Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragón

Análisis Comparativo en Problemas de Clasificación Multiclase mediante Redes Neuronales Superficiales y Máquinas de Soporte Vectorial

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Ingeniero en Computación

PRESENTA:

Luis Alberto Ochoa Esparza

DIRECTOR DE TESIS:

Carlos Oliver Morales

México Ciudad Nezahualcóyotl, Estado de México 2019

Agradecimientos

Este trabajo culmina un largo camino de esfuerzos y satisfacciones.

En primer lugar quiero agradecer a mis padres Ma. de Lourdes y Jesús por su amor, apoyo y verdadera paciencia y a mi hermana Gabriela por su amor y apoyo. Muchas gracias por estar ahí. ¡Son increíbles!

A mi asesor, el M. en C. Carlos Oliver Morales por la confianza depositada en mí.

A los miembros del Jurado: M. en I. Martha Chapa Plata, Ing. Martín Manuel Romero Ugalde, M. en A. María Angélica Feria Victoria, M. en C. Jorge Iván Campos Bravo por todo su apoyo en la revisión y todas sus valiosas recomendaciones que enriquecieron mi proyecto de tesis.

A la Universidad Nacional Autónoma de México y a la Facultad de Estudios Superiores Aragón.

Índice general

Índice de figuras

Lista de tablas

Abreviaturas

ANN red neuronal artificial

AUC área bajo la curva ROC

BFGS Broyden–Fletcher–Goldfarb–Shanno

BP retropropagación

CE entropía cruzada

CG gradiente conjugado

DNN red neuronal profunda

ECOC códigos de salida de corrección de errores

FNN red neuronal prealimentada

IA inteligencia artificial

LM Levenberg-Marquardt

ML aprendizaje automático

MLP perceptrón multicapa

MSE suma del error cuadrático

QP programación cuadrática

ROC característica operativa del receptor

Rprop Resilient propagation

SCG gradiente conjugado escalado

SNN red neuronal superficial

SPM plasticidad sináptica y memoria

SRM minimización del riesgo estructural

SVM máquina de soporte vectorial

XOR or exclusiva

Notación

$\mathbb{N}$ Conjunto de números naturales

$\mathbb{R}^n$ Conjunto de vectores en un espacio de dimensión n

$\mathbb{R}$ Conjunto de números reales

$\in$ Pertenencia. $\alpha \in A$, α pertenece al conjunto A

$\subset$ Subconjunto. $A \subset B$, A es un subconjunto del conjunto B

$\mathbf{x}^T\mathbf{y}$ Producto punto

sgn Función signo

tr Traza. Es la suma de elementos de la diagonal principal de una matriz

lg Logaritmo binario, $\log_2$

ln Logaritmo natural, $\log_e$

$\lceil \cdot \rceil$ Función techo

$\mathcal{D}$ Conjunto de datos

$\mathbf{x}_i$ Patrón. Definido como un vector, $\mathbf{x} \in \mathbb{R}^n$

$\hat{y}$ Salida predicha durante el proceso de aprendizaje de una red neuronal artificial

y Clase (o categoría) asociada al patrón x_i

$\mathcal{X}$ Conjunto de patrones

$\mathcal{Y}$ Conjunto de clases

E Función de error

$\mathbf{K}$ Matriz de Gram

$\mathcal{H}$ Espacio de Hilbert

k Función Kernel

$\mathbf{M}$ Matriz de codificación

H_0 Hipótesis nula, $H_0 : \mu_1 = \mu_2 = \cdots = \mu_n$, indica que no hay diferencia entre las medias

H_1 Hipótesis alternativa, $H_1 : \mu_1 \neq \mu_2 \neq \cdots \neq \mu_n$, indica que hay diferencia entre las medias

p Valor-p

Introducción

Inicialmente, las máquinas fueron desarrolladas para facilitar el trabajo manual del hombre. Hoy en día, se busca que las máquinas también realicen el trabajo intelectual con el objetivo de ayudar en la toma de decisiones. Por esta razón la comunidad científica han trabajado para lograr que las máquinas piensen, entiendan los problemas y tomen decisiones. El aprendizaje automático es una rama de la inteligencia artificial que simula el aprendizaje humano. Por ejemplo, un algoritmo puede "aprender" a determinar si un correo electrónico es spam o no. En la ciencia, los científicos, a través de las micromatrices de ADN, analizan miles de genes simultáneamente y determinar si esos genes son activos, hiperactivos o silenciosos en tejido normal o canceroso; un algoritmo puede determinar si los tejidos cancerosos tienen firmas distintivas de expresión génica sobre los tejidos normales u otros tipos de tejidos cancerosos. Estas tareas pueden ser realizadas por un mismo algoritmo de aprendizaje.

Los ejemplos mencionados anteriormente hacen referencia a un problema de clasificación. Entre las técnicas empleadas para resolver este tipo de problemas se encuentran las redes neuronales artificiales, inspiradas en el sistema nervioso, y las máquinas de soporte vectorial, inspiradas en la teoría de aprendizaje estadístico descrita por Vapnik en *Statistical learning theory*. Ambas técnicas han sido ampliamente estudiadas como clasificadores siendo las máquinas de soporte vectorial que proporcionan un mayor desempeño. En este trabajo de investigación, se analiza el desempeño en problemas de clasificación multiclase y el tiempo requerido de entrenamiento de una máquina (proceso de aprendizaje de una máquina) usando ambas técnicas.

Este trabajo de investigación es el resultado de entender el proceso de simulación del aprendizaje humano y así poder realizar un análisis estadístico que permita seleccionar un modelo de aprendizaje adecuado con base en el nivel de desempeño y tiempo de entrenamiento.

Para realizar el experimento se han seleccionado seis bases de datos de "UCI Machine Learning Repository" proporcionadas por la Universidad de California. Estas son: *Vino*, *Identificación de vidrio*, *Transbordador espacial*, *Levadura*, *Reconocimiento de letras habladas* y *Reconocimiento de letras*. Se ha optado por el uso de MATLAB, ya que ofrece diversas y

completas funcionalidades mediante las toolboxes para la comunidad investigadora completamente documentadas. Las toolboxes usadas son Deep Learning Toolbox™ y Statistics and Machine Learning Toolbox™ para las redes neuronales superficiales y máquinas de soporte vectorial respectivamente. También se ha hecho uso de Parallel Computing Toolbox™ para resolver los problemas de cómputo mediante procesadores multinúcleo y GPU.

Con base en lo anterior, el objetivo general de este trabajo de investigación es crear la mejor arquitectura posible de una red neuronal superficial y el mejor modelo de máquinas de soporte vectorial para cada una de las bases de datos. La finalidad es realizar un análisis estadístico que permita evaluar el desempeño y el tiempo de entrenamiento de ambos clasificadores.

En tanto que los objetivos particulares, derivados del anterior, son los siguientes:

1. Entrenar diversas redes neuronales superficiales con los algoritmo: trainbfg, trainbr, traincgb, traincgf, traincgp, traingdx, trainlm, trainoss, trainrp, trainscg; configurar el número de unidades en la capa oculta usando el esquema propuesto por Elisseeff y Paugam–Moisy como punto de partida. Evaluar los modelos generados empleando la prueba de Kruskal–Wallis y la prueba de comparación múltiple con un nivel de significación con valor de 0.05 para determinar la mejor arquitectura que dé solución a una base de datos específica.

2. Modelar mediante la técnica de códigos de salida de corrección de errores y los esquemas de codificación: onevsone, binarycomplete, denserandom, onevsall, ordinal, sparserandom y ternarycomplete distintas máquinas de soporte vectorial con las funciones kernel: linear, gaussian y polynomial. Determinar el modelo óptimo mediante la búsqueda de hiperparámetros usando la técnica de malla y optimización bayesiana.

3. Evaluar los resultados obtenidos considerando los índices de clasificación correcta en cada experimento empleando el análisis de varianza de una vía de Kruskal–Wallis y la prueba de comparación múltiple con un nivel de significación con valor de 0.05.

4. Evaluar los resultados obtenidos de los modelos óptimos seleccionados empleando el análisis de varianza de una vía de Kruskal–Wallis y la prueba de comparación múltiple con un nivel de significación con valor de 0.05.

Consideremos esta hipótesis: existe una ventaja significativa en el desempeño y el tiempo en el proceso de entrenamiento de las máquinas de soporte vectorial sobre las redes neuronales superficiales. Debido a la ausencia de múltiples soluciones locales mínimas en las máquinas de soporte vectorial.

El presente trabajo se divide en los siguientes tres capítulos y dos anexos: En el primer capítulo se describe las bases teóricas de las de la inteligencia artificial, el aprendizaje automático,

las redes neuronales superficiales, las máquinas de soporte vectorial. Explica los algoritmos de entrenamiento. El segundo capítulo desarrolla la metodología de análisis empleada, como la validación y pruebas de los modelos de aprendizaje desarrollados. También explica la prueba de Kruskal–Wallis para determinar las diferencias estadísticas de ambos algoritmos. El tercer capítulo reúne los resultados obtenidos de la investigación y la interpretación de estos. En el anexo A se presentan las descripciones de las bases de datos empleadas en la investigación y resume las características esenciales de estas. Finalmente, en el anexo B se presenta la tabla parcial de distribución χ^2 empleada para realizar el análisis estadístico utilizando la prueba de Kruskal–Wallis.

Capítulo 1

Marco Teórico

1.1 Inteligencia Artificial

Inteligencia artificial (IA), ese apasionante esfuerzo por hacer pensar a las computadoras. No se trata de ciencia ficción sino de verdadera ciencia, basada en la idea teórica en la que nosotros mismos somos computadoras. Así, las ideas de pensar y computar son exactamente lo mismo (Haugeland, 2001).

A lo largo de los siglos, el hombre ha imaginado con la posibilidad de máquinas que realicen tareas intelectuales (Hofstadter, 2007). Comenzando con las primeras obras mitológicas y literarias, seguidas de tratados filosóficos, formulaciones matemáticas, autómatas y otros tipos de dispositivos, sobre todo la computadora digital (McCorduck, 2004).

La siguiente línea de tiempo, desarrollada por McCorduck (2004), resume los esfuerzos que se han propuesto como formas de pensamiento mecánico:

Antes de Nuestra Era Aristóteles introduce la lógica silogística, el primer sistema formal de razonamiento deductivo

Finales del primer siglo Herón of Alexandría construye autómatas legendarios y otras maravillas mecánicas

1580 "El Golem", se dice que fue creado por el rabino Judah ben Loew en Praga

1642 Blaise Pascal inventa una calculadora mecánica, la Pascalina

1822 Charles Babbage comienza pero nunca termina la Máquina Analítica

1843 Ada, Countless Lovelace, publica su relato de la máquina analítica de Charles Babbage

1854 George Boole publica "Una investigación de las leyes del pensamiento"

1937 Alan Turing propone una máquina de computación universal abstracta

1943 McCulloch y Pitts publican "Un cálculo lógico de las ideas inmanentes en la actividad nerviosa"

1956 Conferencia de Dartmouth, donde John McCarthy propuso el término "inteligencia artificial" y Newell, Shaw y Simon muestran el primer programa de AI en funcionamiento, el teórico de la lógica.

Hofstadter (2007) afirma que la IA existe desde el momento en que las invenciones mecánicas toman a su cargo diversas tareas que únicamente eran realizables por la mente humana.

El desarrollo de la inteligencia artificial ha pasado por periodos de grandes expectativas alternando con periodos de decepción. El primer periodo de expectativas se dio en la conferencia de Dartmouth en 1956, donde los investigadores desarrollaron sistemas con el objetivo de refutar las ideas como "¡Las máquinas jamás podrán hacer…!". Uno de estos sistemas logro probar la mayoría de los teoremas del segundo capítulo de *Principia mathematica* publicado por Bertrand Russell y Alfred North Whitehead. Estos primeros sistemas que solían ser exitosos en problemas pequeños resultaron difíciles de extenderse a problemas de combinatoria y sumado a las grandes limitaciones de hardware referente a la capacidad de la memoria y velocidad de procesamiento dieron lugar al primer invierno en la inteligencia artificial, periodo en el cual se retiran los fondos de investigación y aumenta el escepticismo (Bostrom, 2014, pp. 5–7).

Actualmente, el término *inteligencia artificial* carece de una definición. Por otra parte, algunos investigadores han brindado su propia definición y algunas de estas han sido ampliamente aceptadas:

- Es la ciencia y la ingeniería de la creación de máquinas inteligentes, especialmente programas de computadora inteligentes. Está relacionada con la tarea de usar computadoras para comprender la inteligencia humana, pero la inteligencia artificial no tiene que limitarse a métodos que sean biológicamente observables (McCarthy, 2007).
- El estudio de los cálculos que hacen posible percibir, razonar y actuar (Winston, 1992).
- La inteligencia artificial es el intento de una máquina de explicar el comportamiento del sistema (humano)… (Schank, 1991).

Una rama de la inteligencia artificial es el aprendizaje automático. Así, uno de los objetivos de la inteligencia artificial es enseñar a las computadoras a hacer lo que actualmente los humanos hacemos mejor (Domingos, 2015).

1.2 Aprendizaje Automático

La mayoría de las máquinas son desarrolladas con fines comerciales con la intención de llevar a cabo algún trabajo muy específico con certeza y velocidad considerable. Todo el tiempo las máquinas realizan la misma serie de operaciones una y otra vez sin ninguna variedad (Turing, 1951). Este hecho es un argumento para no esperar que una máquina comprenda la obra Romeo y Julieta, saltar sobre un agujero en la calle o interactuar con otras máquinas a través de un lenguaje común. Sin embargo, también ha sido un argumento para proponer el reto de lograr que las máquinas piensen, entiendan los problemas y tomen decisiones adecuadas como seres humanos (Mohammed, 2017). A la rama de la IA dedicada a éste esfuerzo se le conoce como aprendizaje automático (ML), del inglés *machine learning*.

La investigación sobre ML no solo se centra en que las máquinas adquieran conocimiento e inteligencia, sino que también descubren los principios y secretos del pensamiento y el aprendizaje humano. También se ha tenido un gran impacto en los patrones de almacenamiento de la memoria, los métodos de entrada de información y la arquitectura de computadoras (Shi, 2011).

Cada algoritmo tiene una entrada y una salida. Los datos ingresan a la computadora, un algoritmo hace lo que tiene que hacer con ellos y se produce una salida. El aprendizaje automático da un giro a esto. Los *datos y la salida* ingresan a la computadora, un algoritmo hace lo que tiene que hacer con ellos y se produce una salida, que resulta ser otro algoritmo. En otras palabras, con el aprendizaje automático obtenemos *programas que escriben sus propios programas* (ver figura 1.1). Escribir programas puede requerir inteligencia, creatividad y habilidad para resolver problemas; justo las características que una computadora no tiene. La gente puede escribir muchos programas que las computadoras no pueden aprender, pero las computadoras pueden escribir programas que la gente no puede escribir (Domingos, 2015).

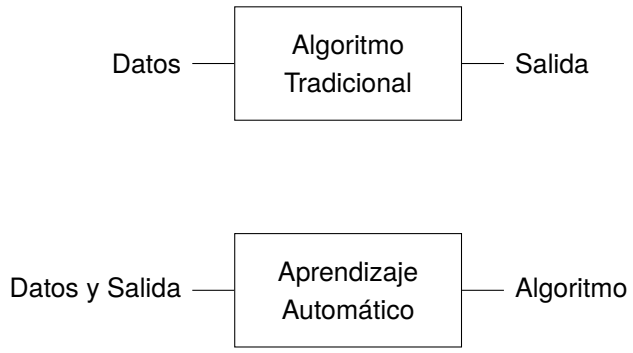


Figura 1.1: Aprendizaje Automático.
Fuente: Adaptado de "Machine Learning wih TikZ" de Ochoa, 2019a.

Quizá no nos damos cuenta, pero vivimos entre algoritmos de aprendizaje. Cuando reali-

zamos una consulta en algún buscador de internet, este determina cuales son los resultados relevantes para cada uno de nosotros. Permiten que un análisis de sangre sea más preciso. Incluso, hacen que los servicios de citas le encuentren un compañero potencial. Netflix tiene miles de títulos, un algoritmo de aprendizaje automático resuelve nuestros gustos y en base a ellos nos recomienda títulos de una gran lista (Y. S. Abu-Mostafa, 2012; Domingos, 2015). En la política, el aprendizaje automático fue el factor que decidió las elecciones presidenciales de 2012. El expresidente Barack Obama contrato a Rayid Ghani, experto en aprendizaje automático, como científico en jefe de su campaña. Ghani procedió a organizar la mayor operación de análisis en la historia de la política (Domingos, 2015). Y así, los demócratas sufrieron su peor derrota en décadas (Issenberg, 2012).

El *aprendizaje automático* es una rama de las ciencias de la computación que registrar minuciosamente conjuntos de datos para hacer predicciones sobre el futuro (Y. S. Abu-Mostafa, 2012).

Mitchell (2006) describe el *aprendizaje automático* como una consecuencia natural de la intersección de las ciencias de la computación y la estadística. Así, el aprendizaje consiste en construir un modelo estadístico de los procesos que generan los datos. Esto permite lograr una alta generalización, que es, realizar buenas predicciones sobre un nuevo conjunto de datos (Bishop, 1995).

1.2.1 Problema del aprendizaje

En un concepto hipotético, el *aprendizaje* se define como *un cambio permanente de conductas como resultado de experiencias pasadas* (Gross, 1987). El desafío que actualmente enfrenta la inteligencia artificial ha renovado el interés de los investigadores por el aprendizaje. De hecho, a medida que las técnicas de representación del conocimiento se vuelven más numerosas y efectivas, el principal obstáculo en la realización de un sistema basado en el conocimiento es precisamente la adquisición de este conocimiento. Está claro que imaginar la realización de un sistema general capaz de aprender es totalmente irreal (LeCun, 1987, p. 9).

En el aprendizaje automático, el problema de aprendizaje considera un conjunto de datos de entrenamiento $\mathcal{D}$, el cual consiste en pares entrada–salida

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_\ell, y_\ell)\}$$

donde $\mathbf{x}$ es un vector (también llamado patrón) conocido como *entrada* y y es una *clase*.

Cada par es generado conforme a una *función desconocida*

$$f : \mathcal{X} \to \mathcal{Y}$$

la cual es llamada *función objetivo*. Por lo tanto $y_i = f(x_i)$. El objetivo consiste en inferir la función objetivo basándose en el conjunto de datos de entrenamiento. El aprendizaje implica elegir una función de hipótesis

$$h : \mathcal{X} \to \mathcal{Y}$$

de un conjunto de funciones candidatas para inferir la función objetivo (Y. Abu-Mostafa, Song, Nicholson & Magdon-Ismail, 2004).

1.2.2 Aprendizaje supervisado

En el aprendizaje supervisado (ver figura 1.2), un supervisor proporciona una clase para cada dato en un conjunto de entrenamiento (Duda, Hart & Stork, 2012).

Durante el entrenamiento, al algoritmo de aprendizaje se le introduce un dato y este produce una salida. El algoritmo modifica sus parámetros internos ajustables para reducir la distancia entre la salida producida y la salida deseada (LeCun, Bengio & Hinton, 2015).

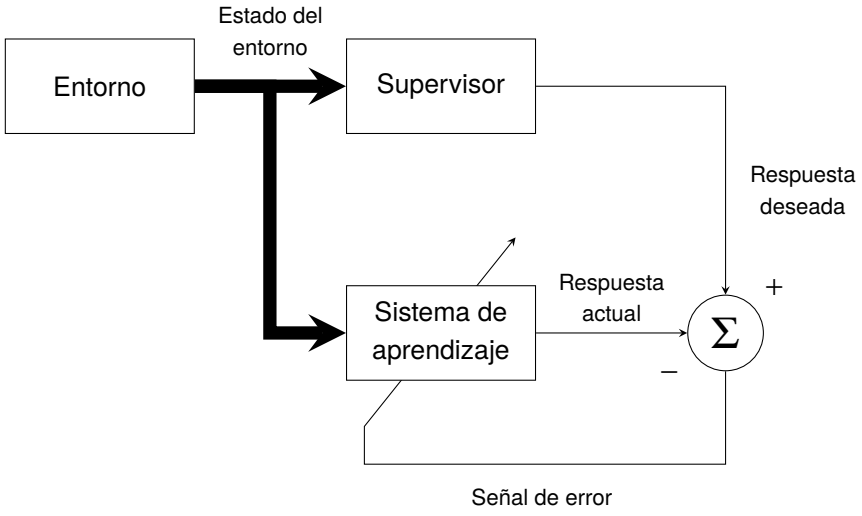


Figura 1.2: Diagrama de bloques del aprendizaje supervisado.
Fuente: Adaptada de *Neural Networks and Learning Machines* de Haykin, 2009.

Como resultado del proceso de entrenamiento, el modelo creado debe ser capaz de asignar un dato desconocido a una de las clases (Bishop, 1995).

1.2.3 Aprendizaje no supervisado

En el aprendizaje no supervisado no hay un supervisor. El objetivo es formar agrupaciones naturales de los datos (o patrones) de entrada (Duda y col., 2012). La agrupación (o clustering) es una de las tareas que se incluyen en esta categoría. El objetivo es encontrar una división natural de los datos en grupos homogéneos (Shawe–Taylor & Cristianini, 2004).

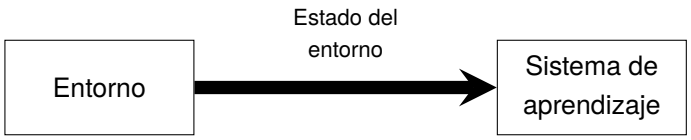


Figura 1.3: Diagrama de bloques del aprendizaje no supervisado.
Fuente: Adaptada de *Neural Networks and Learning Machines* de Haykin, 2009.

1.3 Reconocimiento de Patrones

El *reconocimiento de patrones* se ocupa de la detección automática de patrones en los datos y desempeña un papel central en muchos problemas modernos de la inteligencia artificial y ciencias de la computación. Por patrón entendemos cualquier relación, regularidad o estructura inherente en alguna fuente de datos. Al detectar patrones significativos en los datos disponibles, un sistema puede hacer predicciones sobre nuevos datos que provienen de la misma fuente. En consecuencia, el sistema ha adquirido la habilidad de generalizar al "aprender" algo sobre la fuente de datos (Shawe–Taylor & Cristianini, 2004).

Este descubrimiento automático de regularidades en los datos se realiza mediante el uso de algoritmos informáticos y con el uso de estas regularidades realiza acciones tales como clasificar los datos en diferentes categorías (Bishop, 2006). De modo que, es el proceso por el cual, un patrón recibido se asigna a una de las clases prescritas (Haykin, 2009).

Un *patrón* es un vector que describe a un objeto (Jordan, 1986). Por ejemplo, la información de la flor de Iris setosa con las siguientes características (Fisher, 1936):

1. Longitud del sépalo: 5.1 cm
2. Anchura del sépalo: 3.5 cm
3. Longitud del pétalo: 1.4 cm
4. Anchura del pétalo: 0.2 cm

se puede resumir mediante el vector $\mathbf{x} = [5.1, 3.5, 1.4, 0.2]^T$. Cuando un vector no tiene más de tres componentes, este puede ser representado en un espacio de tres dimensiones (Jordan, 1986).

1.3.1 Características de los patrones

Las variables que conforman un patrón pueden ser *cuantitativas* o *cualitativas* (Hastie, Tibshirani & Friedman, 2009; James, Witten, Hastie & Tibshirani, 2013):

Cuantitativa Toma valores numéricos. Estos pueden ser discretos o continuos.

Cualitativas Toma valores de un conjunto finito sin un orden explícito.

Las variables cualitativas son representadas por valores numéricos. Por ejemplo, las clases "lento" y "veloz" se pueden representan ya sea con $\{0, 1\}$ o $\{-1, 1\}$ (Hastie y col., 2009). En los problemas de regresión, la respuesta es cuantitativa mientras que en los problemas de clasificación es cualitativa (Hastie y col., 2009; James y col., 2013).

1.3.2 Clasificación de los patrones

En el problema de clasificación se considera un conjunto fijo de categorías (o clases) en las que se deben clasificar los patrones. El objetivo es aprender a clasificar correctamente los patrones para que en el futuro, cuando se presente uno particular o una versión distorsionada de uno de ellos, el sistema lo clasifique correctamente (Rumelhart & Zipser, 1986).

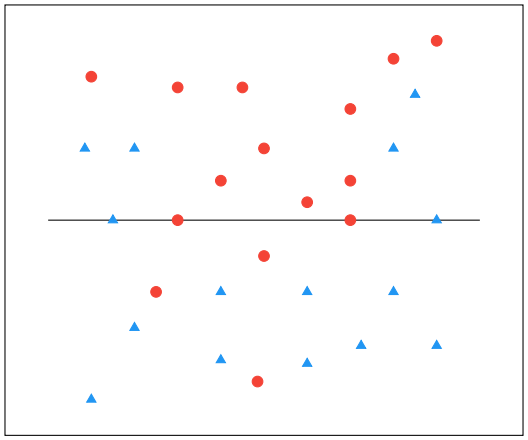
Un *clasificador* es un algoritmo que asigna una categoría (o clase) a un patrón dado (Dreyfus, 2005). Así, dado un conjunto de patrones

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_\ell, y_\ell) \tag{1.1}$$

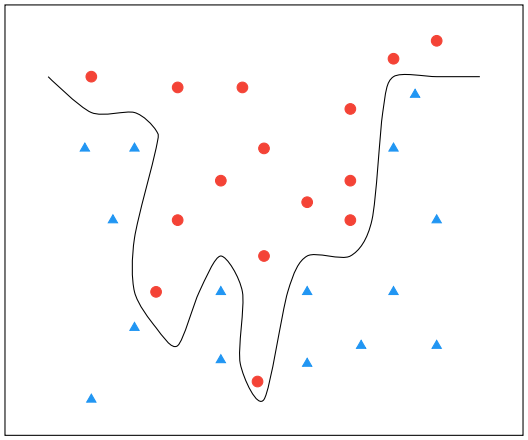
donde $\mathbf{x} \in \mathbb{R}^d$ es un patrón y $y \in \{0, 1\}$ es una categoría asociada a dicho patrón; el problema de la clasificación es encontrar un modelo funcional que permita una predicción razonable de categorías para nuevos patrones (Kramer, 2016).

1.3.3 Sobreajuste

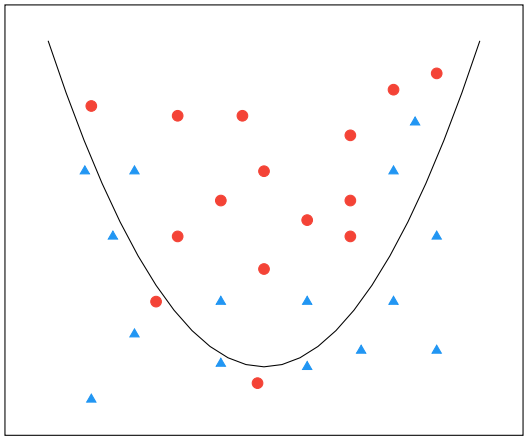
Durante el entrenamiento, el modelo desarrollado debe minimizar el error de su desempeño, pero durante las pruebas, tiene que maximizar sus habilidades para hacer predicciones correctas sobre patrones desconocidos. Este doble objetivo podría llevar al modelo a memorizar el conjunto de entrenamiento, en vez de entender las características esenciales de los patrones, que debería ser su objetivo principal (Chicco, 2017). En la figura 1.4



(a)



(b)



(c)

Figura 1.4: La figura 1.4a corresponde a un modelos no óptimo; la figura 1.4b a un modelo con sobreajuste y la figura 1.4c a un modelo óptimo.
Fuente: Adaptado de "Machine Learning wih TikZ" de Ochoa, 2019a.

En la figura 1.4a se distingue que la solución (línea en color negro) para discriminar entre dos clases de patrones no es la adecuada ya que en ambos lados de la solución se encuentras ambos patrones.

En la figura 1.4b se observa que todos los patrones se encuentran correctamente clasificados, el error se redujo a 0%, lo patrones de color rojo se encuentran por encima de la línea que los separa de los patrones de color azul. Sin embargo, esto corresponde a un sobreajuste.

Finalmente, en la figura 1.4c se halla el modelo óptimo sin importar que el error no se haya reducido a 0%.

1.4 Redes Neuronales Superficiales

1.4.1 Sistema Nervioso y Plasticidad Neuronal

Los seres humanos somos capaces de pensar abstractamente, comunicar pensamientos complejos y acumular información a través de generaciones mejor que cualquier otra especie en el planeta (Bostrom, 2014).

Estas capacidades dependen de una sofisticada gama de receptores sensoriales conectados al cerebro que es capaz de discriminar una gran variedad de eventos en el entorno. El cerebro organiza todo el flujo de información de estos receptores en percepciones, las cuales se almacenan en la memoria para una referencia futura, y luego en respuestas conductuales apropiadas. Esto lo logra el cerebro que usa neuronas y la conexión entre ellas (Kandel, Schwartz & Jessell, 2000).

Neurona

Las neuronas (células nerviosas) son las unidades básicas del cerebro. A pesar de que el cerebro humano contiene un extraordinario número de células (alrededor de 10^{11}) las cuales se pueden clasificar en cientos de tipos diferentes, todas comparten la misma arquitectura básica (Kandel y col., 2000).

Una neurona típica consiste en cuatro regiones definidas morfológicamente: el cuerpo celular (soma), dendritas, axón y terminales presinápticas, como se muestra en la figura 1.5. Cada una de estas regiones tiene su propio rol en la generación de señales y comunicación entre otras neuronas (Kandel y col., 2000).

El cuerpo celular es el centro metabólico de la célula. Las dendritas son ramificaciones en forma de árbol y reciben las señales entrantes de otras neuronas. En contraste, el axón se extiende lejos del cuerpo celular y conduce las señales a otras células (Kandel y col., 2000).

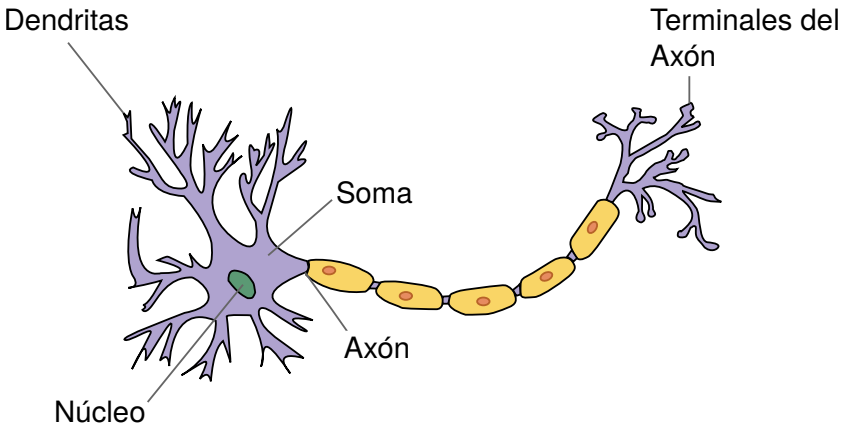


Figura 1.5: Dibujo esquemático de una neurona biológica.
Fuente: Adaptado de "File:Neuron.svg — Wikimedia Commons, the free media repository" de Wikimedia Commons, 2018a.

Un axón puede transportar señales eléctricas a lo largo de distancias que van desde 0.1mm a 3mm. Estas señales eléctricas, llamadas potenciales de acción, son impulsos nerviosos rápidos, transitorios todo o nada (Kandel y col., 2000).

Cerca del final, el axón se divide en finas ramas que forman comunicaciones con otras células. El punto donde dos células se comunican es conocido como sinapsis (Foster, 1897; Kandel y col., 2000). A la célula que transmite una señal es llamada célula presináptica y la que recibe una señal; célula postsináptica (Kandel y col., 2000) (ver figura 1.6). Las sinapsis provocan cambios en el sistema nervioso, a esto se le llama plasticidad neuronal (Kandel y col., 2000). Estos cambios están relacionados con el aprendizaje y la memoria (Takeuchi, Duszkiewicz & Morris, 2013).

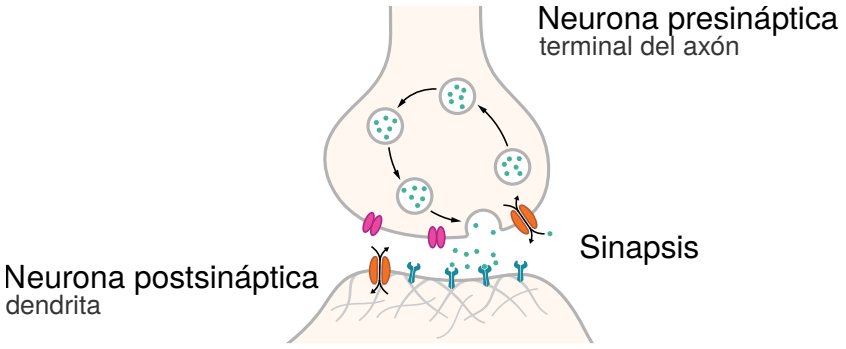


Figura 1.6: Dibujo esquemático de la sinapsis.
Fuente: Adaptado de "File:SynapseSchematic unlabeled.svg — Wikimedia Commons, the free media repository" de Wikimedia Commons, 2018b.

Teoría Hebbiana

La hipótesis de la plasticidad sináptica y memoria (SPM) afirman que la plasticidad sináptica dependiente de la actividad se induce en las sinapsis apropiadas durante la formación de la memoria y es tanto necesaria como suficiente para la codificación y almacenamiento de la información que subyace al tipo de memoria mediada por el área cerebral en la que se observa esa plasticidad (Takeuchi y col., 2013). Las hipótesis teóricas sobre el crecimiento de las conexiones neuronales en el cerebro y las circunstancias en las que tal crecimiento podría tener lugar datan de Cajal (1894), Hebb (1949) y Konorski (1948) (Takeuchi y col., 2013). Hebb (1949) propuso que el aprendizaje se da en el fortalecimiento de las conexiones de las neuronas presinápticas y postsinápticas cuando ocurre la sinapsis. Declaró lo siguiente,

> Cuando un axón de la célula A esta lo suficientemente cerca para excitar a la célula B y repetidamente o persistentemente toma parte en su disparo, algún proceso creciente o cambio metabólico toma lugar en una o ambas células tal que la eficiencia de A, como una de las células que descarga a B, se incrementa. (Hebb, 1949, p. 62)

También declaró (sin evidencia) un procedimiento para el cambio estructural,

> Cuando una célula ayuda repetidamente a otra a disparar, el axón de la primer célula desarrolla botones sinápticos (o los amplia si ya existen) en contacto con el soma de la segunda célula [1]. (Hebb, 1949, p. 63)

El postulado de Hebb puede ser formulado en términos matemáticos. Las señales de las neuronas presináptica y postsináptica se indican como x_i y y_j respectivamente, la sinapsis se denota como w_{ij}. El aprendizaje de Hebb se formula como

$$\Delta w_{ij} = \eta x_i y_j \tag{1.2}$$

donde η es una constante positiva que determina la *proporción de aprendizaje* (Haykin, 2009).

1.4.2 Neurona Artificial

McCulloch y Pitts (1943) formularon el primer modelo matemático de una neurona biológica con el objetivo entender la inteligencia y el aprendizaje basándose en las siguientes suposiciones físicas:

[1] "Soma" se refiere a las dendritas y al cuerpo, o a toda la célula, excepto a su axón.

1. La actividad de la neurona es un proceso *todo o nada*.

2. Un cierto número fijo de sinapsis debe excitarse dentro del período de adición latente para excitar una neurona en cualquier momento, este número es independiente de la actividad previa y posición en la neurona.

3. El único retraso significativo dentro del sistema nervioso es el retraso sináptico.

4. La actividad de cualquier sinapsis inhibidora impide absolutamente la excitación en la neurona en ese momento.

5. La estructura de la red no cambia con el tiempo.

La neurona de McCulloch–Pitts (ver figura 1.7) es, una unidad de umbral binario cuyo disparo o activación es independiente de los valores de sus entradas. La neurona recibe una serie de entradas excitatorias (positivas) o inhibitorias (negativas) a través de conexiones *sinápticas*. Si la suma de las señales positivas excede un valor ϑ, la neurona dispara, de otra manera, nunca realizará el disparo. Si una neurona recibe una señal en cualquiera de sus entradas inhibitorias, la neurona no puede disparar, a pesar de los valores exitatorios de entrada. El valor ϑ, es un umbral fijo único en la neurona (Scott, 2005). Así, la inhibición es la terminación o prevención de la actividad de un grupo de neuronas (McCulloch & Pitts, 1943).

McCulloch y Pitts (1943) demostraron que una red neuronal puede solucionar cualquier problema de computación que pueda ser resuelto por una máquina de Turing sin la necesidad de un cabezal y una cinta. Esto resultó de gran interés ya que proporciona una justificación psicológica de la computabilidad de Turing y sus equivalentes.

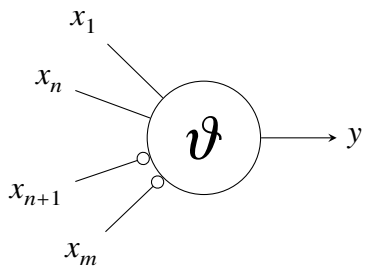


Figura 1.7: Neurona de McCulloch–Pitts. Las señales de entrada $x_1, \dots, x_n$ representan las conexiones excitatorias; las señales $x_{n+1}, \dots, x_m$, las conexiones inhibitorias y ϑ, es el umbral de la neurona.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Los cerebros pueden ser comparados con las computadoras, ya que (Khosrowpour, 2012, p. 2):

1. Las neuronas son dispositivos de encendido y apagado que pueden representar la información digital.

2. El umbral de la neurona permite crear compuertas lógicas.

3. El cerebro tiene canales de entrada y salida al igual que las computadoras.

4. El cerebro funciona con electricidad como las computadoras.

5. Una computadora tiene muchos transmisores como un cerebro tiene muchas neuronas (alrededor de 10^{11}).

1.4.3 Perceptrón

Rosenblatt (1957) desarrolló el perceptrón como un "modelo cerebral" con el objetivo de explicar el funcionamiento psicológico del cerebro en términos de las leyes de la física y las matemáticas. Es un descendiente directo de la neurona de McCulloch–Pitts; el enfoque filosófico está basado en las teorías cognitivas de Hayek y Hebb.

El perceptrón no se desarrolló como una copia detallada de algún sistema nervioso real. Se considera una red simplificada, diseñada para permitir el estudio de las relaciones entre la organización de una red nerviosa, la organización de su entorno y los funcionamientos "psicológicos" de los que es capaz la red. Lo más probable es que representen simplificaciones extremas del sistema nervioso central, en el que algunas propiedades se exageran, otras se suprimen. En este caso, las perturbaciones y refinamientos sucesivos del sistema pueden proporcionar una aproximación más cercana (Rosenblatt, 1961).

Un *perceptrón* es esencialmente una red neuronal de una sola capa con unidades *threshold* sin retroalimentación (Rumelhart, Hinton & McClelland, 1986). En ella se encuentran tres elementos básicos (Rosenblatt, 1957), como se muestra en la figura 1.8:

Sistema sensorial Es un conjunto de entradas, cada una se caracteriza por tener un peso propio. Una señal x_i de entrada es multiplicada por el peso de la sinapsis w_i.

Sistema de asociación Es la suma de las señales de entrada multiplicadas por sus respectivas sinapsis.

Sistema de respuesta Es la función de activación, para limitar la amplitud de la salida de la unidad.

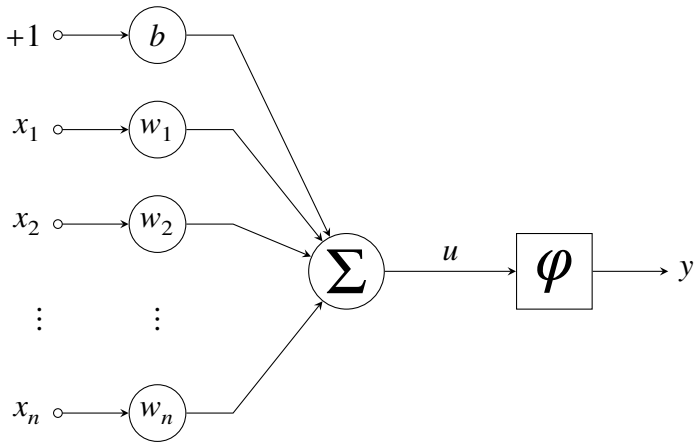


Figura 1.8: Perceptrón.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Modelo matemático

De acuerdo con Rumelhart, Hinton y Williams (1986b) las unidades reciben una entrada adicional con un valor constante de 1 asociada a un peso adicional w_0. A este peso se le llama *bias* y se define como *b*. La unidad se define como

$$u = \sum_{i=1}^{n} w_i x_i + b = \mathbf{x}^T \mathbf{w} \tag{1.3}$$

donde $\mathbf{x} = [+1, x_1, x_2, \dots, x_n]$ son las señales de entrada, $\mathbf{w} = [b, w_1, w_2, \dots, w_n]$ son los pesos sinápticos y *b* es el bias (Haykin, 2009). Así, el valor de la unidad es el producto interno del vector de entrada y el vector de los pesos (Jordan, 1986). La respuesta de activación de la unidad se determina por la *función threshold*, ver figura 1.9,

$$y = \begin{cases} 1 & u > 0 \\ 0 & u \leq 0, \end{cases} \tag{1.4}$$

donde 0 indica que la unidad esta inactiva y 1 que está activa. Otro modelo típico, es la *función signo*, ver figura 1.10,

$$y = \operatorname{sgn}(u) = \begin{cases} +1 & u > 0 \\ -1 & u \leq 0 \end{cases} \tag{1.5}$$

donde sus valores de activación están restringidos a $\{-1, +1\}$ (Rumelhart, Hinton & McClelland, 1986).

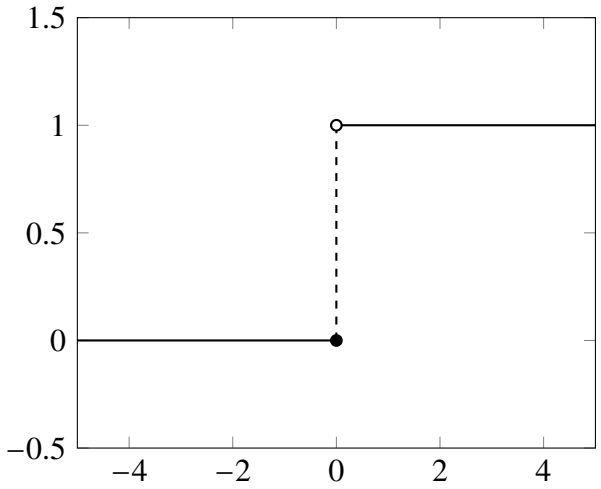


Figura 1.9: Función de activación threshold.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

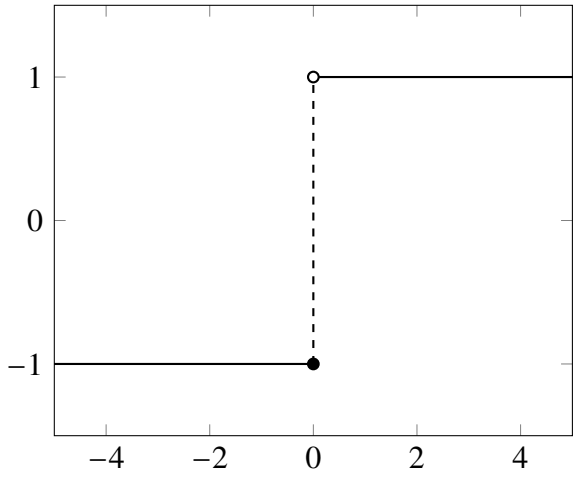


Figura 1.10: Función de activación signo.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Algoritmo de convergencia del perceptrón

El teorema de convergencia del perceptrón, garantiza que si un conjunto de patrones puede ser aprendido por un perceptrón, el procedimiento de aprendizaje encontrará un conjunto de pesos sinápticos que le permiten responder correctamente a todos los patrones de entrada (Rumelhart, Hinton & McClelland, 1986). Resumen del algoritmo de convergencia del perceptrón (Lippmann, 1987; Haykin, 2009):

1. Inicializar los pesos con valores pequeños
2. Presentar una nueva entrada y la salida deseada
3. Calcular la respuesta
$$y = \text{sgn}(\mathbf{x}^T(t)\mathbf{w}(t)) \tag{1.6}$$
4. Adaptación del vector de pesos. Los pesos no cambian si la respuesta del perceptrón es correcta
$$\mathbf{w}(t+1) = \mathbf{w}(t)\eta(d(t) - y(t))\mathbf{x}(t) \tag{1.7}$$
donde
$$d(t) = \begin{cases} +1 & \text{Si la entrada pertenece a la clase A} \\ -1 & \text{Si la entrada pertenece a la clase B} \end{cases} \tag{1.8}$$
y η es la proporción del aprendizaje.
5. Continuar con el paso 2

Prácticamente todas las reglas de aprendizaje para modelos en los cuales las fortalezas de las conexiones se modifican a través de la experiencia pueden considerarse una variante de la regla de aprendizaje de Hebb (Rumelhart, Hinton & McClelland, 1986).

1.4.4 Redes Neuronales Superficiales

El éxito de las redes neuronales artificiales (ANNs) en la resolución de problemas de reconocimiento de patrones fue un desafío para los investigadores. Aquí está el porqué. Cuando se trata de entender cómo funciona el cerebro, surgen dos preguntas diferentes:

1. ¿Qué sucede? ¿Cuáles son los principios de generalización que ejecuta el cerebro?
2. ¿Cómo sucede? ¿Cómo ejecuta el cerebro estos principios?

Las redes neuronales artificiales intentan responder la segunda pregunta utilizando un modelo de cerebro artificial motivado por los neurofisiólogos (Vapnik, 2006, p. 429).

Haykin (2009, p. 2) describe una red neuronal artificial como un procesador distribuido masivamente en paralelo formado por unidades de procesamiento simples que tiene una propensión natural a almacenar el conocimiento experiencial y ponerlo a disposición para su uso. Por lo que, una ANN se parece al cerebro en los siguientes dos aspectos:

1. El conocimiento es adquirido por la red desde su entorno a través de un proceso de aprendizaje.

2. Las fortalezas de conexión interna, conocidas como pesos sinápticos, se utilizan para almacenar el conocimiento adquirido.

Entre los tipos de redes neuronales artificiales se encuentra la red neuronal prealimentada. Esta es una red neuronal artificial que no contiene ciclos cerrados. La arquitectura de una red neuronal prealimentada (FNN) está definida por un grafo dirigido acíclico (Baum & Haussler, 1989). Un perceptrón multicapa (MLP) es una red neuronal prealimentada que está creada de por nodos de entrada, una o varias capas ocultas y una capa de salida (Baum & Wilczek, 1988), como se muestra en la figura 1.11.

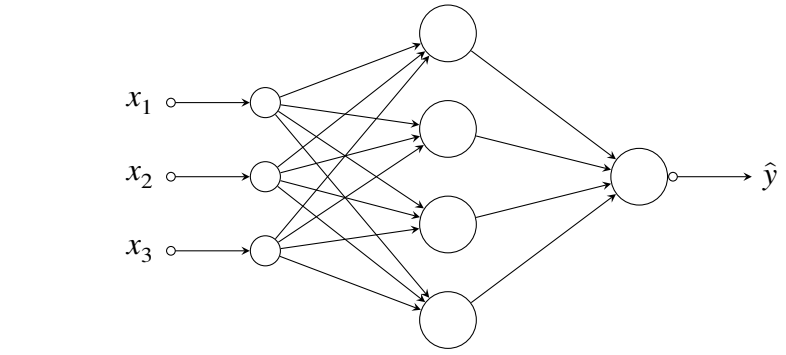
La arquitectura de una red neuronal superficial se caracteriza por tener una capa oculta (ver figura 1.11a) mientras que una red neuronal profunda se caracteriza por tener más de una capa oculta (ver figura 1.11b) (Mhaskar & Poggio, 2016).

Las redes neuronales superficiales en la que las unidades de la capa oculta usan funciones de activación sigmoidales han mostrado ser aproximadores universales (Cybenko, 1989; Funahashi, 1989; Hornik, Stinchcombe & White, 1989; Leshno, Lin, Pinkus & Schocken, 1993)

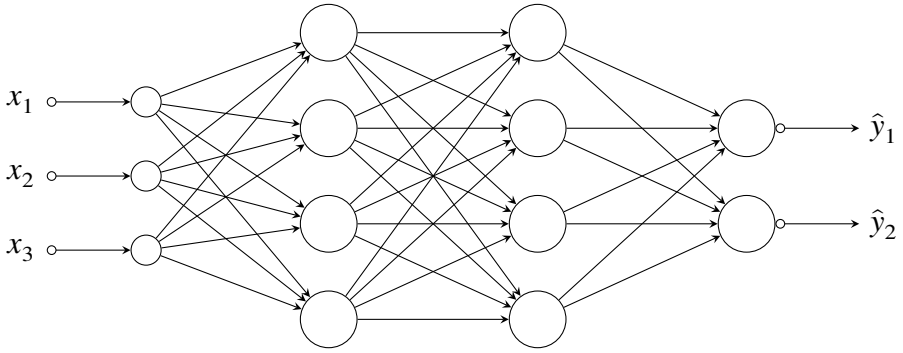
Capas ocultas

Las unidades de las capas ocultas actúan como *detectores de características* o *filtros*. Combinando estas características, las unidades de salida pueden llevar a cabo una mejor clasificación que sin las unidades ocultas (Touretzky & Pomerleau, 1989). Conforme el proceso de aprendizaje progresa a través del perceptrón multicapa, las unidades de las capas ocultas comienzan a *descubrir* gradualmente las características sobresalientes de los patrones de entrenamiento (Haykin, 2009).

El número de unidades ocultas debe calcularse. Si el número de unidades es menor a la complejidad del problema, entonces puede ocurrir un *ajuste bajo*. Esto ocurre cuando hay muy pocas unidades en la capa oculta para detectar de forma correcta las señales (patrones) de



(a)



(b)

Figura 1.11: Arquitecturas comunes de un perceptrón multicapa. La figura 1.11a corresponde a una ANN de dos capas, una capa oculta y una capa de salida. La figura 1.11b pertenece a una ANN de tres capas, dos capas ocultas y una capa de salida.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

entrada. Por otra parte, si hay más unidades de las necesarias, puede producirse un *sobreajuste*. El número de unidades en la capa oculta generalmente es determinado por alguna *regla del pulgar* (Karsoliya, 2012).

Elisseeff y Paugam–Moisy (1997) propusieron un modelo para calcular en número de unidades necesarias y suficientes en la capa oculta con funciones de activación no lineales y varias unidades en la capa de salida. El número de unidades necesarias se define como

$$N_H = \left\lceil \frac{N_P \, N_S}{N_I + N_S} \right\rceil, \tag{1.9}$$

mientras que el número de unidades suficientes es

$$N_H = 2 \left\lceil \frac{N_P}{N_I + N_S} \right\rceil N_S \tag{1.10}$$

donde N_I es el número de unidades de entrada; N_S, el número de unidades de salida y N_P, el número de ejemplares. Otras reglas del pulgar comunes son las siguientes (Karsoliya, 2012):

1. El número de unidades en la capa oculta es entre el 70%—90% el número de las entradas. Si no es suficiente, el número de unidades de la capa de salida se puede sumar más adelante.

2. El número de unidades de la capa oculta debe ser menos que dos veces el número de entradas.

3. El número de unidades en la capa oculta se encuentra entre el número de las entradas y el número de las unidades de salida.

Capa de salida

En una red neuronal artificial para el problema de clasificación, el número de unidades en la capa de salida corresponde al número de clases (Pal & Mitra, 1992). De modo que, cada unidad de salida representa una clase. Por lo cual, cada unidad es útil como discriminante entre la clase que representa y las demás (Allwein, Schapire & Singer, 2000). Es por esto que, un elemento del vector de salida corresponde a una clase al que se le asigna el valor uno, mientras que los demás elementos son cero (Pal & Mitra, 1992). Así, por ejemplo, a las clases *setosa*, *versicolor* y *virginica* de la flor de Iris (Fisher, 1936) se les puede asignar los siguientes

vectores de salida

$$\text{setosa} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{versicolor} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{virginica} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Funciones de activación

La función de activación calcula la salida de una unidad en función a sus entradas (Williams, 1986). Para lograr el entrenamiento de la red neuronal, se requiere de una función de activación no lineal continua (Rumelhart, Hinton & Williams, 1986a).

Función logística sigmoidal La curva "S" es tal vez la curva más importante. Al principio, la salida aumenta lentamente con la entrada, de modo que parece ser constante. Después comienza a cambiar cada vez más rápido y luego más y más lento hasta que se vuelve casi constante de nuevo (Domingos, 2015). Se dice que una función es sigmoidal si

$$\sigma(t) = \begin{cases} 1 & \to +\infty \\ 0 & \to -\infty. \end{cases} \tag{1.11}$$

Tales funciones surgen de la teoría de las redes neuronales artificiales como la función de activación de una unidad (Cybenko, 1989).

La función logística sigmoidal es la función más conocida que se utiliza en las redes neuronales prealimentadas debido a su no linealidad y la simplicidad computacional de su derivada (Minai & Williams, 1993). Se define como

$$\varphi(u) = \frac{1}{1 + \exp(-au)} \tag{1.12}$$

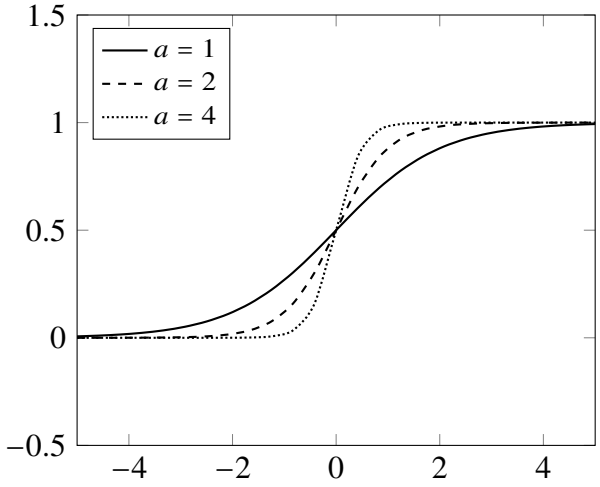
donde a indica la inclinación, su gráfica se muestra en la figura 1.12a. La derivada se resuelve como

$$\frac{\mathrm{d}\varphi}{\mathrm{d}u} = \varphi(u)(1 - \varphi(u)). \tag{1.13}$$

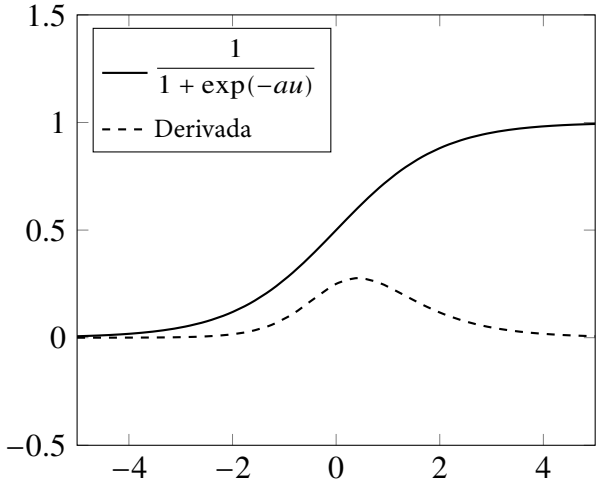
Su gráfica se muestra en la figura 1.12b.

Función tangente hiperbólica sigmoidal La función tangente hiperbólica (ver figura 1.13) también es una función sigmoidal en un rango de $(-1, 1)$, se define como

$$\varphi(u) = \tanh(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)} = \frac{2}{1 + \exp(-2u)} - 1 \tag{1.14}$$



(a)



(b)

Figura 1.12: En la figura 1.12a se muestra la función de activación logística sigmoidal con tres grados de inclinación diferentes. En la figura 1.12b se presenta la derivada de la función. Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

y su derivada se resuelve como

$$\frac{\mathrm{d}\varphi}{\mathrm{d}u} = 1 - \varphi^2(u). \tag{1.15}$$

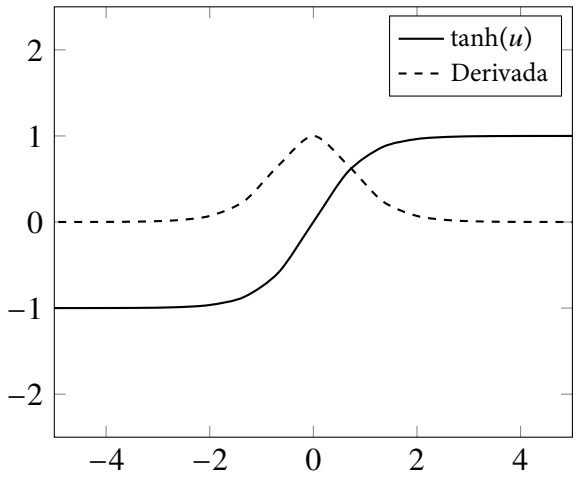


Figura 1.13: Función de activación tangente hiperbólica y su derivada.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Tanto la función logística sigmoidal y tangente hiperbólica son generalmente seleccionadas para ser usadas en la capa oculta (Bishop, 2006).

Función softmax También es llamada *exponencial normalizada*. Es usada en la capa de salida junto con la función de error entropía cruzada. Se define como

$$\varphi_i(\mathbf{u}) = \frac{\exp(u_i)}{\sum_{j=1}^{\ell} \exp(u_j)}, \tag{1.16}$$

y satisface $0 \leq y_i \leq 1$ y $\sum_i y_i = 1$ (Bishop, 2006). Su derivada se resuelve como

$$\frac{\partial \varphi_i}{\partial u_j} = \varphi_i(\mathbf{u})(\delta_{ij} - \varphi_j(\mathbf{u})) \tag{1.17}$$

donde δ_{ij} es la delta de Kronecker

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases} \tag{1.18}$$

Su uso en común en los problemas de clasificación multiclase (Bishop, 2006).

1.4.5 Funciones de error

El entrenamiento de una red neuronal artificial se basa en la minimización de una función de error (Bishop, 1996, p. 2). El objetivo es encontrar un procedimiento para evaluar las derivadas de la función de error respecto a los pesos y bias de la red (Bishop, 1995).

Suma de cuadrados

La suma de cuadrados es la función estándar del algoritmo retropropagación calcula la diferencia entre $\hat{y}$ y y de cada nodo de salida, la eleva al cuadrado y finalmente suma todos los resultados (Marsland, 2015). La función de error se define como

$$E = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{k} (\hat{y}_{ij} - y_{ij})^2 \tag{1.19}$$

donde n es el número de patrones, k es el número de clases, $\hat{y}$ es la salida de la red y y es la salida deseada.

Error cuadrático medio

La suma del error cuadrático (MSE) es un modelo de evaluación métrica usado en modelos de regresión. El error de predicción es la diferencia entre la salida de la red y la salida deseada (Sammut & Webb, 2010). La función de error se define como

$$E = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{k} (\hat{y}_{ij} - y_{ij})^2 \tag{1.20}$$

donde n es el número de patrones, k es el número de clases, $\hat{y}$ es la salida de la red y y es la salida deseada.

Entropía cruzada

La entropía cruzada (CE) presenta un método simple, eficiente y general para resolver una gran variedad de problemas de estimación y optimización (Rubinstein & Kroese, 2004). Se ha propuesto usar la función entropía cruzada con la finalidad de reducir el tiempo de entrenamiento (Hinton, 1989; Baum & Wilczek, 1988) La función de error se define como

$$E = -\sum_{j=1}^{n} \sum_{i=1}^{k} y_{ij} \ln \hat{y}_{ij} + (1 - y_{ij}) \ln(1 - \hat{y}_{ij}) \tag{1.21}$$

donde n es el número de patrones, k es el número de clases, $\hat{y}$ es la salida de la red y y es la salida deseada.

1.4.6 Superficie de error

El procedimiento de aprendizaje tiene una interpretación geométrica simple. Se construye un espacio multi-dimensional llamado *espacio sináptico* el cual tiene un eje por cada peso sináptico y un eje adicional, llamado *altura*, que corresponde a la medida de error. Por cada combinación de pesos la red neuronal tendrá un cierto error representado por un punto sobre el eje de la altura. Estos puntos forman una superficie llamada *superficie de error* (Hinton, 1989, p. 194). Si consideramos que $\mathbf{w} \subset \mathbb{R}^2$, la superficie de error se puede visualizar en un espacio de tres dimensiones, como se muestra en la figura 1.14, en el que se debe buscar el global mínimo.

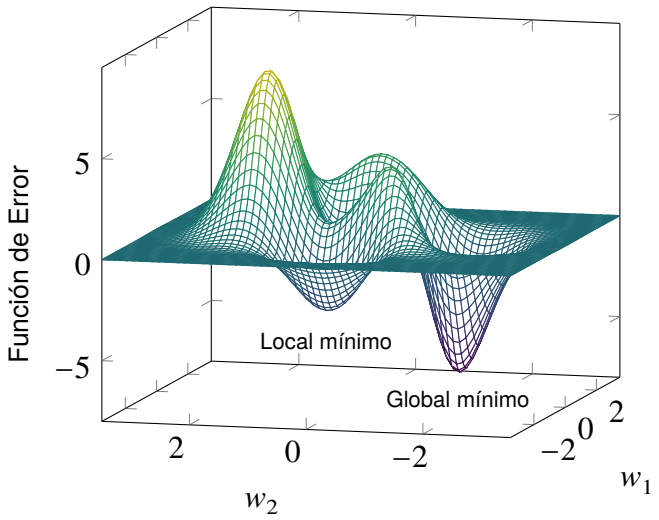


Figura 1.14: Superficie de error de una sola unidad sobre un espacio sináptico de dos dimensiones y un eje adicional para representar el error.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Una red neuronal con varias unidades ocultas genera una superficie irregular con muchos locales mínimos, por lo que el método del descenso del gradiente no garantiza encontrar el global mínimo y puede quedar atrapado en un local mínimo (Hinton, 1989; Rumelhart, Hinton & Williams, 1986b).

La función de error se minimiza mediante la selección de $\mathbf{w}$ que le permita a la red neuronal responder correctamente a todos los patrones (Rumelhart, Hinton & McClelland, 1986). Para minimizar la medida de error mediante el método del descenso del gradiente se deben calcular las derivadas parciales de E respecto a cada peso w de la red neuronal (Rumelhart,

Hinton & Williams, 1986b).

1.4.7 Inicialización

Cuando inicializa el proceso de aprendizaje, se deben inicializar los pesos de la red neuronal, con la elección adecuada de los valores se puede obtener una red neuronal exitosa. Cuando se asignan valores iniciales grandes, es muy probable que las unidades de la red se vean saturadas. Si esto sucede, el proceso de aprendizaje ira más despacio. Si los pesos se inicializan con valores muy pequeños el algoritmo puede operar en un área muy plana de la superficie de error, lo que da origen a un punto de silla (Haykin, 2009).

Nguyen y Widrow (1990) desarrollaron un método para inicializar los pesos de la red neuronal para reducir el tiempo de aprendizaje. La idea que presentan los autores es la siguiente. Se eligen valores aleatorios pequeños como pesos iniciales de la red neuronal. Los pesos se actualizan de tal manera que la región de interés se divide en pequeños intervalos. Entonces, es razonable acelerar el proceso de aprendizaje estableciendo los pesos iniciales de la capa oculta de modo que a cada nodo se le asigne su propio intervalo al comienzo del aprendizaje. A medida que el proceso de aprendizaje avanza, todos nodos de la capa oculta tienen la libertad de ajustar el tamaño de su intervalo y ubicación.

1.4.8 Retropropagación

Desde los primeros días del reconocimiento de patrones, el objetivo de los investigadores ha sido reemplazar el trabajo manual con perceptrones multicapa entrenables, que a pesar de su simplicidad, la solución no se entendió ampliamente hasta mediados de los años ochenta. El algoritmo retropropagación que permite entrenar una arquitectura multicapa usando el descenso de gradiente estocástico, fue descubierto independientemente por Werbos (1974), LeCun (1985), Rumelhart, Hinton y Williams (1986b) como se cita en LeCun y col. (2015).

Rumelhart, Hinton y Williams (1986b) describieron el algoritmo de aprendizaje retropropagación (BP) para entrenar una red neuronal con capas ocultas. El objetivo consiste en encontrar un conjunto de pesos que aseguren que para cada vector de entrada el vector de salida producido por la red neuronal sea lo suficientemente cercano al vector de salida deseado minimizando la función de error E (Rumelhart, Hinton & Williams, 1986b). En general, el algoritmo retropropagación consiste en dos etapas (Rumelhart, Hinton & Williams, 1986b):

Propagación hacia delante Las entradas alimentan la red y las unidades de cada capa almacenan sus estados determinados por la entrada que reciben de unidades de las capas anteriores.

Actualización de pesos El error de predicción es calculado en la capa de salida, este error es propagado hacia atrás donde se presentan dos casos para ajustar los pesos. El primer caso se ocupa de propagar el error hacia la capa de salida mientras que el segundo caso se encarga de propagar el error a la capa oculta. Finalmente, en ambos casos se ajustan los pesos. El algoritmo utiliza la optimización determinista para minimizar la suma del error cuadrático utilizando el método del descenso del gradiente.

El proceso de aprendizaje puede desarrollarse de dos modos distintos (Hinton, 1989):

Por bloques El aprendizaje considera todos los patrones de entrenamiento que se acumulan antes de realizar la actualización de los pesos.

En línea Requiere menos memoria. La actualización de los pesos se realiza después de la presentación de cada patrón de entrenamiento.

Una de las razones a favor del aprendizaje en línea es que posee algo de *aleatoriedad* que podría ayudar a escapar de un local mínimo (Battiti, 1992).

Descenso del gradiente

Para describir el algoritmo retropropagación en una red neuronal artificial de dos capas emplearé la siguiente notación:

- Los índices, i, j y k se usan para describir las unidades de entrada, las unidades de la capa oculta y las unidades de la capa de salida respectivamente.
- Las entradas se describen como x_i.
- Los pesos sinápticos se describen con el símbolo w. De modo que w_{ij} son los pesos sinápticos que se encuentran entre la capa de entrada y la capa oculta; y w_{jk} son los pesos sinápticos que se encuentran entre la capa oculta y la capa de salida.
- La salida deseada se describe como y.
- Las salidas de las unidades de la capa oculta se describen como y_j, y las salidas de las unidades de la capa de salida se describen como $\hat{y}_k$.
- La función de error es la suma de cuadrados y la función de activación es la función logística sigmoidal.

Propagación hacia delante La salida es calculada. Y cada una de las unidades de la red establece su estado aplicando las ecuaciones (1.3) y (1.12). Todas las unidades dentro de una capa establecen sus estados en paralelo, pero las de más capas establecen sus estados secuencialmente comenzando con la capa oculta hasta determinar el estado de las unidades ocultas.

Propagación hacia la capa de salida Se calculan las derivadas parciales de E respecto a w_{jk} de la capa de salida.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial u_k}\frac{\partial u_k}{\partial w_{jk}} \tag{1.22}$$

Resolviendo las derivadas parciales se obtiene

$$\frac{\partial E}{\partial w_{jk}} = \left[(\hat{y}_k - y)\hat{y}_k(1-\hat{y}_k)\right] y_j = \delta_k y_j \tag{1.23}$$

Propagación hacia la capa oculta Se calculan las derivadas parciales de E respecto a w_{ij} de la capa oculta teniendo en cuenta todas las conexiones de la unidad j.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial w_{ij}} \tag{1.24}$$

Resolviendo las derivadas parciales se obtiene

$$\frac{\partial E}{\partial w_{ij}} = \left[\hat{y}_i(1-\hat{y}_i)\sum \delta_k w_{jk}\right] x_i = \delta_j x_i \tag{1.25}$$

Actualización de pesos Para actualizar cada peso se usa el método del descenso del gradiente

$$\Delta w = -\gamma \frac{\partial E}{\partial w} \tag{1.26}$$

donde γ es la razón de aprendizaje (Rumelhart, Hinton & Williams, 1986b).

Descenso del gradiente con impulso El método del *descenso del gradiente* no converge tan rápido como otros métodos que usan la segunda derivada. Este se puede mejorar agregando el termino *impuso* (o *momentum*)

$$\Delta w(t) = -\gamma \frac{\partial E}{\partial w}(t) + \alpha \Delta w(t-1) \tag{1.27}$$

donde γ es la razón de aprendizaje y α es la razón de impulso (Rumelhart, Hinton & Williams, 1986b). El valor de la razón de aprendizaje debe ser lo suficientemente grande para lograr un

aprendizaje rápido y suficientemente pequeño para evitar la oscilación. El termino *impulso* permite incrementar la razón de aprendizaje sin llevar a la oscilación (Rumelhart, Hinton & Williams, 1986a).

Descenso del gradiente con impulso y razón de aprendizaje adaptativo La razón de aprendizaje es un factor que determina el tamaño de los ajustes de los pesos en cada iteración. Cuando su valor es muy grande la búsqueda oscilara. Si es muy pequeña el tiempo de convergencia incrementara significativamente (Yu & Liu, 2002).

Es posible acelerar la convergencia adaptando la razón de aprendizaje conforme avanza sobre la superficie de error. Una razón de aprendizaje adaptiva intenta mantener el tamaño de los pasos de aprendizaje lo más grande posible mientras se mantiene el aprendizaje estable. Por lo tanto, la velocidad de aprendizaje responde a la complejidad de la superficie del error local (The MathWorks, Inc, 2006a). Las siguientes modificaciones se realizan al método retropropagación (Vogl, Mangis, Rigler, Zink & Alkon, 1988):

1. En vez de actualizar los pesos de la red cada vez que un patrón es presentado a la red, la red es actualizada solamente después de que todos los patrones hayan ingresado. La actualización de los pesos es

$$\Delta w(t) = -\gamma \sum_{j=1}^{n} \frac{\partial E}{\partial w}(t) + \alpha \Delta w(t-1) \tag{1.28}$$

2. Si una actualización reduce el error, γ es multiplicada por un factor $\phi > 1$ para la siguiente iteración. Si un paso incrementa el error más de un pequeño porcentaje (típicamente entre $1\% - 5\%$) por encima del valor anterior, se rechazan todos los cambios en los pesos, γ se multiplica por un factor $\beta < 1$, α se establece en cero y el paso se repite. Cuando se da un paso exitoso, α retoma su valor original.

Gradiente conjugado

El método del gradiente conjugado es un método iterativo para resolver sistemas de ecuaciones lineales

$$Ax = b, \tag{1.29}$$

donde A es una matriz simétrica positiva. El problema (1.29) es equivalente al problema

$$\min \quad \phi(x) \stackrel{\text{def}}{=} \frac{1}{2} x^T A x - b^T x, \tag{1.30}$$

por lo que ambos problemas tienen la misma solución (Nocedal & Write, 2006).

Actualizaciones Fletcher–Reeves Fletcher y Reeves (1964) dieron a conocer el método del gradiente conjugado para resolver ecuaciones no lineales haciendo dos simples cambios al algoritmo 1.1. Primero, en lugar de la fórmula para la longitud del paso α_k, necesitamos realizar una búsqueda lineal que identifique un mínimo aproximado de la función no lineal f a lo largo de p_k. Segundo, el residual r, debe reemplazarse por el gradiente de la función objetivo no lineal f (Nocedal & Write, 2006).

$$\beta_{k+1}^{FR} = \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k} \tag{1.37}$$

Actualizaciones Polak–Ribière Es una variante del método de Fletcher–Reeves que difiere en la elección del parámetro β_{k+1} (Nocedal & Write, 2006).

$$\beta_{k+1}^{PR} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} \nabla f_k)}{\|\nabla f_k\|^2} \tag{1.38}$$

Gradiente conjugado con reinicio Powell–Beale El método de gradiente conjugado es particularmente útil para minimizar funciones de varias variables ya que no requiere el almacenamiento de alguna matriz. Sin embargo, la tasa de convergencia del algoritmo es solo lineal a menos que el procedimiento iterativo se "reinicie" (Powell, 1977).

Powell (1977) sugirió que se debería usar la técnica de reinicio de Beale estableciendo $t = k - 1$ cuando

$$|g_{k-1}^T g_k| \geq 0.2\|g_k\|^2. \tag{1.46}$$

se mantiene o cada n iteraciones, lo que ocurra primero. Además, Powell sugirió que uno debería verificar si la dirección de búsqueda es una dirección de descenso suficiente verificando si

$$-1.2\|g_k\|^2 \leq d_k^T g_k \leq -0.8\|g_k\|^2 \tag{1.47}$$

se mantiene. En caso de que no se cumpla, entonces Powell recomienda que el procedimiento se puede reiniciar poniendo $t = k - 1$ y redefiniendo d_k por medio de

$$d_k = -g_k + \beta_k d_{k-1} + \gamma d_t \tag{1.48}$$

donde

$$\gamma_k = \frac{g_k^T (g_{t+1} - g_t)}{d_t^T (g_{t+1} - g_t)} \tag{1.49}$$

usando $\gamma_{t+1} = 0$ (Sherali & Ulular, 1990).

Algoritmo 1.1: Gradiente conjugado.

Datos: x_0
$r_0 \leftarrow Ax_0 - b$
$p_0 \leftarrow -r_0$
$k \leftarrow 0$
mientras $r_k \neq 0$ **hacer**

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k} \tag{1.31}$$
$$x_{k+1} \leftarrow x_k + \alpha_k p_k \tag{1.32}$$
$$r_{k+1} \leftarrow r_k + \alpha_k A p_k \tag{1.33}$$
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{1.34}$$
$$p_{k+1} \leftarrow - r_{k+1} + \beta_{k+1} p_k \tag{1.35}$$
$$k \leftarrow k + 1 \tag{1.36}$$

fin

Algoritmo 1.2: Gradiente conjugado con actualizaciones Fletcher–Reeves.

Datos: x_0
$f_0 = f(x_0)$
$\nabla f_0 = \nabla f(x_0)$
$p_0 \leftarrow -\nabla f_0$
$k \leftarrow 0$
mientras $\nabla f_k \neq 0$ **hacer**

$$\text{Calcular } \alpha_k \tag{1.39}$$
$$x_{k+1} \leftarrow x_k + \alpha_k p_k \tag{1.40}$$
$$\text{Evaluar } \nabla f_{k+1} \tag{1.41}$$
$$r_{k+1} \leftarrow r_k + \alpha_k A p_k \tag{1.42}$$
$$\beta_{k+1}^{FR} \leftarrow \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k} \tag{1.43}$$
$$p_{k+1} \leftarrow - \nabla f_{k+1} + \beta_{k+1} p_k \tag{1.44}$$
$$k \leftarrow k + 1 \tag{1.45}$$

fin

Gradiente conjugado escalado Møller (1993) desarrolló el algoritmo gradiente conjugado escalado (SCG) basado en la técnica de optimización en análisis numérico gradiente conjugado (CG). Su propósito fue desarrollar un algoritmo para eliminar las desventajas del algoritmo estándar BP, el cual tiene un mal funcionamiento en problemas de gran escala y el cual depende de los parámetros establecidos por el usuario *razón de aprendizaje* y la constante *momentum*.

La idea básica es combinar el enfoque del modelo de región de confianza usado en el algoritmo Levenberg-Marquardt (LM) con el enfoque del gradiente conjugado (Møller, 1993).

Método de Newton

BFGS quasi-Newton El método quasi-Newton usa la segunda derivada de la función de error E para ajustar los pesos. La actualización de los pesos se da por

$$\Delta w = -\mathbf{H}^{-1} \nabla E \tag{1.50}$$

donde $\mathbf{H}$ es una matriz Hessiana simétrica de $n \times n$ y n es el número de *pesos* y *bias* de la red neuronal.

La matriz Hessiana se actualiza en cada iteración del algoritmo usando el método Broyden–Fletcher–Goldfarb–Shanno (BFGS), definido como

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{1.51}$$

donde $B = \mathbf{H}^{-1}$ (Mandic & Chambers, 2001).

Algoritmo Levenberg-Marquardt Hagan y Menhaj (1994) presentaron una modificación al método de Newton para entrenar por bloques un MLP. El algoritmo LM es una aproximación al método de Newton. El paso clave es calcular la Matriz Jacobiana (Hagan & Menhaj, 1994). Teniendo una función E que se desea minimizar respecto a cada w, el método de Newon sería

$$\Delta w = -\left[\nabla^2 E\right]^{-1} \nabla E \tag{1.52}$$

donde $\nabla^2 E$ es la matriz Hessiana y ∇E es el gradiente. Asumiendo que E es una suma de cuadrados

$$E = \sum_{i=1}^{\ell} e_i^2 \tag{1.53}$$

entonces

$$\nabla E = \mathbf{J}^T e \tag{1.54}$$

$$\nabla^2 E = \mathbf{J}^T \mathbf{J} + S \tag{1.55}$$

donde $\mathbf{J}$ es la matriz Jacobiana

$$\mathbf{J} = \begin{bmatrix} \frac{\partial E_1}{\partial w_1} & \frac{\partial E_1}{\partial w_2} & \cdots & \frac{\partial E_1}{\partial w_n} \\ \frac{\partial E_2}{\partial w_1} & \frac{\partial E_2}{\partial w_2} & \cdots & \frac{\partial E_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E_m}{\partial w_1} & \frac{\partial E_m}{\partial w_2} & \cdots & \frac{\partial E_m}{\partial w_n} \end{bmatrix} \tag{1.56}$$

y

$$S = \sum_{i=1}^{\ell} e_i \nabla^2 e_i. \tag{1.57}$$

La actualización de los pesos se da por

$$\Delta w = \left[\mathbf{J}^T \mathbf{J} + \mu \mathbf{I}\right]^{-1} \mathbf{J}^T, \tag{1.58}$$

donde el parámetro μ es multiplicado por un factor β cuando un paso incremente E. Cuando un paso reduzca E, μ es dividido por β.

Secante de un paso El método secante de un paso reduce el tiempo computacional y el almacenamiento de la matriz $\mathbf{H}$. La dirección de búsqueda es

$$p_{k+1} = -g_k + A_k s_k + B_k y_k \tag{1.59}$$

donde los escalares A_k y B_k son la combinación de los productos de los vectores s_k, g_k y y_k:

$$A_k = -\left(1 + \frac{y_k^T y_k}{s_k^T y_k}\right) \frac{s_k^T g_k}{s_k^T y_k} + \frac{y_k^T g_k}{s_k^T y_k}; \quad B_k = \frac{s_k^T g_k}{s_k^T y_k}. \tag{1.60}$$

Regularización bayesiana

La regularización es un método que mejora la generalización restringiendo el tamaño de los pesos de la red neuronal. Cuando los pesos se mantienen pequeños, la respuesta de la red

neuronal será "suave" (Foresee & Hagan, 1997).

En el entrenamiento típico se reduce la función de error

$$E = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_{ij} - y_{ij})^2. \tag{1.61}$$

En la regulación la función de error se convierte en

$$E = \beta E_D + \alpha E_w \tag{1.62}$$

donde E_D es la MSE de la respuesta de la red, E_w es la MSE de los pesos de la red, β y α son parámetros de la función (Foresee & Hagan, 1997).

Los pesos se actualizan de acuerdo a la regla de Bayes

$$P(\mathbf{w}|D, \alpha, \beta, M) = \frac{P(D|\mathbf{w}, \beta, M)P(\mathbf{w}|\alpha, M)}{P(D|\alpha, \beta, M)} \tag{1.63}$$

donde D es el conjunto de datos, M es el modelo de la red neuronal y $\mathbf{w}$ es el vector de los pesos de la red. El ajuste de los parámetros β y α se da por la regla de Bayes

$$P(\alpha, \beta|D, M) = \frac{P(D|\alpha, \beta, M)P(\alpha, \beta|M)}{P(D|M)} \tag{1.64}$$

Si asumimos una densidad previa uniforme $P(\alpha, \beta|M)$ para los parámetros de regularización α y β, a continuación, la maximización de la parte posterior se logra maximizando la función de probabilidad $P(M|\alpha, \beta, M)$. La optimización bayesiana regula los parámetros del algoritmo Levenberg–Marquardt usando la aproximación de Gauss–Newton a la matriz de Hessiana (Foresee & Hagan, 1997):

1. Se inicializan los parámetros $\alpha = 0$ y $\beta = 1$. Los pesos se inicializan con el algoritmo de Nguyen y Widrow

2. Minimizar la función de error con el algoritmo Levenberg–Marquardt

3. Calcular $\gamma = N - 2\alpha\,\mathrm{tr}(\mathbf{H})^{-1}$ haciendo uso de la aproximación Gauss–Newton para la Hessiana del algoritmo Levenberg–Marquardt: $\mathbf{H} = 2\,\beta\,\mathbf{J}^T\mathbf{J} + \alpha\,\mathbf{I}_N$

4. Calcular los nuevos valores de los parámetros

$$\alpha = \frac{\gamma}{2E_w} \tag{1.65}$$

$$\beta = \frac{\gamma}{2E_D} \tag{1.66}$$

5. Ahora itere los pasos del 1 al 3 hasta la convergencia

Donde γ es el número efectivo de parámetros, es una medida de los parámetros en la red neuronal que se utilizan efectivamente para reducir la función de error. Y N es el número total de parámetros en la red neuronal.

Resilient propagation

El algoritmo Resilient propagation (Rprop), fue desarrollado por Riedmiller y Braun. Es un esquema de aprendizaje por bloque que realiza una adaptación directa de los pesos basada en la información del gradiente local (Riedmiller & Braun, 1993).

El procedimiento introduce un *valor de actualización*, descrito como Δ_{ij}^t, el cual solo determina el tamaño de la actualización del peso. Este valor de actualización se adapta durante el proceso de aprendizaje mediante la siguiente regla de aprendizaje

$$\Delta_{ij}^t = \begin{cases} \eta^+ \times \Delta_{ij}^{t-1} & \text{Si } \frac{\partial E}{\partial w_{ij}}^{t-1} \times \frac{\partial E}{\partial w_{ij}}^{t} > 0 \\ \eta^- \times \Delta_{ij}^{t-1} & \text{Si } \frac{\partial E}{\partial w_{ij}}^{t-1} \times \frac{\partial E}{\partial w_{ij}}^{t} < 0 \\ \Delta_{ij}^{t-1} & \text{de lo contrario} \end{cases} \tag{1.67}$$

donde $0 < \eta^- < 1 < \eta^+$, $\eta^- = 0.5$ y $\eta^+ = 1.2$. Para comprender mejor, cada vez que la derivada parcial respecto al peso w_{ij} cambia de signo, lo que indica que la última actualización fue demasiado grande y el algoritmo ha saltado sobre un local mínimo, el valor de actualización disminuye por el factor η^-. Si la derivada retiene su signo, el valor de actualización se incrementa ligeramente para acelerar la convergencia sobre regiones superficiales (Riedmiller & Braun, 1993).

Una vez que el valor de actualización para cada peso ha sido adaptado, la actualización de los pesos sigue la siguiente regla

$$\Delta w_{ij}^t = \begin{cases} -\Delta_{ij}^t & \text{Si } \frac{\partial E}{\partial w_{ij}}^{t} > 0 \\ +\Delta_{ij}^t & \text{Si } \frac{\partial E}{\partial w_{ij}}^{t} < 0 \\ 0 & \text{de lo contrario} \end{cases} \tag{1.68}$$

$$w_{ij}^{t+1} = w_{ij}^t + \Delta w_{ij}^t. \tag{1.69}$$

Al incio del algoritmo, todos los valores de actualización Δ_{ij} se establecen con un valor inicial $\Delta_0 = 0.1$ (Riedmiller & Braun, 1993).

1.4.9 Interrupción anticipada

Como se ha dicho anteriormente, el objetivo es crear una red neuronal artificial que proporcione predicciones útiles sobre patrones desconocidos, y esto no es la red neuronal artificial que provee el error más pequeño sobre el conjunto de entrenamiento (Bishop, 1995).

Cuando entrenamos una ANN esperamos obtener una óptima generalización. Sin embargo, todas las arquitecturas son propensas al sobreajuste. Esto quiere decir, que mientras la red parece mejorar, el error de entrenamiento decrementa, pero en algún punto comienza a empeorar. Por lo que el error sobre los datos de prueba incrementa (Prechelt, 2012). Este efecto se observa en la figura 1.15.

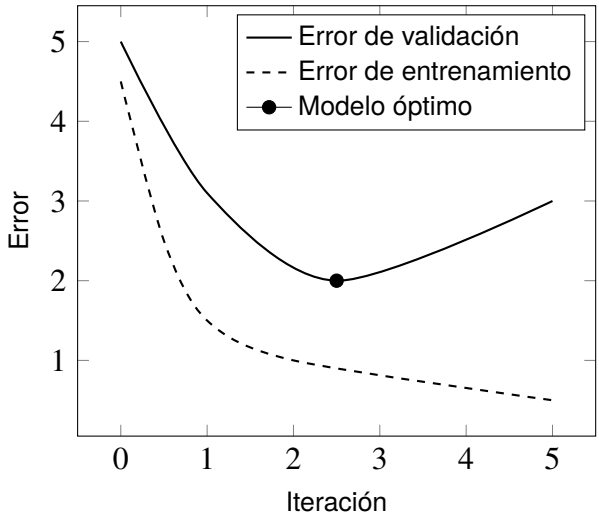


Figura 1.15: Interrupción anticipada.
Fuente: Adaptado de "Multilayer Perceptron with TikZ" de Ochoa, 2019b.

Esta técnica es una alternativa a la regularización como una forma de controlar la complejidad de una red neuronal (Bishop, 2006). Ha sido ampliamente usada ya que es fácil de entender e implementar. Consiste en el siguiente procedimiento (Prechelt, 2012):

1. Dividir los patrones de entrenamiento en un conjunto de entrenamiento y un conjunto de validación.

2. Entrenar solo con el conjunto de entrenamiento y evaluar de vez en cuando con el conjunto de validación. Por ejemplo, evaluar cada quinta época.

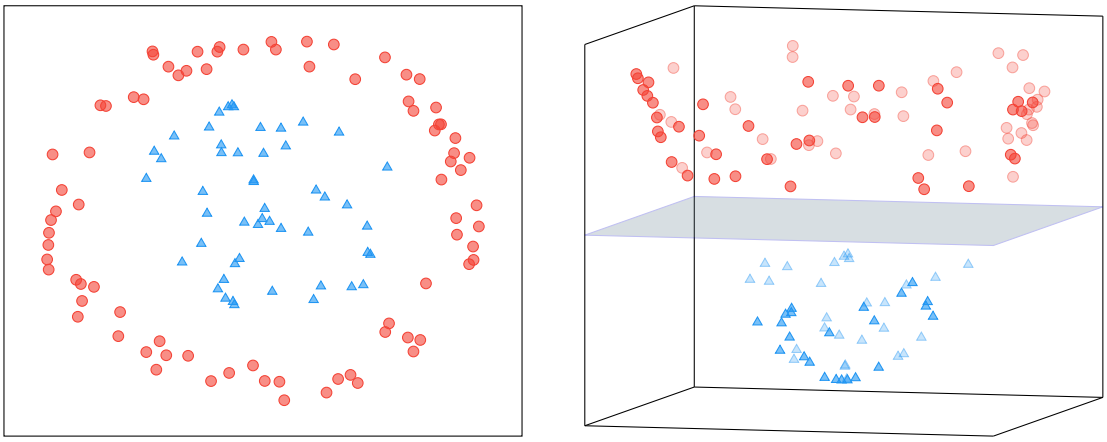
3. Detener el entrenamiento tan pronto el error del conjunto de evaluación sea más alto que la última vez que se verificó.

4. Usar los pesos que tenía la red en el paso anterior como resultado del entrenamiento.

1.5 Máquinas de Soporte Vectorial

Cortes y Vapnik (1995) desarrollaron una máquina de aprendizaje inspirada en la teoría de aprendizaje estadístico para resolver problemas de clasificación binaria. A la que llamaron *Support-Vector Network*.

Cortes y Vapnik (1995) implementaron la siguiente idea: cuando se tiene un conjunto de patrones de entrenamiento que no es linealmente separable, una máquina de soporte vectorial (SVM) mapea los datos hacia un espacio de dimensión mayor donde se podrá crear una superficie lineal (o hiperplano) que permita separar ambas clases, como se muestra en la figura 1.16.



(a) (b)

Figura 1.16: Ejemplo de la transformación de un conjunto de patrones de entrenamiento en un espacio de entrada $\mathbb{R}^2$ hacia una dimensión mayor, $\mathbb{R}^3$. $\phi = \mathbb{R}^2 \rightarrow \mathcal{H} = \mathbb{R}^3$.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

El problema de separar dos conjuntos de datos con un hiperplano de margen máximo en el espacio de entrada ha sido estudiado por Cover (1965), Vapnik y Lerner (1963). No obstante, es la combinación de este problema de optimización con funciones kernel lo que produjo máquinas de soporte vectorial (Shawe–Taylor & Cristianini, 2004). La arquitectura de las máquinas de soporte vectorial se observa en la figura 1.17.

Las características clave de las máquinas de soporte vectorial son el uso de métodos kernel, la ausencia de mínimos locales y el control de capacidad obtenido al optimizar el margen (Shawe–Taylor & Cristianini, 2004).

De acuerdo con Boser, Guyon y Vapnik (1992), el maximizar el margen entre dos clases del conjunto de entrenamiento presenta algunas ventajas:

- Es posible que el resultado de la clasificación logre una separación de los datos de entrenamiento sin errores.

- Los patrones atípicos son identificados por el algoritmo y pueden ser eliminados fácilmente con o sin supervisión.
- La sensibilidad del clasificador a la limitada exactitud computacional es mínima a comparación con otras separaciones de margen menor.

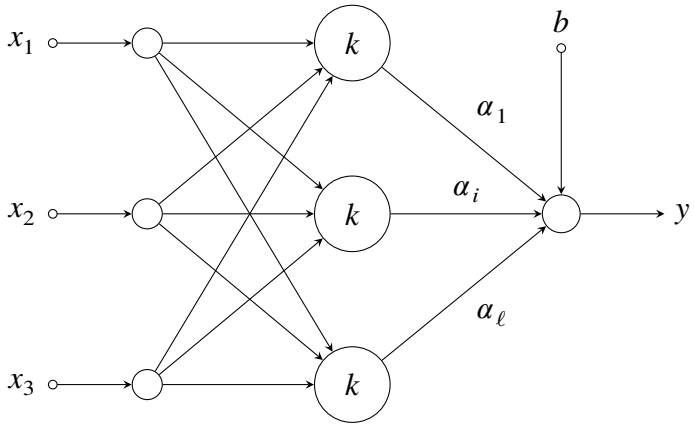


Figura 1.17: Arquitectura de una máquina de soporte vectorial.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

De acuerdo con Vapnik (2006), el desarrollo de las SVMs se dio en tres etapas:

1. *Hiperplano de separación óptima.* Se busca construir un hiperplano que separe un conjunto de datos de entrenamiento con el margen más amplio.
2. *Capacidad de control en el espacio de Hilbert.* Se usa el teorema de Mercer para construir un hiperplano de separación en el espacio de Hilbert.
3. *Máquinas de soporte vectorial.* Se generaliza la idea para construir un hiperplano de separación cuando los datos de entrenamiento no son separables.

1.5.1 Teoría del Aprendizaje Estadístico

A fin de elegir la mejor aproximación disponible para la respuesta del supervisor, debemos medir la pérdida o discrepancia $L(y, f(x, \alpha))$ entre la respuesta del supervisor a una entrada dada y la respuesta proporcionada por el algoritmo de aprendizaje. El valor de la discrepancia dada por el riesgo funcional

$$R(\alpha) = \int L(y, f(x, \alpha)) \, dP(x, y) \tag{1.70}$$

donde $\alpha \in A$ es un parámetro ajustable. El objetivo es encontrar una función $f(x, \alpha)$, de un conjunto de funciones, que minimice el riesgo funcional $R(\alpha)$ (Vapnik, 1999).

El problema de minimizar el riesgo funcional incluye tres problemas estadísticos básicos:

Reconocimiento de patrones La salida del supervisor toma solo dos valores $y = \{0, 1\}$.

$$L(y, f(x, \alpha)) = \begin{cases} 0 & y = f(x, \alpha) \\ 1 & y \neq f(x, \alpha) \end{cases} \tag{1.71}$$

La función de perdida describe una función indicatriz (funciones que toman solo dos valores, cero y uno) (Vapnik, 1999).

Estimación de regresión La salida del supervisor son valores reales.

$$L(y, f(x, \alpha)) = (y - f(x, \alpha))^2. \tag{1.72}$$

Una característica importante del problema de estimación de regresión es que la función de perdida puede tomar valores no negativos arbitrarios, mientras que el problema del reconocimiento de patrones solo puede tomar dos valores (Vapnik, 1998).

Estimación de densidad El problema de estimación de densidad considera de un conjunto de densidades. La función de perdida es

$$L(y, f(x, \alpha)) = -\log p(x, \alpha). \tag{1.73}$$

Tomar valores arbitrarios en el intervalo $(-\infty, \infty)$, mientras que el problema de estimación de regresión solo toma valores no negativos (Vapnik, 1998).

1.5.2 Minimización del riesgo estructural

Una estructura es el conjunto de subconjuntos anidados de funciones admisible.

$$S_1 \subset S_2 \subset \ldots S_k. \tag{1.74}$$

El principio de la minimización del riesgo estructural (SRM) requiere minimizar el riesgo empírico de cada uno de los elementos de la estructura. El elemento óptimo S^* se selecciona para minimizar el riesgo garantizado, definido como la suma del riesgo empírico y el intervalo de confianza (Vapnik, 1992).

1.5.3 Hiperplano de separación en patrones separables

Si un conjunto de patrones de entrenamiento asociados a una clase

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_\ell, y_\ell)\}, \quad y_i \in \{+1, -1\} \tag{1.75}$$

es linealmente separable en el espacio de entrada, entonces existe un vector $\mathbf{w}$ y un escalar b que cumplen las desigualdades

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b \geq +1 \qquad & \text{Si } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 \qquad & \text{Si } y_i = -1 \end{aligned} \tag{1.76}$$

para todos los elementos del conjunto entrenamiento. Las desigualdades (1.76) se pueden reducir a la forma (Cortes & Vapnik, 1995):

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1. \tag{1.77}$$

La función del hiperplano óptimo se define como

$$D(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \tag{1.78}$$

donde $\mathbf{w}$ y $\mathbf{x}$ son vectores y b es el bias (Boser y col., 1992). El vector $\mathbf{w}$ que determina el hiperplano óptimo se describe como una combinación lineal de los vectores de soporte, estos son todos aquellos vectores que cumplen con $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$,

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i y_i \tag{1.79}$$

y $\hat{\mathbf{w}}$ determina su orientación (Cortes & Vapnik, 1995). El valor del bias

$$b = \langle y_i - \mathbf{w}^T\mathbf{x}_i \rangle \tag{1.80}$$

es el promedio los vectores de soporte (Guyon, Weston, Barnhill & Vapnik, 2002).

La distancia Euclidiana desde los patrones de entrenamiento de una clase al hiperplano óptimo se da por

$$\frac{|D(\mathbf{x})|}{\|\mathbf{w}\|} \tag{1.81}$$

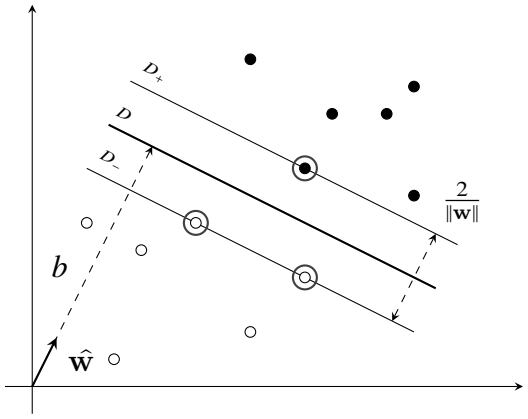


Figura 1.18: SVM con un hiperplano de margen máximo entre muestras de dos clases. D_+ y D_- son los hiperplanos creados para cada clase y D es el hiperplano óptimo de separación. Los vectores de soporte se encuentran rodeados.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

de modo que los datos de entrenamiento deben satisfacer la expresión

$$\frac{y_i D(\mathbf{x}_i)}{\|\mathbf{w}\|} \geq \rho, \quad i = 1, \ldots, \ell \tag{1.82}$$

donde ρ es el margen. Ya que existe un número infinito de soluciones, debemos establecer la siguiente restricción

$$\rho\|\mathbf{w}\| = 1 \tag{1.83}$$

de este modo el margen total entre los hiperplanos D_+ y D_- es

$$\rho = \frac{2}{\|\mathbf{w}\|}. \tag{1.84}$$

Por lo tanto, maximizar el margen ρ es equivalente a minimizar la norma $\|\mathbf{w}\|$. Entonces, se debe resolver el siguiente problema de optimización principal

$$\min \quad \|\mathbf{w}\|^2 \tag{1.85}$$

sujeto a la restricción (1.77). Para resolver el problema (1.85) se utiliza el método de los multiplicadores de Lagrange. Por lo que se obtiene la función

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{\ell} \alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right] \tag{1.86}$$

donde α son los multiplicadores de Lagrange. La solución óptima se da por el punto de

silla donde es minimizada respecto a $\mathbf{w}$ y b, y maximizada respecto a los multiplicadores de Lagrange. De las derivadas respecto de $\mathbf{w}$ y b se obtiene

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i y_i = 0 \tag{1.87}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{1.88}$$

de la ecuación (1.87) se obtiene

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i y_i \tag{1.89}$$

sustituyendo las ecuaciones (1.89) y (1.88) en la ecuación (1.86) se obtiene el problema de programación cuadrática (QP)

$$\max \quad W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{1.90}$$

sujeta a las restricciones

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0 \quad i = 1, \ldots, \ell \tag{1.91}$$

$$\alpha_i \geq 0 \quad i = 1, \ldots, \ell. \tag{1.92}$$

1.5.4 Hiperplano de separación en patrones no separables

En la mayoría de las aplicaciones prácticas no es posible crear un hiperplano de separación en el espacio de entrada (Schölkopf y col., 1997). En este caso se introducen las variables de holgura no negativas ξ_i para separar un conjunto irregular de entrenamiento con un número mínimo de errores (Cortes & Vapnik, 1995), como se muestra en la figura 1.19.

Para reducir las irregularidades en el conjunto de entrenamiento, actualmente se presentan dos esquemas: norma L_1 y norma L_2 (Howlett & Jain, 2001). Cortes y Vapnik (1995) proponen encontrar un subconjunto mínimo de errores en el conjunto de entrenamiento. Para esto se debe minimizar la función

$$\Phi = \sum_{i=1}^{\ell} \xi_i \tag{1.93}$$

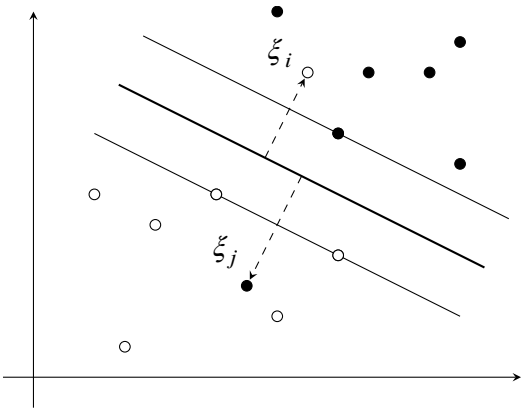


Figura 1.19: SVM con un hiperplano de margen flexible entre muestras de dos clases. Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

que corresponde a la norma L_1 y está sujeta a las restricciones

$$y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, \ell \tag{1.94}$$

$$\xi_i \geq 0 \qquad i = 1, \ldots, \ell \tag{1.95}$$

de modo que, al excluir este subconjunto del conjunto de entrenamiento, se puede construir un hiperplano óptimo de separación entre ambas clases. Esta idea se puede expresar formalmente como

$$\min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{\ell}\xi_i \tag{1.96}$$

sujeta a las restricciones (1.94) y (1.95). El parámetro C es una contante, controla la compensación entre el margen y las variables de holgura (Shawe–Taylor & Cristianini, 2004), como se observa en la figura 1.20. Aplicando los multiplicadores de Lagrange se obtiene la función

$$\begin{aligned} L(\mathbf{w}, b, \xi, \alpha, \beta) =& \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{\ell}\xi_i \\ & - \sum_{i=1}^{\ell}\alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i] - \sum_{i=1}^{\ell}\beta_i\xi_i \end{aligned} \tag{1.97}$$

donde α y β son los multiplicadores de Lagrange. De las derivadas respecto de $\mathbf{w}$, b y ξ se obtiene

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell}\alpha_i\mathbf{x}_i y_i = 0 \tag{1.98}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{\ell}\alpha_i y_i = 0 \tag{1.99}$$

$$\frac{\partial L}{\partial \xi} = C - \alpha_i - \beta_i = 0 \tag{1.100}$$

de las ecuaciones (1.98) y (1.100) se obtiene

$$w = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i \tag{1.101}$$

$$C = \alpha_i + \beta_i \tag{1.102}$$

sustituyendo las ecuaciones (1.101), (1.99) y (1.102) en la ecuación (1.97) se obtiene el problema de QP

$$\max \quad W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{1.103}$$

sujeta a las restricciones

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0 \quad i = 1, \ldots, \ell \tag{1.104}$$

$$\alpha_i \geq 0 \quad i = 1, \ldots, \ell. \tag{1.105}$$

En la norma L_2 se debe minimizar la función (Cortes & Vapnik, 1995)

$$\min \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^{\ell} \xi_i^2 \tag{1.106}$$

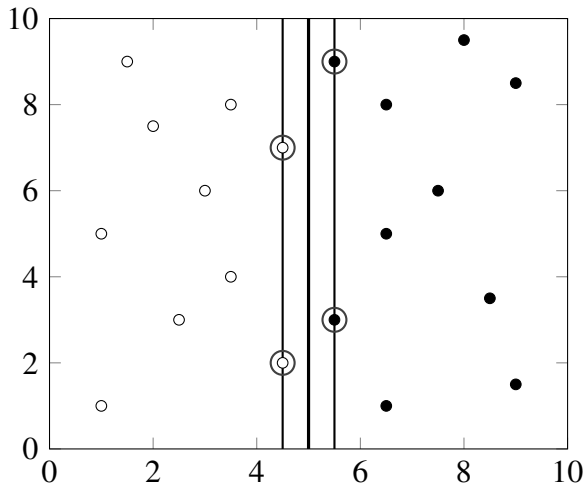
sujeta a las restricciones (1.94) y (1.95). Lo cual nos lleva al problema de QP (Howlett & Jain, 2001)

$$\max \quad W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{4C} \sum_{i=1}^{\ell} \alpha_1^2 \tag{1.107}$$

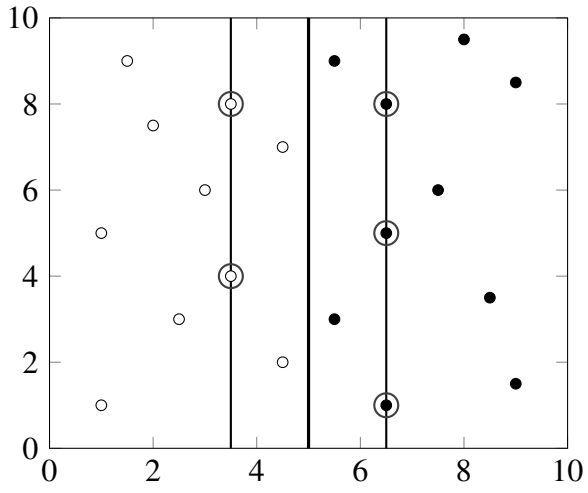
sujeta a las restricciones

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0 \quad i = 1, \ldots, \ell \tag{1.108}$$

$$\alpha_i \geq 0 \quad i = 1, \ldots, \ell. \tag{1.109}$$



(a)



(b)

Figura 1.20: El parámetro *C* controla la compensación entre el margen y las variables de holgura.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

1.5.5 Métodos de Kernel

Los algoritmos de aprendizaje automático han sido desarrollados para resolver problemas donde existe una separación lineal. Por otro lado, los problemas del mundo real requieren de métodos no lineales que permitan una generalización exitosa. Usando una función kernel se puede obtener una separación lineal y una predicción deseada. Estos métodos se han vuelto bastante populares ya que, a diferencia de otros, como las redes neuronales artificiales, tienen una fuerte inclinación hacia las matemáticas (Hofmann, Schölkopf & Smola, 2008).

De acuerdo con Chen, Lin y Schölkopf (2005) existen tres ventajas de llevar los vectores de entrada hacia un espacio de características:

1. Nos permite definir una medida de similitud a partir del producto punto en $\mathcal{H}$;
2. Nos permite tratar los patrones geométricamente y, por lo tanto, nos permite estudiar el algoritmo de aprendizaje utilizando álgebra lineal y geometría analítica.
3. La libertad de elegir la asignación ϕ nos permitirá diseñar una gran variedad de algoritmos de aprendizaje.

Cuando los patrones no son separables en el espacio de entrada, estos deben ser transformados en un *vector de características* mediante una función ϕ

$$\phi = \mathbb{R}^n \mapsto \mathbb{R}^N \tag{1.110}$$

donde n es la dimensión de entrada y N es la dimensión del espacio de características. Así, el hiperplano de separación se construirá sobre el conjunto de vectores transformados. La clasificación de un vector desconocido $\mathbf{x}$ realizada primero por la transformación del vector

$$\mathbf{x} \mapsto \phi(\mathbf{x}) \tag{1.111}$$

y después se toma el signo de la función

$$f(x) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{1.112}$$

donde

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \phi(\mathbf{x}_i). \tag{1.113}$$

Sustituyendo (1.112) en (1.113) obtenemos

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} y_i \alpha_i \phi(\mathbf{x})^T \phi(\mathbf{x}_i), \tag{1.114}$$

donde se puede observar que la clasificación del vector desconocido depende del producto punto (Cortes & Vapnik, 1995).

Aunque se podría usar el método principal, surgirán problemas si N es muy grande, lo que hace que la solución del sistema $N \times N$ de la ecuación (1.113) sea muy costosa. Si, consideramos la solución dual, toda la información que requiere el algoritmo son los productos internos entre los $\phi(x_i)$ y $\phi(x_j)$ en el espacio de características $\mathcal{H}$. La complejidad de evaluar cada producto interno es proporcional a la dimensión del espacio de características. Sin embargo, a veces, los productos internos se pueden calcular de manera más eficiente como una función directa de las características de entrada, sin calcular explícitamente el mapeo ϕ. En otras palabras, el paso de representación del vector de características se puede omitir. Esta función es conocida como *función kernel* (Shawe–Taylor & Cristianini, 2004).

Dada una función kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{1.115}$$

y un conjunto de entrenamiento se puede obtener una matriz Gram. Esta contiene la evaluación de la función kernel de todos los pares del conjunto (Shawe–Taylor & Cristianini, 2004)

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_m) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \cdots & k(\mathbf{x}_n, \mathbf{x}_m) \end{bmatrix}. \tag{1.116}$$

Toda función kernel debe cumplir con el teorema de Mercer para que pueda ser usada para construir un hiperplano de separación en algún espacio $\mathcal{H}$. El teorema define la forma general de los productos punto en espacios Hilbert (Vapnik, 1999).

Teorema 1 (Teorema de Mercer)**.** *La forma general del producto punto en un espacio Hilbert está definido por una función simétrica positiva $k(x, y)$ que satisface la condición*

$$\int k(x, y)\ \phi(x)\ \phi(y)\ \mathrm{d}x\ \mathrm{d}y \geq 0, \tag{1.117}$$

para todas las funciones ϕ tal que

$$\int \phi^2(x) \mathrm{d}x \leq \infty. \tag{1.118}$$

La función (1.115) se sustituye en el producto punto de la ecuación (1.90) de modo que

el problema se reescribe de la siguiente forma (Chapelle, Haffner & Vapnik, 1999)

$$\max \quad W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.119}$$

sujeta a las restricciones

$$\sum_{i=1}^{\ell} y_i \alpha_i = 0 \quad i = 1, \ldots, \ell \tag{1.120}$$

$$\alpha_i \geq 0 \quad i = 1, \ldots, \ell. \tag{1.121}$$

Con una elección adecuada del Kernel los datos pueden volverse separables en un espacio característico a pesar de no ser separable en el espacio de entrada original (Howlett & Jain, 2001).

Kernel lineal

Un conjunto de patrones es linealmente separable cuando el hiperplano logra separar los patrones sin error (Guyon y col., 2002). En este caso, el kernel lineal es válido ya que no es necesario transformar los patrones a una dimensión mayor (Vert, Tsuda & Schölkopf, 2004).

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j. \tag{1.122}$$

Kernel gaussiano

El kernel gaussiano es la función más utilizada y se ha estudiado ampliamente en campos vecinos (Shawe–Taylor & Cristianini, 2004). Se define como

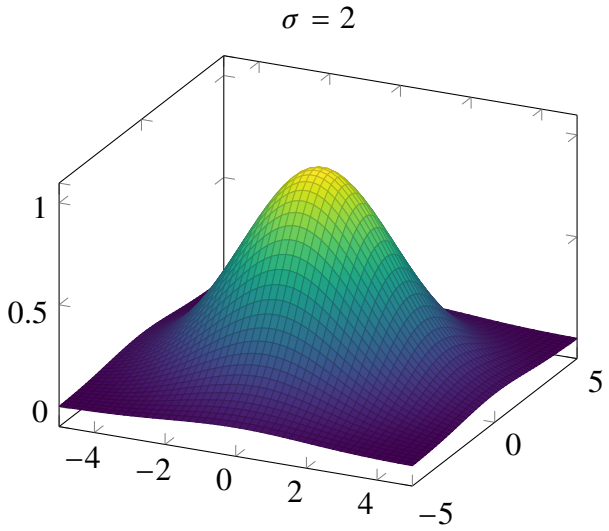
$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \tag{1.123}$$

donde

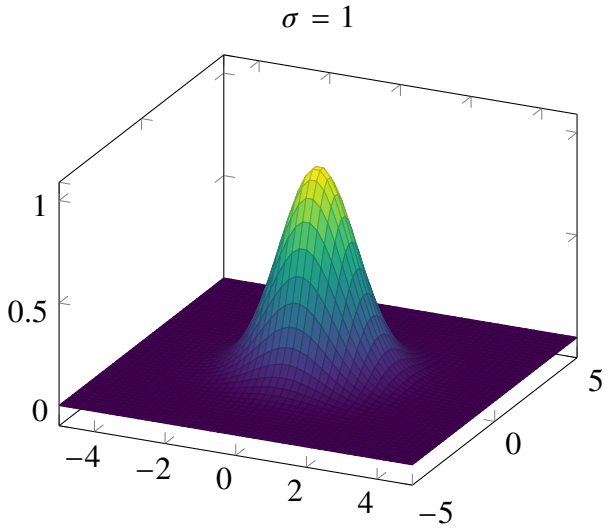
$$\gamma = \frac{1}{2\sigma^2} \tag{1.124}$$

y $\sigma > 0$ es un parámetro libre que determina la amplitud (ver figura 1.21) (Veropoulos, Campbell & Cristianini, 1999; Vapnik & Mukherjee, 2000).

Los valores pequeños de σ permiten que los clasificadores se ajusten a cualquier patrón, por lo tanto, existe riesgo de un sobreajuste. Por otro lado, valores grandes de σ reducen gradualmente el kernel a una función constante, haciendo imposible el aprendizaje de cualquier clasificador (Shawe–Taylor & Cristianini, 2004). Esto quiere decir que el kernel



(a)



(b)

Figura 1.21: Efecto del parámetro σ del kernel gaussiano. Los valores grandes de σ se aproximan a un clasificador lineal mientras que los valores pequeños de σ conducen al sobreajuste. Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

gaussiano se aproxima a un clasificador lineal (Boser y col., 1992).

Kernel polinomial

El kernel polinomial de grado d se describe como

$$k(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^d \tag{1.125}$$

donde d es el grado del polinomio. Cuando $d = 1$ el kernel es lineal (Boser y col., 1992; Weston, Chapelle, Elisseeff, Schölkopf & Vapnik, 2002). La dimensión del espacio de características para el kernel polinomial es

$$\binom{n+d}{d} \tag{1.126}$$

donde n es la dimensión de entrada y d la dimensión del espacio característico. Valores grandes de d permiten que los clasificadores se ajusten a cualquier patrón, por lo tanto, existe riesgo de un sobreajuste (Shawe–Taylor & Cristianini, 2004).

Por ejemplo, cuando $d = 2$ y $\mathbf{x} \in \mathbb{R}^2$ se obtiene un vector de seis dimensiones

$$k(\mathbf{x}_i, \mathbf{x}_j) = 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + 2x_{i1}x_{j1} + x_{i2}^2 x_{j2}^2 + 2x_{i2}x_{j2} \tag{1.127}$$

$$= \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j) \tag{1.128}$$

donde

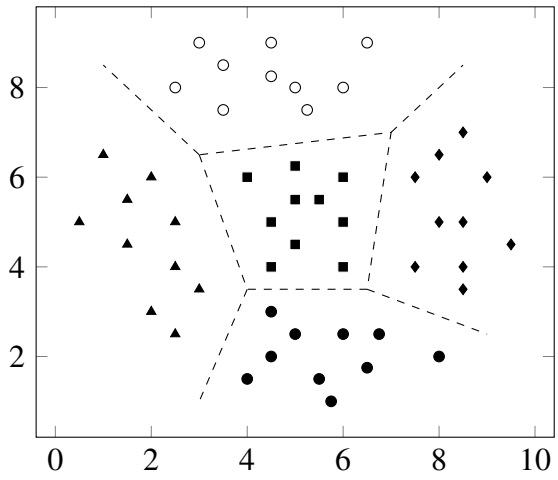
$$\boldsymbol{\phi}(\mathbf{x}_i) = 1 + x_{i1}^2 + \sqrt{2}x_{i1}x_{i2} + \sqrt{2}x_{i1} + x_{i2}^2 + \sqrt{2}x_{i2} \tag{1.129}$$

$$\boldsymbol{\phi}(\mathbf{x}_j) = 1 + x_{j1}^2 + \sqrt{2}x_{j1}x_{j2} + \sqrt{2}x_{j1} + x_{j2}^2 + \sqrt{2}x_{j2}. \tag{1.130}$$

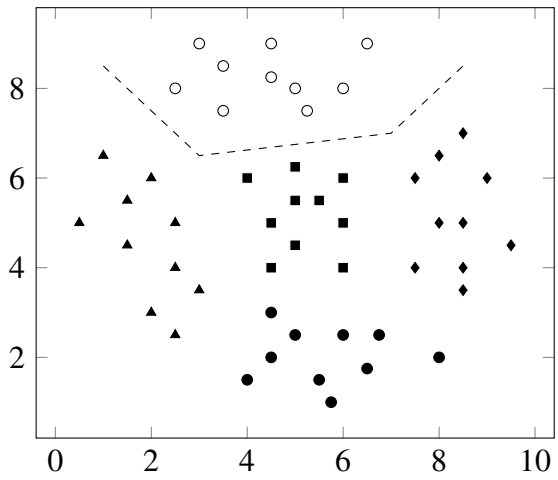
1.5.6 Códigos de Salida de Corrección de Errores

Muchos algoritmos de aprendizaje son clasificadores binarios a pesar de que muchos problemas del mundo real tienen múltiples clases. Estos algoritmos pueden ser aplicados mediante técnicas de binarización, lo que reduce un problema de aprendizaje multiclase en varios problemas de aprendizaje binarios (ver figura 1.22) que pueden ser resueltos separadamente. Después de que se hayan resuelto los problemas binarios, el conjunto resultante debe ser combinado de alguna manera (Allwein y col., 2000; Fürnkranz, 2002).

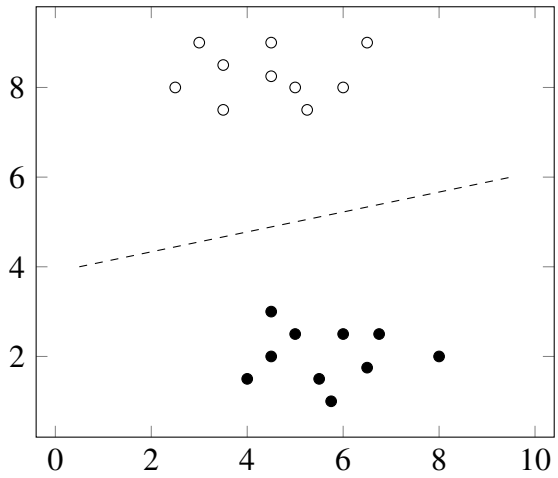
Los códigos de salida de corrección de errores (ECOC) son una técnica de binarización que emplea códigos de corrección de errores como una representación de salida distribuida. Cada clase es asociada a una cadena única de longitud n llamada *palabra código* asignada por un esquema de codificación. Entonces, se implementan n clasificadores binarios por cada



(a)



(b)



(c)

Figura 1.22: En la figura 1.22a se muestra un problema de clasificación de cinco clases. En la figura 1.22b se muestra la solución con el esquema de codificación *uno-vs-todos* y en la figura 1.22c se muestra la solución con el esquema de codificación *uno-vs-uno*.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

posición en la palabra código (Dietterich & Bakiri, 1995). La matriz resultante es llamada *matriz de codificación* (Fürnkranz, 2002).

Códigos exhaustivos

Dietterich y Bakiri (1995) recomiendan usar el esquema de códigos exhaustivos cuando el número de clases es $3 \leq k \leq 7$, donde se construyen $2^{k-1} - 1$ clasificadores usando el siguiente procedimiento:

1. Fila 1 Todos los elementos son 1;
2. Fila 2 Consiste en 2^{k-2} ceros seguido de $2^{k-2} - 1$ unos;
3. Fila 3 Consiste en 2^{k-3} ceros, 2^{k-3} unos, 2^{k-3} ceros y $2^{k-3} - 1$ unos;
4. Fila i Hay 2^{k-i} ceros y unos alternadamente.

En la tabla 1.1 se muestra la matriz resultante para un problema de cinco clases. En esta se observa que el esquema de códigos exhaustivos 15 clasificadores binarios para dar solución a cinco clases.

Tabla 1.1
Matriz de codificación para un problema de cinco clases diseñado con códigos exhaustivos.

Clases	Palabra código														
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fuente: Adaptada de "Solving Multiclass Learning Problems via Error-Correcting Output Codes" de Dietterich y Bakiri, 1995.

Distancia de Hamming

La distancia de Hamming de dos palabras código es el número de posiciones en las cuales son dispares (Hamming, 1950). La distancia se formula aplicando el operador lógico XOR (Wikipédia, 2018)

$$d(a,b) = \sum_{i=0}^{n-1} (a_i \oplus b_i) \tag{1.131}$$

Por ejemplo, aplicando el operador lógico XOR a las palabras `01010000 01010011 01010101 01001010` y `01001100 01000001 01001111 01000101`, obtenemos una distancia de Hamming igual a 12.

```
01010000 01010011 01010101 01001010
01001100 01000001 01001111 01000101
-------- -------- -------- --------
00011100 00010010 00011010 00001111
```

La distancia de Hamming tiene una interpretación geométrica mediante un cubo de n-dimensiones, como se observa en la figura 1.23. Una posición de la palabra corresponde a una coordenada. Así, la distancia entre dos puntos, es el número de coordenadas en las que son diferentes (Hamming, 1950).

Diseño de códigos

Los ECOC se definen en una matriz donde una fila es una palabra código y el número de filas es el número de clases. La longitud de la palabra código es el número de columnas y el número de clasificadores binarios (Dietterich & Bakiri, 1995).

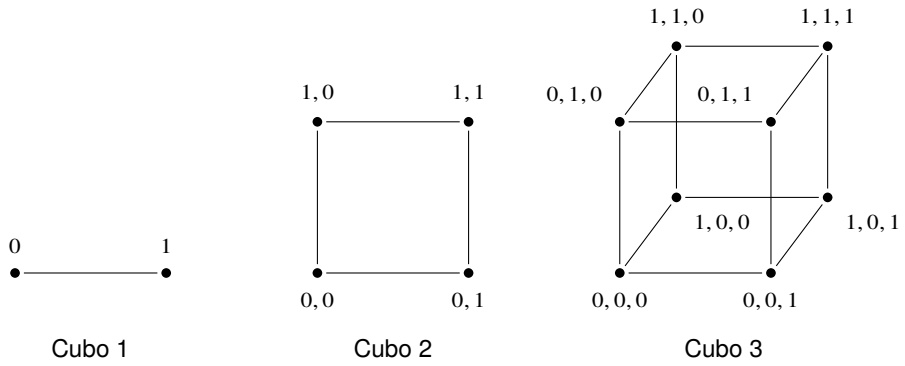


Figura 1.23: Los cubos *n*-dimensional son la interpretación geométrica de la distancia de Hamming.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

La capacidad de un código para corregir errores está directamente relacionado a la separación de las filas (clases). Por lo que los ECOCs son ventajoso para un problema de k-clases debe tener una separación óptima en distancia de Hamming (Dietterich & Bakiri, 1995). En la figura 1.24a en la página 59 se muestra el crecimiento del número de clasificadores binarios requeridos para k-clases. Y en la figura 1.24b en la página 59 se ilustra la distancia Hamming entre k-clases.

Códigos binarios

Mediante el uso códigos binarios, la matriz resultante toma la siguiente forma

$$\mathbf{M} \in \{+1, -1\}^{k \times \ell}. \tag{1.132}$$

Uno *vs* todos Es el diseño más simple ya que crea un problema binario por cada clase (Allwein y col., 2000). Por lo tanto, origina k problemas binarios y mantiene una distancia mínima entre cada par de clases de 2 (The MathWorks, Inc, 2014).

$$\mathbf{M} = \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix}$$

Ordinal Origina $k - 1$ problemas binarios. Mantiene una distancia mínima entre cada par de clases de 1. Para el primer clasificador binario, la primera clase es negativa y el resto positiva. Para el segundo clasificador binario, las dos primeras clases son negativas, el resto positivas, Para el tercer clasificador binario, las tres primeras clases son negativas, el resto positivas y así sucesivamente (The MathWorks, Inc, 2014).

$$\mathbf{M} = \begin{pmatrix} -1 & -1 \\ +1 & -1 \\ +1 & +1 \end{pmatrix}$$

Binario completo Este diseño divide las clases en todas las combinaciones binarias y no ignora ninguna clase. Para cada problema binario, todas las asignaciones de clase son -1 y 1 con al menos una clase positiva y negativa. Origina $2^{k-1} - 1$ problemas binarios. Mantiene una distancia mínima entre cada par de clases de 2^{k-2} (The MathWorks, Inc, 2014).

$$\mathbf{M} = \begin{pmatrix} +1 & +1 & +1 \\ +1 & -1 & -1 \\ -1 & +1 & -1 \end{pmatrix}$$

Denso al azar Origina aproximadamente $10 \lg k$ problemas binarios. La distancia entre cada par de clases es variable (The MathWorks, Inc, 2014).

Códigos ternarios

Allwein y col. (2000) añadieron el cero a la matriz, indica que no se tomaran en cuenta los patrones que corresponden a dicha clase. Así, mediante códigos ternarios, la matriz resultante toma la siguiente forma

$$\mathbf{M} \in \{+1, 0, -1\}^{k \times \ell}. \tag{1.133}$$

Uno *vs* uno Para cada problema binario una clase es positiva, otra negativa y las clases restantes son ignoradas. Origina

$$\frac{k(k-1)}{2} \tag{1.134}$$

problemas binarios. Mantiene una distancia mínima entre cada par de clases de 1 (The MathWorks, Inc, 2014).

$$\mathbf{M} = \begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix}$$

Ternario completo Divide las clases en todas las combinaciones ternarias. Todas las asignaciones de clase son 0, -1 y 1 con al menos una clase positiva y una negativa. Origina

$$\frac{3^k - 2^{k+1} + 1}{2} \tag{1.135}$$

problemas binarios. Mantiene una distancia mínima entre cada par de clases de 3^{k-2} (The MathWorks, Inc, 2014).

$$\mathbf{M} = \begin{pmatrix} -1 & -1 & 0 & +1 & -1 & -1 \\ +1 & -1 & -1 & -1 & 0 & +1 \\ 0 & +1 & +1 & +1 & +1 & +1 \end{pmatrix}$$

En comparación con otros diseños de codificación, en el diseño ternario completo, el número de clasificadores generados es significativamente más alto y la fase de entrenamiento es más lenta en la práctica (Joutsijoki, Haponen, Rasku, Aalto-Setälä & Juhola, 2016).

Disperso al azar Origina aproximadamente 15 lg k problemas binarios. La distancia entre cada par de clases es variable (The MathWorks, Inc, 2014).

En la figura 1.24 se muestra el crecimiento del número de clasificadores binarios y la distancia de Hamming para k-clases para los diferentes diseños de códigos.

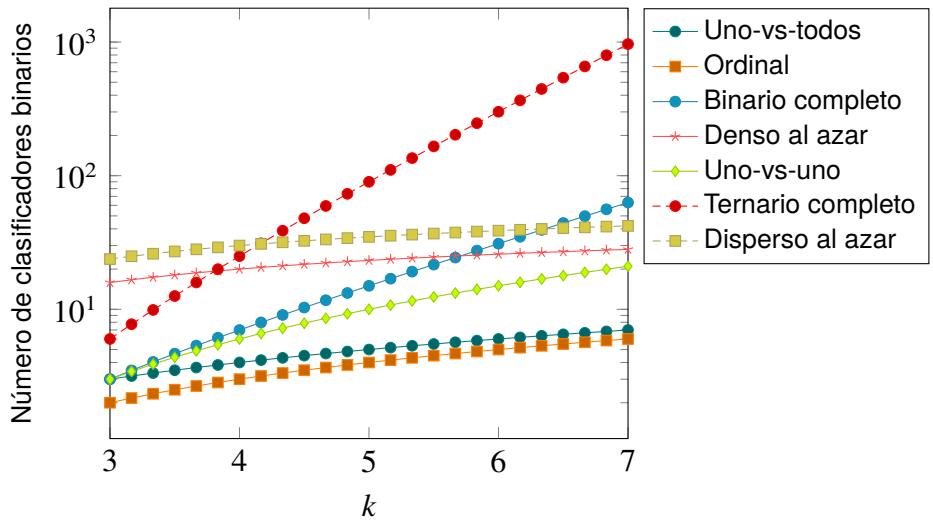
Decisiones de clasificación

Algunos métodos de agrupamiento construyen múltiples hipótesis las cuales votan para determinar la clasificación de un patrón. Un código de corrección de errores puede verse como una forma de votación en la que se puede corregir un cierto número de votos (Dietterich & Bakiri, 1995).

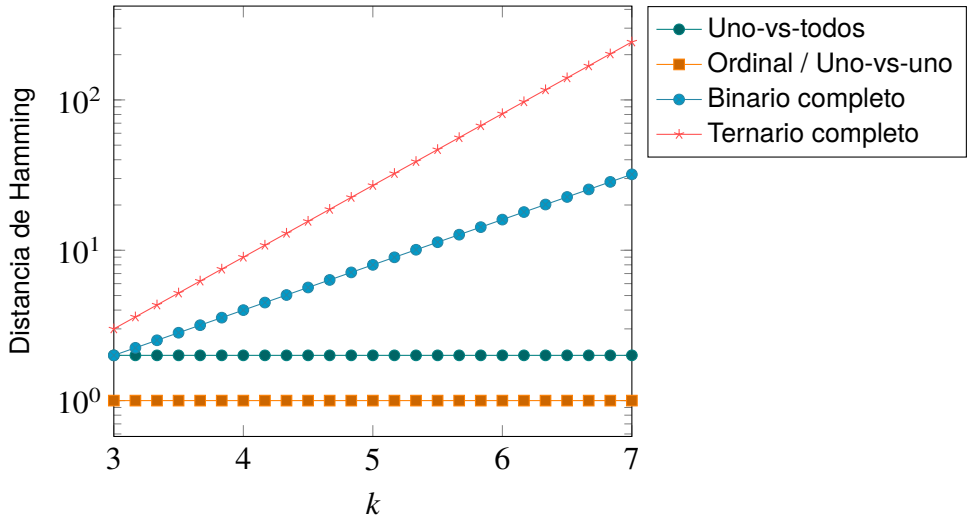
Cada salida puede ser vista como el cálculo de la probabilidad de que su bit correspondiente en la palabra código sea 1. Estos valores de probabilidad son $b = \{b_1, b_2, \dots, b_n\}$ donde n es la longitud de la palabra código. Para clasificar un nuevo patrón, se calcula la distancia entre el vector b y cada una de las palabras código w_i. La distancia L^1 entre b y w_i) se define como

$$L^1(b, w_i) = \sum_{j=0}^{L} |b_j - w_{i,j}|. \tag{1.136}$$

La palabra código que tenga la distancia más corta entre L^1 y B es asignada como la clase del nuevo patrón (Dietterich & Bakiri, 1995).



(a)



(b)

Figura 1.24: En la figura 1.24a se muestra el crecimiento de número de clasificadores binarios en una escala logarítmica en base 10 (The MathWorks, Inc, 2014). Y en la figura 1.24b se presenta el crecimiento de la distancia mínima de Hamming entre k-clases. La distancia entre los códigos denso al azar y disperso al azar es variable.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

1.5.7 Validación Cruzada

Para construir modelos con un buen desempeño, se requiere utilizar la mayor cantidad posible de datos en el entrenamiento. Sin embargo, en muchas aplicaciones, la cantidad de datos disponibles está limitada. Por lo que los conjuntos de entrenamiento y de pruebas estarán limitados. Una solución para determinar un buen modelo es el uso de la *validación cruzada* (Bishop, 2006).

La *validación cruzada* es un método para seleccionar modelos de acuerdo a su desempeño. El conjunto de datos se divide en dos partes. La primer parte (conjunto de entrenamiento) es usada para ajustar un modelo, mientras que la segunda parte (conjunto de validación) evalúa el desempeño de predicción (Shao, 1993).

Algunas técnicas de validación cruzada son el método de retención, validación cruzada de k-iteraciones y validación cruzada dejando uno fuera.

Método de retención

Divide el conjunto en dos subconjuntos. El conjunto de entrenamiento y el conjunto de pruebas también llamado conjunto retenido (ver figura 1.25). Este método, garantiza con alta probabilidad que la estimación del error estará cerca del error verdadero (Blum, Kalai & Langford, 1999).

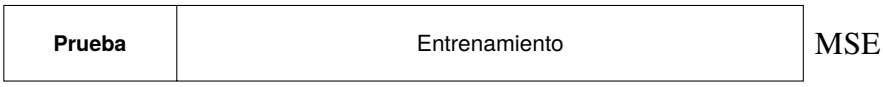


Figura 1.25: Método de retención.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

Validación cruzada de k-iteraciones

Divide el conjunto de entrenamiento en k subconjuntos aproximadamente del mismo tamaño (ver figura 1.26) (Kohavi, 1995; James y col., 2013).

La primera división se usa como el conjunto de validación y las $k-1$ divisiones restantes se usan para el entrenamiento. Así, el procedimiento resulta en k pruebas de error. Le estimación de la validación cruzada se calcula por el promedio de los valores de error (James y col., 2013)

$$\mathrm{CV}_k = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i. \tag{1.137}$$

Figura 1.26: Esquema de validación cruzada con 5 divisiones.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

Validación cruzada dejando uno fuera

Consiste en eliminar un elemento del conjunto de entrenamiento para construir la regla de decisión sobre la base de los datos de entrenamiento restantes y luego probar el elemento eliminado (ver figura 1.27). De esta manera, uno prueba todos los elementos del conjunto de entrenamiento usando ℓ reglas de decisión diferentes, donde ℓ es el número de datos (Chapelle & Vapnik, 2000).

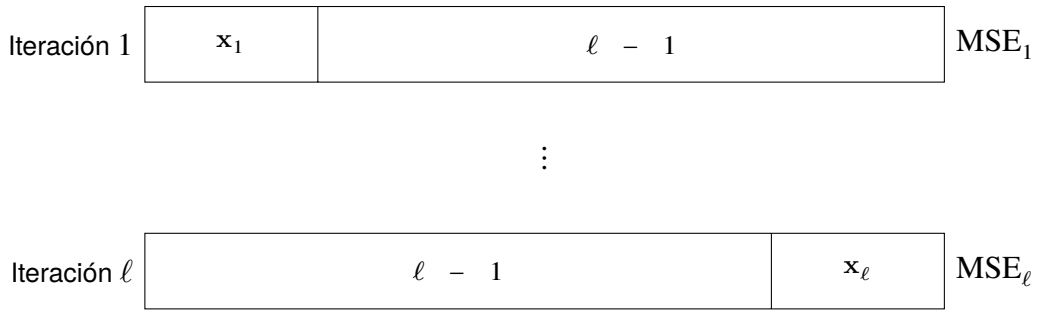


Figura 1.27: Validación cruzada dejando uno fuera.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

1.5.8 Optimización de Hiperparámetros

Los algoritmos de aprendizaje automático con frecuencia requieren un ajuste cuidadoso de los parámetros del modelo. Desafortunadamente, estos ajustes requieren de experiencia, reglas del pulgar no escritas o, a veces, búsqueda de fuerza bruta (Snoek, Larochelle & Adams, 2012).

Las máquinas de soporte vectorial y los métodos kernel dependen de varios parámetros. El parámetro C, controla la compensación entre la maximización de márgenes y la minimización de errores. Los parámetros d y σ corresponden a las funciones kernel polinomial y al kernel base radial respectivamente (Chapelle, Vapnik, Bousquet & Mukherjee, 2002). En general, a los parámetros libres se les conoce como *hiperparámetros* (Rasmussen & Williams, 2006).

Es bien sabido que el desempeño de las máquinas de soporte vectorial depende de un ajuste cuidadoso de los hiperparámetros (Cherkassky & Ma, 2004).

Optimización bayesiana

La optimización bayesiana es una estrategia de optimización global de funciones de *caja negra*. Gran parte de la eficiencia se deriva de la capacidad para incorporar ideas previas sobre el problema para ayudar a dirigir el muestreo y compensar la exploración y explotación del espacio de búsqueda (Brochu, Cora & de Freitas, 2010). El objetivo es encontrar la configuración correcta de los hiperparámetros tan rápido como sea posible (Snoek y col., 2012).

Un estudio realizado por Snoek y col. (2012), demostró la efectividad de la optimización bayesiana en problemas que abarcan diferentes áreas del aprendizaje automático obteniendo mejores hiperparámetros significativamente más rápido que otros enfoques.

Búsqueda en malla

La búsqueda en malla es una estrategia ampliamente usada gracias a su simplicidad en su implementación (Bergstra & Bengio, 2012). Sin embargo, este método es potencialmente impráctico, ya que la evaluación de una combinación de hiperparámetros más específicos puede requerir mucho tiempo (Czogiel, Luebke & Weihs, 2006).

Por ejemplo, si se consideran dos parámetros, γ y C y cada uno toma sus valores de un conjunto de candidatos, donde n_γ y n_C es el número de valores, se realizará una búsqueda en malla de $n_\gamma n_C$ por lo que se observa que existe un crecimiento exponencial en la búsqueda (Vert y col., 2004; Hsu, Chang & Lin, 2003).

Hay dos puntos importantes para considerar la búsqueda en malla a pesar del costo computacional (Hsu y col., 2003):

1. Psicológicamente no podemos sentirnos seguros de usar métodos que eviten realizar una búsqueda exhaustiva de los hiperparámetros.

2. El tiempo de cálculo para encontrar buenos parámetros mediante la búsqueda en malla no es mucho mayor que con los métodos avanzados ya que puede ser fácilmente paralelizada porque cada hiperparémetro es independiente.

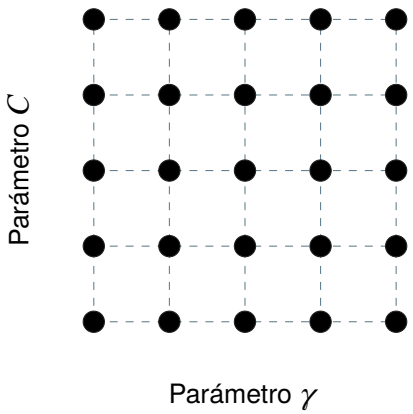


Figura 1.28: Búsqueda en malla de los hiperparámetros C y γ, donde $n_\gamma = n_C = 5$, en la que se evalúan las 25 posibles combinaciones.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

Búsqueda aleatoria

Bergstra y Bengio (2012) demostraron que la búsqueda aleatoria es más eficaz para optimizar los hiperparámetros que la búsqueda en malla. Los autores señalan que encontraron mejores modelos en la mayoría de los casos y requirió menos tiempo de cómputo. También, los experimentos aleatorios son más fáciles de realizar que los experimentos en malla por razones prácticas relacionadas con la independencia estadística de cada prueba.

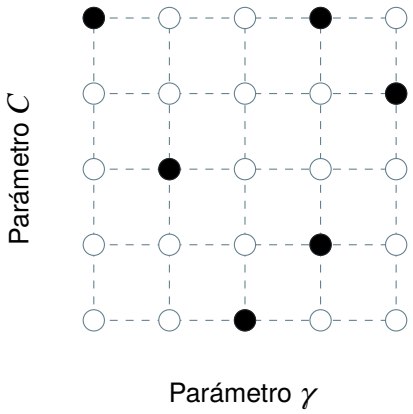


Figura 1.29: Búsqueda aleatoria de los hiperparámetros C y γ, donde $n_\gamma = n_C = 5$, en la que se seleccionan seis combinaciones para evaluar de las 25 posibles.
Fuente: Adaptado de "Support-Vector Machine with TikZ" de Ochoa, 2019c.

A fin de obtener la mejor arquitectura para cada base de datos propuesta, se deben analizar los resultados obtenidos de cada uno de los experimentos mediante el uso de una matriz de confusión que permite visualizar el desempeño de las redes neuronales superficiales y las máquinas de soporte vectorial. Esta técnica se describe en el siguiente capítulo.

Capítulo 2

Marco Metodológico

2.1 Pruebas y Validación de un Modelo de Aprendizaje

Todo clasificador debe ser evaluado y seleccionado en función de su desempeño (Bradley, 1997). Cuando este es capaz de funcionar correctamente sobre un nuevo conjunto de patrones, se dice que el clasificador generaliza correctamente (Foresee & Hagan, 1997). La generalización es un concepto tomado de la psicología, "indica la capacidad de reaccionar ante nuevas situaciones cuando estas se parecen a otras ya aprendidas y que se volvieron familiares" (Galimberti, 2006, p. 525). Acorde con Riedmiller (1994), todo algoritmo de aprendizaje debe cumplir con al menos los siguientes requisitos:

1. Rápida convergencia;
2. Facilidad de elección de parámetros;
3. Buena capacidad de generalización sobre patrones desconocidos.

Cuando se crean distintos modelos (clasificadores), debemos tener en mente un método de selección que permita estimar el desempeño de estos para seleccionar el mejor. Y un método de evaluación para estimar el error de predicción, también llamado error de generalización, sobre un conjunto nuevo de patrones. Es por esto que se recomienda dividir el conjunto de patrones en tres parte, ver figura 2.1 (Hastie y col., 2009):

Entrenamiento El conjunto de entrenamiento es usado para ajustar los modelos.

Validación El conjunto de validación es usado para estimar el error de predicción para la selección de un modelo.

Prueba El conjunto de prueba es usado para evaluar el error de generalización del modelo final elegido.

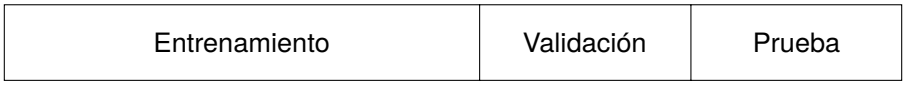


Figura 2.1: Esquema general de división de un conjunto de entrenamiento para validación y prueba. Una división típica puede ser 70% para el entrenamiento y 15% para validación y pruebas.
Fuente: Adaptado de "Machine Learning wih TikZ" de Ochoa, 2019a.

2.1.1 Matriz de confusión

Una *matriz de confusión*, también llamada tabla de contingencia, es un modo de tabular estadísticas para evaluar la calidad de un clasificador (Flach, 2003). Esta resume la exactitud de clasificación respecto a algunos datos de prueba de un clasificador. Considerando un problema de clasificación binaria, se crea una matriz donde se asignan los valores *verdadero positivo* (TP), *falso positivo* (FP), *verdadero negativo* (TN) y *falso negativo* (FN) (Veropoulos y col., 1999), como se muestra en la figura 2.2.

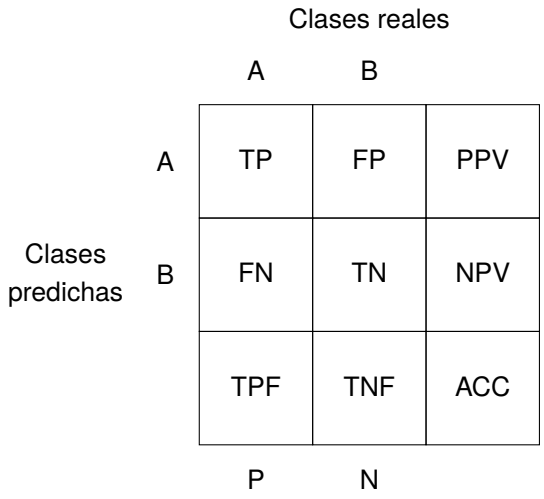


Figura 2.2: Matriz de confusión, con dos clases, A y B. La clase A es designada a la clase *positiva*; la clase B, a la clase *negativa*. P es el número total de datos que corresponden a la clase positiva; N, a la clase negativa.
Fuente: Adaptado de "Machine Learning wih TikZ" de Ochoa, 2019a.

Las columnas corresponden a las clasificaciones reales; las filas, a las clasificaciones predichas. Por lo que TP se refiere al número de clasificaciones correctas; FP, al número de clasificaciones incorrectas para la clase A. Del mismo modo, TN se refiere al número de clasificaciones correctas; FN, al número de clasificaciones incorrectas para la clase B (Bradley, 1997).

El desempeño de un clasificador es generalmente cuantificado por su precisión durante el proceso de prueba. Sin embargo, el desempeño de dicho sistema se describe mejor en

términos de su sensibilidad y especificidad que cuantifican su desempeño para falsos positivos y falsos negativos (Veropoulos y col., 1999).

Exactitud Es el radio entre el número de datos clasificados correctamente y el tamaño del conjunto de prueba (Veropoulos y col., 1999). Aunque la exactitud proporciona un solo simple número para evaluar el desempeño, a menudo es demasiado simple y debe interpretarse con considerable precaución (Metz, 1978).

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.1}$$

$$\mathrm{ERROR} = 1 - \mathrm{ACC} \tag{2.2}$$

Sensibilidad Es la relación entre los números de datos positivos clasificados correctamente TP y el número de ejemplares positivos (Veropoulos y col., 1999).

$$\mathrm{TPF} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.3}$$

$$\mathrm{FNF} = 1 - \mathrm{TPF} \tag{2.4}$$

Especificidad Es la relación entre los números de datos negativos clasificados correctamente TN y el número de ejemplares negativos (Veropoulos y col., 1999).

$$\mathrm{TNF} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{2.5}$$

$$\mathrm{FPF} = 1 - \mathrm{TNF} \tag{2.6}$$

Valores predictivos positivos (Precisión) Es el radio de datos positivos correctos del número de datos positivos actuales (Bradley, 1997).

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.7}$$

Valores predictivos negativos Es el radio de datos negativos correctos del número de datos negativos actuales (Bradley, 1997).

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} \tag{2.8}$$

Puntaje F Es la media armónica entre la precisión y la sensibilidad (Sasaki, 2007). Propor-

ciona una medida única del desempeño de los clasificadores (Chinchor, 1992).

$$F_1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}} \tag{2.9}$$

Si todos los casos fueron evaluados como positivos o negativos, el número de decisiones correctas más el número de decisiones incorrectas debe ser igual al número de casos reales. Por lo tanto, es fácil mostrar que las diversas fracciones definidas anteriormente deben estar relacionadas por

$$\text{TPF} + \text{FNF} = 1$$
$$\text{TNF} + \text{FPF} = 1.$$

Tanto FNF como FPF representan las fracciones de las clases positivas y negativas respectivamente que fueron clasificadas incorrectamente (Metz, 1978).

2.1.2 Curva ROC

La curva ROC (ver figura 2.3) describe la relación entre TPF y FPF (Metz, 1978). Es útil al visualizar el desempeño de un algoritmo de clasificación ya que el umbral de decisión varía (Metz, 1978; Bradley, 1997). El área bajo la curva ROC (AUC) determina el desempeño de un clasificador. De modo que el clasificador con mayor área tendrá el mejor desempeño (Bradley, 1997).

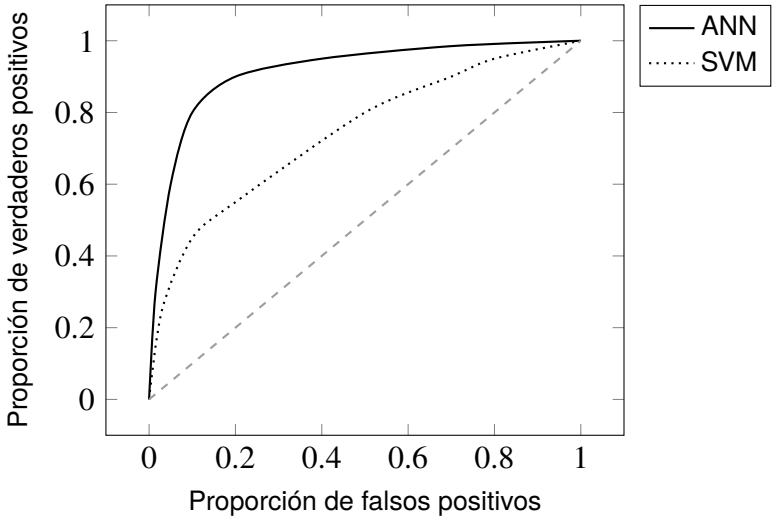


Figura 2.3: Curva ROC.
Fuente: Adaptado de "Machine Learning wih TikZ" de Ochoa, 2019a.

En la curva ROC es importante notar algunos puntos de la gráfica. El punto inferior izquierdo $(0, 0)$ representa la estrategia de nunca emitir una clasificación positiva; dicho clasificador no comete errores falsos positivos, pero tampoco obtiene verdaderos positivos. La estrategia opuesta, de emitir clasificaciones positivas, está representada por el punto superior derecho $(1, 1)$. El punto $(0, 1)$ representa la clasificación perfecta. La línea diagonal representa la estrategia de adivinar aleatoriamente una clase. Los puntos debajo de la diagonal representan la peor clasificación, los puntos encima de la diagonal representan la mejor clasificación. Cuando los puntos están sobre la diagonal, se dice que el clasificador no tiene información sobre dicha clase (Fawcett, 2006).

2.2 Tiempo de ejecución de los algoritmos

Cuando se trata de medir la eficiencia de un algoritmo, es muy útil saber cuánto tiempo tarda en realizar una tarea específica.

Para evaluar el tiempo transcurrido se ha utilizado la información generada por MATLAB durante los procesos de entrenamiento.

2.3 Análisis Estadístico

Existen múltiples algoritmos que resuelven un mismo problema y necesitamos elegir el mejor entre todos ellos. Las pruebas estadísticas permiten evaluar las *diferencias significativas* entre las medias de los grupos a probar.

La prueba de Kruskal–Wallis, también llamada prueba H de Kruskal–Wallis, se utiliza para probar la hipótesis nula H_0 de que k muestras independientes provienen de poblaciones idénticas. Para ello se calcula

$$h = \frac{12}{n(n+1)} \sum_{i=1}^{k} \frac{r_i^2}{n_i} - 3(n+1) \tag{2.10}$$

donde n es el número total de observaciones entre todas las muestras, n_i es el número de observaciones en la muestra i y r_i es la suma de los rangos de la muestra i. Si h cae en la región crítica $H > \chi_\alpha^2$ con $v = k - 1$ grados de libertad, se rechaza H_0 al nivel de significancia α; de lo contrario no se rechaza H_0. Para determinar la región crítica se utiliza la tabla B.1 del anexo B en la página 95. Esta toma el valor de intersección de los grados de libertad y el valor del nivel de significancia (Walpole, Myers & Myers, 2012).

La prueba de Kruskal–Wallis compara las medias de varios grupos para probar la hipótesis de que todas son iguales, frente a la hipótesis alternativa general de que no todas son

iguales. Esta alternativa puede ser demasiado general. Por lo que es posible que se requiera información sobre qué pares de medias son significativamente diferentes y cuáles no. Una prueba de comparación múltiple puede proporcionar esta información (The MathWorks, Inc, 2006b).

Los resultados obtenidos de cada una de las bases de datos son analizados mediante el análisis de varianza de una vía de Kruskal–Wallis y la prueba de comparación múltiple con un nivel de significación con valor de 0.05. Los resultados para determinar las diferencias estadísticas se describen en el siguiente capítulo.

Capítulo 3

Análisis de Resultados y Conclusiones

Se realizaron dos experimentos empleando redes neuronales superficiales y códigos de salida de corrección de errores como modelo de ajuste para las máquinas de soporte vectorial sobre seis bases de datos multiclase tomadas de "UCI Machine Learning Repository". El objetivo es comparar el desempeño de ambos algoritmos de clasificación y el tiempo que demora en el entrenamiento cada algoritmo.

En la tabla 3.1 se describe la división de las seis bases de datos para realizar el experimento. En la primera columna se especifica el número total de ejemplares que conforman las bases de datos. La segunda y tercer columna corresponde a la división del total de ejemplares para realizar en entrenamiento y las pruebas de los modelos creados respectivamente.

El experimento se llevó a cabo con el software matemático MATLAB R2018b sobre el sistema operativo Debian 9 con un procesador Intel i7-4770 de 3.40GHz (cuatro núcleos), memoria ram de 8GB de 1600MHz y una unidad de procesamiento gráfico NVIDIA GeForce GT 635.

Tabla 3.1
División de los conjuntos de entrenamiento.

Base de datos	Ejemplares	Entrenamiento	Pruebas
Vino	178	—	—
Identificación de vidrio	214	—	—
Transbordador espacial	58000	43500	14500
Levadura	1484	—	—
Reconocimiento de letras habladas	7797	5458	2339
Reconocimiento de letras	20000	16000	4000

Fuente: Ochoa, L. A. (2019).

3.1 Análisis de desempeño

El experimento se repitió diez veces. Por lo que los resultados presentados en la tabla 3.2 son el valor promedio del desempeño obtenido de los experimentos realizados.

Tabla 3.2
Promedio del desempeño.

	Desempeño	
Base de datos	SNN	ECOC/SVM's
Vino	99.6067%	**100%**
Identificación de vidrio	**93.7850%**	83.6449%
Transbordador espacial	99.2439%	**99.8828%**
Levadura	**86.1388%**	62.5674%
Reconocimiento de letras habladas	**98.2503%**	95.5537%
Reconocimiento de letras	94.2486%	**95.7167%**

Nota. Los resultados con mayor exactitud están remarcados.
Fuente: Ochoa, L. A. (2019).

El valor estadístico de la prueba de Kruskal–Wallis toma el valor $h = 0$. Ya que el valor obtenido no cae región crítica de 3.841 para un grado de libertad, no se rechaza la hipótesis nula. Lo que quiere decir que no hay evidencia para concluir que existe una diferencia estadísticamente significativa entre el desempeño de ambas técnicas de clasificación. La tabla 3.3 resume el análisis de desempeño obtenido.

Tabla 3.3
Prueba de Kruskal–Wallis: Análisis de desempeño.

Fuente	Suma de Cuadrados	Grados de Libertad	Error Cuadrático Medio	χ^2	Prov > χ^2
Grupos	0	1	0	0	1
Error	143	10	14.3		
Total	143	11			

Fuente: Ochoa, L. A. (2019).

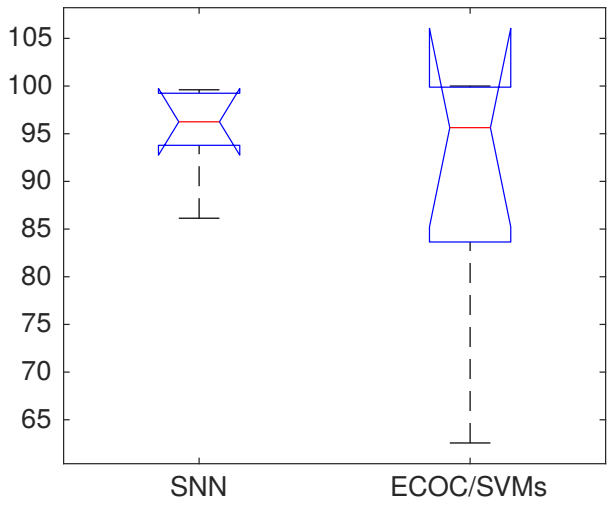


Figura 3.1: Diagrama de cajas del análisis de desempeño.
Fuente: Ochoa, L. A. (2019).

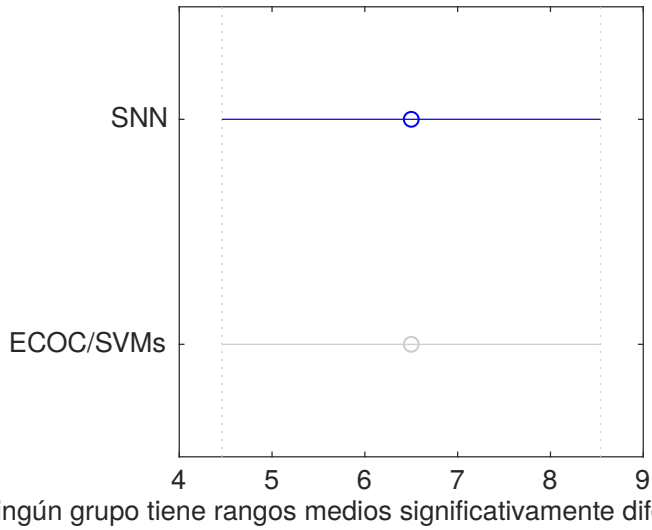


Figura 3.2: Mediante la prueba de análisis de comparación múltiple con un nivel de significación con valor de 0.05 es fácil observar que las medias de los valores obtenidos se encuentran dentro de los rangos logrados por ambos clasificadores. Lo cual establece que no existen diferencias estadísticamente significativas entre ambos algoritmos.
Fuente: Ochoa, L. A. (2019).

3.2 Análisis de tiempo transcurrido durante el proceso de entrenamiento

En la tabla 3.4 se tabula el tiempo promedio en segundos durante el proceso de entrenamiento proporcionado por MATLAB.

Tabla 3.4
Promedio del tiempo transcurrido en segundos durante el proceso de entrenamiento.

	Tiempo transcurrido	
Base de datos	SNN	ECOC/SVM's
Vino	**0.0493**	0.4506
Identificación de vidrio	52.2682	**2.6332**
Transbordador espacial	**5.3029**	323.1063
Levadura	839.3090	**22.8478**
Reconocimiento de letras habladas	**7.7460**	13.1436
Reconocimiento de letras	**7.3075**	41.2276

Nota. Los resultados con menor tiempo de ejecución están remarcados.
Fuente: Ochoa, L. A. (2019).

El valor estadístico de la prueba de Kruskal–Wallis toma el valor $h = 0.0256$. Ya que el valor obtenido no cae región crítica de 3.841 para un grado de libertad, no se rechaza la hipótesis nula. Lo que quiere decir que no hay evidencia para concluir que existe una diferencia estadísticamente significativa entre el tiempo durante el proceso de entrenamiento entre ambas técnicas de clasificación. La tabla 3.5 resume el análisis del tiempo en segundos obtenido en el proceso de entrenamiento.

Tabla 3.5
Prueba de Kruskal–Wallis: Análisis del tiempo en segundos transcurrido en el proceso de entrenamiento.

Fuente	Suma de Cuadrados	Grados de Libertad	Error Cuadrático Medio	χ^2	Prov > χ^2
Grupos	0.3333	1	0.3333	0.0256	0.8728
Error	142.6667	10	14.2667		
Total	143	11			

Fuente: Ochoa, L. A. (2019).

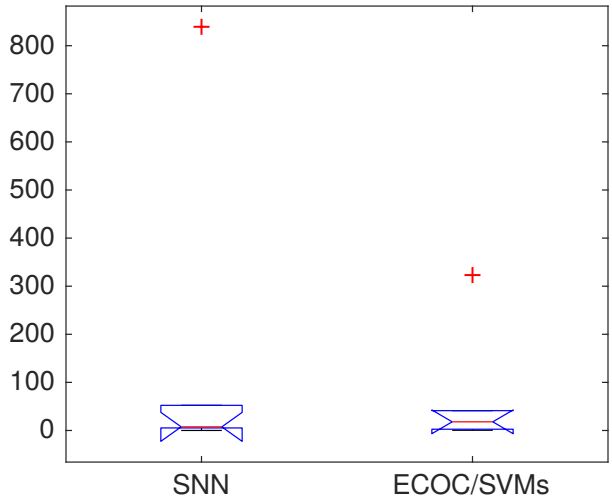


Figura 3.3: Diagrama de cajas del análisis de tiempo transcurrido en el proceso de entrenamiento.
Fuente: Ochoa, L. A. (2019).

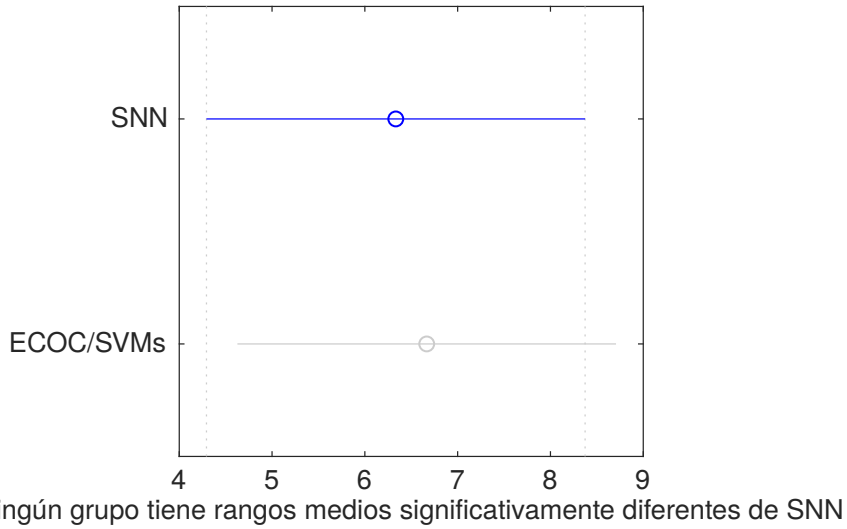


Figura 3.4: Mediante la prueba de análisis de comparación múltiple con un nivel de significación con valor de 0.05 es fácil observar que las medias de los valores obtenidos se encuentran dentro de los rangos logrados por ambos clasificadores. Lo cual establece que no existen diferencias estadísticamente significativas entre ambos algoritmos.
Fuente: Ochoa, L. A. (2019).

En las tablas 3.6 y 3.7 se describen las arquitecturas óptimas obtenidas para las redes neuronales superficiales y máquinas de soporte vectorial respectivamente para cada una de las bases de datos.

Tabla 3.6
Arquitectura de las Redes Neuronales Superficiales.

Parámetros	Valores
Vino	
Función de entrenamiento	traincgp
Función de error	Entropía cruzada
Número de unidades en la capa oculta	5
Computo	Un núcleo
Identificación de vidrio	
Función de entrenamiento	trainbr
Función de error	Error cuadrático medio
Número de unidades en la capa oculta	50
Computo	Un núcleo
Transbordador espacial	
Función de entrenamiento	trainrp
Función de error	Entropía cruzada
Número de unidades en la capa oculta	10
Computo	GPU
Levadura	
Función de entrenamiento	trainbr
Función de error	Error cuadrático medio
Número de unidades en la capa oculta	55
Computo	Un núcleo
Reconocimiento de letras habladas	
Función de entrenamiento	trainscg
Función de error	Entropía cruzada
Número de unidades en la capa oculta	300
Computo	Un núcleo
Reconocimiento de letras	
Función de entrenamiento	traincgp
Función de error	Entropía cruzada
Número de unidades en la capa oculta	50
Computo	Computación paralela

Nota. En cada arquitectura se usó la función de activación tangente hiperbólica sigmoidal en la capa oculta y la función de activación softmax en la capa de salida. Para la inicialización de los pesos sinápticos se empleó el algoritmo de Nguyen y Widrow.
Fuente: Ochoa, L. A. (2019).

Tabla 3.7
Arquitectura de las Máquinas de Soporte Vectorial.

Parámetros	Valores
Vino	
Función Kernel	Lineal
Normalizar	Sí
Matriz de codificación	Binario completo
Validación cruzada	10-fold
Optimización de hiperparámetros	Búsqueda en malla
Identificación de vidrio	
Función Kernel	Gaussiano
Normalizar	No
Matriz de codificación	Uno-vs-uno
Validación cruzada	5-fold
Optimización de hiperparámetros	Optimización bayesiana
Transbordador espacial	
Función Kernel	Polinomial de grado 2
Normalizar	zscore
Matriz de codificación	Uno-vs-uno
Validación cruzada	Holdout (0.15)
Optimización de hiperparámetros	Búsqueda en malla
Levadura	
Función Kernel	Polinomial de grado 4
Normalizar	No
Matriz de codificación	Disperso y denso al azar
Validación cruzada	5-fold
Optimización de hiperparámetros	Búsqueda en malla
Reconocimiento de letras habladas	
Función Kernel	Lineal
Normalizar	zscore
Matriz de codificación	Uno-vs-uno
Validación cruzada	Holdout (0.15)
Optimización de hiperparámetros	Búsqueda en malla
Reconocimiento de letras	
Función Kernel	Polinomial de grado 2
Normalizar	zscore
Matriz de codificación	Uno-vs-uno
Validación cruzada*	Holdout (0.15)
Optimización de hiperparámetros	Búsqueda en malla

Nota. En cada uno de los experimentos se habilitó la opción de computación paralela. En la base de datos *Reconocimiento de letras*, la validación cruzada se utilizó como método de selección.
Fuente: Ochoa, L. A. (2019).

Los resultados de desempeño obtenidos son reforzados con el estudio de las máquinas de soporte vectorial "Increasing Efficiency of Support Vector Machine using the Novel Kernel Function: Combination of Polynomial and Radial Basis Function" de Bhavsar y Ganatra el cual se basó en la metodología "Sequential Minimal optimization" como método de binarización y la propuesta de la función kernel "Radial Basis Polynomial Kernel" llegando al mismo nivel de desempeño. También hacen la observación donde aclaran que las técnicas de binarización (o descomposición) requieren mayor tiempo de entrenamiento.

El estudio de Bhavsar y Ganatra se realizó sobre distintas bases de datos, entre ellas están *Identificación de vidrio* y *Reconocimiento de letras*. En la base de datos *Identificación de vidrio* reportan un desempeño del 64.48% usando el kernel lineal con un tiempo de entrenamiento de 0.078 segundos. Mientras que el desempeño para *Reconocimiento de letras* fue de 84.6% mediante el kernel Gaussiano con un tiempo de entrenamiento de 12.391 segundos.

Otros estudios son los de Park y Fürnkranz, que realizaron una investigación sobre diversas bases de datos, entre estas se encuentran *Reconocimiento de letras habladas*, *Vino* y *Lavadura* donde se obtuvo un desempeño para las máquinas de soporte vectorial de 96.15%, 96.11% y 60.27% mientras que para las redes neuronales superficiales se obtuvo el 97.62%, 96.67% y 60.13% respectivamente.

T. Do y Poulet realizaron una investigación sobre diferentes bases de datos, entre estas están *Vino* y *Transbordador espacial* usando máquinas de soporte vectorial donde reportan un desempeño de 97.22% y 94.78% respectivamente.

Además, Mohit, Katoch, Vanjare y Omkar realizaron un estudio de diversas bases de datos complejas, entre estas se encuentra la base de datos *Lavadura* donde reportan un desempeño del 94.8598% usando una red neuronal artificial.

Por su parte, Grabczewski y Duch efectuaron un análisis sobre la base de datos *Transbordador espacial* donde se obtuvo un desempeño del 96.57% mediante el uso de una red neuronal artificial.

Por último, V. D. Do y Woo realizaron una investigación sobre la base de datos *Reconocimiento de letras* usando una red neuronal artificial donde reportan un desempeño de 94.16%.

En definitiva, ambas técnicas han mejorado en comparación con los primeros algoritmos de clasificación. Respecto a las redes neuronales superficiales, se desarrollaron algoritmos de entrenamiento con el fin de evitar los problemas que surgieron con el método estándar retropropagación, esto es, prevenir que el algoritmo quede atrapado en una solución local mínima o evitar la oscilación dentro de esta; también se han propuesto distintas funciones de activación que permitan acelerar la actualización de los pesos sinápticos; incluso la selección aleatoria de estos tiene un impacto en la aceleración del entrenamiento y finalmente, el uso

de la función de error validación cruzada, siempre y cuando sea conjugable con la función de activación, que permite simplificar el cálculo en la propagación del error.

En el caso de las máquinas de soporte vectorial, es evidente la ventaja del uso de la estandarización de los patrones de entrada que incrementó la predicción. Sin embargo, el tiempo de entrenamiento también creció ya que se requieren múltiples máquinas de soporte vectorial para solucionar un solo problema de múltiples clases y por supuesto la búsqueda de la combinación adecuada de hiperparámetros.

3.3 Conclusiones

El aprendizaje automático es una rama de la inteligencia artificial que simula el aprendizaje humano con aplicaciones en la economía, medicina, seguridad informática, investigación científica, entre otras áreas. Las redes neuronales artificiales y las máquinas de soporte vectorial son las técnicas del aprendizaje automático más utilizadas para lograr dicha simulación.

En este trabajo de investigación se realizan dos análisis estadísticos no paramétricos para evaluar los resultados obtenidos de seis bases de datos que corresponden al problema de clasificación multiclase.

En el primer análisis se evalúa el desempeño midiendo las clasificaciones correctas logradas por cada técnica de clasificación. El segundo, evalúa el tiempo durante el proceso de entrenamiento de las máquinas de soporte vectorial y las redes neuronales superficiales. En ambos experimentos se esperaba encontrar diferencias *estadísticamente* significativas entre ambas técnicas siendo las máquinas de soporte vectorial quienes proporcionan un mejor desempeño en un periodo breve de entrenamiento.

En el primer experimento, tanto las redes neuronales superficiales como las máquinas de soporte vectorial han mostrado ser excelentes técnicas de clasificación. Sin embargo durante la realización del experimento, se presentó el problema para determinar el mejor ajuste de parámetros que permitan obtener el mejor modelo para cada una de las bases de datos usando ambas estrategias. Esto se solucionó aplicando la búsqueda en malla y la optimización bayesiana en las máquinas de soporte vectorial y usando distintas reglas del pulgar para determinar el número de unidades ocultas en el caso de las redes neuronales superficiales.

En los resultados obtenidos, expuestos en la tabla 3.2 en la página 71, se observa que las redes neuronales superficiales desempeñan la clasificación mejor que las máquinas de soporte vectorial. Sin embargo, es importante examinar que no existe una diferencia estadísticamente significativa en el desempeño de ambos clasificadores.

Un factor importante para señalar la eficacia de un algoritmo es el tiempo requerido para realizar su trabajo. En los algoritmos de aprendizaje, el tiempo de entrenamiento es un factor

importante. Por lo que obtener y analizar el tiempo durante dicho proceso de cada técnica fue una tarea esencial dando lugar al segundo experimento.

Como se puede observar en la tabla 3.4 en la página 73, se obtuvo un tiempo de entrenamiento menor de las redes neuronales superficiales en comparación con las máquinas de soporte vectorial; no obstante, la prueba estadística demostró que no hay una diferencia estadísticamente significativa entre ambos clasificadores.

Finalmente, en un caso práctico se deben evaluar ambas técnicas de clasificación, no necesariamente limitarse a estas, a fin de obtener el mejor modelo posible para resolver un caso en específico.

Este trabajo de investigación marca un fin del camino más importante de mi vida en el que he adquirido experiencias. Por lo que brindo algunas breves recomendaciones a los alumnos que comienzan una nueva etapa. Antes que nada, confía en ti, investiga a tus profesores, y por supuesto, ¡estudia!. Si estas por terminar tu formación profesional y haz elegido realizar un trabajo de investigación, entonces haz tomado la mejor decisión. Haz de la lectura y escritura un hábito y se abierto a los comentarios de tu tutor.

Glosario

abstracción Modelo de la realidad. Éste modelo se basa en las características esenciales de un objeto que lo distinguen de otros objetos.

algoritmo Procedimiento computacional bien definido que toma ciertos valores como entrada y produce algún valor como salida.

aprendizaje Cambio permanente de conductas como resultado de experiencias pasadas.

aprendizaje automático Rama de las ciencias de la computación que registrar minuciosamente conjuntos de datos para hacer predicciones sobre el futuro.

binarización Reducción un problema de aprendizaje multiclase en varios problemas de aprendizaje binarios que pueden ser resueltos separadamente.

códigos de salida de corrección de errores Técnica de binarización que emplea códigos de corrección de errores como una representación de salida distribuida.

entrenamiento Algoritmo mediante el cual un modelo de aprendizaje se ajusta para dar solución a un problema determinado.

época Presentación del conjunto de los patrones de entrenamiento a una red neuronal ya sea en bloque o en línea.

generalización Indica la capacidad de reaccionar ante nuevas situaciones cuando estas se parecen a otras ya aprendidas y que se volvieron familiares.

inteligencia artificial Es la ciencia y la ingeniería de la creación de máquinas inteligentes, especialmente programas de computadora inteligentes. Está relacionada con la tarea de usar computadoras para comprender la inteligencia humana, pero la inteligencia artificial no tiene que limitarse a métodos que sean biológicamente observables.

interrupción anticipada Técnica para controlar la complejidad de una red neuronal utilizada para evitar el sobreajuste evaluando cada época del entrenamiento.

máquina de soporte vectorial Algoritmo de aprendizaje automático que separa los patrones para construir un hiperplano de separación. Resuelve problemas de clasificación binaria.

reconocimiento de patrones Descubrimiento de regularidades en los patrones mediante el uso de algoritmos y con el uso de estas regularidades realiza acciones tales como clasificar los datos en diferentes categorías.

red neuronal artificial procesador distribuido masivamente en paralelo formado por unidades de procesamiento simples que tiene una propensión natural a almacenar el conocimiento experiencial y ponerlo a disposición para su uso.

regla del pulgar Método para juzgar una condición que no es exacta pero se basa en la experiencia.

Bibliografía

Abu-Mostafa, Y. S. (2012). Machines that think for themselves: New techniques for teaching computers how to learn are beating the experts. *Scientific American*, 78-81. Recuperado desde https://work.caltech.edu/pubs.html

Abu-Mostafa, Y., Song, X., Nicholson, A. & Magdon-Ismail, M. (2004). The Bin Model. *IEEE Potentials*. Recuperado desde https://work.caltech.edu/pubs.html

Allwein, E. L., Schapire, R. E. & Singer, Y. (2000). Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*. Recuperado desde http://www.jmlr.org/papers/v1/allwein00a.html

Battiti, R. (1992). First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation*, *4*(2), 141-166. doi:10.1162/neco.1992.4.2.141

Baum, E. B. & Haussler, D. (1989). What Size Net Gives Valid Generalization? En D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1* (pp. 81-90). Morgan-Kaufmann. Recuperado desde https://papers.nips.cc/paper/154-what-size-net-gives-valid-generalization.pdf

Baum, E. B. & Wilczek, F. (1988). Supervised Learning of Probability Distributions by Neural Networks. En D. Z. Anderson (Ed.), *Neural Information Processing Systems* (pp. 52-61). American Institute of Physics. Recuperado desde http://papers.nips.cc/paper/3-supervised-learning-of-probability-distributions-by-neural-networks.pdf

Bay, S. D. (2001). Multivariate Discretization for Set Mining. *Knowledge and Information Systems*, *3*(4), 491-512. doi:10.1007/PL00011680

Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res. 13*, 281-305. Recuperado desde http://www.jmlr.org/papers/v13/bergstra12a.html

Bhavsar, H. & Ganatra, A. (2014). Increasing Efficiency of Support Vector Machine using the Novel Kernel Function: Combination of Polynomial and Radial Basis Function. En *International Journal on Advanced Computer Theory and Engineering (IJACTE)*. Recupe-

rado desde https://pdfs.semanticscholar.org/ff6b/d737ac9a9794229589549e589806a93655ab.pdf

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Bishop, C. M. (1996). Neural Networks: A Pattern Recognition Perspective. En *Handbook of Neural Computation* (Handbook of Neural Computation). Oxford University Press y IOP Publishing. Recuperado desde https://www.microsoft.com/en-us/research/publication/neural-networks-a-pattern-recognition-perspective/

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

Blum, A., Kalai, A. T. & Langford, J. (1999). Beating the Holdout: Bounds for K-Fold and Progressive Cross-Validation. En *Proceedings of the Twelfth Annual Conference on Computational Learning Theory* (pp. 203-208). ACM Press. Recuperado desde https://www.microsoft.com/en-us/research/publication/beating-holdout-bounds-k-fold-progressive-cross-validation/

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. En *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (pp. 144-152). Pittsburgh, Pennsylvania, USA: ACM. Recuperado desde http://www.svms.org/training/BOGV92.pdf

Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145-1159. CiteSeerX:10.1.1.93.112

Brochu, E., Cora, V. M. & de Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR*, *abs/1012.2599*. arXiv: 1012.2599

Cajal, S. R. Y. (1894). The Croonian Lecture: La Fine Structure des Centres Nerveux. *Proceedings of the Royal Society of London*, *55*(331-335), 444-468. doi:10.1098/rspl.1894.0063

Chapelle, O., Haffner, P. & Vapnik, V. (1999). Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Networks*, *10*(5), 1055-1064. doi:10.1109/72.788646

Chapelle, O. & Vapnik, V. (2000). Model Selection for Support Vector Machines. En S. A. Solla, T. K. Leen & K. Müller (Eds.), *Advances in Neural Information Processing Systems 12* (pp. 230-236). MIT Press. Recuperado desde http://papers.nips.cc/paper/1663-model-selection-for-support-vector-machines.pdf

Chapelle, O., Vapnik, V., Bousquet, O. & Mukherjee, S. (2002). Choosing Multiple Parameters for Support Vector Machines. *Machine Learning*, *46*(1), 131-159. doi:10.1023/A:1012450327387

Chen, P.-H., Lin, C.-J. & Schölkopf, B. (2005). A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry*, *21*(2), 111-136. doi:10.1002/asmb.537

Cherkassky, V. & Ma, Y. (2004). Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, *17*(1), 113-126. doi:10.1016/S0893-6080(03)00169-2

Chicco, D. (2017). Ten quick tips for machine learning in computational biology. *BioData Mining*, *10*(1). doi:10.1186/s13040-017-0155-3

Chinchor, N. (1992). MUC-4 Evaluation Metrics. En *Proceedings of the 4th Conference on Message Understanding* (pp. 22-29). MUC4 '92. McLean, Virginia: Association for Computational Linguistics. doi:10.3115/1072064.1072067

Cole, R. & Fanty, M. (1990). Spoken Letter Recognition. En *Proceedings of the Workshop on Speech and Natural Language* (pp. 385-390). HLT '90. Hidden Valley, Pennsylvania: Association for Computational Linguistics. doi:10.3115/116580.116725

Cortes, C. & Vapnik, V. (1995). Support–Vector Networks. *Machine Learning*, *20*(3), 273-297. doi:10.1023/A:1022627411411

Cover, T. M. (1965). Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers*, *EC-14*(3), 326-334. doi:10.1109/PGEC.1965.264137

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303-314. doi:10.1007/BF02551274

Czogiel, I., Luebke, K. & Weihs, C. (2006). Response Surface Methodology for Optimizing Hyper Parameters. Recuperado desde https://www.semanticscholar.org/paper/Response-Surface-Methodology-for-Optimizing-Hyper-Czogiel-Luebke/6f08a8d26f8567e8c6f04a7cf628e43f04756292

Dheeru, D. & Karra Taniskidou, E. (2017). UCI Machine Learning Repository. University of California, Irvine, School of Information y Computer Sciences. Recuperado desde http://archive.ics.uci.edu/ml

Dietterich, T. G. & Bakiri, G. (1995). Solving Multiclass Learning Problems via Error-Correcting Output Codes. *CoRR*. arXiv: cs/9501101

Do, T. & Poulet, F. (2006). Kernel-based Algorithms and Visualization for Interval Data Mining. En *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)* (pp. 295-299). doi:10.1109/ICDMW.2006.103

Do, V. D. & Woo, D. (2015). Handwritten Character Recognition Using Feedforward Artificial Neural Network. En *7th International Conference on Latest Trends in Engineering and*

Technology (ICLTET'2015) Nov. 26-27, 2015 Irene, Pretoria (South Africa). International Institute of Engineers. doi:10.15242/iie.e1115044

Domingos, P. (2015). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York: Basic Books.

Dreyfus, G. (2005). *Neural Networks: Methodology and Applications*. Springer-Verlag Berlin Heidelberg. doi:10.1007/3-540-28847-3

Duda, R., Hart, P. & Stork, D. (2012). *Pattern Classification*. Wiley.

Elisseeff, A. & Paugam–Moisy, H. (1997). Size of Multilayer Networks for Exact Learning: Analytic Approach. En M. C. Mozer, M. I. Jordan & T. Petsche (Eds.), *Advances in Neural Information Processing Systems 9* (pp. 162-168). MIT Press. Recuperado desde http://papers.nips.cc/paper/1303-size-of-multilayer-networks-for-exact-learning-analytic-approach.pdf

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861-874. ROC Analysis in Pattern Recognition. doi:10.1016/j.patrec.2005.10.010

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*(2), 179-188. doi:10.1111/j.1469-1809.1936.tb02137.x

Flach, P. A. (2003). The geometry of ROC space: understanding machine learning metrics through ROC isometrics. En *in Proceedings of the Twentieth International Conference on Machine Learning* (pp. 194-201). AAAI Press. CiteSeerX:10.1.1.104.4919

Fletcher, R. & Reeves, C. M. (1964). Function minimization by conjugate gradients. *The Computer Journal*, *7*(2), 149-154. doi:10.1093/comjnl/7.2.149

Foresee, F. D. & Hagan, M. T. (1997). Gauss-Newton approximation to Bayesian learning. En *Neural Networks,1997., International Conference on* (Vol. 3, 1930-1935 vol.3). doi:10.1109/ICNN.1997.614194

Foster, C., M.; Sherrington. (1897). *Textbook of Physiology, volume 3* (seventh). Sinapsis. London : Macmillan.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, *2*, 183-192. doi:10.1016/0893-6080(89)90003-8

Fürnkranz, J. (2002). Round Robin Classification. *Journal of Machine Learning Research*, *2*, 721-747.

Galimberti, U. (2006). Generalización. En *Diccionario de psicología* (p. 252). Psicología y psicoanálisis Series. Gruppo editoriale L'Espresso.

Grabczewski, K. & Duch, W. (2000). The Separability Of Split Value Criterion. En *In Proceedings of the 5th Conference on Neural Networks and Their Applications*. CiteSeerX:10.1.1.43.1719

Gross, R. D. (1987). *Psychology: The Science of Mind and Behaviour*. London : Hodder y stoughton, 1987.

Guyon, I., Weston, J., Barnhill, S. & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, *46*(1), 389-422. doi:10.1023/A:1012487302797

Hagan, M. T. & Menhaj, M. B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, *5*(6), 989-993. doi:10.1109/72.329697

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, *29*(2), 147-160. doi:10.1002/j.1538-7305.1950.tb00463.x

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. doi:10.1007/978-0-387-84858-7

Haugeland, J. (2001). *La inteligencia artificial* (3ª ed.). Ciencia y Técnica. Siglo XXI.

Hayek, F. (1952). *The Sensory Order: An Inquiry Into the Foundations of Theoretical Psychology*. University of Chicago Press.

Haykin, S. (2009). *Neural Networks and Learning Machines*. New York: Prentice Hall/Pearson.

Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. A Wiley book in clinical psychology. John Wiley y Sons Inc.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, *40*(1), 185-234.

Hofmann, T., Schölkopf, B. & Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, *36*(3), 1171-1220. doi:10.1214/009053607000000677

Hofstadter, D. (2007). *Gödel, Escher, Bach: un eterno y grácil bucle*. Fabula (Tusquets Editores) Series. Tusquets.

Hornik, K., Stinchcombe, M. & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359-366. doi:10.1016/0893-6080(89)90020-8

Horton, P. & Nakai, K. (1996). A Probabilistic Classification System for Predicting the Cellular Localization Sites of Proteins. En *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology* (pp. 109-115). AAAI Press. Recuperado desde http://www.aaai.org/Papers/ISMB/1996/ISMB96-012.pdf

Howlett, R. J. & Jain, L. C. (2001). *Radial Basis Function Networks 1: Recent Developments in Theory and Applications*. Physica-Verlag Heidelberg.

Hsu, C., Chang, C. & Lin, C. (2003). *A Practical Guide to Support Vector Classification*. Department of Computer Science, National Taiwan University. Recuperado desde http://www.csie.ntu.edu.tw/~cjlin/papers.html

Issenberg, S. (2012). How Obama's Team Used Big Data to Rally Voters. Recuperado desde https://www.technologyreview.com/s/509026/how-obamas-team-used-big-data-to-rally-voters/

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. doi:10.1007/978-1-4614-7138-7

Jordan, M. I. (1986). An Introduction to Linear Algebra in Parallel Distributed Processing. En D. E. Rumelhart, J. L. McClelland & C. PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (pp. 365-422). Cambridge, MA, USA: MIT Press.

Joutsijoki, H., Haponen, M., Rasku, J., Aalto-Setälä, K. & Juhola, M. (2016). Error-Correcting Output Codes in Classification of Human Induced Pluripotent Stem Cell Colony Images. *BioMed Research International*, *2016*, 1-13. doi:10.1155/2016/3025057

Kandel, E. R., Schwartz, J. H. & Jessell, T. M. (2000). *Principles of Neural Science*. New York; México City: McGraw-Hill, c2000.

Karsoliya, S. (2012). Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture. *International Journal of Engineering Trends and Technology*, *3*, 714-717. Recuperado desde http://ijettjournal.org/archive/ijett-v3i6p206

Khosrowpour, M. (2012). *Machine learning: Concepts, Methodologies, Tools and Applications*. Hershey, Pennsylvania : Information Science Reference, c2012.

Kohavi, R. (1995). A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. En *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (pp. 1137-1143). IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc. Recuperado desde http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.529

Konorski, J. (1948). *Conditioned reflexes and neuron organization.* Cambridge biological studies. Cambridge : University press, 1948.

Kramer, O. (2016). *Machine Learning for Evolution Strategies*. Springer International Publishing. doi:10.1007/978-3-319-33383-0

LeCun, Y. (1985). Une procedure d'apprentissage pour reseau a seuil asymmetrique (A learning scheme for asymmetric threshold networks). En *Proceedings of Cognitiva 85, Paris, France.* Recuperado desde http://yann.lecun.com/exdb/publis/index.html

LeCun, Y. (1987). *Modeles connexionnistes de l'apprentissage (connectionist learning models)* (Tesis doctoral, Université P. et M. Curie (Paris 6)). Recuperado desde http://yann.lecun.com/exdb/publis/index.html

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444. doi:10.1038/nature14539

Leshno, M., Lin, V. Y., Pinkus, A. & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861-867. doi:10.1016/S0893-6080(05)80131-5

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, *4*(2), 4-22. doi:10.1109/MASSP.1987.1165576

Mandic, D. & Chambers, J. (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley.

Marsland, S. (2015). *Machine learning: An algorithmic perspective*. Chapman & Hall/CRC machine learning & pattern recognition series. Boca Raton : CRC Press, 2015.

McCarthy, J. (2007). What is Artificial Intelligence? *John McCarthy's Home Page*. Recuperado desde http://www-formal.stanford.edu/jmc/

McCorduck, P. (2004). *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. AK Peters Ltd.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133. doi:10.1007/BF02478259

Metz, C. E. (1978). Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, *8*(4), 283-298. doi:10.1016/S0001-2998(78)80014-2

Mhaskar, H. & Poggio, T. A. (2016). Deep vs. Shallow Networks: An approximation theory perspective. *CoRR, abs/1608.03287*. arXiv: 1608.03287

Minai, A. A. & Williams, R. D. (1993). On the derivatives of the sigmoid. *Neural Networks*, *6*(6), 845-853. doi:10.1016/S0893-6080(05)80129-7

Mitchell, T. M. (2006). *The Discipline of Machine Learning*. Carnegie Mellon University. Recuperado desde http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf

Mohammed, M. (2017). *Machine Learning: Algorithms and Applications*. Boca Raton: CRC Press.

Mohit, R. R. V., Katoch, S., Vanjare, A. & Omkar, S. (2015). Classification of Complex UCI Datasets Using Machine Learning Algorithms Using Hadoop. *Internation Journal of Computer Sceince and Software Engineering (IJCSSE)*, *4*(7). Recuperado desde http://ijcsse.org/published/volume4/issue7/p4-V4I7.pdf

Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*(4), 525-533. doi:10.1016/S0893-6080(05)80056-5

Nguyen, D. & Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. En *1990 IJCNN International Joint Conference on Neural Networks* (21-26 vol.3). doi:10.1109/IJCNN.1990.137819

Nocedal, J. & Write, S. J. (2006). *Numerical optimization*. New York: Springer.

Ochoa, L. A. (2019a). Machine Learning wih TikZ. doi:10.5281/zenodo.2564739

Ochoa, L. A. (2019b). Multilayer Perceptron with TikZ. doi:10.5281/zenodo.2564472

Ochoa, L. A. (2019c). Support-Vector Machine with TikZ. doi:10.5281/zenodo.2564470

Pal, S. K. & Mitra, S. (1992). Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, *3*(5), 683-697. doi:10.1109/72.159058

Park, S. & Fürnkranz, J. (2007). Efficient Pairwise Classification. En J. N. Kok, J. Koronacki, R. L. d. Mantaras, S. Matwin, D. Mladenič & A. Skowron (Eds.), *Machine Learning: ECML 2007* (pp. 658-665). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-74958-5_65

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, *12*(1), 241-254. doi:10.1007/BF01593790

Prechelt, L. (2012). Early Stopping — But When? En G. Montavon, G. B. Orr & K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade: Second Edition* (pp. 53-67). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-35289-8_5

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press. Recuperado desde http://www.gaussianprocess.org/gpml/

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons–From backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, *16*(3), 265-278. doi:10.1016/0920-5489(94)90017-5

Riedmiller, M. & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm. En *IEEE International Conference on Neural Networks* (586-591 vol.1). doi:10.1109/ICNN.1993.298623

Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory.

Rosenblatt, F. (1961). *Principles of neurodynamics: Perceptions and the theory of brain mechanism*. Cornell Aeronautical Laboratory.

Rubinstein, R. Y. & Kroese, D. P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. New York, NY: Springer-Verlag New York.

Rumelhart, D. E., Hinton, G. E. & McClelland, J. L. (1986). A General Framework for Parallel Distributed Processing. En D. E. Rumelhart, J. L. McClelland & C. PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (pp. 45-76). Cambridge, MA, USA: MIT Press.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986a). Learning Internal Representations by Error Propagation. En D. E. Rumelhart, J. L. McClelland & C. PDP Research Group

(Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (pp. 318-362). Cambridge, MA, USA: MIT Press.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, *323*, 533-536.

Rumelhart, D. E. & Zipser, D. (1986). Feature Discovery by Competitive Learning. En D. E. Rumelhart, J. L. McClelland & C. PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (pp. 151-193). Cambridge, MA, USA: MIT Press.

Sammut, C. & Webb, G. I. (2010). Mean Squared Error. En C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 653-653). Boston, MA: Springer US. doi:10.1007/978-0-387-30164-8_528

Sasaki, Y. (2007). The truth of the F-measure. *Teach Tutor Mater*.

Schank, R. C. (1991). Where's the AI? *AI Mag. 12*(4), 38-49. Recuperado desde https://aaai.org/ojs/index.php/aimagazine/article/view/917/835

Schölkopf, B., Sung, K., Burges, C. J. C., Girosi, F., Niyogi, P., Poggio, T. & Vapnik, V. (1997). Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, *45*(11), 2758-2765. doi:10.1109/78.650102

Scott, A. (2005). *Encyclopedia of Nonlinear Science*. Routledge. Recuperado desde http://pbidi.unam.mx:8080/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=160399&lang=es&site=eds-live

Shao, J. (1993). Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association*, *88*(422), 486-494. Recuperado desde http://www.jstor.org/stable/2290328

Shawe–Taylor, J. & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Sherali, H. D. & Ulular, O. (1990). Conjugate gradient methods using quasi-Newton updates with inexact line searches. *Journal of Mathematical Analysis and Applications*, *150*(2), 359-377. doi:10.1016/0022-247X(90)90109-S

Shi, Z. (2011). *Advanced Artificial Intelligence (Series on Intelligence Science)*. World Scientific Publishing Company.

Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. arXiv: 1206.2944

Takeuchi, T., Duszkiewicz, A. J. & Morris, R. G. M. (2013). The synaptic plasticity and memory hypothesis: encoding, storage and persistence. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *369*(1633). doi:10.1098/rstb.2013.0288

The MathWorks, Inc. (2006a). Gradient descent with adaptive learning rate backpropagation. Recuperado desde https://la.mathworks.com/help/releases/R2018b/nnet/ref/traingda.html

The MathWorks, Inc. (2006b). Multiple comparison test. Recuperado desde https://la.mathworks.com/help/releases/R2018b/stats/multcompare.html

The MathWorks, Inc. (2014). Fit multiclass models for support vector machines or other classifiers. Recuperado desde https://la.mathworks.com/help/releases/R2018b/stats/fitcecoc.html

Touretzky, D. S. & Pomerleau, D. A. (1989). What's Hidden in the Hidden Layers? *BYTE*, *14*(8), 227-233. Recuperado desde https://www.cs.cmu.edu/~dst/pubs/byte-hiddenlayer-1989.pdf

Turing, A. M. (1951). Intelligent Machinery, A Heretical Theory. *The Essential Turing*.

Vapnik, V. (1992). Principles of Risk Minimization for Learning Theory. En J. E. Moody, S. J. Hanson & R. P. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4* (pp. 831-838). Morgan–Kaufmann. Recuperado desde http://papers.nips.cc/paper/506-principles-of-risk-minimization-for-learning-theory.pdf

Vapnik, V. (1998). *Statistical learning theory*. Wiley-Interscience.

Vapnik, V. (1999). An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, *10*(5), 988-999. doi:10.1109/72.788640

Vapnik, V. (2006). *Estimation of Dependences Based on Empirical Data*. Information Science and Statistics. Springer New York.

Vapnik, V. & Lerner, A. (1963). Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control*, *24*.

Vapnik, V. & Mukherjee, S. (2000). Support Vector Method for Multivariate Density Estimation. En S. A. Solla, T. K. Leen & K. Müller (Eds.), *Advances in Neural Information Processing Systems 12* (pp. 659-665). MIT Press. Recuperado desde http://papers.nips.cc/paper/1652-support-vector-method-for-multivariate-density-estimation.pdf

Veropoulos, K., Campbell, C. & Cristianini, N. (1999). Controlling the Sensitivity of Support Vector Machines. En *Proceedings of the International Joint Conference on AI* (pp. 55-60).

Vert, J., Tsuda, K. & Schölkopf, B. (2004). A Primer on Kernel Methods. En B. Schölkopf, K. Tsuda & J. Vert (Eds.), *Kernel Methods in Computational Biology* (pp. 35-70). Cambridge, MA, USA: MIT Press. Recuperado desde https://pdfs.semanticscholar.org/f15f/dea09a1e4be8ff698e34f13e51a0ff66131d.pdf

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T. & Alkon, D. L. (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, *59*(4), 257-263. doi:10.1007/BF00332914

Walpole, R. E., Myers, R. H. & Myers, S. L. (2012). *Probabilidad y estadística para ingeniería y ciencias*. México: Pearson.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* (Tesis doctoral, Harvard University).

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B. & Vapnik, V. (2002). Kernel Dependency Estimation. En *Proceedings of the 15th International Conference on Neural Information Processing Systems* (pp. 897-904). NIPS'02. Cambridge, MA, USA: MIT Press.

Wikimedia Commons. (2018a). File:Neuron.svg — Wikimedia Commons, the free media repository. Recuperado desde https://commons.wikimedia.org/w/index.php?title=File:Neuron.svg&oldid=288844218

Wikimedia Commons. (2018b). File:SynapseSchematic unlabeled.svg — Wikimedia Commons, the free media repository. Recuperado desde https://commons.wikimedia.org/w/index.php?title=File:SynapseSchematic_unlabeled.svg&oldid=313848168

Wikipédia. (2018). Distance de Hamming — Wikipédia, l'encyclopédie libre. Recuperado desde http://fr.wikipedia.org/w/index.php?title=Distance_de_Hamming&oldid=148403120

Williams, R. J. (1986). The Logic of Activation Functions. En D. E. Rumelhart, J. L. McClelland & C. PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (pp. 423-443). Cambridge, MA, USA: MIT Press.

Winston, P. H. (1992). *Artificial Intelligence (3rd Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Yu, C. & Liu, B. (2002). A backpropagation algorithm with adaptive learning rate and momentum coefficient. En *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on* (Vol. 2, pp. 1218-1223). doi:10.1109/IJCNN.2002.1007668

Anexo A

Bases de Datos

Las seis bases de datos en las cuales se llevaron los experimentos fueron obtenidas de "UCI Machine Learning Repository" donde también se incluyen las descripciones de estas. Cada una de estas se seleccionó por tener más de dos clases y no tener valores faltantes. Algunas bases de datos cuentan con un número reducido de ejemplares y otras con un número abundante con el objetivo de evaluar ambos casos. En la tabla A.1 se describen las características esenciales de cada una de las base de datos empleadas.

Vino Estos datos son el resultado de un análisis químico de vinos cultivados en la misma región en Italia pero derivados de tres cultivares diferentes. El análisis determinó las cantidades de 13 componentes encontrados en cada uno de los tres tipos de vinos.

Identificación de vidrio El estudio de clasificación de los tipos de vidrio fue motivado por la investigación criminológica. En la escena del crimen, el vidrio puede ser usado como evidencia…¡si está correctamente identificado!

Transbordador espacial Se ocupa del posicionamiento de los radiadores en el transbordador espacial (Bay, 2001).

Levadura Predicción de los sitios de localización de proteínas de levadura en bacterias gramnegativas, dada la información de la secuencia de aminoácidos (Horton & Nakai, 1996).

Reconocimiento de letras habladas (ISOLET) El reconocimiento automático de letras habladas es una de las tareas más desafiantes en el campo del reconocimiento de voz por computadora. La dificultad de la tarea se debe a la similitud acústica de muchas de las letras. El reconocimiento preciso requiere que el sistema realice distinciones fonéticas finas, tales como B *vs.* D, B *vs.* P, D *vs.* T, T *vs.* G, C *vs.* Z, V *vs.* Z, M *vs.* N y J *vs.* K (Cole & Fanty, 1990).

Reconocimiento de letras El objetivo es identificar cada una de una gran cantidad de pantallas de píxeles rectangulares en blanco y negro como una de las 26 letras mayúsculas en el alfabeto inglés.

Tabla A.1
Características esenciales de las bases de datos utilizadas en los experimentos.

Conjunto de datos	Ejemplares	Atributos	Clases
Vino	178	13	3
Identificación de vidrio	214	9	6
Transbordador espacial	58000	9	7
Levadura	1484	8	10
Reconocimiento de letras habladas	7797	617	26
Reconocimiento de letras	20000	16	26

Fuente: "UCI Machine Learning Repository" de Dheeru y Karra Taniskidou, 2017.

Anexo B

Distribución χ^2

Tabla B.1
Valores críticos de la distribución χ^2 con v grados de libertad.

	Probabilidad de exceder el valor crítico								
v	0.30	0.25	0.20	0.10	0.05	0.025	0.02	0.01	0.005
1	1.074	1.323	1.642	2.706	3.841	5.024	5.412	6.635	7.879
2	2.408	2.773	3.219	4.605	5.991	7.378	7.824	9.21	10.597
3	3.665	4.108	4.642	6.251	7.815	9.348	9.837	11.345	12.838
4	4.878	5.385	5.989	7.779	9.488	11.143	11.668	13.277	14.86
5	6.064	6.626	7.289	9.236	11.07	12.832	13.388	15.086	16.75
6	7.231	7.841	8.558	10.645	12.592	14.449	15.033	16.812	18.548
7	8.383	9.037	9.803	12.017	14.067	16.013	16.622	18.475	20.278
8	9.524	10.219	11.03	13.362	15.507	17.535	18.168	20.09	21.955
9	10.656	11.389	12.242	14.684	16.919	19.023	19.679	21.666	23.589
10	11.781	12.549	13.442	15.987	18.307	20.483	21.161	23.209	25.188
11	12.899	13.701	14.631	17.275	19.675	21.92	22.618	24.725	26.757
12	14.011	14.845	15.812	18.549	21.026	23.337	24.054	26.217	28.3
13	15.119	15.984	16.985	19.812	22.362	24.736	25.471	27.688	29.819
14	16.222	17.117	18.151	21.064	23.685	26.119	26.873	29.141	31.319
15	17.322	18.245	19.311	22.307	24.996	27.488	28.259	30.578	32.801
16	18.418	19.369	20.465	23.542	26.296	28.845	29.633	32	34.267
17	19.511	20.489	21.615	24.769	27.587	30.191	30.995	33.409	35.718
18	20.601	21.605	22.76	25.989	28.869	31.526	32.346	34.805	37.156
19	21.689	22.718	23.9	27.204	30.144	32.852	33.687	36.191	38.582
20	22.775	23.828	25.038	28.412	31.41	34.17	35.02	37.566	39.997

Fuente: Adaptada de *Probabilidad y estadística para ingeniería y ciencias* de Walpole, Myers y Myers, 2012.