mathbf{x}_i \\ &= (\mathbf{w} - \alpha_1^v y_1 \mathbf{x}_1 - \alpha_2^v y_2 \mathbf{x}_2) \cdot \mathbf{x}_i \\ &= \mathbf{w} \cdot \mathbf{x}_i - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i} \\ &= (\mathbf{w} \cdot \mathbf{x}_i + b^v) - b^v - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i} \\ &= f_i^v - b^v - \alpha_1^v y_1 \kappa_{1,i} - \alpha_2^v y_2 \kappa_{2,i} \end{aligned}$$

Donde f_i^v es la salida para \mathbf{x}_i bajo el \mathbf{w}^v correspondiente a los viejos parámetros. Entonces, el coeficiente de α_2 es:

$$\begin{aligned} &\gamma s \kappa_{1,1} - \gamma s \kappa_{1,2} + y_2 (v_1 v_2) + 1 - s \\ &= (\alpha_1^v + s \alpha_2^v) s \kappa_{1,1} - (\alpha_1^v + s \alpha_2^v) s \kappa_{1,2} + y_2 (v_1 - v_2) + 1 - s \\ &= \alpha_1^v s \kappa_{1,1} + \alpha_2^v s \kappa_{1,1} - \alpha_1^v s \kappa_{1,2} - \alpha_2^v s \kappa_{1,2} \\ &\quad + y_2 (v_1 - v_2) + 1 - s \\ &= \alpha_1^v s \kappa_{1,1} + \alpha_2^v s \kappa_{1,1} - \alpha_1^v s \kappa_{1,2} - \alpha_2^v s \kappa_{1,2} \\ &\quad + y_2 (f_1^v - b^v - \alpha_1^v y_1 \kappa_{1,1} - \alpha_2^v y_2 \kappa_{1,2} - f_2^v + b^v + \alpha_1^v y_1 \kappa_{1,2} + \alpha_2^v y_2 \kappa_{2,2}) + 1 - s \\ &= \alpha_1^v s \kappa_{1,1} + \alpha_2^v s \kappa_{1,1} - \alpha_1^v s \kappa_{1,2} - \alpha_2^v s \kappa_{1,2} \\ &\quad + y_2 (f_1^v - \alpha_1^v y_1 \kappa_{1,1} - \alpha_2^v y_2 \kappa_{1,2} - f_2^v + \alpha_1^v y_1 \kappa_{1,2} + \alpha_2^v y_2 \kappa_{2,2}) + 1 - s \\ &= \alpha_1^v s \kappa_{1,1} + \alpha_2^v s \kappa_{1,1} - \alpha_1^v s \kappa_{1,2} - \alpha_2^v s \kappa_{1,2} \\ &\quad + y_2 (f_1^v - f_2^v) - \alpha_1^v s \kappa_{1,1} - \alpha_2^v s \kappa_{1,2} + \alpha_1^v s \kappa_{1,2} + \alpha_2^v s \kappa_{2,2} + 1 - s \\ &= \alpha_1^v s \kappa_{1,1} - \alpha_1^v s \kappa_{1,2} + \alpha_1^v s \kappa_{1,2} - \alpha_1^v s \kappa_{1,1} \\ &\quad + \alpha_2^v s \kappa_{1,1} - \alpha_2^v s \kappa_{1,2} - \alpha_2^v s \kappa_{1,2} + \alpha_2^v s \kappa_{2,2} \\ &\quad + y_2 (f_1^v - f_2^v) + 1 - s \\ &= \alpha_2^v (\kappa_{1,1} - 2\kappa_{1,2} + \kappa_{2,2}) \\ &\quad + y_2 (f_1^v - f_2^v) + 1 - s \\ &= y_2 (f_1^v - f_2^v) + 1 - s - \alpha_2^v \eta \\ &= y_2 (f_1^v - f_2^v) + y_2 y_2 - y_1 y_2 - \alpha_2^v \eta \\ &= y_2 (f_1^v - f_2^v + y_2 - y_1) - \alpha_2^v \eta \\ &= y_2 ((f_1^v - y_1) - (f_2^v - y_2)) - \alpha_2^v \eta \end{aligned}$$

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 queda como:

$$G = \frac{1}{2} \eta \alpha_2^2 + (y_2 (E_1^v - E_2^v) - \alpha_2^v \eta) \alpha_2$$

Hemos llegado a una expresión G tal que $G = \mathcal{L}_D + \text{Constante}$, y por ende es equivalente 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 $\frac{\partial G}{\partial \alpha_2} = 0$, tenemos:

$$\begin{aligned}
0 &= \eta \alpha_2 + y_2 (E_1^v - E_2^v) - \alpha_2^v \eta \\
\alpha_2 &= \frac{-y_2 (E_1^v - E_2^v) + \alpha_2^v \eta}{\eta} \\
\alpha_2 &= \alpha_2^v + \frac{-y_2 (E_1^v - E_2^v)}{\eta} \\
\alpha_2 &= \alpha_2^v + \frac{y_2 (E_2^v - E_1^v)}{\eta} \\
\alpha_2 &= \alpha_2^v + \frac{y_2 (E_2^v - E_1^v)}{\eta}
\end{aligned}$$

Armados con esta última ecuación, que relaciona de forma simple α_2 con α_2^v , procedemos ahora a buscar una regla de actualización de las variables α_2 y α_1 para mejorar la solución actual.

4.6.2. Regla de actualización para α_1 y α_2

Debemos tener 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 llamaremos L y H los límites inferiores y superiores que puede tomar la variable α_2 , cuya existencia asumimos y que derivaremos en la sección siguiente. Estos valores límite asegurarían que la nueva solución cumpla las condiciones KKT 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 $\eta \neq 0$. Podemos ver que $\eta \leq 0$, ya que:

$$\eta = -\kappa_{1,1} + 2\kappa_{1,2} - \kappa_{2,2} = -(\kappa_{1,1} - 2\kappa_{1,2} + \kappa_{2,2})$$

$$= -(\mathbf{x}_2 - \mathbf{x}_1)^T (\mathbf{x}_2 - \mathbf{x}_1) = -\|\mathbf{x}_2 - \mathbf{x}_1\|^2 \leq 0$$

Por ende debemos manejar 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 $\eta < 0$, tenemos 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 en α_2 sería:

$$\Delta \alpha_2 = \frac{y_2 (E_2^v - E_1^v)}{\eta}$$

Entonces, podemos 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 $\operatorname{restrict}_{L,H}(x) = \min(\max(L, x), H)$ que devuelve su argumento restringido al intervalo $[L, H]$, la regla quedaría como:

$$\alpha_2 \leftarrow \operatorname{restrict}_{L,H}(\alpha_2^v + \Delta \alpha_2)$$

En resumen, la regla final de actualización sería:

$$\alpha_2 \leftarrow \begin{cases} \underset{\alpha_2 \in \{L, H\}}{\operatorname{argmax}} y_2 (E_1^v - E_2^v) \alpha_2 & \text{si } \eta = 0 \\ \operatorname{restrict}_{L,H}(\alpha_2^v + \Delta \alpha_2) & \text{si } \eta < 0 \end{cases}$$

A continuación derivamos expresiones para los límites L y H , y tratamos el problema de la actualización de las variables E_i al finalizar un paso de optimización exitoso. Luego continuamos con los dos puntos restantes: la elección de α_i y α_j y la condición de fin del algoritmo.

4.6.2.1. Límites para la variable dual α_2

Sabemos que 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) debemos mantener ambas relaciones.

Si $s = 1$, tenemos $\alpha_1^v + \alpha_2^v = \gamma$. 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 $s = -1$, tenemos $\alpha_1 - \alpha_2 = \gamma$. 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:

$$\begin{aligned} pos(1) &= 1 & neg(1) &= 0 \\ pos(-1) &= 0 & neg(-1) &= 1 \end{aligned}$$

y teniendo en cuenta la relación de los segundos argumentos a *max* y *min* con el signo de *s*, podemos combinar ambos casos:

$$\begin{aligned} L &= \max(0, s(\gamma - pos(s)c)) \\ H &= \min(c, s(\gamma + neg(s)c)) \end{aligned}$$

4.6.2.2. Actualización luego de un paso de optimización exitoso

Cuando α_1 y α_2 se cambian con $\Delta\alpha_1$ y $\Delta\alpha_2$, debemos actualizar los E_i y b para que reflejen los nuevos valores de las variables duales.

Sabemos que el error E_i es:

$$E_i = h(\mathbf{x}_i) - y_i = \mathbf{w} \cdot \mathbf{x}_i + b - y_i = \sum_j a_j \kappa_{i,j} y_j + b - y_i$$

Podríamos calcular los E_i de esta manera, pero no conocemos el nuevo valor de b . Pero si $0 < \alpha_1 < c$ o $0 < \alpha_2 < c$, esto quiere decir que dichas variables, que llamaremos variables de soporte, corresponden a vectores de soporte y por ende $E_1 = 0$ o $E_2 = 0$, respectivamente. Entonces podemos despejar b con alguna de las siguientes ecuaciones:

$$\begin{aligned} b &= -\sum_j a_j \kappa_{1,j} y_j + y_1 \text{ o} \\ b &= -\sum_j a_j \kappa_{2,j} y_j + y_2 \end{aligned}$$

Luego, calculamos E_i para todas las variables duales, utilizando la fórmula anterior o el hecho de que:

$$\Delta E_i = \Delta\alpha_1 y_1 \kappa_{1,i} + \Delta\alpha_2 y_2 \kappa_{2,i} + \Delta b$$

Si $\alpha_1 = 0$ o $\alpha_1 = c$ y $\alpha_2 = 0$ o $\alpha_2 = c$, entonces no podemos calcular directamente b , y en la práctica se computa b tanto con α_1 y α_2 , y se promedian los valores como una forma de aproximar el nuevo b .

4.6.3. Elección de las variables α_i y α_j y pseudocódigo del algoritmo

A continuación, presentamos 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. Dejamos, en esta parte, de utilizar las variables α_1 y α_2 como las seleccionadas, y generalizamos utilizando α_i y α_j , respectivamente.

4.6.3.1. 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 $\mathbf{x}_i \in D$

Una tolerancia a la violación de las KKT, ϵ

Result: Un hiperplano descrito implícitamente por las variables duales $\alpha \in \mathbb{R}^{|D|}$ y un bias $b \in \mathbb{R}$

global $\alpha = (0, \dots, 0)$;

global $b = 0$;

global $E = (-y_1, \dots, -y_n)$;

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 $0 < \alpha_i < c$ **then**

 changed = changed OR findAjAndOptimize(i) ;

end

end

end

 examineAll = NOT (changed OR examineAll) ;

end

Retornar b y α ;

Algoritmo 5: Procedimiento principal del algoritmo SMO

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 preferimos utilizar las variables de soporte es que son las que definen realmente el hiperplano, y por ende cambiarlas nos da 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 α 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.

4.6.3.2. Función *findAjAndOptimize(i)*

Dado el primer α_i , buscaremos un α_j apropiado. Antes de eso, debemos verificar que α_i no cumpla las KKT. Para ello, definimos $R_i = y_i E_i$, una variable fácil de computar, y reescribimos las KKT en términos R_i :

$$\alpha_i = 0 \rightarrow 0 \leq R_i$$

$$0 < \alpha_i < c \rightarrow R_i \simeq 0$$

$$\alpha_i = c \rightarrow R_i \leq 0$$

que, introduciendo además la tolerancia, se pueden escribir más sintéticamente como:

$$(0 < \alpha_i \wedge R_i > 0) \vee (\alpha_i < c \wedge R_i < 0) \rightarrow \alpha_i \text{ no cumple las KKT}$$

Si α_i viola dichas condiciones, procedemos a buscar el α_j apropiado. Utilizaremos una heurística que intenta tres cosas en sucesión:

- Primero, tratamos de encontrar 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, tratamos de encontrar un α_j que sea una variable de soporte, por las mismas razones que antes.
- Por último, intentamos 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 comenzamos siempre desde una posición aleatoria del vector para no sesgar el algoritmo.

En pseudocódigo:

```

function findAndOptimize(i) ;
 $R_i = y_i * E_i$  ;
if ( $0 < \alpha_i$  OR  $R_i > 0$ ) AND ( $\alpha_i < c$  OR  $R_i < 0$ ) then
    | return false ;
end
if  $\exists \alpha_j: 0 < \alpha_j < c$  then
    | randomOffset=random(1,|D|) ;
    | max=0 ;
    | for index=1 to |D| do
    | |  $j = (\text{index} + \text{randomOffset}) \% |D|$  ;
    | | if  $0 < \alpha_j < c$  AND  $|E_i - E_j| \geq \text{max}$  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
    | |  $j = (\text{index} + \text{randomOffset}) \% |D|$  ;
    | | if  $0 < \alpha_j < c$  AND optimize(i,j) then
    | | | return true;
    | | end
    | end
    | randomOffset=random(1,|D|) ;
    | for index=1 to |D| do
    | |  $j = (\text{index} + \text{randomOffset}) \% |D|$  ;
    | | if optimize(i,j) then
    | | | return true;
    | | end
    | end
end
return false;

```

Función findAndOptimize(i)

4.6.3.3. Función optimize(i,j)

Finalmente, llegamos al proceso de optimización de \mathcal{L}_D respecto a α_i y α_j . Primero, se calculan las variables relevantes: $H, L, s, \gamma, \eta, \Delta\alpha_j, \Delta\alpha_i$, 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 mencionamos. Para ello, queremos que $\Delta\alpha_j/\alpha_j$ sea más grande que un valor pequeño ϵ' , a definir.

```

function findAndOptimize(i) ;
    Calcular  $H, L, s, \gamma, \eta, \Delta\alpha_j, \Delta\alpha_i$  según las fórmulas dadas anteriormente ;
    if ( $L \geq H$  OR  $\Delta\alpha_j/\alpha_j < \epsilon$ ) then
        | return false ;
    end
    Actualizar  $\alpha_i$  y  $\alpha_j$  ;
    Actualizar  $b$  ;
    Actualizar  $E$  ;
    return true ;

```

Función optimize(i,j)

Los cálculos y actualizaciones e hacen en base a ls fórmulas de las secciones anteriores, y para un pseudocódigo más detallado se puede consultar el artículo original de Platt [64].

4.7. Adaptación para el reconocimiento multiclase

El modelo de SVM que desarrollamos solo considera dos clases. Para resolver problemas multiclase, debemos 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.

Nosotros tomaremos este último camino, que a su vez tiene dos estrategias posibles.

La primera es entrenar $C(C-1)/2$ clasificadores, 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 descrita 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 $\arg\max_c f_c(\mathbf{x})$.

4.8. Apéndice: Condiciones KKT, Lagrangiano y Dual de Wolfe

A continuación describimos brevemente las condiciones KKT y el problema dual [67]. Este apéndice no intenta ser exhaustivo ni explicativo, sino de referencia para las secciones anteriores.

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 $h(x) = 0$, $h \in \mathbb{C}^1$ 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, transformamos nuestro problema a otro que es, en cierto modo, equivalente, para simplificar la tarea original de optimización.

4.8.1. 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:

$$\begin{aligned} \text{Min}_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{sujeto a} \quad & h_i(\mathbf{x}) = 0, i = 0, \dots, n \\ & g_i(\mathbf{x}) \leq 0, i = 0, \dots, m \end{aligned}$$

Asumiendo que f , g_i y h_i son funciones suaves en un entorno del óptimo \mathbf{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:

$$\mathcal{L}(\mathbf{x}, \lambda, \mathbf{u}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) + \sum_{i=1}^n u_i g_i(\mathbf{x})$$

Las condiciones KKT para un mínimo local \mathbf{x}^* del problema son que existan constantes $\lambda = (\lambda_1, \dots, \lambda_m)$ y $\mathbf{u} = (u_1, \dots, u_n)$, llamadas multiplicadores KKT, tales que:

- Estacionalidad

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^*} = 0 \qquad \frac{\partial \mathcal{L}}{\partial \lambda} = 0 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{u}} = 0$$

- Feasibilidad Primal

$$\begin{aligned} g_i(\mathbf{x}^*) &\leq 0, i = 1, \dots, n \\ h_i(\mathbf{x}^*) &= 0, i = 1, \dots, m \end{aligned}$$

- Feasibilidad dual

$$u_i \geq 0, i = 1, \dots, m$$

- Holgura complementaria

$$u_i g_i(\mathbf{x}^*) = 0, i = 1, \dots, n$$

Para que dicho punto \mathbf{x}^* satisfaga las condiciones KKT, el problema deberá también satisfacer una condición de regularidad. Existen varias condiciones de regularidad posibles; la más simple para nuestro propósito es la condición de linealidad de las restricciones, o sea;

$$\begin{aligned} g_i(\mathbf{x}) &\text{ es afín, } i = 1, \dots, n \\ h_i(\mathbf{x}) &\text{ es afín, } i = 1, \dots, m \end{aligned}$$

Estas son condiciones **necesarias** para cualquier óptimo del problema \mathbf{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.

4.8.2. Problema Dual

Para todo problema de optimización de estas características existe un **problema dual** asociado. El problema dual está relacionado al problema original, que llamamos primal, por las condiciones de optimalidad KKT. Tiene la siguiente forma:

$$\begin{aligned}
 &\text{Max}_{\lambda, u} && q(\lambda, u) \\
 &\text{sujeto a} && \lambda_i \geq 0, i=0, \dots, m \\
 &\text{donde} && q(\lambda, u) = \min_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^m \lambda_i h_i(\mathbf{x}) + \sum_{i=1}^n u_i g_i(\mathbf{x}) = \min_{\mathbf{x}} \mathcal{L}
 \end{aligned}$$

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 factibles \mathbf{x} y (λ, u) se cumple que $f(\mathbf{x}) \geq q(\lambda, u)$.

En particular, para los dos óptimos $f(\mathbf{x}^*)$ y $q(\lambda^*, u^*)$ también se cumple, por ende $f(\mathbf{x}^*) \geq q(\lambda^*, u^*)$; a la cantidad $f(\mathbf{x}^*) - q(\lambda^*, u^*)$ 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 $f(\mathbf{x}^*) = q(\lambda^*, u^*)$ y entonces no hay hueco dual. De esta forma, si podemos encontrar relaciones entre las variables del dual y del primal que nos permiten calcular las segundas en base a las primeras, podemos resolver el problema dual, y luego calcular el valor de la solución del primal (\mathbf{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 [67].

Redes neuronales

“No, I’m not interested in developing a powerful brain [..]. All I’m after is just a mediocre brain.”
Alan M. Turing

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

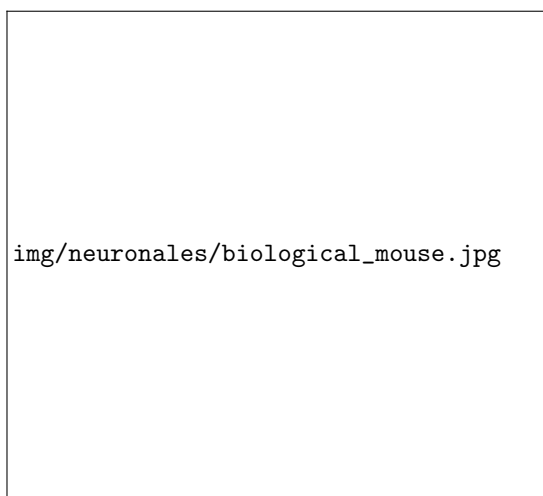


Figura 5.1: Red neuronal biológica. Porción del giro cingular del cerebro de un ratón.

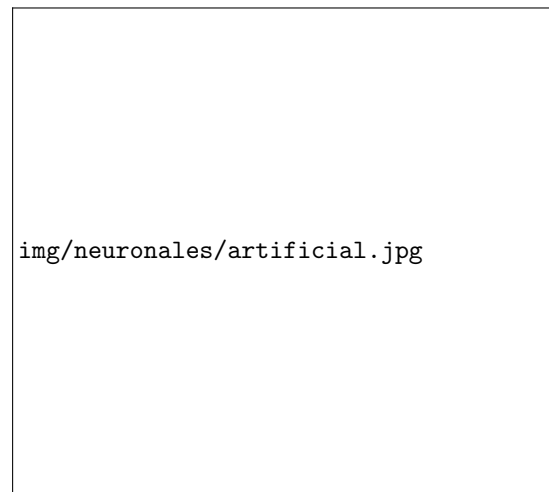


Figura 5.2: Red neuronal artificial. Los círculos representan neuronas y las flechas conexiones entre las mismas.

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**, en donde se estimulan mutuamente. Los estímulos que genera una neurona son función de los estímulos actualmente recibidos, su estado interno (generado por la historia de estímulos pasados) 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. Podemos 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.

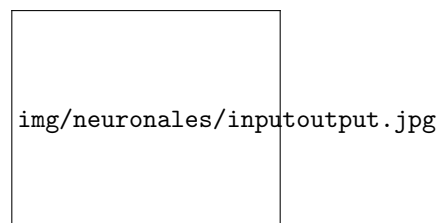


Figura 5.3: Relación entre las neuronas sensoriales, interneuronas y motrices (TODO)

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 actúan 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.

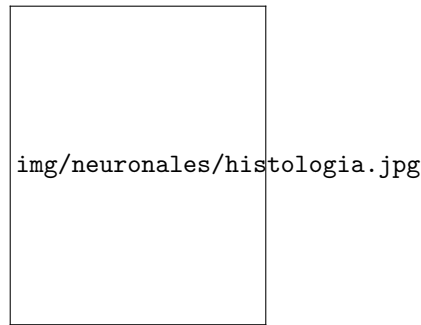


Figura 5.4: Diagrama de las partes que componen una neurona y su interconexión

La estimulación entre neuronas ocurre mediante señales electroquímicas, que se propagan a través de las **sinapsis**, que son conexiones especializadas con otras células 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 siguen los patrones recién descritos.

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 como 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.

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 [68, 69]. 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 [70]. Además, como mencionamos antes, 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.

En general, los modelos de neuronas artificiales toman en cuenta la idea de neuronas de sensoriales, interneuronas y motrices que llamaremos 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 evalúan 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.

A continuación, describimos un modelo general de red neuronal sincrónica, de tiempo discreto, sin recurrencias ni estado permanente. Luego, profundizamos 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).

5.1. 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 mencionamos, 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 (asumimos que todas al mismo tiempo), lo cual causa que estas estimulen a sus adyacentes, y así sucesivamente, en algún orden determinado. Trataremos solo grafos acíclicos, en donde el orden de estimulación es el topológico ¹.

imagen: propagación de estímulos desde la neurona de más a la izquierda al resto de la red

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 $f_i :: \mathbb{R}^{n_i} \mapsto \mathbb{R}$, donde n_i es el grado de entrada de la neurona v_i ².

imagen: red con funciones f_i adentro de cada vértice

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

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 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.

Data:

Un conjunto de ejemplares $D \in \mathbb{R}^d$, posiblemente con una clase una clase $y_i \in \{-1, 1\}$ asociada a cada $\mathbf{x}_i \in D$

Una topología de la red $G = (V, E)$, y una función f_i asociada a cada v_i .

Una tolerancia al error, ϵ

Un valor inicial para el conjunto de parámetros de las funciones, W

Result: Un valor final del conjunto de parámetros de las funciones, W

while $error(D, W) > \epsilon$ **do**

 | Modificar W para que la red tenga menor error con los ejemplares de D ;

end

Retornar W

Algoritmo 6: Esquema general de un algoritmo de entrenamiento para una red neuronal

5.1.1. El perceptrón: una red neuronal simple

Podemos revisar el perceptrón como una red neuronal para un problema de clasificación de 2 clases, donde si $\mathbf{x} \in \mathbb{R}^d$, entonces tenemos d neuronas de entrada, todas conectadas a una neurona de salida que calcula la clase estimada del ejemplar. Llamaremos $o(\mathbf{x})$ a la salida de la red:

IMAGEN: d neuronas de entrada, una de salida, $o(\mathbf{x})$

Cada neurona de entrada representa un componente del ejemplar \mathbf{x} , simulando la estimulación de un órgano sensorial con los valores del ejemplar.

La neurona de salida utiliza la función:

$$f(\mathbf{x}) = \theta_t(h(\mathbf{x})) \quad \text{donde:}$$

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

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

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

Dado un ejemplar \mathbf{x} , se estimulan las neuronas de entrada con los d elementos del vector \mathbf{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 $o(\mathbf{x})$ que representa su estímulo de salida. Como esta neurona no transmite ese estímulo a otra neurona, asumimos que es la salida observable de la red.

Para entrenar esta versión del perceptrón se puede utilizar el mismo algoritmo que el descrito en la sección 3.4.1, donde el vector normal \mathbf{w} corresponde al parámetro de la función de la neurona de salida de la red.

El perceptrón es una red neuronal que tiene una capacidad de separación limitada a un hiperplano. A continuación, presentamos la arquitectura feedforward de red neuronal, una extensión natural del perceptrón. Se desarrollará el algoritmo tradicional de entrenamiento para esas redes, backpropagation, y una de sus variantes, de mayor efectividad, resilient backpropagation. Luego, trataremos las redes competitivas, también con su algoritmo de entrenamiento clásico, vector quantization.

5.2. 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

¹Si 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.

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

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

⁴Estamos asumiendo 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.

el resultado de la red. Es uno de los modelos más populares de red neuronal, uno de los que popularizó la técnica.

imagen: red feedforward con varias capas y funciones no lineales caption: Una red feedforward con cinco capas: la primera, de entrada, luego tres capas ocultas, y finalmente una capa de salida, con varias neuronas de salida. Las funciones de las neuronas pueden ser no-lineales.

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

5.2.1. Varias clases

Supongamos que buscamos clasificar C clases de ejemplares. Podemos 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.

Imagen: Caption: Una red neuronal con 2 neuronas de entrada y 3 de salida.

Imagen: Ejemplares de tres clases distintas. Cada grupo de ejemplares está separado del resto por un hiperplano. Caption: Ejemplares de tres clases distintas. Cada grupo de ejemplares está separado del resto por un hiperplano.

Interpretamos la salida de esta red como un vector σ , donde σ_i es el resultado calculado para la neurona de salida i . En este caso, un vector de la clase c está bien clasificado si $\sigma_c = 1$ y $\sigma_j = -1, j \neq c$. Podemos entrenar esta red aplicando el algoritmo del perceptrón a cada neurona por separado. Podemos ver la topología de la red como dos capas de neuronas, las de entrada y las de salida. A continuación, generalizamos dicha noción.

5.2.2. Capas ocultas

Supongamos que buscamos clasificar ejemplares de las 3 clases de la figura X. Se ve claramente que no existe un hiperplano que pueda separar la clase 2 del resto de las clases.

Imagen: Ejemplares de tres clases distintas, la 2da entre las otras dos. La clase 2 no puede separarse del resto con un solo hiperplano. Caption: Ejemplares de tres clases distintas. La clase 2 no puede separarse del resto con un solo hiperplano.

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 podemos separar a la clase 2 del resto utilizando *dos* hiperplanos, entre la clase 1 y la 2, y la 2 y la 3.

Imagen: Ejemplares de tres clases distintas, la 2da entre las otras dos. Dos hiperplanos separan a las clases 1 y 2 y 2 y 3 figura Y

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

Podemos 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.

Imagen: La misma imagen de antes con 2 entradas y 3 salidas pero además con una oculta de dos neuronas caption: Un perceptrón multiclase con una capa oculta de dos neuronas.

En el ejemplo, podemos 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, $(x,y) = (1,1)$.

Imagen: plano centrado en 0,0; los ejemplares de cada clase en su cuadrante cerca del origen (no separables) caption: Transformación de las neuronas de entrada a las neuronas ocultas. Se muestran los ejemplares originales en el rango de la transformación.

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 practicidad 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 esta red, utilizaremos el algoritmo backpropagation, a describirse luego. Antes, introduciremos la necesidad de emplear funciones no lineales en las neuronas para resolver problemas más complejos.

5.2.3. Funciones no lineales

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

Reconsideremos ahora la estructura de las funciones de las neuronas. A las funciones f_i 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 nuestro 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.

Supongamos que tenemos un problema de dos clases, en donde las mismas no pueden separarse de ninguna manera por un clasificador lineal, ni siquiera uno de dos capas.

Imagen: dos clases, una metida en la otra tipo ojo caption: Dos clases que no son linealmente separables

Debemos 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, podemos definir una función:

$$f(\mathbf{x}) = \theta_t(c(\mathbf{x}))$$

$$c(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}\| - r$$

que representa la pertenencia de un ejemplar a un círculo con centro \mathbf{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, obtendremos un clasificador cuya salida es 1 para los ejemplares dentro del círculo y 2 para los otros.

Imagen: dos clases, una metida en la otra tipo ojo caption: 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, obtuvimos una transformación a un espacio donde los ejemplares son separables.

Por otro lado, podemos tener también un cambio en la función de activación. Si tenemos un problema que no es linealmente separable, donde la separación de las clases está dada por una superficie parecida a un hiperplano, pero ligeramente no lineal, se puede doblar el hiperplano utilizando una función de activación no lineal.

Imagen: dos clases, tipo ying yang, un hiperplano $((\mathbf{w} \cdot \mathbf{x})^3)$ doblado las separaría. caption: Ejemplares de dos clases que no son linealmente separables.

Aplicamos entonces la función de combinación lineal junto con una función de activación no lineal $\theta_3(x) = x^3$, de manera que $f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x})^3$. Optimizando el error respecto a los parámetros de \mathbf{w} , se puede obtener una superficie de separación adecuada.

Imagen: dos clases, tipo ying yang, un hiperplano $((\mathbf{w} \cdot \mathbf{x})^3)$ doblado las separa. caption: Ejemplares de dos clases que no son linealmente separables.

REVISAR TODO ESTO; ME PARECE QUE LE ESTOY PIFIANDO CON LOS DIBUJOS

Es importante notar que en el caso de utilizar una red con una capa oculta, 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.

5.2.3.1. 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 "primeros" separadores, debido a que particionan el espacio *grosso modo*. En el caso más simple, se puede utilizar la función de activación θ_t , que nos da 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 *tapanán* 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 saturan en dos valores, típicamente 0 y 1 o -1 y 1 [71]. Las sigmoideas más utilizadas son la función logística S y la tangente hiperbólica \tanh :

$$S(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

Imagen: logística y tanh al lado caption: La función logística $S(x) = \frac{1}{1 + e^{-x}}$ y 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 saturan 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. Utilizamos 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 \mathbf{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.

5.2.3.2. El modelo feedforward

Combinando estas tres ideas, podemos definir el modelo feedforward como una red acíclica con L capas, donde la primera es una capa de entrada, las $L - 2$ siguientes son capas ocultas, y la última capa es de salida. Las neuronas de la capa k solo se conectan con las de la capa $k + 1$, exceptuando las de la capa de salida que no tienen conexiones salientes. Cada capa puede tener un número distinto de neuronas n_i .

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.

Imagen: red neuronal feedforward caption: Red neuronal feedforward con 3 capas.

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

- Utilizaremos 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 $c(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$.

Nos referiremos 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 de salida de dicha capa son $\mathbf{o}^0 = (o_1^0, \dots, o_d^0)$. Las neuronas de la capa de entrada no tienen ninguna función (o utilizan la función identidad), y entonces al ser estimulada con un ejemplar \mathbf{x} , los valores de sus neuronas son $\mathbf{o}^0 = \mathbf{x}$.

Tendremos una función de activación θ^j para las neuronas de las otras capas ($j > 0$). Además, para la neurona i de la capa j , tendremos un vector de coeficientes del hiperplano de la función de combinación $\mathbf{w}_i^j \in \mathbb{R}^{n_{j-1}}$. Entonces, para cada capa $j > 0$ tenemos una matriz de coeficientes $\mathbf{W}^j \in \mathbb{R}^{n_j \times n_{j-1}}$ que *pesan* las salidas de las neuronas de la capa $j - 1$.

De esta manera, la salida de la neurona i de la capa j , o_i^j , puede expresarse como:

$$\begin{aligned} o_i^j &= \theta_i(\text{net}_i^j) \\ \text{net}_i^j &= \sum_{k=1}^{n_{j-1}} o_k^{j-1} \mathbf{w}_i^j(k) = \mathbf{o}^{j-1} \cdot \mathbf{w}_i^j \end{aligned}$$

En esta expresión, net_i^j se conoce como la entrada neta a la neurona, asumiendo que el escalado de las salidas de las neuronas de la capa anterior por los pesos \mathbf{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:

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

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:

$$\begin{aligned} o_i^2 &= \theta_2(\text{net}_i^2) \\ &= \theta_2(\mathbf{o}^1 \cdot \mathbf{w}_i^2) \\ &= \theta_2([\theta_1(\text{net}_1^1), \dots, \theta_1(\text{net}_{n_1}^1)] \cdot \mathbf{w}_i^2) \\ &= \theta_2([\theta_1(\mathbf{o}^0 \cdot \mathbf{w}_1^1), \dots, \theta_1(\mathbf{o}^0 \cdot \mathbf{w}_{n_1}^1)] \cdot \mathbf{w}_i^2) \\ &= \theta_2([\theta_1(\mathbf{x} \cdot \mathbf{w}_1^1), \dots, \theta_1(\mathbf{x} \cdot \mathbf{w}_{n_1}^1)] \cdot \mathbf{w}_i^2) \end{aligned}$$

Este modelo tiene, a priori, tantos parámetros como pesos $w_i^j(k)$. 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 descrito el modelo, tratamos ahora el algoritmo backpropagation para entrenar la red, y luego una de sus mejoras, el resilient backpropagation.

5.2.4. Algoritmo de entrenamiento Backpropagation

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

5.2.4.1. Error cuadrático de una red neuronal

Dada una red neuronal con $k+1$ capas y un ejemplar \mathbf{x}_i , la salida de la red es $o^k(\mathbf{x}_i)$. Si la clase asociada al ejemplar es \mathbf{y}_i , donde ahora $\mathbf{y}_i = (y_1, \dots, y_C)$ es un vector que señala la pertenencia del ejemplar a cada clase, podemos definir el error cuadrático la red sobre un conjunto de datos D como⁵:

$$E = \sum_{\mathbf{x}_i \in D} E(\mathbf{x}_i)$$

$$E(\mathbf{x}) = \sum_{l=1}^C (o_l^k - y_l)^2$$

Donde en la expresión E_x asumimos que o_l^k es la salida de la neurona l de la capa de salida para el ejemplar \mathbf{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 α , nuestro objetivo es:

$$\text{Min}_{\alpha} E(\alpha) = \sum_{\mathbf{x}_i \in D} E_{\mathbf{x}_i}$$

En nuestro caso, si utilizamos funciones de activación sin parámetros, y tenemos una red de $k+1$ capas, nuestros parámetros estarán dados por las matrices de pesos de cada capa $\alpha = \{W^1, \dots, W^k\}$, y el problema definitivo es:

$$\text{Min}_{\{W^1, \dots, W^k\}} E(W^1, \dots, W^k) = \sum_{\mathbf{x}_i \in D} E_{\mathbf{x}_i}$$

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

$$\frac{\partial E}{\partial W^1}, \dots, \frac{\partial E}{\partial W^k}$$

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

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

Donde α es una tasa de aprendizaje que podemos utilizar para regular la magnitud de los cambios. Esta regla nos permite movernos de a pasos en el espacio de parámetros en la dirección en que el error decrece.

Dado que el desarrollo de las $\frac{\partial E}{\partial W^j}$ suele traer confusión debido a la repetida aplicación de la regla de la cadena para volver para atrás por las capas de la red, llegaremos 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 $k+1$ capas.

5.2.4.2. Backpropagation con una red de dos capas y neuronas lineales

5.2.4.2.1. Derivación de $\frac{\partial E}{\partial W^j}$: Como ejemplo, supongamos 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 $\theta(x) = x$. Asumimos también un conjunto de datos D con un solo ejemplar, $\mathbf{x} = (1, 3)$ y salida esperada 0,1. Entonces, la función error está definida en base a los dos pesos de la red, $\mathbf{w}_1^1(1), \mathbf{w}_1^1(2)$ que llamaremos w_1 y w_2 por simplicidad. La función de error para esta red es:

$$E(w_1, w_2) = (x_1 w_1 - y_1)^2 + (x_2 w_2 - y_2)^2$$

$$E(w_1, w_2) = (1w_1 - 0)^2 + (3w_2 - 1)^2$$

Imagen: Superficie de la función de error $E(w_1, w_2)$. caption: Superficie de la función de error $E(w_1, w_2)$.

Imagen: Contorno de la función de error $E(w_1, w_2)$. caption: Contorno de la función de error $E(w_1, w_2)$.

El opuesto del gradiente de la función nos indica la dirección del espacio de parámetros que minimiza E :

$$-\nabla E = -\left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}\right) = -((x_1 w_1 - y_1)x_1 + (x_2 w_2 - y_2)x_2)$$

$$= -\left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}\right) = -((w_1 - 0) + (3w_2 - 1)3)$$

Imagen: Superficie y contorno, con el gradiente, de la función de error $E(w_1, w_2)$. caption: Opuesto del gradiente de la función de error $E(w_1, w_2)$.

⁵Incluimos la constante $\frac{1}{2}$ para simplificar la derivada de $E(\mathbf{x})$.

⁶Si bien es poco claro que $E(\mathbf{x})$ depende de un ejemplar en particular viendo solamente su definición, hacemos 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.

5.2.4.2.2. 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.

Imagen: Superficie y contorno, con gradiente de la función de error $E(w_1, w_2)$. Camino desde una posición inicial hasta una posición con poco error en el espacio de parámetros caption: 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 $\mathbf{x} = (1, 3) \in \mathbb{R}^2$, con clase $\mathbf{y} = (0, 1)$

Una tolerancia al error ϵ

Una tasa de aprendizaje α

Result: Un vector de parámetros $\mathbf{w} = (w_1, w_2)$ optimizado para clasificar \mathbf{x} con un error menor a ϵ

$\mathbf{w} = \text{random}(2)$;

$\text{error} = \infty$;

while $\text{error} > \epsilon$ **do**

$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$;

$w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2}$;

$\text{error} = E(w_1, w_2)$;

end

Retornar \mathbf{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.

Imagen: Una función con un mínimo local, un punto en el espacio de parámetros de un lado de un mínimo local, y un paso al otro lado del mínimo local. caption: 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 funciones de activación arbitrarias.

Generalizando este algoritmo para un conjunto de ejemplares D , podemos escribir el error de un ejemplar \mathbf{x} con clase \mathbf{y} como:

$$\begin{aligned} -\nabla E(\mathbf{x}) &= -\left(\frac{\partial E(\mathbf{x})}{\partial w_1}, \frac{\partial E(\mathbf{x})}{\partial w_2}\right) \\ &= -((x_1 w_1 - y_1)x_1, (x_2 w_2 - y_2)x_2) \end{aligned}$$

Data:

Un conjunto de ejemplares $\mathbf{x}_i \in D$, con $D \subset \mathbb{R}^2$, y clase $\mathbf{y}_i \in \mathbb{R}^2$

Una tolerancia al error ϵ

Una tasa de aprendizaje α

Result: Un vector de parámetros $\mathbf{w} = (w_1, w_2)$ optimizado para clasificar D con un error menor a ϵ

$\mathbf{w} = \text{random}(2)$;

$\text{error} = \infty$;

while $\text{error} > \epsilon$ **do**

$dw_1 = 0$;

$dw_2 = 0$;

for $\mathbf{x}_i \in D$ **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

$w_1 = w_1 - \alpha dw_1$;

$w_2 = w_2 - \alpha dw_2$;

$\text{error} = E(w_1, w_2)$;

end

Retornar \mathbf{w} ;

Algoritmo 8: Esquema del algoritmo Backpropagation para el problema de ejemplo con un conjunto de datos arbitrario D .

Continuando con la generalización, podemos formular este algoritmo para una red neuronal feedforward de 3 capas y una función de activación arbitraria y \mathbb{C}^1 .

5.2.4.3. Derivación de $\frac{\partial E}{\partial \mathbf{W}}$ con una red de tres capas

5.2.4.3.1. Derivación de $\frac{\partial E}{\partial \mathbf{W}}$: 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, tenemos solo dos capas con pesos para optimizar, la 1, oculta, y la 2, de salida. Entonces, tendremos que derivar sólo dos reglas de actualización, una para los pesos de cada capa.

Comenzamos la más simple, que involucra la derivada de la salida o_m^2 respecto a un peso $w_i^2(p)$ para un patrón \mathbf{x} :

$$\frac{\partial o_m^2}{\partial w_i^2(p)} = \begin{cases} \frac{\partial o_m^2}{\partial w_i^2(p)} & \text{si } i = m \\ 0 & \text{de lo contrario} \end{cases}$$

debido a que el vector de pesos \mathbf{w}_i^2 no interviene en el cálculo de o_m^2 para $i \neq m$. Si $i = m$, continuamos expandiendo:

$$\begin{aligned} \frac{\partial o_i^2}{\partial w_i^2(p)} &= \frac{\partial \theta_2(\mathbf{o}^1 \cdot \mathbf{w}_i^2)}{\partial w_i^2(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_i^2) \frac{\partial \mathbf{o}^1 \cdot \mathbf{w}_i^2}{\partial w_i^2(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_i^2) \frac{\partial \sum_{l=1}^{n_1} o_l^1 w_i^2(l)}{\partial w_i^2(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_i^2) \frac{\partial (o_i^1 w_i^2(p))}{\partial w_i^2(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_i^2) o_i^1 \end{aligned}$$

Entonces, la derivada del error para la neurona m de la capa de salida y el patrón \mathbf{x} respecto a $w_i^2(p)$ es:

$$\begin{aligned} \frac{\partial E_m(\mathbf{x})}{\partial w_i^2(p)} &= \frac{\partial (\frac{1}{2}(o_m^2 - y_m)^2)}{\partial w_i^2(p)} \\ &= (o_m^2 - y_m) \frac{\partial o_m^2}{\partial w_i^2(p)} \\ &= \begin{cases} (o_i^2 - y_i) \frac{\partial o_i^2}{\partial w_i^2(p)} & \text{si } i = m \\ 0 & \text{de lo contrario} \end{cases} \end{aligned}$$

Y la derivada total del error para la capa de salida dado un patrón \mathbf{x} :

$$\frac{\partial E(\mathbf{x})}{\partial w_i^2(p)} = \sum_{m=1}^{n_2} \frac{\partial E_m(\mathbf{x})}{\partial w_i^2(p)} = \frac{\partial E_i(\mathbf{x})}{\partial w_i^2(p)}$$

Para los pesos de la capa 1, tenemos la derivada de la salida o_m^2 respecto a un peso $w_i^1(p)$ para un patrón \mathbf{x} :

$$\begin{aligned} \frac{\partial o_m^2}{\partial w_i^1(p)} &= \frac{\partial \theta_2(\mathbf{o}^1 \cdot \mathbf{w}_m^2)}{\partial w_i^1(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_m^2) \frac{\partial \mathbf{o}^1 \cdot \mathbf{w}_m^2}{\partial w_i^1(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_m^2) \frac{\partial \sum_{l=1}^{n_1} o_l^1 w_m^2(l)}{\partial w_i^1(p)} \\ &= \theta_2'(\mathbf{o}^1 \cdot \mathbf{w}_m^2) \sum_{l=1}^{n_1} w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)} \end{aligned}$$

Expandiendo $\frac{\partial o_l^1}{\partial w_i^1(p)}$, de forma análoga a como expandimos $\frac{\partial o_m^2}{\partial w_i^2(p)}$:

$$\frac{\partial o_l^1}{\partial w_i^1(p)} = \begin{cases} \frac{\partial o_l^1}{\partial w_i^1(p)} & \text{si } i = l \\ 0 & \text{de lo contrario} \end{cases}$$

Y entonces si $i = l$:

$$\begin{aligned}
&= \frac{\partial \theta_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1)}{\partial w_i^1(p)} \\
&= \theta'_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1) \frac{\partial \mathbf{o}^0 \cdot \mathbf{w}_i^1}{\partial w_i^1(p)} \\
&= \theta'_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1) \frac{\partial \sum_{q=1}^{n_1} o_q^0 w_i^1(q)}{\partial w_i^1(p)} \\
&= \theta'_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1) o_p^0 \\
&= \theta'_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1) x_p
\end{aligned}$$

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 podremos luego generalizar la regla de actualización a $k+1$ capas.

Volviendo al caso de 3 capas, la expresión para $\frac{\partial o_m^2}{\partial w_i^1(p)}$ es entonces:

$$\begin{aligned}
\frac{\partial o_m^2}{\partial w_i^1(p)} &= \theta'_2(\mathbf{o}^1 \cdot \mathbf{w}_m^2) \sum_{l=1}^{n_1} w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)} \\
&= \theta'_2(\mathbf{o}^1 \cdot \mathbf{w}_m^2) w_m^2(l) \frac{\partial o_l^1}{\partial w_i^1(p)} \\
&= \theta'_2(\mathbf{o}^1 \cdot \mathbf{w}_m^2) w_m^2(l) \theta'_1(\mathbf{o}^0 \cdot \mathbf{w}_i^1) x_p
\end{aligned}$$

La derivada del error para la neurona de salida m respecto al peso $w_i^1(p)$ es entonces:

$$\begin{aligned}
\frac{\partial E_m(\mathbf{x})}{\partial w_i^1(p)} &= \frac{\partial (\frac{1}{2} (o_m^2 - y_m)^2)}{\partial w_i^1(p)} \\
&= (o_m^2 - y_m) \frac{\partial o_m^2}{\partial w_i^1(p)}
\end{aligned}$$

Y la derivada total del error para la capa de salida dado un patrón \mathbf{x} :

$$\frac{\partial E(\mathbf{x})}{\partial w_i^1(p)} = \sum_{m=1}^{n_2} \frac{\partial E_m(\mathbf{x})}{\partial w_i^1(p)}$$

Finalmente, las reglas de actualización para los pesos de las neuronas de las capas 1 y 2, respectivamente, son:

$$\begin{aligned}
w_i^1(p) &\leftarrow w_i^1(p) - \alpha \frac{\partial E(\mathbf{x})}{\partial w_i^1(p)} \\
w_i^2(p) &\leftarrow w_i^2(p) - \alpha \frac{\partial E(\mathbf{x})}{\partial w_i^2(p)}
\end{aligned}$$

5.2.4.3.2. Algoritmo Para el algoritmo, utilizaremos como variables las matrices de pesos de cada capa llamándolas \mathbf{W}^1 y \mathbf{W}^2 , y matrices de derivadas de esos pesos $d\mathbf{W}^1$ y $d\mathbf{W}^2$, del mismo tamaño, y asumiremos que $\frac{\partial E}{\partial \mathbf{W}^1}$ calcula una matriz de derivadas del error respecto a cada peso de la capa 1, y lo mismo con $\frac{\partial E}{\partial \mathbf{W}^2}$. Entonces, $\mathbf{W}^j(i, p) = \mathbf{w}_i^j(p)$ y $\frac{\partial E}{\partial \mathbf{W}^j(i, p)} = \frac{\partial E}{\partial \mathbf{w}_i^j(p)}$.

Modificando el algoritmo visto antes para actualizar ahora los pesos de cada capa, tenemos:

Data:

Un conjunto de ejemplares $\mathbf{x}_i \in D$, con $D \subset \mathbb{R}^d$, y clase $\mathbf{y}_i \in \mathbb{R}^{n_2}$.

Una tolerancia al error ϵ .

Una tasa de aprendizaje α .

Una cantidad de neuronas ocultas n_1 .

Result: Pesos \mathbf{W}^1

y \mathbf{W}^2 de cada neurona oculta \mathbf{w}_i^1 y de salida \mathbf{w}_i^2 , optimizados para clasificar D con un error menor a ϵ

$\mathbf{W}^1 = \text{random}(n_1, d)$;

$\mathbf{W}^2 = \text{random}(n_2, n_1)$;

$\text{error} = \infty$;

while $\text{error} > \epsilon$ **do**

$d\mathbf{W}^1 = \frac{\partial E}{\partial \mathbf{W}^1}$;

$d\mathbf{W}^2 = \frac{\partial E}{\partial \mathbf{W}^2}$;

$\mathbf{W}_1 = \mathbf{W}_1 - \alpha d\mathbf{W}_1$;

$\mathbf{W}_2 = \mathbf{W}_2 - \alpha d\mathbf{W}_2$;

$\text{error} = E(\mathbf{W}^1, \mathbf{W}^2)$;

end

Retornar $\mathbf{W}^1, \mathbf{W}^2$;

Algoritmo 9: Esquema del algoritmo Backpropagation para una red de tres capas.

5.2.4.4. Derivación de $\frac{\partial E}{\partial w_i^j}$ para cualquier topología de red

Podemos escribir la forma genérica de la derivada de la salida de cada neurona i' de cada capa j' y para un ejemplar particular \mathbf{x} , respecto a un peso $w_i^j(p)$, $j' \geq j$ ⁷:

$$\begin{aligned} \frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} &= \frac{\partial \theta_{j'}(net_{i'}^{j'})}{\partial w_i^j(p)} \\ &= \frac{\partial \theta_{j'}(net_{i'}^{j'})}{\partial net_{i'}^{j'}} \frac{\partial net_{i'}^{j'}}{\partial w_i^j(p)} \\ &= \theta'_{j'}(net_{i'}^{j'}) \frac{\partial net_{i'}^{j'}}{\partial w_i^j(p)} \\ &= \theta'_{j'}(net_{i'}^{j'}) \frac{\partial \mathbf{w}_{i'}^{j'} \cdot \mathbf{o}^{j'-1}}{\partial w_i^j(p)} \\ &= \theta'_{j'}(net_{i'}^{j'}) \sum_{l=1}^{n_{(j'-1)}} \frac{\partial (w_{i'}^{j'}(l) o_l^{j'-1})}{\partial w_i^j(p)} \end{aligned}$$

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, debemos utilizar la regla de la cadena con todas estas entradas; como estas entradas son salidas de otras neuronas, podemos utilizar la regla recursiva:

$$\frac{\partial o_{i'}^{j'}}{\partial w_i^j(p)} = \theta'_{j'}(net_{i'}^{j'}) \sum_{l=1}^{n_{(j'-1)}} w_{i'}^{j'}(l) \frac{\partial o_l^{j'-1}}{\partial w_i^j(p)}$$

⁷El peso $w_i^j(p)$ es el componente p del vector de pesos $\mathbf{w}_i^j \in \mathbb{R}^{n_{j-1}}$ para la conexiones de la capa $j-1$ a la neurona i de la capa j .

Y entonces la derivada del error para la neurona de salida o_i^k , dado un ejemplar \mathbf{x} , es:

$$\frac{\partial E_i(\mathbf{x})}{\partial w_i^j(p)} = \frac{\partial(\frac{1}{2}(o_i^p - y_i^p)^2)}{\partial w_i^j(p)} = (o_i^{j'} - y_i^{j'}) \frac{\partial o_i^{j'}}{\partial w_i^j(p)}$$

Y la derivada del error para la capa de salida respecto a $w_i^j(p)$:

$$\frac{\partial E(\mathbf{x})}{\partial w_i^j(p)} = \sum_{i=1}^{n_k} \frac{\partial E_i(\mathbf{x})}{\partial w_i^j(p)}$$

Donde como ahora tenemos una fórmula genérica para $\frac{\partial o_i^{j'}}{\partial w_i^j(k)}$ podemos expandirla de acuerdo a la topología de la red.

Entonces, podemos calcular la derivada del error para cualquier peso, $\frac{\partial E}{\partial w_i^j(p)} = \sum_{\mathbf{x}_i \in D} \frac{\partial E(\mathbf{x}_i)}{\partial w_i^j(p)}$, y podemos aplicar la regla de actualización $w_i^j(p) \leftarrow w_i^j(p) - \alpha \frac{\partial E}{\partial w_i^j(p)}$.

5.2.4.4.1. Algoritmo El algoritmo final queda como:

Data:

Una topología dada por $k+1$, el número de

capas, y n_i la cantidad de neuronas de la capa i . Un conjunto de ejemplares $\mathbf{x}_i \in D$, con $D \subset \mathbb{R}^d$, y clase $\mathbf{y}_i \in \mathbb{R}^{n_k}$.

Una tolerancia al error ϵ .

Una tasa de aprendizaje α .

Result: Pesos \mathbf{W}^j de las neuronas de cada capa $j > 0$, optimizados para clasificar D con un error menor a ϵ

for $j=1$ to k **do**

$\mathbf{W}^1 = \text{random}(n_j, n_{j-1})$;

end

$\text{error} = \infty$;

while $\text{error} > \epsilon$ **do**

for $j=1$ to k **do**

$d\mathbf{W}^j = \frac{\partial E}{\partial \mathbf{W}^j}$;

end

for $j=1$ to k **do**

$\mathbf{W}_j = \mathbf{W}_j - \alpha d\mathbf{W}_j$;

end

$\text{error} = E(\mathbf{W}^1, \dots, \mathbf{W}^k)$;

end

Retornar $\mathbf{W}^1, \dots, \mathbf{W}^k$;

Algoritmo 10: Esquema del algoritmo Backpropagation para una red de tres capas.

5.2.4.5. 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 condición de corte del algoritmo. En la línea del algoritmo:

$$\text{error} = E(\mathbf{W}^1, \dots, \mathbf{W}^k)$$

Estamos asumiendo que el error se calcula sobre los patrones del conjunto de entrenamiento D_e . Una mejora posible es dividir el conjunto de datos original D en tres; el conjunto de entrenamiento D_e , el de prueba D_p , y otro de validación, 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 estamos entrenando 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 la capa oculta, h . 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.

Imagen: Un conjunto de datos con muchísimos hiperplanos superpuestos, que clasifican cada ejemplar casi individualmente caption: Hiperplanos definidos por las neuronas ocultas de una red con demasiadas neuronas para la dificultad inherente del problema.

Por otro lado, 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.

Imagen: Un conjunto de datos con cinco clases y solo dos hiperplanos para separarlas caption: 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.

5.2.4.6. 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 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, lo consideramos un lugar a evitar ya que el algoritmo backpropagation puede quedarse atascado⁸.

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:

- **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 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 varíe 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, podemos aumentar la granularidad haciéndolas variar para cada capa, o grupo arbitrario de neuronas.
- **Métodos de segundo orden:** Además de calcular el gradiente del error, se puede calcular o aproximar el Hessiano 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 decrece 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 es mucho más estable.
- **Obviar la magnitud del gradiente:** La dirección del gradiente nos 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.

5.2.5. Algoritmo de entrenamiento Resilient Backpropagation

La idea principal del algoritmo Resilient Backpropagation, o Rprop [72]⁹, como ya fue mencionado, es no utilizar la magnitud de la derivada del error para actualizar cada peso w .

En esta sección utilizaremos una notación más simple para numerar los pesos, escribiendo simplemente w para referirnos a un peso cualquiera, asumiendo un conjunto de pesos W de la red y alguna correspondencia entre esos pesos a las posiciones exactas en la red.

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.

⁸Nos referimos a un *paso* o *movimiento* en el espacio de parámetros a un cambio de los pesos w 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.

⁹En esta sección, estamos desarrollando la variable iRprop- de Rprop, una mejora del mismo introducida en [73, 74].

■ 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 se computa dinámicamente una velocidad Δ_w^t , y se mueve w en la dirección opuesta a la que indica la derivada, en un paso con magnitud Δ_w^t , es decir:

$$\Delta w^t \leftarrow -\text{sign}\left(\frac{\partial E(t)}{\partial w^t}\right) \Delta_w^t$$

$$w^{t+1} \leftarrow w^t + \Delta w^t$$

■ Evitar fluctuaciones

La segunda idea importante en el algoritmo es tratar de no hacer cambios cuando la derivada fluctúa para evitar pasos innecesarios.

Entonces, 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. Aún más, 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 asumiendo que el último cambio de w es producto de una fluctuación de la derivada. En ese caso, se deshace ese cambio con la regla:

$$w^{t+1} \leftarrow w^t - \Delta w^{t-1}$$

Además, siempre que $\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}} < 0$ 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 . Las variables reset_w se inician en *true* para todo w .

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. Por otro lado, 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 $\text{reset}_w = \text{true}$.

■ 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 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

$$s = \frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}}:$$

$$\Delta_w^{t+1} \leftarrow \begin{cases} \max(\Delta_w^t \eta^+, \Delta_{max}) & \text{si } s > 0 \\ \min(\Delta_w^t \eta^-, \Delta_{min}) & \text{si } s < 0 \\ \Delta_w^t & \text{si } s = 0 \end{cases}$$

Juntando estas tres ideas podemos la regla de actualización completa para cada peso w del algoritmo:

```

for  $w \in W$  do
   $s \leftarrow \text{sign}\left(\frac{\partial E(t)}{\partial w^t} \cdot \frac{\partial E(t-1)}{\partial w^{t-1}}\right);$ 
   $\Delta_w^{t+1} \leftarrow \begin{cases} \max(\Delta_w^t \eta^+, \Delta_{\max}) & \text{si } s > 0 \\ \min(\Delta_w^t \eta^-, \Delta_{\min}) & \text{si } s < 0 ; \\ \Delta_w^t & \text{si } s = 0 \end{cases}$ 
  if  $\frac{\partial E(t-1)}{\partial w^{t-1}} = 0$  OR  $\text{reset}_w$  then
     $\Delta w^t \leftarrow -\text{sign}\left(\frac{\partial E(t)}{\partial w^t}\right) \Delta_w^t ;$ 
     $w^{t+1} \leftarrow w^t + \Delta w^t ;$ 
     $\text{reset}_w \leftarrow \text{false}$ 
  else if  $s > 0$  then
     $\Delta w^t \leftarrow -\text{sign}\left(\frac{\partial E(t)}{\partial w^t}\right) \Delta_w^t ;$ 
     $w^{t+1} \leftarrow w^t + \Delta w^t ;$ 
  else if  $s < 0$  then
    if  $E(t) > E(t-1)$  then
       $w^{t+1} \leftarrow w^t - \Delta w^t ;$ 
    end
     $\text{reset}_w \leftarrow \text{true}$ 
  else
    { Caso  $\frac{\partial E(t)}{\partial w^t} = 0$  }
     $\Delta w^t \leftarrow 0 ;$ 
  end
end

```

Algoritmo 11: Regla de actualización de los pesos del algoritmo Rprop

5.3. Redes Competitivas

5.3.1. Vector Quantization

Gestos

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

Edward Sapir

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 modelo y proceso de reconocimiento con características particulares. A continuación, describimos 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, lingüística, 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. Si bien no existe una taxonomía definitiva para clasificar distintos tipos de gestos, existen ciertos criterios tanto matemáticos como lingüísticos para distinguirlos, y algunas clasificaciones establecidas.

6.1. Clasificaciones

6.1.1. Clasificación de McNeill

Desde el punto de vista del uso comunicacional de los gestos, encontramos prominentemente 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 podemos 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 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 un "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 rompiera o abriera 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 pictorial. 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 intencionalmente. 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 movemos nuestro brazo para atrás para señalar el pasado o indicamos 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 muy intrínsecamente relacionados al discurso que acompañan y a relaciones sutiles con el contexto que son muy difíciles de reconocer y traducir en una interfaz.

6.1.2. Clasificación de Cadoz (SI HAY TIEMPO)

6.1.3. Clasificación de Kendon (SI HAY TIEMPO)

6.1.4. Clasificación de Caridakis (SI HAY TIEMPO)

6.1.5. Lenguajes de señas (SI HAY TIEMPO)

6.1.6. Clasificación para su reconocimiento

Desde la perspectiva de las ciencias de la computación, consideramos 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 distintas partes del cuerpo tarden distinto tiempo en realizar su parte del gesto o ciertas partes deban realizarse 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.

Podemos 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,

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 una vidriera interactiva [?].

En esta tesina, nos enfocamos en gestos corporales dinámicos y unimodales, dependientes del usuario, y realizados con una sola parte del cuerpo. Además, consideramos 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).

6.2. 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 gestos a reconocer, todavía podemos distinguir diversas propiedades de los gestos dinámicos.

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

En la etapa de reconocimiento, asignaremos una clase a un nuevo ejemplar de un gesto en a partir de ejemplares previamente obtenidos, que llamaremos **ejemplares de entrenamiento**. Para ello, utilizaremos algoritmos de clasificación basados en aprendizaje automático que de alguna manera comparen 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).

Entonces, procedemos a modelar un ejemplar de un gesto dinámico 3D unimodal c como una trayectoria. En este caso tenemos dos parametrizaciones posibles: una **temporal** y una por **longitud de arco**. Procederemos primero a definir el modelo de ejemplar de gesto a reconocer como una trayectoria con parametrización temporal, con ciertas propiedades e invarianzas. Luego haremos lo mismo con la parametrización por longitud de arco, y finalmente compararemos ambas. En estos modelos de gestos dichas trayectorias serán representadas por una función continua, que en secciones posteriores adaptaremos al caso discreto debido a la naturaleza discreta del proceso de captura de los sensores.

6.2.1. Modelo de gestos con parametrización temporal

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

$$\begin{aligned} c: \mathbb{R} &\rightarrow \mathbb{R}^3 \\ c(t) &= (x, y, z) \quad t = 0 \dots T \end{aligned} \tag{6.1}$$

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. Podemos asumir c continua dado que dicha trayectoria surge del movimiento de la mano, aunque en la práctica eso 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 obtendremos una definición de clase de gestos, definido como una clase de equivalencia en base a una relación de equivalencia entre ejemplares de gestos. Vamos a definir 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.

$$c \equiv_n c' \iff \begin{aligned} c(t) &= c'(t) \\ t &= 0 \dots T, T = T' \end{aligned} \tag{6.2}$$

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

$$\mathbb{V}_{T,T'} = \left\{ v(t) \mid \begin{array}{l} v(0) = 0, \quad v(T) = T' \\ v \in \mathbb{C}, \quad \frac{\partial v}{\partial t} \geq 0 \end{array} \right\} \tag{6.3}$$

Entonces, definimos c y c' como equivalentes con invarianza a la velocidad:

$$c \equiv_v c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'} \\ &c(t) = c'(v(t)), t = 0 \dots T \end{aligned} \quad (6.4)$$

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

La invarianza a la velocidad es en cierto modo ortogonal al resto de las invarianzas. Podríamos definirlas exigiendo $T = T'$ como condición en todas ellas, o exigiendo la existencia de v , o sea, $c \equiv_v c'$. Dado que en nuestra aplicación queremos reconocer gestos sin importar la velocidad con la que fueron realizados, utilizaremos la segunda opción.

Con ella podemos definir una equivalencia con **invarianza a la traslación**:

$$c \equiv_t c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'}, \exists b \in \mathbb{R}^3: \\ &c(t) + b = c'(v(t)) \end{aligned} \quad (6.5)$$

También podemos definir **invarianza a la escala** del gesto:

$$c \equiv_s c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'}, \exists a \in \mathbb{R} \\ &a c(t) = c'(v(t)) \end{aligned} \quad (6.6)$$

Definimos 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 deseamos 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' ,

$$c \equiv_c c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'}, \exists \omega \in \mathbb{R} \\ &c((t + \omega) \% T) = c'(v(t)) \end{aligned} \quad (6.7)$$

Donde $\%$ es el operador módulo. Además, podemos definir una equivalencia con **invarianza a la dirección** hacia donde se realiza la trayectoria.

$$c \equiv_c c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'} \\ &c(t) = c'(v(t')) \\ &t' = T - t \end{aligned} \quad (6.8)$$

Por ejemplo, un gesto circular puede hacerse en dirección horaria o anti-horaria. Hay que tener cuidado ya que con esta invarianza ocurren ciertas equivalencias potencialmente indeseables, ya que, por ejemplo, un gesto de “swipe” de izquierda a derecha es equivalente a uno de derecha a izquierda.

Por ese motivo dejaremos esta última equivalencia definida, pero no la incluiremos en nuestro modelo final de gesto.

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

$$c \equiv_{\mathbf{R}_i} c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'}, \exists \mathbf{R}_i \in \mathbb{R}^{3 \times 3} \\ &c(t) = \mathbf{R}_i c'(v(t)) \end{aligned} \quad (6.9)$$

Donde \mathbf{R}_i es una matriz de rotación alrededor del eje \mathbf{i} .

Con estas relaciones, tenemos distintas clases de equivalencia entre ejemplares de gestos. En nuestro caso, buscamos una combinación de las equivalencias con invarianza 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.

Ignoramos la invarianza a la posición de comienzo para gestos cerrados ya que trabajamos en un modelo dependiente del usuario en donde el mismo grabará sus gestos, y dicha invarianza es difícil de incluir en un clasificador. Como mencionamos anteriormente, no incluimos la invarianza 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, obtenemos el siguiente modelo:

$$c \equiv_{m'} c' \iff \begin{aligned} &\exists v \in \mathbb{V}_{T,T'}, \exists \mathbf{R}_i \in \mathbb{R}^{3 \times 3} \\ &\exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\ &c(t) = a(\mathbf{R}_i c'(v(t))) + b \end{aligned} \quad (6.10)$$

Por último, es obvio que un gesto no puede replicarse exactamente, aún con estas equivalencias. Por ende, relajamos la condición de igualdad entre posiciones a una de ϵ – equivalencia

$$\begin{aligned} &\|a - b\| < \epsilon \\ &a =_\epsilon b \iff \text{donde } \epsilon \in \mathbb{R}, a, b \in \mathbb{R}^3 \\ &\text{y } \|\cdot\| \text{ es una norma en } \mathbb{R}^3 \end{aligned} \quad (6.11)$$

De esta manera, finalmente llegamos a la equivalencia deseada:

$$\begin{aligned}
& \exists v \in \mathbb{V}_{T,T'}, \exists \mathbf{R}_i \in \mathbb{R}^{3 \times 3} \\
c \equiv_m c' & \iff \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\
& c(t) = {}_e a(\mathbf{R}_i c'(v(t))) + b
\end{aligned} \tag{6.12}$$

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 invarianzas, que pueden lograrse con transformaciones más tradicionales de la entrada.

Esta última adición colapsa todas las clases

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 uno de otro.

6.2.2. 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:

$$\begin{aligned}
c: \mathbb{R} & \rightarrow \mathbb{R}^3 \\
c(l') & = (x, y, z) \quad l' = 0 \dots L
\end{aligned} \tag{6.13}$$

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

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

$$\begin{aligned}
c: \mathbb{R} & \rightarrow \mathbb{R}^3 \\
c(l) & = (x, y, z) \quad l = 0 \dots 1
\end{aligned} \tag{6.14}$$

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 queremos agregar. Al igual que en el caso temporal, podemos definir una equivalencia con invarianza a la escala y traslación (y a la velocidad, trivialmente, ya que no forma parte de esta definición de gesto).

$$\begin{aligned}
c \equiv_{m'} c' & \iff \exists \mathbf{R}_i \in \mathbb{R}^{3 \times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\
& c(l) = a(\mathbf{R}_i c'(l)) + b
\end{aligned} \tag{6.15}$$

y nuevamente podemos agregar cierta tolerancia al error:

$$\begin{aligned}
c \equiv_m c' & \iff \exists \mathbf{R}_i \in \mathbb{R}^{3 \times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\
& c(l) = {}_e a(\mathbf{R}_i c'(l)) + b
\end{aligned} \tag{6.16}$$

Este es, entonces, el modelo de gestos a reconocer. Preferiremos esta 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, vemos 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 $c'(l)$.

Parte II

Aplicaciones

Bases de datos y características para el reconocimiento de gestos

Para probar los algoritmos, se generó una base de datos con gestos de números y letras utilizando un dispositivo Kinect y su SDK. A continuación describimos el proceso de captura, el funcionamiento del Kinect, y las etapas preprocesamiento y generación de características para cada ejemplar de gesto. La captura se realizó con el SDK del Kinect, que provee las posiciones de las partes del cuerpo en tiempo real. El preprocesamiento constó en rotar los gestos a una dirección canónica, suavizarlos para quitar el posible *jitter* en la captura, y volver a muestrear cada gesto para que todos tengan una cantidad de posiciones igual y constante, interpolando con los valores originales para llegar a los nuevos. Como características se utilizaron las direcciones entre posiciones consecutivas del gesto, normalizadas para que sean invariantes a la escala.

También presentamos un experimento sobre una base de datos ajena, que llamaremos VizApp [], para la cual obtuvimos resultados comparables al algoritmo de Dynamic Time Warping presentado en el artículo original que dio lugar a la base de datos, si tenemos en cuenta ciertas diferencias importantes entre los experimentos.

7.1. Base de datos de letras y números arábigos

Para probar los algoritmos, creamos la Letters and Numbers Hand Gesture Database (LNHGDB) ¹, una pequeña DB con 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 5 minutos entre la grabación de cada 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.

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 comfortable, para lo cual solo era necesario presionar la tecla *Enter*.

Cada ejemplar $s_i \in S$, donde S es nuestra base de datos de gestos, consiste en una secuencia:

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

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

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

y etiquetas de clase c_i . Cada muestra 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 s_i es una versión discreta del modelo de gestos continuos descrito en un capítulo anterior.

7.2. El Kinect y su SDK

7.2.1. 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 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.

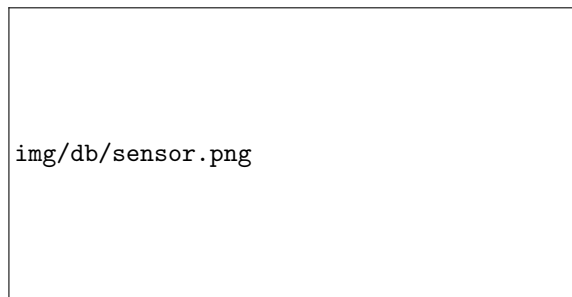


Figura 7.1: Kinect

¹ En <https://sites.google.com/site/dbanhg/> se encuentra una versión anterior de la base de datos que sólo contiene gestos de números arábigos

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 la información esencial para ubicar objetos 3D en un espacio 2D, su posición en el eje z, invaluable para el proceso de segmentación. Al utilizar un láser infrarrojo las condiciones de luz ambiental no afectan tanto este proceso.

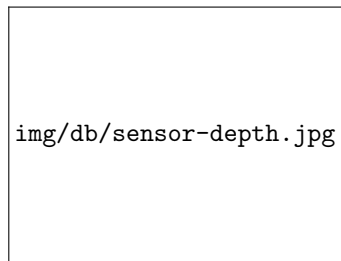


Figura 7.2: Imagen capturada con el sensor de profundidad del Kinect

Acompaña al dispositivo un software de detección de personas y las partes de sus cuerpos en tiempo real. Este 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.

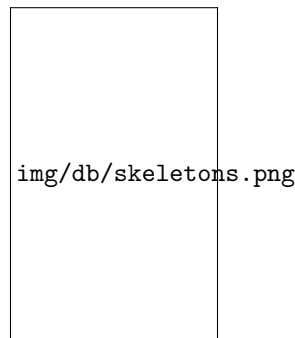


Figura 7.3: Esqueletos detectados en modo normal y modo sentado

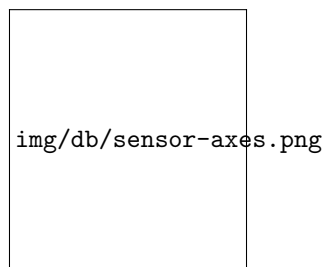


Figura 7.4: Ejes del kinect. El centro de coordenadas corresponde a la posición de la cámara

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.

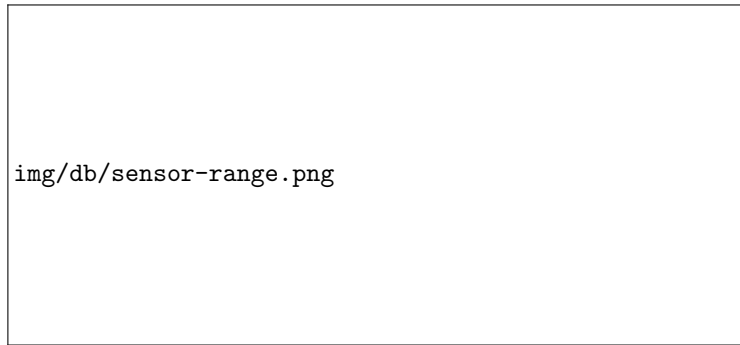


Figura 7.5: Rango del sensor

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.

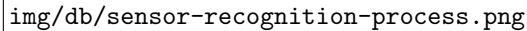
7.2.2. SDK y Algoritmo de tracking del cuerpo

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 la cual las partes del cuerpo se etiquetaron a mano. En base a eso, entrenaron 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 árbol profundo de decisión aleatorizado que evita el sobreentrenamiento y obtiene invarianza al gracias a esta gigantesca cantidad de datos. Esta manera, la técnica obtiene invarianza a la traslación 3D manteniendo la eficiencia computacional. Finalmente, utilizando el algoritmo mean-shift, se infieren los modos espaciales de las distribuciones por pixel de las cuales extraen distintas propuestas de las posiciones 3D de las articulaciones.

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



img/db/sensor-recognition-process.png

Figura 7.6: Imagen de profundidad → Partes del cuerpo → Modelos 3D propuestos

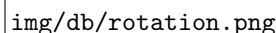
El SDK se puede utilizar con un mecanismo de polling o mediante callbacks que se realizan a la tasa máxima de 30fps mencionada anteriormente. En ambos casos, en cada poll o callback la información que se provee es la cantidad de esqueletos (personas) detectados, con un identificador que se mantiene en 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.

7.3. 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 descartaron los frames en donde el SDK no ha detectado todas las articulaciones relevantes. Luego los ejemplares fueron rotados, suavizados, y resampleados.

7.3.1. Rotación

La rotación del usuario respecto a la cámara introduce diferencias en las posiciones de los ejemplares de gestos irrelevantes para el modelo de gestos pero que dificultan significativamente su reconocimiento. Para evitar dichas dificultades, rotamos 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.



img/db/rotation.png

Figura 7.7: 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 \mathbf{h} para aproximar el grado de rotación del usuario. Luego, se calcula la matriz de rotación \mathbf{R} para llevar a dicho vector \mathbf{h} al vector $(1,0,0)$,

que representaría la rotación canónica. Finalmente, se aplica la matriz de rotación R a todos los puntos del ejemplar, obteniendo una posición canónica para los mismos.

7.3.2. Suavizado

Las muestras fueron suavizadas individualmente utilizando la técnica de la média móvil 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.

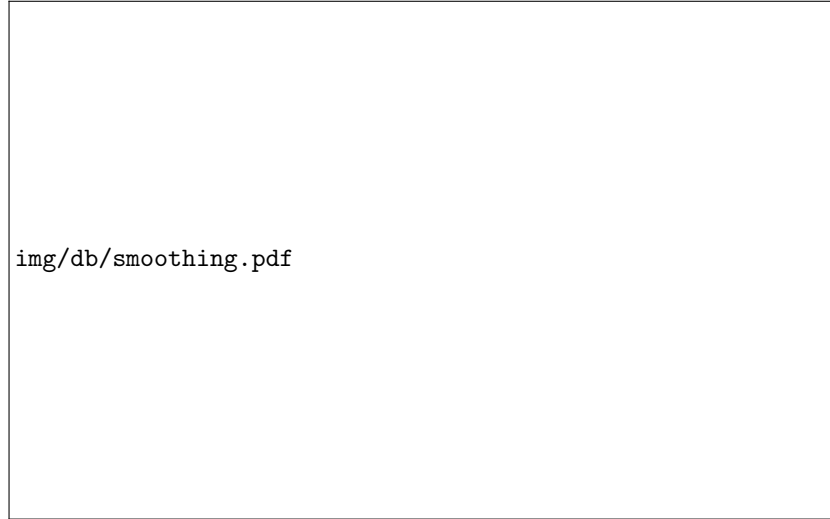


Figura 7.8: Proyección en el plano xy de un ejemplar del gesto 1, antes y después de suavizar con $w = 5$

7.3.3. Re-muestreo

La naturaleza de las arquitecturas de las redes neuronales feedforward, SVM y el método de templates requiere que el número de posiciones de cada ejemplar n_i sea el mismo para todas las muestras, es decir $n = n_i$, $i = 1, \dots, |S|$. 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 primer 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 inavariantes a la velocidad.

Para lograr ambas cosas, cada muestra se vuelve a muestrear a una secuencia de longitud constante n utilizando interpolación cúbica con una parametrización de longitud de arco.

La parametrización de longitud de arco de cada muestra s de longitud original q nos da la posición de la mano p en el espacio 3D en función de la longitud de arco recorrida desde la primer posición del gesto, p_0 hasta la posición p . Entonces, por cada posición $s[j]$ calculamos la longitud de arco desde la primer posición $l_j = \sum_{k=2}^j \|s[k] - s[k-1]\|$, donde $\|\cdot\|$ es la norma Euclídea. El re-muestreo se realiza en n puntos de control distribuidos de manera uniforme a través de la longitud de arco total $L = l_q$, dada por $k_j = \frac{j-1}{n-1} * L$, $j = 1 \dots n$. Obtenemos entonces un vector de puntos r de longitud n tal que $r[j] = \text{cubic}(k_j, \text{near}_4(k_j))$ $j = 1 \dots n$ donde $\text{cubic}(x, (x_1, x_2, x_3, x_4))$ realiza una interpolación cúbica en el punto de control dado por la distancia x , utilizando las distancias (x_1, x_2, x_3, x_4) cuyas posiciones se conocen y $\text{near}_4(x)$ retorna las 4 posiciones más cercanas a x (es decir, $\text{min}_4 = (|l_1 - x|, \dots, |l_q - x|)$) tal que $x_1 \leq x_2 \leq x \leq x_3 \leq x_4$.

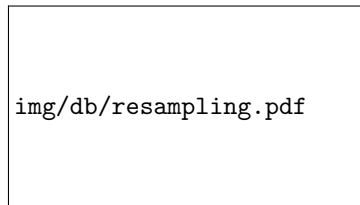


Figura 7.9: Proyección en el plano xy de un ejemplar del gesto 1, antes y después de resamplear con $n = 10,30$. La mayor densidad de posiciones muestreadas, representadas con puntos, en la parte superior en relación con la parte inferior del original fue causada por una diferencia en la velocidad utilizada para realizar el gesto. Dicha diferencia fue compensada, con distintos grados de detalle, al resamplear.

Este proceso da como resultado, para cada ejemplar x_i , un vector 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.

7.4. Características

A partir de la secuencia de puntos rotada, suavizada y resampleada r_i calculamos el vector de primeras diferencias normalizado d_i , donde $d_i[j] = \frac{r_i[j+1] - r_i[j]}{\|r_i[j+1] - r_i[j]\|}$, $j = 1 \dots n-1$, $d_i[j] \in \mathbb{R}^3$. Este vector será nuestra característica para el gesto, y cada uno de sus elementos representa la dirección relativa entre las posiciones consecutivas del mismo. El vector de primeras diferencias, sin normalizar, nos da una representación equivalente invariante a la traslación. Al normalizar, removemos 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.

Como una alternativa a los vectores dirección, empleamos también los ángulos de la representación esférica de los vectores dirección, obteniendo una representación a_i , donde $a_i[j] = (\theta, \phi)_j \in \mathbb{R}^2$. La coordenada z de la representación esférica se obvia debido a que es siempre 1 a consecuencia de la normalización previa, y reescalamos los ángulos de manera que $\theta, \phi \in [-1, 1]$.

Dada la naturaleza periódica de la representación con ángulos, en todas las diferencias entre ángulos calculadas para los algoritmos de clasificación utilizamos la función $d(\alpha, \beta) = \min(|\alpha - \beta|, 2 - |\alpha - \beta|)$.

Aunque ambas características son isomorfas y tienen las mismas propiedades deseables (invarianza a la escala, traslación y velocidad [77]), producen resultados ligeramente distintos en nuestros experimentos. En las siguientes secciones nos referiremos a las características calculadas a partir de un ejemplar de gesto como s , donde $s[j] \in \mathbb{R}^d$ puede referirse a cualquiera de las dos características (con $d=2$ para ángulos y $d=3$ para direcciones cartesianas).

7.4.1. Versiones discretas de las propiedades de los gestos

Previamente definimos 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, necesitamos una versión discreta de dichas propiedades, aplicables a la representación que utilizamos en los algoritmos de clasificación. La principal diferencia entre ambas es la función de correspondencia entre gestos, ya que de todas maneras buscamos una matriz de rotación R_i , y constantes de traslación y escala a y c . Recordando la definición para el caso continuo:

$$c \equiv_m c' \iff \exists R_i \in \mathbb{R}^{3 \times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\ c(l) = {}_e a(R_i c'(l)) + b \quad (7.1)$$

Queremos aproximar dicha definición en el caso concreto. Para ello, el resampleo de un gesto nos da una representación de sus posiciones 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, definimos la equivalencia en el caso discreto simplemente como:

$$r \equiv_m r' \iff \exists R_i \in \mathbb{R}^{3 \times 3}, \exists b \in \mathbb{R}^3, \exists a \in \mathbb{R} \\ r[j] = {}_e a(R_i r'[j]) + b, j = 1, \dots, n \quad (7.2)$$

Como el espacio de búsqueda de dicha transformación es muy grande, buscamos una equivalencia entre características dadas por $\phi: \mathcal{P} \mapsto \mathcal{F}$, donde \mathcal{F} es el espacio de características, de manera que si $\phi(r) = {}_e \phi(r') \rightarrow r \equiv_m r'$. Es decir, equivalencia en el espacio de las características implica equivalencia \equiv_m .

En nuestro caso, llamaremos $\phi_d(x) = r$ a la función característica que dado un ejemplar de gesto realizar la normalización, incluyendo rotación, suavizado y re sampleado, y el cálculo de la primera diferencia con norma 1.

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 resampleado 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 $r[j]$ represente siempre la misma parte del gesto, aunque sea de forma aproximada.

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

Entonces, nuestra característica r es invariante, como deseábamos a la rotación, traslación, velocidad y escala.

Por último, definimos la propiedad de invarianza 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 $x[1] = {}_e x[n]$. Entonces, una característica ϕ es invariante a la posición de comienzo si:

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

donde

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

y% es el operador modulo

La característica ϕ_d que desarrollamos 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 describimos en el capítulo 8 *si* provee invarianza 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, hemos definido una característica ϕ_d para una secuencia de puntos 3D \mathbf{x} , y establecido que es invariante a la rotación, traslación, velocidad y escala, propiedades de utilidad para el reconocimiento de gestos.

7.5. Base de datos de gestos Visapp2013

Se realizaron experimentos con la base de datos **Celebi2013** [78], 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 de 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
- Swipe hacia la derecha con la mano izquierda
- Swipe hacia la izquierda con la mano derecha
- Saludo con la mano izquierda
- Saludo con la mano derecha



Figura 7.10: 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 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.

²Visitar <http://datascience.sehir.edu.tr/visapp2013/> para más información.

Experimentos y resultados

"Torture
data long enough, and they'll confess to anything"

A continuación, describimos los clasificadores de gestos propuestos a comparar, generados con la unión de los métodos de clasificación y las características para gestos desarrollados. Mostramos los resultados de los mismos aplicados a las base de datos de Letras y Números y a la base de datos Celebi2013, y finalizamos con una discusión sobre la aplicabilidad y efectividad de los métodos propuestos.

8.1. Modelos de reconocimiento

Combinando la característica descrita en el capítulo 7 para el modelo del capítulo 6, con los clasificadores de los capítulos 4 y 5, y otros desarrollos, es posible construir clasificadores efectivos para el reconocimiento de gestos.

Podemos distinguir dos tipos de características: las globales, que representan alguna característica global del ejemplar, y las locales, que describen el ejemplar en términos de cada parte del mismo, típicamente respetando su topología. En nuestro caso, la característica derivada es del tipo local, ya que cada dirección nos da información local del gesto. De todos modos, pueden generarse características globales en base a ella con la ventaja de que la misma posee las invarianzas ya mencionadas.

A continuación, describimos la aplicación de los modelos SVM y Redes Neuronales Feedforward al problema de clasificación con la característica local desarrollada. Luego, se describe el Clasificador Neuronal Competitivo (CNC), que a partir del vector de direcciones genera un descriptor de tipo global del gesto y lo usa para clasificarlo. Finalmente, describimos un método simple basado en templates que también utiliza la característica desarrollada sin cambios.

8.1.1. SVM

Utilizando con la característica directamente, podemos pensar en un gesto como un punto en un espacio n -dimensional de direcciones. Nuestro clasificador toma toda la secuencia de direcciones y genera un modelo que distinga gestos.

En el caso de SVM, asumimos 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 entrenamos C clasificadores para que distingan los ejemplares de cada clase de los ejemplares de las otras, encontraremos superficies que separen dichas clases; dichas superficies dependen del kernel elegido. Utilizamos dos kernels en nuestras pruebas: el lineal y el gaussiano.

En el caso del lineal, estamos simplemente buscando un hiperplano separador. En el caso del kernel gaussiano, podemos notar que la fórmula resultante de f es:

$$\begin{aligned} f(\mathbf{x}) &= \sum_i \alpha_i y_i \kappa_{\mathbf{x}_i, \mathbf{x}} + b \\ &= \sum_i \alpha_i y_i e^{-\|\mathbf{x}_i - \mathbf{x}\|/\sigma} + b \end{aligned}$$

donde σ es un parámetro de desviación estándar y $\|\cdot\|$ es la norma euclídea. Entonces, podemos interpretar esta fórmula de la siguiente manera.

Cada función gaussiana es una función de similitud del ejemplar \mathbf{x} al ejemplar de entrenamiento \mathbf{x}_i , escalado por σ . En el mejor caso $\mathbf{x}_i = \mathbf{x}$, por ende $e^{-\|\mathbf{x}_i - \mathbf{x}\|/\sigma} = 1$ y entonces, el término i -ésimo de la sumatoria es $\alpha_i y_i$. A medida que \mathbf{x} es cada vez más diferente de \mathbf{x}_i , $e^{-\|\mathbf{x}_i - \mathbf{x}\|/\sigma}$ tiende a 0, y entonces dicho término prácticamente se anula en la sumatoria.

Como $\alpha_i \geq 0$ y $y_i = \pm 1$, podemos interpretar esto como que se seleccionan los vectores del conjunto de entrenamiento más cercanos a \mathbf{x} , pesados por la distancia entre \mathbf{x} y \mathbf{x}_i , y además por la variable dual 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 \mathbf{x} sea considerado de clase 1, con una fuerza de votación $\alpha_i e^{-\|\mathbf{x}_i - \mathbf{x}\|/\sigma}$, y viceversa si $y_i = -1$.

Por último, b ajusta la línea de corte, por las mismas razones mencionadas en el capítulo 3 para agregar un bias en el perceptrón. En este caso, \mathbf{x} y \mathbf{x}_i son gestos; el kernel gaussiano entonces implementaría nuestra noción del capítulo 6 de una equivalencia aproximada $=_\epsilon$ entre gestos.

8.1.2. Redes Neuronales Feedforward (FF)

Para las redes neuronales feedforward, así como para SVM, utilizaremos 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** [79], 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 tomamos entonces; la red toma un vector de n direcciones, y por ende utiliza n neuronas de entrada. utilizamos una sola capa de entrada con h neuronas ocultas, determinadas experimentalmente. La capa de salida tiene tantas neuronas como clases, C ; cada salida o_c representa la confianza de que el ejemplar x pertenezca a la clase c .

8.1.3. CNC: El clasificador neuronal competitivo

8.1.4. Clasificador basado en templates

8.1.4.1. Arquitectura y entrenamiento

8.2. Experimentos con base de datos de LNHGDB

Comparamos el error de los métodos en la base de datos de gestos, utilizando los mismos parámetros de preprocesamiento para todos. El tamaño de resamplio fue $n = 65$ y el de la ventana de suavizado $w = 5$.

Por cada algoritmo se realizaron 500 experimentos utilizando validación cruzada estratificada con resamplio aleatorio, con 3 muestras por clase para el conjunto de entrenamiento ($3 \times 26 = 78$ 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 por la naturaleza de los algoritmos de entrenamiento (Backpropagation y iRprop+) sería muy difícil de entrenar con solo 3 muestras.

Para los clasificadores FF+BP, FF+iRP y CNC, mostramos resultados para redes con $h = 30$, $h = 30$ y $h = 70$ neuronas ocultas, respectivamente, que dieron los mejores resultados en nuestros experimentos (probamos $h \in 5, 10, \dots, 150$ para determinar esos valores).

El CNC tuvo $p = 5$ subclasificadores. 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 $class = \max_c(o_c)$.

Clasificador	Parámetros	Algoritmo de entrenamiento	Error (μ)
SVM	Kernel Lineal	SMO	60%
SVM	Kernel Gaussiano, $\sigma = 4$	SMO	60%
CNC	$h = 70, p = 5$	CNC, $\alpha = 0,5$	60%
FF	$h = 80$	iRprop+	60%
FF	$h = 80$	Backpropagation	60%
Template	$\sigma = 4$	-	60%

Cuadro 8.1: Resultados de los experimentos de clasificación para la base de datos Celebi2013.

Presentamos también el error del CNC sin utilizar el proceso de resamplio, debido a que no necesita un vector de entrada de longitud fijo, aunque en este caso los gestos no serían verdaderamente invariantes a la velocidad.

TABLA

Podemos ver que los errores de reconocimiento son ligeramente peores pero no significativamente, lo cual muestra, en principio, que el método podría utilizarse sin el resamplado en los casos donde un peor desempeño sea preferible a un coste computacional elevado.

8.3. 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. Se realizaron todos los pasos de preprocesamiento y extracción de características descritas anteriormente salvo el proceso de normalización de la rotación del usuario, ya que no había información de las posiciones de las otras partes del cuerpo para tal fin.

Se empleó la misma metodología de validación cruzada con sampleo aleatorio y 500 iteraciones. Los resultados se presentan en la siguiente tabla:

Clasificador	Parámetros	Algoritmo de entrenamiento	Error (μ)
SVM	Kernel Lineal	SMO	60%
SVM	Kernel Gaussiano, $\sigma = 4$	SMO	60%
CNC	$h = 70, p = 5$	CNC, $\alpha = 0,5$	60%
FF	$h = 80$	iRprop+	60%
FF	$h = 80$	Backpropagation	60%
Template	$\sigma = 4$	-	60%

Cuadro 8.2: Resultados de los experimentos de clasificación para la base de datos Celebi2013.

8.4. Conclusiones

A novel approach for gesture recognition with small training samples has been presented that is time, scale and translation invariant, and for closed gestures, starting-point invariant as well, while achieving high recognition rates comparable to other approaches and with a learning algorithm that requires few iterations to converge.

In our experiments, the CNC, ST-OMM and the ID-FF all perform reasonably well, especially first two. In addition, the CNC has the advantage of being quick to train and starting-point invariant without any modification, although at a greater intrinsic computational cost. For the other two methods, the same effect can be achieved by shifting the sample in a circular way in all n possible combinations which also scales up by a factor n their execution order.

In future work, we hope to determine the method's performance on larger gesture databases, and apply it in a real-time setting to test its ability to recognize gestures in a unsegmented stream of hand positions. Finally, we intend to improve and extend our current database with 3D gesture data to provide a reference point for future comparisons and benchmarkings.

Bibliografia

- [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 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. “your 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, Yadollah 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-Aurélié 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] 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.
 - [58] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.
 - [59] Abraham Wald. Statistical decision functions. 1950.
 - [60] 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.
 - [61] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Mach. Learn.*, 52(3):239–281, September 2003.
 - [62] Vladimir N Vapnik. Statistical learning theory. 1998.
 - [63] Yaser S. Abu-Mostafa, Malik Magdon-Ismael, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.
 - [64] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.
 - [65] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964.
 - [66] Richard Bellman and R. Kalaba. On adaptive control processes. *Automatic Control, IRE Transactions on*, 4(2):1–9, 1959.
 - [67] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
 - [68] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university

- press, 2002.
- [69] 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.
 - [70] 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.
 - [71] 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.
 - [72] 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.
 - [73] Christian Igel and Michael Husken. Improving the rprop learning algorithm, 2000.
 - [74] Christian Igel and Michael Husken. Empirical evaluation of the improved rprop learning algorithms, 2003.
 - [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] Volodymyr V. Kindratenko. On using functions to describe the shape. *J. Math. Imaging Vis.*, 18(3), May 2003.
 - [78] 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.
 - [79] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.