TECNOLOGÍA

JORDI GIRONÉS ROIG
JORDI CASAS ROMA
JULIÀ MINGUILLÓN ALFONSO
RAMON CAIHUELAS QUILES

MINERÍA DE DATOS

MODELOS Y ALGORITMOS

EDITORIAL UOC

Minería de datos: modelos y algoritmos

Minería de datos
Modelos y algoritmos

Jordi Gironés Roig
Jordi Casas Roma
Julià Minguillón Alfonso
Ramon Caihuelas Quiles

Director de la colección «Manuales» (Tecnología): Antoni Pérez

Diseño de la colección: Editorial UOC
Diseño de la cubierta: Natàlia Serrano

Primera edición en lengua castellana: julio 2017
Primera edición digital: septiembre 2017

Rambla del Poblenou, 156
08018 Barcelona
http://www.editorialuoc.com

Realización editorial: dâctilos
ISBN: 978-84-9116-904-8

Autores

Jordi Gironés Roig

Licenciado en Matemáticas por la Universidad Autónoma de Barcelona y diplomado en Empresariales por la Universitat Oberta de Catalunya. Ha desarrollado la mayor parte de su carrera profesional en torno de la solución SAP, en sus vertientes operativa (R3) y estratégica (BI). Actualmente trabaja en la industria farmacéutica como responsable de aplicaciones corporativas para los Laboratorios del Doctor Esteve.

Jordi Casas Roma

Licenciado en Ingeniería Informática por la Universidad Autónoma de Barcelona (UAB), tiene un máster en Inteligencia Artificial Avanzada por la Universidad Nacional de Educación a Distancia (UNED) y es doctor en Informática por la UAB. Desde 2009 ejerce como profesor en los Estudios de Informática, Multimedia y Telecomunicación de la Universitat Oberta de Catalunya (UOC), y recientemente también como profesor asociado en la UAB. Es director del máster universitario en Ciencia de Datos de la UOC. Sus actividades docentes se centran en la minería de datos, el análisis en entornos big data, las bases de datos y la teoría de grafos. Desde 2010 pertenece al grupo de investigación KISON (K-ryptography and Information Security for Open Networks). Sus intereses de investigación incluyen temas relacionados con la privacidad, la minería de datos y los sistemas de big data.

Julià Minguillón Alfonso

Licenciado en Ingeniería Informática por la Universidad Autónoma de Barcelona (UAB) en 1995, obtuvo un máster en Combinatoria y Comunicación Digital por la UAB en 1997 y se graduó como doctor ingeniero en Informática por la UAB en 2002. Desde 2001 ejerce como profesor en los Estudios de Informática, Multimedia y Telecomunicación de la Universitat Oberta de Catalunya (UOC). Pertenece al grupo de investigación LAIKA (*Learning Analytics for Innovation and Knowledge Application*), donde desarrolla proyectos de investigación relacionados con el análisis y visualización del comportamiento de los usuarios de entornos virtuales de aprendizaje y redes sociales.

Ramon Caihuelas Quiles

Licenciado en Ciencias de la Información por la Universidad Autónoma de Barcelona (UAB). Tiene un posgrado en Diseño de Aplicaciones por la Universitat Politècnica de Catalunya (UPC) y un máster en Gestión de Tecnologías de Información por La Salle (Universidad Ramon Llull). Doctorando en Informática por La Salle (URL). Actualmente trabaja como responsable del grupo de bases de datos y soporte al desarrollo del área de Tecnologías de la Universidad de Barcelona y colaborador docente de los estudios de Informática de Ingeniería La Salle (URL) y la Universitat Oberta de Catalunya.

A Toni Pellisa. Siempre tendrás
un rincón en mi corazón.
Jordi Gironés Roig

A mi hijo Arnau, quien, con su mirada y su sonrisa, me recuerda cada día cuáles son las cosas importantes en la vida.
Jordi Casas Roma

A mi maestro Jaume Pujol, que fue quien
me orientó en este camino.
Julià Minguillón Alfonso

Índice

Prólogo

En el discurso de la defensa de Sócrates, este trata de protegerse de quienes le acusan de haber cultivado una mala reputación. Él mismo explica que en una ocasión su amigo Quérefon tuvo el atrevimiento de preguntar al oráculo de Delfos si había en el mundo un hombre más sabio que Sócrates, a lo que el oráculo respondió que no.

Al conocer tal respuesta, Sócrates se propuso demostrar que la Pythia se había equivocado, de modo que emprendió la aventura de encontrar un hombre más sabio que él.

Empezó por visitar al que era considerado uno de los más sabios de la ciudad, un político de la época. Sócrates, al darse cuenta de que el personaje en cuestión, aun teniéndose por sabio, en realidad no lo era, hizo la siguiente reflexión:

Luego que de él me separé, razonaba conmigo mismo, y me decía:

—Yo soy más sabio que este hombre. Puede muy bien suceder, que ni él ni yo sepamos nada de lo que es bello y de lo que es bueno; pero hay esta diferencia, que él cree saberlo aunque no sepa nada, y yo, no sabiendo nada, creo no saber. Me parece, pues, que en esto yo, aunque poco más, era mas sabio, porque no creía saber lo que no sabía.

Platón, Apología de Sócrates, 399 a. C. aprox.

En este bello pasaje, Platón sienta las bases de lo que acabará siendo el pensamiento científico: el análisis crítico y el aprendizaje continuo. Este último vive en la actualidad uno de los momentos más felices de la historia de la humanidad con las últimas dos revoluciones: internet con la democratización del conocimiento y la movilidad con la inmediatez de la información.

La inteligencia artificial, sustentada de lleno en las matemáticas y la computación, plantea a analistas, programadores e ingenieros el reto del aprendizaje continuo e incansable que persigue un conocimiento inalcanzable y palpable a la vez. Si las ciencias exactas son el mejor idioma de que disponemos para hablar con el universo, también son la mejor herramienta que tenemos para tratar de imitar la inteligencia humana.

Hoy en día el talento puede encontrarse en cualquier lugar del mundo, el maestro es también alumno y viceversa. La fuerza de la curiosidad por la verdad sigue siendo el motor, pero ahora es un motor nuevo, un motor mejorado que nos permite, con más computación y con más inteligencia artificial, ir más lejos para poder seguir yendo más lejos.

Los algoritmos que se presentan en este libro son representativos de lo que consideramos inteligencia artificial y son fruto del trabajo de rastreadores del saber, que en su momento abrieron un camino que todos nosotros estamos invitados a seguir construyendo. La búsqueda de patrones y relaciones, la clasificación, la segmentación y la regresión son paradigmas que nos servirán para adentrarnos en el apasionante mundo del «dato como unidad básica del conocimiento».

A pesar de todo, sin duda, si el oráculo de Delfos hubiera conocido la inteligencia artificial, Sócrates volvería a ser su apuesta.

Es de agradecer la inspiración del profesor Jordi Berrio durante las entrañables tertulias que mantuvimos en el foyer del Palau de la Música Catalana.

Jordi Gironés Roig

Introducción

El enfoque del libro es claramente descriptivo, con el objetivo de que el lector entienda los conceptos e ideas básicos detrás de cada algoritmo o técnica, sin entrar en excesivos detalles técnicos, tanto matemáticos como algorítmicos. Siempre que es posible, se hace referencia a los trabajos originales que describieron por primera vez el problema a resolver, los cuales pueden ser rastreados para obtener la literatura más actualizada.

Este libro se divide en seis partes principales:

- El primer bloque constituye la introducción a la minería de datos, presentando las etapas de un proceso de minería de datos, así como una clasificación básica de algoritmos y problemas relacionados. A continuación se presentan algunos conceptos matemáticos que son necesarios para el correcto seguimiento de los modelos descritos más adelante. Esta parte finaliza con una breve descripción de las tareas de preparación de datos, previa a todo proceso de minería de datos.

- En la segunda parte, se describen dos conceptos importantes para todo proceso de minería de datos: por un lado la preparación de los conjuntos de entrenamiento

y test, aplicable a los métodos supervisados; y por otro lado las métricas empleadas en la evaluación de los resultados de un modelo de predicción, que permiten comparar el rendimiento de los distintos algoritmos sobre un mismo conjunto de datos.

- La tercera parte presenta los principales métodos relacionados con la selección y extracción de atributos, que permiten reducir el conjunto de datos original para un ahorro en espacio y tiempo de cálculo de los modelos empleados posteriormente en el proceso de minería de datos, así como para la mejora de sus capacidades predictivas en algunos casos.

- El cuarto bloque de este libro se centra en los métodos de aprendizaje no supervisado, es decir, aquellos métodos que utilizan conjuntos de datos no etiquetados. En concreto, se verán algoritmos jerárquicos, el método *k-means* y sus derivados, y el concepto y proceso de *preclustering* empleando el algoritmo Canopy.

- De forma complementaria, en el quinto bloque se describen los principales métodos de aprendizaje supervisado, formado por algoritmos que requieren un conjunto de entrenamiento etiquetado con las clases o variables objetivo del análisis. En esta parte se verá el algoritmo k-NN, las máquinas de soporte vectorial, las redes neuronales, los árboles de decisión y los métodos probabilísticos.

- Finalmente, la sexta parte del libro está dedicada a la técnica de combinación de clasificadores, que se basa en combinar distintos modelos de minería de datos, tra-

bajando en serie o en paralelo, para obtener un mejor resultado de forma global.

En los distintos capítulos de este libro se describen algunos ejemplos sencillos, con el objetivo de que puedan ser interpretados de forma fácil y ayuden en la comprensión de los detalles teóricos de los métodos descritos. En el libro, sin embargo, no se incluye el código asociado a estos ejemplos, dado que este puede sufrir cambios debido a las actualizaciones de librerías y versiones de los lenguajes de programación empleados. En su lugar, se proporciona una página web que permite la descarga de los ejemplos actualizados, ya sea algunos de los descritos en el libro, como también otros ejemplos más complejos que permiten ver la aplicación real de los algoritmos presentados en este texto. El detalle de los ejemplos, así como el enlace para su descarga, se puede encontrar en el apéndice VII.

Jordi Gironés, Jordi Casas,
Julià Minguillón y Ramon Caihuelas

Parte I

Introducción

Capítulo 1

Introducción a la minería de datos

Los juegos de datos encierran estructuras, patrones y reglas de los que es posible extraer conocimiento sobre los eventos que los han generado. Los física teórica más avanzada, cada vez más, habla de un universo de eventos y no de partículas, de modo que tratar de entender o incluso predecir estos eventos se ha convertido en un reto para muchas disciplinas y la razón de ser para la minería de datos, que ha pasado de tratar de entender los datos a tratar de comprender los eventos que hay detrás.

Las cosas no son como aparentan ser, por esta razón técnicas como la visualización de datos, aunque necesarias, son del todo insuficientes para llegar hasta el conocimiento que se esconde detrás de estructuras y relaciones no triviales en los juegos de datos.

Esta nueva visión convierte necesariamente los equipos de analistas de datos en equipos interdisciplinares donde se requieren habilidades matemáticas e informáticas, pero también

conocedoras del negocio y de la organización empresarial. Solo así se podrá cubrir el proceso de extracción de conocimiento de principio a fin.

Por medio de una metodología adecuada y de entender tanto qué tipo de problemas trata de resolver, como con qué técnicas hace frente a estos retos, trataremos de ofrecer una visión lo más completa posible de lo que es hoy en día la minería de datos.

1.1. Etapas de un proyecto DM

Desde las organizaciones de hoy en día, nos enfrentamos a proyectos complejos con multitud de tareas interdisciplinares e interdependientes, que además mezclan intereses y necesidades de diferentes grupos de personas y que normalmente están condicionados por limitaciones económicas y tecnológicas.

Lo recomendable en estos casos es diseñar una hoja de ruta que nos va a permitir saber dónde estamos, dónde queremos llegar y las medidas a tomar para corregir periódicamente las desviaciones del rumbo seguido.

La metodología CRISP-DM nació en el seno de dos empresas, DaimlerChrysler y SPSS, que en su día fueron pioneras en la aplicación de técnicas de minería de datos (en inglés, *data mining*) en los procesos de negocio. CRISP-DM se ha convertido *de facto* en la metodología del sector. Su éxito se debe a que está basada en la práctica y experiencia real de analistas de minería de datos que han contribuido activamente al desarrollo de la misma.

Veremos que hay dos aspectos clave en esta metodología: la adopción de la estrategia de calidad total y la visión de un proyecto de minería de datos como una secuencia de fases.

1.1.1. Calidad total

El compromiso con la calidad en el mundo de la gestión de proyectos pasa por seguir de forma iterativa lo que se conoce como ciclo de Deming o ciclo PDCA:

- **Planificar (*Plan*)**: establecer los objetivos y los procesos necesarios para proporcionar resultados de acuerdo con las necesidades del cliente y con las políticas de la empresa.
- **Hacer (*Do*)**: implementar los procesos.
- **Verificar (*Check*)**: monitorizar y medir los procesos y los servicios contrastándolos con las políticas, los objetivos y los requisitos, e informar sobre los resultados.
- **Actuar (*Act*)**: emprender las acciones necesarias para mejorar continuamente el rendimiento y comportamiento del proceso.

Un aspecto a destacar es que la iteración y revisión de fases y procesos se remarca como un aspecto clave si se quiere ejecutar un proyecto de calidad.

De este modo se establecen microciclos de planificación, ejecución y revisión, de los que solo se sale cuando el proceso de revisión es satisfactorio. Este principio está muy presente tanto en la norma ISO 9000 como en la ISO 20000.

Todas las fases son importantes, por supuesto, pero cabe subrayar que la tendencia natural de la condición humana, por experiencia propia, es la de concentrar recursos en exceso al final del proyecto, en la fase de despliegue, por no haber hecho las cosas bien en las fases anteriores.

Merece la pena y es más óptimo y económico no escatimar recursos en las fases iniciales de preparación, planificación, construcción e iteración.

Conviene mencionar también que la metodología debe ser entendida siempre como una guía de trabajo que permite garantizar calidad en la entrega del proyecto. Para conseguir que efectivamente sea una guía de trabajo útil y práctica, deberemos adaptarla a las necesidades, limitaciones y urgencias que en cada momento tengamos.

Vamos a estudiar todas las fases que nos propone la metodología CRISP-DM. Observad que en el centro del esquema que la resume se encuentra el objetivo de la misma, es decir, la conversión de los datos en conocimiento.

La figura 1.1 esquematiza el ciclo de fases que propone CRISP-DM.

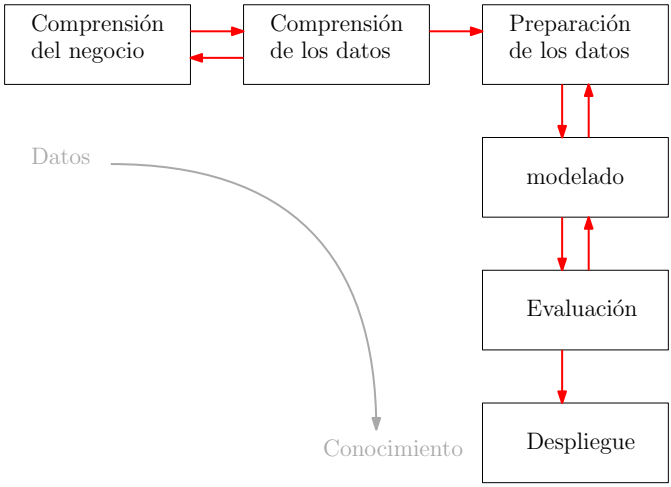


Figura 1.1. Fases de la metodología CRISP-DM

1.1.2. Comprensión del negocio

En esta fase trataremos de descubrir, desde una perspectiva de negocio, cuáles son los objetivos del mismo, tratando de evitar el gran error de dedicar el esfuerzo de todo el proyecto a proporcionar respuestas correctas a preguntas equivocadas.

Con los objetivos de negocio en mente, elaboraremos un estudio de la situación actual del negocio respecto de los objetivos planteados. En este punto, trataremos de clarificar recursos, requerimientos y limitaciones, para así poder concretar objetivos de la minería de datos que contribuyan claramente a la consecución de los objetivos primarios.

Finalmente, elaboraremos un plan de proyecto en el que detallaremos las fases, tareas y actividades que nos deberán llevar a alcanzar los objetivos planteados.

En esta fase deberemos ser capaces de:

- Establecer los objetivos de negocio.
- Evaluar la situación actual.
- Fijar los objetivos a nivel de minería de datos.
- Obtener un plan de proyecto.

1.1.3. Comprensión de los datos

Comprensión se refiere a trabajar los datos con el objetivo de familiarizarse al máximo con ellos, saber de dónde provienen, en qué condiciones nos llegan, cuál es su estructura, qué propiedades tienen, qué inconvenientes presentan y cómo podemos mitigarlos o eliminarlos.

Se trata de una fase crítica puesto que es donde trabajamos de lleno con la calidad de los datos, que además debemos ver como la materia prima para la minería de datos.

Tener una buena calidad de los datos será siempre una condición necesaria aunque no suficiente para tener éxito en el proyecto.

En esta fase deberemos ser capaces de:

- Ejecutar procesos de captura de datos.
- Proporcionar una descripción del juego de datos.
- Realizar tareas de exploración de datos.
- Gestionar la calidad de los datos, identificando problemas y proporcionando soluciones.

1.1.4. Preparación de los datos

El objetivo de esta fase es disponer del juego de datos final sobre el que se aplicarán los modelos. Además, también se desarrollará la documentación descriptiva necesaria sobre el juego de datos.

Deberemos dar respuesta a la pregunta: ¿qué datos son los más apropiados para alcanzar los objetivos marcados? Esto significa evaluar la relevancia de los datos, la calidad de los mismos y las limitaciones técnicas que se puedan derivar de aspectos como el volumen de datos. Documentaremos los motivos tanto para incluir datos como para excluirlos.

Nos replantearemos los criterios de selección de datos basándonos, por un lado, en la experiencia adquirida en el proceso de exploración de datos, y por otro lado, en la experiencia adquirida en el proceso de modelado.

Consideraremos el uso de técnicas estadísticas de muestreo y técnicas de relevancia de atributos, que nos ayudarán, por ejemplo, a plantear la necesidad de iniciar actividades de reducción de la dimensionalidad.

Prestaremos atención a la incorporación de datos de diferentes fuentes y por supuesto a la gestión del ruido.

En esta fase deberemos ser capaces de:

- Establecer el universo de datos con los que trabajar.
- Realizar tareas de limpieza de datos.
- Construir un juego de datos apto para ser usado en modelos de minería de datos.
- Integrar datos de fuentes heterogéneas si es necesario.

1.1.5. Modelado

El objetivo último de esta fase será el de disponer de un modelo que nos ayude a alcanzar los objetivos de la minería de datos y los objetivos de negocio establecidos en el proyecto. Podemos entender el modelo como la habilidad de aplicar una técnica a un juego de datos con el fin de predecir una variable objetivo o encontrar un patrón desconocido.

El hecho de que esta fase entre en iteración tanto con su antecesora, la preparación de los datos, como con su sucesora, la evaluación del modelo, nos da una idea de la importancia de la misma en términos de la calidad del proyecto.

Dado un problema en el ámbito de la minería de datos, pueden existir una o varias técnicas que den respuesta al mismo, por ejemplo:

- Un problema de segmentación puede aceptar técnicas de *clustering*, de redes neuronales o simplemente técnicas de visualización.

- Un problema de clasificación puede aceptar técnicas de análisis discriminante, de árboles de decisión, de redes neuronales, máquinas de soporte vectorial o de k-NN.
- Un problema de predicción aceptará técnicas de análisis de regresión, de árboles de regresión, de redes neuronales o de k-NN.
- Un problema de análisis de dependencias puede afrontarse con técnicas de análisis de correlaciones, análisis de regresión, reglas de asociación, redes bayesianas o técnicas de visualización.

En definitiva, un mismo problema puede resolverse con varias técnicas y una técnica puede servir para resolver varios problemas.

En esta fase deberemos ser capaces de:

- Seleccionar las técnicas de modelado más adecuadas para nuestro juego de datos y nuestros objetivos.
- Fijar una estrategia de verificación de la calidad del modelo.
- Construir un modelo a partir de la aplicación de las técnicas seleccionadas sobre el juego de datos.
- Ajustar el modelo evaluando su fiabilidad y su impacto en los objetivos anteriormente establecidos.

1.1.6. Evaluación del modelo

En fases anteriores nos hemos preocupado de asegurar la fiabilidad y plausibilidad del modelo; en cambio, en esta fase

nos centraremos en evaluar el grado de acercamiento a los objetivos de negocio y en la búsqueda, si las hay, de razones de negocio por las cuales el modelo es ineficiente.

Una forma esquemática y gráfica de visualizar el propósito de un proyecto de minería de datos es pensar en la siguiente ecuación:

$$\text{Resultados} = \text{Modelos} + \text{Descubrimientos}$$

Es decir, el propósito de un proyecto de minería de datos no son solo los modelos, que son por supuesto importantes, sino también los descubrimientos, que podríamos definir como cualquier cosa aparte del modelo que contribuye a alcanzar los objetivos de negocio o que contribuye a plantear nuevas preguntas, que a su vez son decisivas para alcanzar los objetivos de negocio.

Siempre y cuando sea posible probaremos el modelo en entornos de prueba para asegurarnos de que el posterior proceso de despliegue se realiza satisfactoriamente y para asegurarnos también de que el modelo obtenido es capaz de dar respuesta a los objetivos de negocio.

Estableceremos un ranquin de resultados con respecto a los criterios de éxito con relación al grado de cumplimiento de los objetivos de negocio.

Adicionalmente, también emitiremos opinión sobre otros descubrimientos que se hayan realizado aparte del modelado que, aunque probablemente no contribuyan directamente a los objetivos planteados, quizá puedan abrir puertas a nuevos planteamientos y líneas de trabajo.

En esta fase deberemos ser capaces de:

- Evaluar el modelo o modelos generados hasta el momento.

- Revisar todo el proceso de minería de datos que nos ha llevado hasta este punto.
- Establecer los siguientes pasos a tomar, tanto si se trata de repetir fases anteriores como si se trata de abrir nuevas líneas de investigación.

1.1.7. Despliegue

En esta fase organizaremos y ejecutaremos tanto las tareas propias del despliegue de los resultados como del mantenimiento de las nuevas funcionalidades, una vez el despliegue haya finalizado.

El plan deberá contemplar todas las tareas a realizar en el proceso de despliegue de resultados, e incorporará medidas alternativas en forma de planes alternativos o versiones del plan inicial, que deberán permitir tener varias visiones y escoger la mejor.

Deberemos definir cómo el conocimiento obtenido en forma de resultados será propagado hacia los usuarios interesados.

En el caso de que haya que instalar o distribuir software por nuestros sistemas, deberemos gestionarlo para minimizar posibles efectos negativos y planificarlo para que se ejecute con suficiente antelación.

Habrá que prever cómo mediremos el beneficio producido por el despliegue y cómo monitorizaremos todo el proceso.

Identificaremos los posibles inconvenientes que pueda ocasionar nuestro despliegue.

En esta fase deberemos ser capaces de:

- Diseñar un plan de despliegue de modelos y conocimiento sobre nuestra organización.

- Realizar seguimiento y mantenimiento de la parte más operativa del despliegue.
- Revisar el proyecto en su globalidad con el objetivo de identificar lecciones aprendidas.

1.2. Tipología de tareas y métodos

En esta sección presentaremos, en primer lugar, las principales tareas que se pueden resolver utilizando técnicas de minería de datos. Veremos las características básicas de estas tareas y los objetivos principales que persiguen. En segundo lugar, describiremos brevemente los dos grandes grupos de métodos que existen en la minería de datos y el aprendizaje automático (*machine learning*).

1.2.1. Tipología de tareas

Las tres tareas o problemas clave de la minería de datos son la clasificación, regresión y agrupamiento. Todas ellas se basan en el paradigma del aprendizaje inductivo. La esencia de cada una de ellas es derivar inductivamente a partir de los datos (que representan la información del entrenamiento) un modelo (que representa el conocimiento) que tiene utilidad predictiva, es decir, que puede aplicarse a nuevos datos.

Clasificación

La clasificación (*classification*) es uno de los procesos cognitivos importantes, tanto en la vida cotidiana como en los negocios, donde podemos clasificar clientes, empleados, transacciones, tiendas, fábricas, dispositivos, documentos o cualquier

otro tipo de instancias en un conjunto de clases o categorías predefinidas con anterioridad.

La tarea de clasificación consiste en asignar instancias de un dominio dado, descritas por un conjunto de atributos discretos o de valor continuo, a un conjunto de clases, que pueden ser consideradas valores de un atributo discreto seleccionado, generalmente denominado clase. Las etiquetas de clase correctas son, en general, desconocidas, pero se proporcionan para un subconjunto del dominio. Por lo tanto, queda claro que es necesario disponer de un subconjunto de datos correctamente etiquetado, y que se usará para la construcción del modelo.

La función de clasificación puede verse como:

$$c : X \to C \tag{1.1}$$

donde c representa la función de clasificación, X el conjunto de atributos que forman una instancia y C la etiqueta de clase de dicha instancia.

Un tipo de clasificación particularmente simple, pero muy interesante y ampliamente estudiado, hace referencia a los problemas de clasificación binarios, es decir, problemas con un conjunto de datos pertenecientes a dos clases, por ejemplo, $C = \{0, 1\}$.

Regresión

Al igual que la clasificación, la regresión (*regression*) es una tarea de aprendizaje inductivo que ha sido ampliamente estudiada y utilizada. Se puede definir, de forma informal, como un problema de *clasificación con clases continuas*. Es decir, los modelos de regresión predicen valores numéricos en lugar de etiquetas de clase discretas. A veces también nos podemos referir a la regresión como *predicción numérica*.

La tarea de regresión consiste en asignar valores numéricos a instancias de un dominio dado, descritos por un conjunto de atributos discretos o de valor continuo. Se supone que esta asignación se aproxima a alguna función objetivo, generalmente desconocida, excepto para un subconjunto del dominio. Este subconjunto se puede utilizar para crear el modelo de regresión.

En este caso, la función de regresión se puede definir como:

$$f : X \to \mathbb{R} \tag{1.2}$$

donde f representa la función de regresión, X el conjunto de atributos que forman una instancia y $\mathbb{R}$ un valor en el dominio de los números reales.

Agrupamiento

El agrupamiento (*clustering*) es una tarea de aprendizaje inductiva que, a diferencia de las tareas de clasificación y regresión, no dispone de una etiqueta de clase a predecir. Puede considerarse como un problema de clasificación, pero donde no existen un conjunto de clases predefinidas, y estas se «descubren» de forma autónoma por el método o algoritmo de agrupamiento, basándose en patrones de similitud identificados en los datos.

La tarea de agrupamiento consiste en dividir un conjunto de instancias de un dominio dado, descrito por un número de atributos discretos o de valor continuo, en un conjunto de grupos (clústeres) basándose en la similitud entre las instancias, y crear un modelo que puede asignar nuevas instancias a uno de estos grupos o clústeres.

Un proceso de agrupación puede proporcionar información útil sobre los patrones de similitud presentes en los datos

como, por ejemplo, segmentación de clientes o creación de catálogos de documentos. Otra de las principales utilidades del agrupamiento es la detección de anomalías. En este caso, el método de agrupamiento permite distinguir instancias que con un patrón absolutamente distinto a las demás instancias «normales» del conjunto de datos, facilitando la detección de anomalías y posibilitando la emisión de alertas automáticas para nuevas instancias que no tienen ningún clúster existente.

La función de agrupamiento o *clustering* se puede modelar mediante:

$$h : X \rightarrow C_h \tag{1.3}$$

donde h representa la función de agrupamiento, X el conjunto de atributos que forman una instancia y C_h un conjunto de grupos o clústeres. Aunque esta definición se parece mucho a la tarea de clasificación, una diferencia aparentemente pequeña pero muy importante consistente en que el conjunto de «clases» no está predeterminado ni es conocido *a priori*, sino que se identifica como parte de la creación del modelo.

1.2.2. Tipología de algoritmos

Los algoritmos de minería de datos se dividen en dos grandes grupos:

- Los métodos supervisados, que requieren de un conjunto de datos previamente etiquetado con el conjunto de clases.
- Los métodos no supervisados, donde los datos no tienen ninguna etiqueta o clasificación previa.

Veamos a continuación las principales características, diferencias y métodos de estos dos grandes grupos.

Métodos supervisados

Los métodos supervisados (*supervised methods*) son algoritmos que basan su proceso de aprendizaje en un juego de datos de entrenamiento convenientemente etiquetado. Por etiquetado entendemos que para cada ocurrencia del juego de datos de entrenamiento conocemos el valor de su atributo objetivo o clase. Esto le permitirá al algoritmo poder deducir una función capaz de predecir el atributo objetivo para un juego de datos nuevo.

Las grandes familias de algoritmos de aprendizaje supervisado son:

- Algoritmos de clasificación, indicados cuando el atributo objetivo es categórico.
- Algoritmos de regresión, indicados cuando el atributo objetivo es numérico.

Estos dos grandes grupos corresponden, en esencia, con la tipología de problemas descrita en la sección anterior.

Métodos no supervisados

Los métodos no supervisados (*unsupervised methods*) son algoritmos que basan su proceso de entrenamiento en un juego de datos sin etiquetas o clases previamente definidas. Es decir, *a priori* no se conoce ningún valor objetivo o de clase, ya sea categórico o numérico.

El aprendizaje no supervisado está dedicado a las tareas de agrupamiento, también llamadas *clustering* o segmentación,

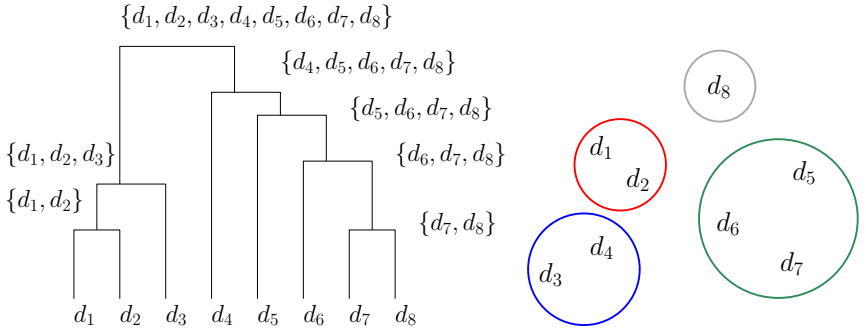
donde su objetivo es encontrar grupos similares en los juegos de datos.

Existen dos grupos principales de métodos o algoritmos de agrupamiento:

1. Los métodos jerárquicos, que producen una organización jerárquica de las instancias que forman el conjunto de datos, posibilitando de esta forma distintos niveles de agrupación.

2. Los métodos particionales o no jerárquicos, que generan grupos de instancias que no responden a ningún tipo de organización jerárquica.

En la figura 1.2 podemos distinguir de forma visual la diferencia entre los métodos de agrupamiento jerárquico y los métodos no jerárquicos o particionales. La principal diferencia es que los métodos jerárquicos permiten escoger el nivel de granularidad, es decir, dependiendo de la altura se obtiene un distinto número de conjuntos con distinto número de elementos.



(a) Agrupamiento jerárquico **(b)** Agrupamiento particional

Figura 1.2. Tipologías de métodos de agrupamiento

Ambos métodos proporcionan una forma directa de descubrir y representar la similitud en los datos, formando conjuntos de datos «similares» entre ellos. El concepto de «similitud» deberá ser escogido para cada problema y representado según una métrica concreta. Una vez creados los grupos, estos algoritmos también pueden predecir a qué conjunto corresponde una nueva instancia, lo que les permite implementar análisis predictivos (*predictive analysis*). Para este tipo de tarea se suele emplear la misma medida de similitud entre la nueva instancia y los conjuntos definidos.

Métodos	**Supervis.**		**No super.**
	Clasificación	Regresión	Agrupamiento
Agrupamiento jerárquico			X
k-means y derivados			X
k-NN	X		
SVM	X	X	
Redes neuronales	X	X	
Árboles de decisión	X	X	
Métodos probabilísticos	X	X	

Cuadro 1.1. Tipología de los algoritmos de minería de datos

Finalmente, el cuadro 1.1 identifica los principales métodos de minería de datos, indicando en cada caso la tipología de algoritmo y de la tarea o problema que puede resolver. En los siguientes capítulos veremos en detalle cada uno de estos métodos.

Capítulo 2

Conceptos preliminares

En este capítulo revisaremos algunos conceptos matemáticos que son especialmente útiles en los algoritmos de minería de datos y que, por lo tanto, es interesante conocer antes de iniciar el estudio de los diferentes métodos o algoritmos.

En primer lugar, veremos muy brevemente los conceptos más básicos del análisis estadístico, en la sección 2.1. A continuación, en la sección 2.2 repasaremos algunas de las métricas de distancia o similitud más empleadas por los algoritmos de minería de datos. Finalizaremos este repaso de conceptos preliminares con una breve reseña de la entropía y la ganancia de información, en la sección 2.3.

2.1. Análisis estadístico

La estadística mira a los datos y trata de analizarlos, por este motivo la minería de datos utiliza tantos conceptos que provienen de este campo de conocimiento. La recogida de muestras, la exploración de los datos, la inferencia estadística y la búsqueda de patrones forman parte de ambas disciplinas.

2.1.1. La distribución de una muestra

A continuación veremos las bases estadísticas que pueden ayudarnos en la minería de datos, centrándonos en el concepto de distribución con el objetivo de introducirnos en el ámbito de conocimiento de la estadística muestral.

Principales estimadores

La estadística entiende los datos como observaciones de una variable. Generalmente, se suele trabajar con el alfabeto griego y latino[1] más que con números, de modo que se suele representar una variable con la mayúscula X y sus observaciones con la minúscula $X = \{x_1, x_2, \ldots, x_n\}$.

La distribución de una variable son las propiedades de los valores observados como, por ejemplo, los valores mayor y menor o los valores con mayor o menor frecuencia de aparición. Un aspecto interesante en una distribución es el estudio de su zona central. Es decir, los valores que aproximadamente tienen la mitad de las observaciones por debajo (con menos frecuencia) y la mitad por arriba (con más frecuencia). Aquí es donde encontramos los dos estimadores más básicos: la mediana y la media aritmética.

La mediana (*median*) corresponde al valor central de la distribución, que podemos expresar como:

$$M_e = x_{\frac{n+1}{2}} \tag{2.1}$$

donde n es el número total de observaciones.

La media aritmética (*arithmetic mean*) de un conjunto de datos numéricos es su valor medio, representado por:

[1]Normalmente se utiliza el alfabeto griego para hacer referencia a la población y el latino para la muestra.

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.2}$$

Tanto la mediana como la media aritmética miden el centro de una distribución, pero lo hacen de forma distinta. Solo cuando la distribución es simétrica, ambos valores coinciden. Los valores asimétricos y alejados atraen la media, mientras que no afectan a la mediana. Esto hace que la media por sí sola no sea un estimador suficiente para describir la distribución de una variable.

La media aritmética, en ocasiones, puede llegar a ser muy poco representativa del juego de datos. Por este motivo suele ir acompañada de la varianza, donde una varianza pequeña indica que el juego de datos es compacto y en consecuencia la media será muy representativa.

La varianza, entendida como la suma de los cuadrados de las distancias a la media, permite medir la dispersión alrededor de la media aritmética. Se calcula de la siguiente forma:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1} \tag{2.3}$$

La desviación estándar es simplemente la raíz cuadrada de la varianza, de modo que nos da una medida de la dispersión en torno a la media, pero expresada en la misma escala que la variable. Su fórmula de cálculo es:

$$s = \sqrt{s^2} \tag{2.4}$$

La importancia de la desviación estándar, como medida de dispersión de la distribución de una variable, radica en el he-

cho de que la *mayoría*[2] de las desviaciones respecto de la media caen dentro de un intervalo igual a una o dos veces la desviación estándar, es decir, $(\overline{x}-s, \overline{x}+s)$ o $(\overline{x}-2s, \overline{x}+2s)$, como se puede ver en la figura 2.1.

La distribución normal

La curva de densidad es la representación gráfica de la distribución de una variable suponiendo que fuéramos capaces de obtener muchísimas observaciones. Una distribución normal[3] cumple la propiedad de que su curva de densidad es aproximadamente simétrica, como se puede ver en la figura 2.1, y por lo tanto la mediana y la media aritmética coinciden en su centro. La fórmula matemática de la distribución normal viene dada por:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.5}$$

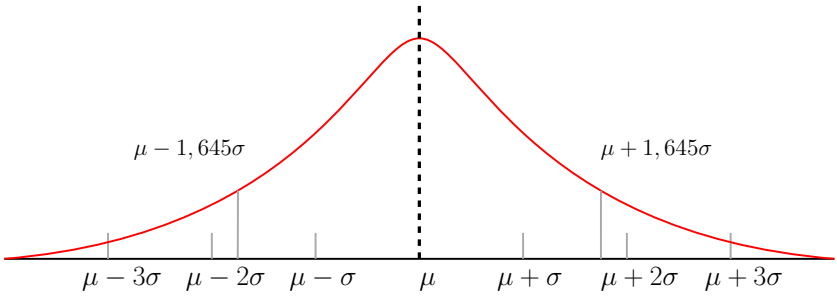


Figura 2.1. Curva de densidad de una distribución normal

[2]El concepto de la *mayoría* lo definiremos con el estudio de la distribución normal.

[3]La distribución normal también es conocida como distribución de Gauss o distribución gaussiana.

Debido a las propiedades especiales que posee la distribución normal, reservamos letras específicas para referirnos a su media y a su desviación estándar:

- La letra μ se utiliza para referirse a la media aritmética de una distribución normal.
- La letra σ se utiliza para referirse a la desviación estándar de una distribución normal.

Si nos fijamos en estos intervalos, sucede que:

- En toda distribución normal, el área debajo de la curva para el intervalo de una desviación estándar es de 0,68. Es decir, en el intervalo $(\mu - \sigma, \mu + \sigma)$ encontraremos el 68 % de la población.
- En el intervalo $(\mu - 1{,}645\sigma, \mu + 1{,}645\sigma)$ encontramos el 90 % de la población.
- En el intervalo $(\mu - 2\sigma, \mu + 2\sigma)$ encontraremos el 95 % de la población.
- Y finalmente, en el intervalo $(\mu - 3\sigma, \mu + 3\sigma)$ encontraremos el 99,7 % de la población.

La distribución normal estándar

La distribución normal estándar es una distribución normal que tiene una media aritmética con valor 0 y una desviación estándar con valor igual a 1.

Es interesante, dada una variable, poder estandarizarla, es decir, realizar los cálculos de probabilidades (áreas debajo de

la curva de densidad) necesarios y regresar de nuevo a la variable original con la traducción de las probabilidades calculadas. Para realizar este proceso, partiremos de una variable X normalmente distribuida, con media aritmética μ y desviación estándar σ, y la estandarizaremos restando a cada observación la media aritmética y dividiendo el resultado por la desviación estándar. El resultado será la variable normal estandarizada, que se calcula de la siguiente forma:

$$Z = \frac{X - \mu}{\sigma} \tag{2.6}$$

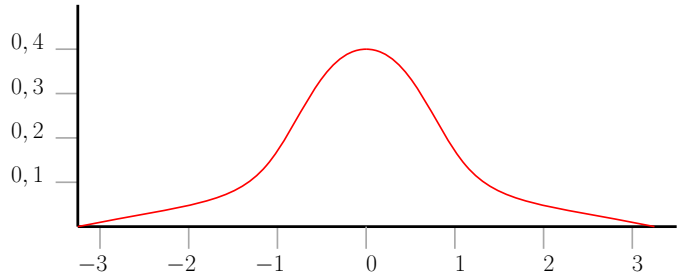


Figura 2.2. Curva de densidad de una distribución normal estándar

Si llamamos P a la probabilidad o área debajo de la curva de densidad, viendo la figura 2.2 y consultando las tablas de densidad de la distribución normal estándar sabemos que, al ser una distribución simétrica centrada en el cero se cumple que:

$$P(Z < -1) = P(Z > 1) = 0{,}1587 \tag{2.7}$$

y por deducción, como todo el área bajo la curva de dicha distribución es 1, podemos calcular la probabilidad de que un valor de Z se encuentre en la zona central, que corresponde a:

$$\begin{aligned} P(-1 < Z < 1) &= 1 - P(Z < -1) - P(Z > 1) \\ &= 1 - 2 \times 0{,}1587 = 0{,}6826 \end{aligned} \tag{2.8}$$

Para realizar el camino inverso, es decir, de la distribución normal estándar a la distribución normal original, usaremos la siguiente expresión:

$$X = \mu + Z\sigma \tag{2.9}$$

2.2. Métricas de distancias o similitud

Antes de adentrarnos en la sección de algoritmos, nos centraremos en los conceptos de distancia y similitud (o disimilitud) puesto que estos constituyen la base sobre la que se construyen muchos de los algoritmos de segmentación y clasificación.

La elección de una métrica apropiada para la distancia tiene mucho impacto en el resultado final de cualquier algoritmo de segmentación. Es más, la elección de la métrica de distancia adecuada suele ser uno de los aspectos clave en la construcción de este tipo de algoritmos.

Cuando hablamos de distancia, en realidad nos estamos refiriendo a una forma de cuantificar cuán similares (o disimilares) son dos objetos, variables o puntos.

Planteado de esta forma, el concepto de distancia es muy abstracto y etéreo, por este motivo los científicos quieren ponerle algunos límites o condiciones. Para un conjunto de elementos X se considera distancia a cualquier función $f : X \times X \to \mathbb{R}$ que cumpla las siguientes tres condiciones:

- No negatividad: la distancia entre dos puntos siempre debe ser positiva.

$$d(a,b) \geq 0 \quad \forall a,b \in X \tag{2.10}$$

- Simetría: la distancia entre un punto a y un punto b debe ser igual a la distancia entre el punto b y el punto a.

$$d(a,b) = d(b,a) \quad \forall a,b \in X \tag{2.11}$$

- Desigualdad triangular: la distancia debe coincidir con la idea intuitiva que tenemos de ella en cuanto a que es el camino más corto entre dos puntos.

$$d(a,c) \leq d(a,b) + d(b,c) \quad \forall a,b,c \in X \tag{2.12}$$

Veamos, a continuación, algunas de las métricas de distancia o similitud más empleadas por los algoritmos de minería de datos.

Distancia euclídea

La distancia euclídea o euclidiana es una de las distancias más utilizadas, especialmente en el caso de tratar con atributos numéricos. La distancia euclídea coincide plenamente con lo que nuestra intuición entiende por «distancia». Su expresión matemática para un espacio de m dimensiones es:

$$d(x,y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2} \tag{2.13}$$

Uno de sus principales inconvenientes se presenta al utilizar esta distancia en un proceso de segmentación, donde este

cálculo de la distancia no tiene en cuenta las diferentes unidades de medida en las que pueden estar expresadas la variables x e y. Por ejemplo, si estamos segmentando familias o razas de ratones, donde la variable x representa la edad de los ratones y la variable y representa la longitud de la cola del ratón (en milímetros), entonces la variable x tomará valores en el rango $[0, 3]$, mientras que la variable y utilizará el rango $[90, 120]$. Parece claro que cuando calculemos la distancia o similitud entre dos individuos, pesará injustamente mucho más la longitud de la cola que la edad.

Distancia estadística o de Gauss

Para superar la distorsión provocada por las diferentes unidades de medida usadas en las distintas variables, tenemos la distancia estadística, que simplemente normaliza la variables para situarlas todas bajo la misma escala.

Su expresión analítica para un espacio de m dimensiones, en la que nuestra distribución de puntos en estas dimensiones tuviera una desviación estándar σ, viene dada por la fórmula:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} \left(\frac{x_i - y_i}{\sigma_i}\right)^2} \tag{2.14}$$

Nuevamente, este concepto de distancia sigue teniendo problemas. No tiene en cuenta la correlación entre las variables, es decir, si nuestras variables de trabajo fueran totalmente independientes, no habría ningún problema, pero si tienen algún tipo de correlación, entonces una influye sobre la otra y esta «influencia» no queda bien reflejada si usamos la distancia estadística.

Por ejemplo, las variables *entrenar* y *rendimiento* están correlacionadas, de modo que más entrenamiento siempre implica más rendimiento, pero esta regla no se cumple infinitamente, ya que entrenamiento infinito no implica rendimiento infinito (lo vemos continuamente en el deporte). Si no tenemos en cuenta esta correlación y queremos comparar a dos deportistas (medir su distancia) podemos llegar a conclusiones erróneas.

Distancia de Mahalanobis

Esta métrica pretende corregir la distorsión provocada por la correlación de las variables mediante la siguiente expresión para un espacio de m dimensiones es:

$$d(x,y) = \sqrt{(x_1 - y_1, \ldots, x_m - y_m) \begin{pmatrix} \sigma_x^2 & \sigma(x,y) \\ \sigma(x,y) & \sigma_y^2 \end{pmatrix}^{-1} \begin{pmatrix} x_1 - y_1 \\ x_m - y_m \end{pmatrix}} \tag{2.15}$$

donde σ_x^2 y σ_y^2 representan la varianza de las variables x e y, respectivamente, y $\sigma(x,y)$ la covarianza entre ambas variables.

La distancia de Mahalanobis es una generalización de la distancia de Gauss, la cual, a su vez, es una generalización de la distancia Euclídea:

- Si en la distribución de puntos, las m dimensiones son totalmente independientes, tendremos que su covarianza es 0, de manera que la distancia de Mahalanobis coincide con la distancia estadística.
- Si en la misma distribución con variables independientes tenemos que su distribución está normalizada, es decir,

que las dos variables tienen la misma escala, entonces su varianza es 1, de manera que la distancia estadística coincide con la distancia euclídea.

Distancia de Hamming

Hasta ahora hemos revisado algunas de las distancias más importantes para lidiar con valores numéricos. En el caso de tratar con atributos nominales o binarios, una de las métricas más utilizadas es la distancia de Hamming. Esta métrica se define como el número de atributos iguales que existen entre dos vectores de atributos.

$$d(x, y) = \sum_{i=1}^{m} \delta(x_i, y_i) \tag{2.16}$$

donde $\delta(x_i, y_i)$ toma el valor 0 si $x_i = y_i$ y 1 en caso contrario.

El lector interesado en otras métricas posibles adaptadas a la naturaleza de los datos puede consultar el trabajo de Cha [5], por ejemplo.

2.3. Entropía y ganancia de información

En la base conceptual de varios métodos de minería de datos se encuentra el concepto de ganancia de la información, como hilo heurístico que puede guiar el proceso de aprendizaje.

Empezaremos por introducir la idea de entropía (*entropy*) como medida de cómo de predecible es un resultado en un juego de datos. También puede ser pensada como el grado de desorden o de incertidumbre presente en un juego de datos. Su expresión matemática es la siguiente:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.17}$$

donde X es una variable aleatoria discreta con un conjunto de posibles valores $X = \{x_1, \ldots, x_n\}$ y $p(x_i)$ es la probabilidad de que la variable X tome el valor x_i.

Valores de entropía altos indican que el resultado es muy aleatorio y en consecuencia, poco predecible. Veamos, como ejemplo, los siguientes experimentos:

- Tirar un dado con 6 caras al aire puede darnos 6 posibles resultados, $X \in \{1, 2, 3, 4, 5, 6\}$. La entropía de este experimento es:

$$H(X) = -6 \cdot \left(\frac{1}{6} \log_2 \left(\frac{1}{6}\right)\right) \approx 2{,}58$$

- Tirar una moneda al aire puede darnos 2 posibles resultados, $X \in \{cara, cruz\}$. La entropía de este experimento es:

$$H(X) = -2 \cdot \left(\frac{1}{2} \log_2 \left(\frac{1}{2}\right)\right) = 1$$

Observamos cómo la entropía de lanzar un dado con 6 caras es mucho más alta y, por lo tanto, mucho más aleatoria que la entropía de lanzar una moneda.

La ganancia de información (*information gain*) nos da una medida de cómo de relevante es un atributo dentro de un juego de datos, de modo que un atributo con mucha ganancia será muy relevante en el juego de datos, es decir, muy determinante para predecir el atributo objetivo o clase.

La ganancia de información refleja el cambio en la entropía del juego de datos cuando tomamos parte de la información como dada. Dicho de otro modo, la ganancia de información del atributo a respecto del atributo objetivo x es la diferencia entre la entropía del atributo objetivo y la entropía del atributo a:

$$IG(X, a) = H(X) - H(X, a) \tag{2.18}$$

$$H(X, a) = \sum_{v \in valores(a)} p(v) H(\{x \in X \mid valor(x, a) = v\}) \tag{2.19}$$

donde $H(X, a)$ es la entropía del juego de datos cuando lo particionamos en base al atributo a, o visto de otro modo, es la entropía del atributo a en el juego de datos.

En la expresión anterior, el valor $H(\{x \in X \mid valor(x, a) = v\})$ es la entropía del juego de datos cuando fijamos el atributo a al valor v o, visto de otro modo, es la entropía del atributo a cuando toma el valor v en el juego de datos X.

Capítulo 3

Preparación de los datos

Las tareas o técnicas de preparación de datos en un proyecto de minería de datos se orientan a la adecuación del juego de datos para que pueda ser usado, posteriormente, por algoritmos de clasificación, segmentación o regresión.

Estas tareas las podemos agrupar en los siguientes bloques temáticos:

- Tareas de limpieza de datos, que permiten corregir o eliminar ruido o datos no válidos.
- Tareas de normalización de datos, que facilita la presentación de los datos en el mismo rango.
- Tareas de discretización, entendidas como procesos de conversión de variables continuas a categóricas.
- Tareas de reducción de la dimensionalidad, que nos ayudará a desarrollar modelos con juegos de datos reducidos.

3.1. Limpieza de datos

En el proceso de limpieza de datos (en inglés, *data cleansing* o *data scrubbing*) se llevan a cabo actividades de detección, eliminación o corrección de instancias corrompidas o inapropiadas en los juegos de datos.

A nivel de valores de atributos se gestionan los valores ausentes, los erróneos y los inconsistentes. Un ejemplo podrían ser los valores fuera de rango (*outliers*).

El proceso de integración de datos puede ser una de las principales fuentes de incoherencias en los datos. Fruto de la fusión de juegos de datos distintos, se pueden generar inconsistencias que deben ser detectadas y subsanadas.

3.2. Normalización de datos

La normalización de datos consiste en modificar los datos para lograr que estén en una escala de valores equivalentes que simplifique la comparación entre ellos. La normalización es útil para varios métodos de minería de datos, que tienden a quedar sesgados por la influencia de los atributos con valores más altos, distorsionando de esta forma el resultado del modelo.

Algunos de los principales métodos de normalización de atributos son:

- Normalización por el máximo, que consiste en encontrar el valor máximo del atributo a normalizar y dividir el resto de los valores por este valor máximo, asegurándonos, por tanto, que el máximo recibe el valor 1 y el resto valores en el rango $[0, 1]$.

$$z_i = \frac{x_i}{x_{\max}} \tag{3.1}$$

donde z_i es el valor normalizado, x_i el valor original del atributo y $x_{\max}$ el valor máximo para este atributo de todo el conjunto de datos.

- Normalización por la diferencia, que intenta compensar el efecto de la distancia del valor que tratamos con respecto al máximo de los valores observados. Su fórmula de cálculo es:

$$z_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{3.2}$$

donde $x_{\min}$ es el valor mínimo para este atributo en el conjunto de datos.

- Normalización basada en la desviación estándar, también llamada estandarización de valores, que asegura que se obtienen valores dentro del rango elegido que tienen como propiedad que su media es el 0 y su desviación estándar es 1.

$$z_i = \frac{x_i - \mu}{\sigma} \tag{3.3}$$

donde μ representa la media y σ la varianza.

3.3. Discretización de datos

La discretización es el proceso mediante el cual los valores de una variable continua se incluyen en contenedores, intervalos o grupos, con el objetivo de que haya un número limitado de estados posibles. Los contenedores se tratarán entonces como si fueran categorías.

La discretización es una tarea habitual en los procesos de minería de datos puesto que muchos algoritmos de clasificación requieren que sus atributos sean discretizados, bien porque solo aceptan valores nominales, bien porque trabajar con atributos nominales en lugar de continuos disminuye el coste computacional y acelera el proceso inductivo. Además, la discretización de atributos puede, en ocasiones, mejorar el rendimiento de un clasificador.

Los métodos de discretización se pueden clasificar según tres grandes categorías:

- Supervisados o no supervisados. Métodos que tienen en cuenta los valores del atributo clase o no.
- Locales o globales. Métodos limitados a un atributo o, en general, a un subconjunto horizontal (subconjunto de instancias) o vertical (subconjunto de atributos) del conjunto de datos; o bien actuando sobre todos los atributos y todos los datos a la vez.
- Parametrizados y no parametrizados. Los primeros empiezan conociendo el número máximo de intervalos que hay que generar para un atributo específico, mientras que los demás tienen que encontrar este número automáticamente.

3.3.1. Método de igual amplitud

El método de partición en intervalos de la misma amplitud es un método no supervisado que busca crear grupos con el mismo número de instancias en su interior.

El método consiste en los siguientes pasos:

1. Se parte de un atributo con n valores, el cual queremos discretizar en k intervalos de igual longitud.

2. La longitud de cada intervalo será:

$$\alpha = \frac{x_{\max} - x_{\min}}{k} \tag{3.4}$$

donde $x_{\min}$ y $x_{\max}$ representan el valor mínimo y máximo, respectivamente, que toman los atributos.

3. Entonces, los intervalos quedan definidos por la expresión:

$$Intervalo_i = x_{\min} + i\alpha,\ \forall i \in \{1, \ldots, k\} \tag{3.5}$$

Algunos de los principales problemas de este método son:

1. Da la misma importancia a todos los valores, independientemente de su frecuencia de aparición.

2. Puede generar intervalos poco homogéneos, donde se mezclen dentro de un mismo intervalo valores que corresponden a clases diferentes.

3. La elección del parámetro k no es trivial en la mayoría de casos.

3.3.2. Método de igual frecuencia

Este método es similar al método de igual amplitud, pero en este caso se requiere que cada grupo contenga el mismo número de instancias en su interior.

El método consiste en los pasos siguientes:

- Ordenar los valores del atributo X de menor a mayor: $x_1 \leq \ldots \leq x_n$.
- Fijar un número de intervalos k.
- Calcular el valor de la frecuencia de cada intervalo: $f = \frac{n}{k}$.
- Asignar a cada intervalo los f valores ordenados, de tal forma que cada cada intervalo o grupo contendrá f elementos ordenados, donde el valor máximo del intervalo i será menor o igual que el valor mínimo del intervalo $i+1$.

El principal inconveniente de este método es que valores muy dispares pueden ir a parar al mismo intervalo, dado que el criterio es mantener la misma frecuencia en todos los intervalos. Por el contrario, el método funciona correctamente cuando se da una distribución uniforme de los valores del atributo a discretizar.

3.3.3. Método *chi merge*

Chi merge es un algoritmo simple que utiliza el estadístico chi-cuadrado para discretizar los atributos numéricos. Se trata de un método de discretización de datos supervisado.

Este método intenta que dentro de cada intervalo aparezcan representadas todas las clases de forma parecida, y muy diferente respecto de otros intervalos. En términos de frecuencias, eso quiere decir que dentro de un intervalo hay que esperar que la frecuencia de aparición de valores de cada clase sea parecida, y muy diferente de la de otros intervalos.

Si la primera condición no se cumple, quiere decir que tenemos un intervalo demasiado heterogéneo y que sería preciso

subdividirlo. Si no se cumple la segunda, entonces quiere decir que el intervalo es muy parecido a algún otro intervalo adyacente, de manera que, en tal caso, sería necesario unirlos.

Una manera de medir estas propiedades es recurriendo al estadístico chi-cuadrado χ^2 para averiguar si las frecuencias correspondientes a dos intervalos son significativamente diferentes (y, por lo tanto, ya es correcto que los intervalos estén separados), o bien que no lo son (y hay que unirlos).

3.3.4. Métodos basados en entropía

Los métodos basados en medidas de entropía parten de la idea de encontrar particiones del conjunto original de datos que sean internamente tan homogéneas como sea posible, y tan diferentes como sea posible con respecto al resto de las particiones.

Estos métodos trabajan sobre variables continuas con el objetivo de crear un punto de decisión (llamado límite o umbral) sobre un atributo determinado tal que permita alcanzar la mejor separación de los datos en dos subconjuntos disjuntos.

Se pueden definir distintos criterios para escoger la partición «óptima» en cada iteración del algoritmo, pero una de las métricas más empleadas para tal fin es la entropía de la información asociada a cada posible partición. Es decir, se toma como criterio de discretización el valor que minimice la entropía de los subconjuntos generados utilizando ese punto de corte.

3.4. Reducción de la dimensionalidad

En algunos procesos de minería de datos podemos encontrarnos con el problema de que el método de construcción

del modelo elegido no puede tratar la cantidad de datos de que disponemos, o simplemente que el tiempo requerido para construir el modelo no es adecuado para nuestras necesidades.

En estos casos, es posible aplicar una serie de operaciones con el fin de facilitar dos objetivos, y asegurando, además, que se mantiene la calidad del modelo resultante, es decir:

- la reducción del número de atributos a considerar
- y la reducción del número de casos que hay que tratar.

Si denotamos como $D_{n,m}$ a nuestro conjunto de datos, en el primer caso el objetivo será reducir el valor de m (reducción de columnas), mientras que en el segundo caso el objetivo será reducir el valor de n (reducción de filas).

3.4.1. Reducción del número de atributos

La reducción del número de atributos consiste en encontrar un subconjunto de los atributos originales que permita obtener modelos de la misma calidad que los que se obtendrían utilizando todos los atributos, pero con una reducción en la complejidad temporal y/o de cálculo del método de minería de datos.

Existen dos grandes grupos:

- Los métodos de selección de atributos, que consisten en saber qué subconjunto de atributos puede generar un modelo con la misma, o similar, calidad que el modelo obtenido con el conjunto de atributos original. En esencia, se trata de encontrar métricas que indiquen los atributos que pueden ignorarse en la creación del modelo sin sufrir una considerable pérdida de información.

Algunos métodos realizan este tipo de pruebas a cada atributo de forma individual, como por ejemplo la prueba de significación, que compara si un atributo es relevante o no a partir de sus valores. Otros métodos, en cambio, parten de que la suposición de independencia entre los atributos puede ser un poco excesiva, y analizan las dependencias y correlaciones entre atributos para determinar grupos de atributos.

- La fusión y creación de nuevos atributos, que consiste en crear nuevos atributos «híbridos» a partir de la integración o fusión de los atributos existentes. En general, podemos crear un atributo nuevo a_{n+1} mediante la aplicación de una combinación lineal de pesos a los atributos $a_1, \ldots, a_n$:

$$a_{n+1} = \sum_{i=1}^{n} a_i w_i \tag{3.6}$$

donde w_i es el peso que indica en qué grado contribuye el atributo a_i a la creación del nuevo atributo.

El método de análisis de componentes principales (*Principal Component Analysis*, PCA) es uno de los métodos más conocidos para la fusión y creación de nuevos atributos.

3.4.2. Métodos de reducción de casos

En minería de datos no siempre es posible disponer de todos los datos de la población que se estudia. Es en esta circunstancia en la que el concepto de *muestra* se convierte en relevante para el analista. A una muestra de una población se le exige

que sus propiedades sean extrapolables a las propiedades de la población.

Llamaremos espacio muestral al conjunto de muestras que se extrae de una población. La variable que asocia a cada muestra su probabilidad de extracción, sigue lo que se llama una distribución muestral cuyo estudio nos permitirá calcular la probabilidad que se tiene, dada una sola muestra, de acercarse a las propiedades de la población.

Respecto de las técnicas de muestreo, podemos clasificarlas en dos grupos: muestreo probabilístico y muestreo no probabilístico o subjetivo.

Técnicas de muestreo probabilístico

Se trata del conjunto de técnicas más aconsejable, puesto que permiten calcular la probabilidad de extracción de cualquiera de las muestras posibles.

A continuación enumeramos algunas de las más importantes:

- Muestreo estratificado: consiste en dividir la población de estudio en grupos homogéneos respecto de alguna de las características que se quiere medir. Una vez determinados los grupos a estos se les aplicaría, por ejemplo, una técnica de muestreo sistemática. Casos de grupos pueden ser por sexo, por sector laboral, etc.

- Muestreo sistemático: si el número total de miembros de nuestra población es N y el número de miembros de la muestra que queremos tomar es n, entonces tendremos que el intervalo de selección será $\frac{N}{n}$. Escogemos al azar un miembro de la población q y, a partir de este, se

escogen los siguientes miembros siguiendo el orden del intervalo de selección. Esto es $\{q+i\frac{N}{n} \mid i = 0, \ldots, n-1\}$.

- Muestreo por estadios múltiples: se divide la población en niveles estratificados, de modo que vamos tomando muestras de los distintos niveles considerados. Por ejemplo, si queremos realizar un estudio de intención de voto, empezaríamos por dividir la población total en regiones, localidades y barrios.

- Muestreo por conglomerado: esta técnica está indicada para poblaciones que ya de forma natural se encuentran divididas en grupos que contienen toda la variabilidad de la población total. Es decir, que el estudio de uno de los grupos es suficiente para extrapolar los resultados al resto de población.

Técnicas de muestreo no probabilístico o subjetivo

Se dan cuando no es posible determinar la probabilidad de extracción de una determinada muestra o incluso cuando ciertos elementos de la población no tienen posibilidades de ser seleccionados (individuos fuera de cobertura).

Como la selección de muestras no es aleatoria tampoco aplica el concepto de estimación de errores de la muestra. Esta circunstancia provoca lo que se conoce como sesgo de selección, que es la distorsión provocada por el hecho de que la muestra tiene muchas limitaciones a la hora de representar toda la variabilidad de la población total.

A continuación enumeramos algunas de las más importantes:

- Muestreo por cuotas: podemos pensarlo como el muestreo por estratos, pero ponderado por cuotas. Es decir,

se divide la población en estratos definidos por alguna variable de distribución conocida (por ejemplo, el sexo). Posteriormente calculamos la proporción de cada estrato respecto de la población total. Y finalmente se multiplica cada proporción por el tamaño de cada muestra. Esta técnica es habitual en sondeos de opinión y estudios de mercado.

- Muestreo de bola de nieve: En esta técnica se selecciona inicialmente un pequeño grupo de individuos que nos ayudarán a encontrar más miembros representativos de la población. Está indicada para los casos en que la población se encuentra muy dispersa o incluso escondida.

Parte II

Validación y evaluación de resultados

Capítulo 4

Entrenamiento y test

En este capítulo se describe el procedimiento previo a la construcción de modelos una vez los datos ya han sido preparados y se consideran adecuados. El objetivo es asegurar que los modelos construidos a partir de los datos disponibles funcionan correctamente para nuevos datos que haya que procesar (clasificar, agrupar, etc.) en un futuro, es decir, asegurar que el modelo es válido y capaz de ser usado en producción.

4.1. Conjuntos de entrenamiento y test

Para validar un algoritmo de aprendizaje o modelo es necesario asegurar que este funcionará correctamente para los datos de prueba o test futuros, de forma que capture la esencia del problema a resolver y generalice correctamente. En esencia se trata de evitar que sea dependiente de los datos utilizados durante su entrenamiento, evitando el problema conocido como *sobreentrenamiento* (en inglés, *overfitting*).

El sobreentrenamiento se puede definir, informalmente, como el peligro que corremos al sobreentrenar un modelo. Con-

siste en que este acabe respondiendo estrictamente a las propiedades del juego de datos de entrenamiento y que sea incapaz de extrapolarse con niveles de acierto adecuados a otros juegos de datos que puedan aparecer en un futuro.

En la figura 4.1, queremos subrayar que cuando hablamos de datos de entrenamiento y de test, estamos refiriéndonos en exclusiva a los algoritmos de aprendizaje supervisado, ya que es necesario evaluar los resultados obtenidos sobre datos etiquetados nunca vistos anteriormente y se pueden comparar los errores cometidos para cada conjunto.

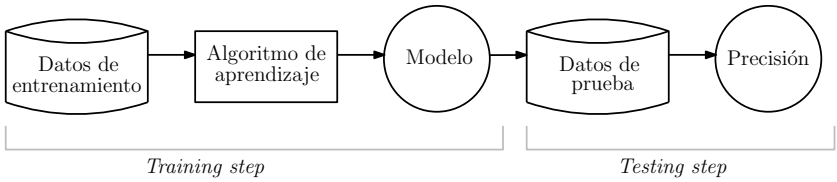


Figura 4.1. Proceso de creación y validación de un modelo basado en aprendizaje supervisado

Así, usaremos el conjunto de datos de entrenamiento para crear el modelo supervisado, mientras que el conjunto de datos de test se usará para medir la precisión alcanzada por el modelo. Formará parte del proceso de construcción del modelo la repetición iterativa de entrenamiento y verificación hasta conseguir unos niveles de precisión y de capacidad de predicción aceptables.

Habitualmente los juegos de datos de entrenamiento y de test suelen ser extracciones aleatorias del juego de datos inicial. En función del número de datos disponibles, existen diferentes técnicas para la construcción de los conjuntos de entrenamiento y de prueba. Se trata de un compromiso entre la robustez del modelo construido (a mayor número de datos usados para el entrenamiento, más robusto será el modelo)

y su capacidad de generalización (a mayor número de datos usados para la validación, más fiable será la estimación del error cometido).

4.1.1. *Leave-one-out*

Tal y como su nombre indica, la técnica *leave-one-out* consiste en utilizar todos los datos menos uno para construir el modelo y utilizar el dato no usado para evaluarlo. Obviamente, si el número de datos es n, este proceso puede repetirse n veces, siendo posible calcular el error promedio de los n modelos construidos.

Este método suele usarse cuando n es pequeño, del orden de cientos de datos o menos, dada la necesidad de construir tantos modelos como datos, lo cual puede ser computacionalmente muy costoso.

4.1.2. *Leave-p-out*

Una generalización del método anterior es utilizar un subconjunto de $p \ll n$, de forma que se utilizan $n - p$ datos para entrenar el modelo y p para validarlo, siguiendo el mismo proceso. El problema es que el número de subconjuntos posibles de p elementos tomados de un conjunto de n crece exponencialmente, con lo cual para n y/o p grandes el problema puede ser intratable computacionalmente, por lo que p suele ser muy pequeño. Por ejemplo, si $n = 100$ y $p = 10$, el número de modelos a construir sería de:

$$\binom{100}{10} = \frac{100!}{90!\,10!} = 17310309456440 \tag{4.1}$$

Obviamente, dicha cantidad está fuera de toda consideración. Incluso con valores más pequeños de p el número de

modelos a construir es enorme si n es grande, por lo que este método queda restringido a conjuntos realmente pequeños donde no pueden usarse las técnicas descritas a continuación.

4.1.3. *k-fold cross validation*

Para conjuntos del orden de cientos o miles de muestras, una solución de compromiso es realizar una partición aleatoria del conjunto de datos en k conjuntos del mismo tamaño, usando $k-1$ conjuntos para entrenar el modelo y el conjunto restante para evaluarlo, repitiendo el proceso k veces y promediando el error estimado. Nótese que el método *leave-one-out* descrito anteriormente es el caso particular $k = n$. Este método permite trabajar con conjuntos más grandes, incluso hasta decenas de miles de elementos.

Habitualmente, se escoge un valor de $k = 5$ o $k = 10$, aunque no existe ninguna base teórica que sustente dichos valores, sino que es resultado de la experimentación [13]. También es típico usar $k = 3$, dando lugar a la que se conoce como regla de los dos tercios, debido al tamaño relativo del conjunto de entrenamiento con respecto al conjunto de datos original.

Por otra parte, dado que la partición en k conjuntos es aleatoria, este proceso podría repetirse un cierto número de ocasiones, promediando todos los errores cometidos (que, de hecho, ya eran promedios del resultado de aplicar *k-fold cross validation*), siempre y cuando se utilice el mismo valor de k.

Como siempre que hay una partición aleatoria, es importante comprobar que la distribución de los valores de la variable objetivo sea similar, para evitar la creación de modelos muy sesgados. De hecho, la partición aleatoria se debe realizar teniendo en cuenta dicha distribución, para evitar el posible sesgo.

4.1.4. Otras consideraciones

En el caso de que el valor de n sea muy grande, del orden de cientos de miles o millones de datos, pueden usarse otros criterios para la partición de los datos originales en un conjunto de entrenamiento y otro de prueba, intentando reducir el coste computacional de estimar el error cuando se deben generar diferentes modelos. Así, una opción habitual es utilizar la regla de los dos tercios en una única partición, entrenando una única vez con dos tercios de los datos originales y validando con el tercio restante. Además, conforme aumenta el número de datos n, se puede reducir el tamaño del conjunto de test, utilizando una fracción más pequeña (p. ej. un quinto), incrementando así el número de datos usados como entrenamiento para construir el modelo y mejorando su robustez.

Por otra parte, la dimensionalidad m del conjunto de datos también juega un papel relevante. Es aconsejable que el numero de datos n satisfaga $n \gg m$, recomendando habitualmente que $n > 10m$, o incluso más (veinte o treinta veces) en función del tipo de problema a resolver (clasificación, regresión, etc.) [23]. Por lo tanto, para valores de n pequeños la única opción posible es utilizar técnicas para reducir la dimensionalidad m del conjunto de entrada, mejorando así el ratio n/m.

Finalmente, la complejidad intrínseca [9] del modelo construido también determina el número de datos necesarios para asegurar la robustez del modelo construido. Mientras más complejo sea el modelo y más parámetros necesite para su construcción, mayor será el número necesario de elementos del conjunto de entrenamiento.

4.2. Conjunto de validación

En algunos casos es posible afinar ciertos parámetros del modelo construido para intentar mejorar su eficiencia. Por ejemplo, en el caso del modelo k-NN es posible probar diferentes valores de k y también diferentes métricas usadas como distancia, en función de la naturaleza de los datos.

En este caso, es necesario disponer de tres conjuntos diferentes, los dos ya comentados conjuntos de entrenamiento (para construir el modelo) y de test (usado únicamente para estimar el error cometido por el modelo ante datos nuevos); y un tercero llamado de validación, con el cual el mejor modelo (con respecto al error estimado mediante el conjunto de test) es afinado antes de ser validado mediante el conjunto de test.

Desafortunadamente, en la literatura del tema es habitual usar indistintamente un conjunto de test o un conjunto de validación dado el uso de cada conjunto. No obstante, siguiendo el esquema representado por la figura 4.1, la idea es que cada etapa donde se toma una decisión con respecto al modelo construido utilice un conjunto de datos diferente, para asegurar su capacidad de generalización.

Como resumen, el lector debe tener una idea clara: la evaluación final del modelo construido debe hacerse con un conjunto de datos que no haya sido usado de ninguna manera durante su proceso de construcción.

Capítulo 5

Evaluación de resultados

En este capítulo revisaremos los principales indicadores o métricas para evaluar los resultados de un proceso de minería de datos. El objetivo es cuantificar el grado o valor de «bonanza» de la solución encontrada, permitiendo la comparación entre distintos métodos sobre los mismos conjuntos de datos.

Las métricas para realizar este tipo de evaluación dependen, principalmente, del tipo de problema con el que se está lidiando. En este sentido, existen métricas específicas para problemas de clasificación, regresión y agrupamiento.

5.1. Evaluación de modelos de clasificación

Las medidas de calidad de modelos de clasificación se calculan comparando las predicciones generadas por el modelo en un conjunto de datos D con las etiquetas de clase verdaderas de las instancias de este conjunto de datos.

5.1.1. Matriz de confusión

La matriz de confusión (*confusion matrix*) presenta en una tabla una visión gráfica de los errores cometidos por el modelo de clasificación. Se trata de un modelo gráfico para visualizar el nivel de acierto de un modelo de predicción. También es conocido en la literatura como *tabla de contingencia* o *matriz de errores*.

La figura 5.1 presenta la matriz de confusión para el caso básico de clasificación binaria.

		Clase predicha	
		P	N
Clase verdadera	P	TP	FN
	N	FP	TN

Figura 5.1. Matriz de confusión binaria

En esencia, esta matriz indica el número de instancias correcta e incorrectamente clasificadas. Los parámetros que nos indica son:

- Verdadero positivo (*True Positive*, TP): número de clasificaciones correctas en la clase positiva (P).
- Verdadero negativo (*True Negative*, TN): número de clasificaciones correctas en la clase negativa (N).
- Falso negativo (*False Negative*, FN): número de clasificaciones incorrectas de clase positiva clasificada como negativa.

- Falso positivo (*False Positive*, FP): número de clasificaciones incorrectas de clase negativa clasificada como positiva.

Algoritmos como los árboles de decisión, k-NN y *support vector machine* son ejemplos de la tipología supervisada sobre los que cabría utilizar una matriz de confusión para medir su grado de acierto.

Matriz de confusión para k clases

La matriz de confusión se puede extender a más de dos clases de forma natural, tal y como podemos ver en la figura 5.2. Esta representación nos permite identificar de forma rápida el número de instancias correctamente clasificadas, que se corresponde con la diagonal de la tabla.

		Clase predicha		
		C_1	$\cdots$	C_k
Clase verdadera	C_1			
	$\cdots$			
	C_k			

Figura 5.2. Matriz de confusión para k clases

Por otro lado, en el caso de problemas de clasificación que impliquen más de dos clases, se puede realizar la evaluación de dos formas distintas:

- *1-vs-1* (OvO): midiendo la capacidad de discriminar entre instancias de una clase, considerada la clase positiva, frente a las instancias de otra clase, consideradas negativas.
- *1-vs-all* (OvA): midiendo la capacidad de discriminar entre instancias de una clase, considerada la clase positiva, frente a las instancias de las demás clases, consideradas negativas.

La aproximación de *1-vs-1* nos conduce a una matriz de confusión, con formato 2×2, para cada par de clases existentes. Por otra parte, la aproximación *1-vs-all* produce una matriz de confusión 2×2 para cada clase.

5.1.2. Métricas derivadas de la matriz de confusión

A partir de la matriz de confusión, definimos un conjunto de métricas que permiten cuantificar la bondad de un modelo de clasificación.

El error de clasificación (*misclassification error*, ERR) y la exactitud (*accuracy*, ACC) proporcionan información general sobre el número de instancias incorrectamente clasificadas. El error, ecuación 5.1, es la suma de la predicciones incorrectas sobre el número total de predicciones. Por el contrario, la precisión es el número de predicciones correctas sobre el número total de predicciones, como se puede ver en la ecuación 5.2.

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} \tag{5.1}$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR \tag{5.2}$$

En algunos problemas nos puede interesar medir el error en los falsos positivos o negativos. Por ejemplo, en un sistema de diagnosis de tumores, nos interesa centrarnos en los casos de tumores malignos que han sido clasificados incorrectamente como tumores benignos. En estos casos, la tasa de verdaderos positivos (*True Positive Rate*, TPR) y la tasa de verdaderos negativos (*False Positive Rate*, FPR), que definimos a continuación, pueden ser muy útiles:

$$TPR = \frac{TP}{FN + TP} \tag{5.3}$$

$$FPR = \frac{FP}{FP + TN} \tag{5.4}$$

La precisión (*precision*, PRE) mide el rendimiento relacionado con las tasas de verdaderos positivos y negativos, tal y como podemos ver a continuación.

$$PRE = \frac{TP}{TP + FP} \tag{5.5}$$

El recall (*recall*, REC) y la sensibilidad (*sensitivity*, SEN) se corresponden con la tasa de verdaderos positivos (TPR), mientras que la especificidad (*specificity*, SPE) se define como la tasa de instancias correctamente clasificadas como negativas respecto a todas las instancias negativas.

$$REC = SEN = TPR = \frac{TP}{FN + TP} \tag{5.6}$$

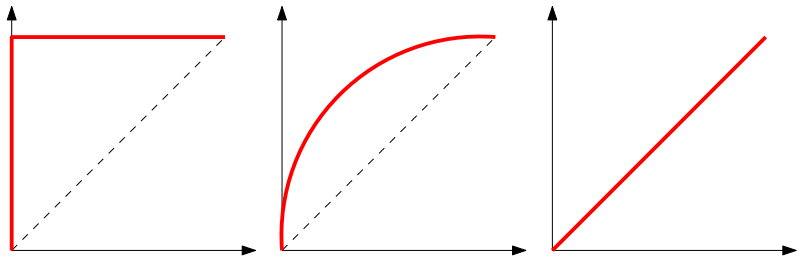
$$SPE = \frac{TN}{TN + FP} = 1 - FPR \tag{5.7}$$

Finalmente, en la práctica se suelen combinar la precisión y el recall en una métrica llamada F1 (*F1 score*), que se define de la siguiente forma:

$$F1 = 2 \times \frac{PRE \times REC}{PRE + REC} \tag{5.8}$$

5.1.3. Curvas ROC

Una curva ROC (acrónimo de *Receiver Operating Characteristic*) mide el rendimiento respecto a los falsos positivos (FP) y verdaderos positivos (TP). La diagonal de la curva ROC se interpreta como un modelo generado aleatoriamente, mientras que valores inferiores se consideran peores que una estimación aleatoria de los nuevos datos.



(a) AUC = 1 **(b)** AUC ≈ 0.8 **(c)** AUC = 0.5

Figura 5.3. Ejemplo de curvas ROC

En esta métrica, un clasificador perfecto ocuparía la posición superior izquierda de la gráfica, con una tasa de verdaderos positivos (TP) igual a 1 y una tasa de falsos positivos (FP) igual a 0. A partir de la curva ROC se calcula el área bajo la curva (*area under the curve*, AUC) que permite caracterizar el rendimiento del modelo de clasificación. La figura 5.3 ejem-

plifica un rendimiento excelente, bueno y malo de una curva ROC.

A modo de guía para interpretar las curvas ROC se han establecido los siguientes intervalos para los valores de AUC:

- [0,5, 0,6): Test malo
- [0,6, 0,75): Test regular
- [0,75, 0,9): Test bueno
- [0,9, 0,97): Test muy bueno
- [0,97, 1): Test excelente

5.2. Evaluación de modelos de regresión

La evaluación de modelos de regresión comparte el mismo principio que la evaluación de modelos de clasificación, en el sentido de que se compara los valores predichos con los valores reales de las instancias del conjunto de datos. En esta sección veremos algunas de las métricas más empleadas para evaluar el rendimiento de un modelo de regresión.

El error absoluto medio (*mean absolute error*, MAE) es la métrica más simple y directa para la evaluación del grado de divergencia entre dos conjuntos de valores, representado por la ecuación:

$$MAE = \frac{1}{|D|} \sum_{d \in D} |f(d) - h(d)| \tag{5.9}$$

donde D es el conjunto de instancias, $h : X \to \mathbb{R}$ es la función del modelo y $f : X \to \mathbb{R}$ es la función objetivo con las etiquetas correctas de las instancias.

En este caso, todos los residuos tienen la misma contribución al error absoluto final.

El error cuadrático medio (*mean square error*, MSE) es, probablemente, la métrica más empleada para la evaluación de modelos de regresión. Se calcula de la siguiente forma:

$$MSE = \frac{1}{|D|}\sum_{d\in D}(f(d) - h(d))^2 \tag{5.10}$$

Utilizando esta métrica se penaliza los residuos grandes. En este sentido, si el modelo se aproxima correctamente a gran parte de las instancias del conjunto de datos, pero comete importantes errores en unos pocos, la penalización en esta métrica será muy superior a la indicada si se emplea la métrica anterior (MAE).

La raíz cuadrada del error cuadrático medio (*root mean square error*, RMSE) se define aplicando la raíz cuadrada al error cuadrático medio (MSE). Tiene las mismas características que este, pero aporta la ventaja adicional de que el error se expresa en la misma escala que los datos originales. En el caso del MSE no era así, ya que la diferencia de valores son elevados al cuadrado antes de realizar la suma ponderada. En algunos casos esta diferencia puede ser importante, mientras que irrelevante en otros.

$$RMSE = \sqrt{MSE} \tag{5.11}$$

Finalmente, en algunos modelos de regresión se pueden tolerar diferencias importantes entre los valores predichos y verdaderos, siempre que el modelo tenga un comportamiento general similar a los valores verdaderos, prestando especial atención a la monotonicidad. En estos casos, las métricas vistas anteriormente pueden no ser representativas del rendimiento

de un modelo de regresión. Por el contrario, los índices de correlación lineal o de rango, como por ejemplo Pearson o Spearman, pueden ser una buena métrica para medir la bondad del modelo generado.

5.3. Evaluación de modelos de agrupamiento

En el caso de los modelos de agrupamiento, de forma contraria a las dos secciones vistas anteriormente, no se dispone de «etiquetas» que indiquen *a priori* la clase a la que pertenecen las instancias de entrenamiento o test. Por lo tanto, las métricas que veremos en esta sección intentan capturar el grado de similitud o disimilitud de los patrones detectados en el conjunto de datos. Generalmente, este tipo de métricas son referenciadas como métricas de calidad.

5.3.1. Métricas de calidad por partición

Este conjunto de métricas de calidad tienen como objetivo medir el nivel de cohesión de una partición (o *cluster*) y/o el nivel de separación de esta con las demás particiones obtenidas en el proceso de agrupamiento. Aunque estas métricas no proporcionan información directa sobre la calidad del modelo completo, sí que aportan información relativa a las propiedades de cada partición y las diferencias entre ellas.

El diámetro (*diameter*) de una partición es la máxima disimilitud entre cualquier par de instancias de la partición. Podemos expresar el diámetro de la siguiente forma:

$$DIAM_{\delta}(p) = \max \delta(x_1, x_2) \mid x_1, x_2 \in p \tag{5.12}$$

donde δ se refiere a la métrica de distancia o similitud empleada y p es la partición que evaluamos.

Claramente, las particiones compactas, es decir, con un diámetro pequeño, suelen ser las más deseadas. Cuando aparecen particiones con diámetros muy dispares, puede ser un signo de que debemos ampliar (o reducir) el número de particiones.

La separación (*separation*) indica el grado de disimilitud entre una partición y las demás. Se define, generalmente, como la mínima disimilitud entre cualquier instancia de una partición y cualquier otra instancia de las demás particiones. Matemáticamente se puede representar de la siguiente forma:

$$SEP_\delta(p) = \min \delta(x_1, x_2) \mid x_1 \in D^p, x_2 \in D - D^p \qquad (5.13)$$

donde D^p referencia al conjunto de instancias de la partición p, y $D - D^p$ indica el conjunto de instancias excepto las incluidas en la partición p.

Idealmente, sería preferible que las distancias entre todas las particiones fueran similares, pero es difícil en problemas y conjuntos de datos reales. En todo caso, que exista diferencia entre el valor de separación de las distintas particiones no se debe entender como una debilidad del modelo.

Para finalizar, definiremos el aislamiento (*isolation*) como una propiedad que dice que el diámetro de una partición debería ser inferior al valor de separación del mismo. Debemos remarcar que el aislamiento no es una métrica en el sentido estricto, si no más bien una propiedad deseable en el resultado de un proceso de agrupamiento; aunque no siempre es posible que se cumpla dicha propiedad, y su ausencia no indica, necesariamente, que el modelo generado no sea válido.

5.3.2. Métricas de calidad general

Aunque las métricas de calidad asociadas a cada partición proporcionan información muy relevante, a veces es preferible un indicador simple que pueda ser utilizado para comparar directamente modelos alternativos sobre un mismo conjunto de datos. Este es el objetivo de las métricas de calidad general, que combinan distintos indicadores sobre la cohesión y separación de todas las particiones en una única métrica.

Aunque es posible realizar la media de los valores de calidad por partición obtenidos en las distintas particiones, existen métricas específicas para evaluar la calidad del modelo de forma global, obteniendo una única puntuación o valor para el modelo de agrupamiento analizado.

El índice Dunn (*Dunn index*) mide la calidad del modelo utilizando el valor mínimo de separación y el valor máximo de diámetro de las particiones, tal y como se describe a continuación:

$$DUNN_\delta = \frac{\min\ sep_\delta(p)}{\max\ diam_\delta(p)} \mid p \in P \tag{5.14}$$

donde p representa una partición y P es el conjunto de todas la particiones que ha generado el modelo de agrupamiento.

Claramente, valores altos de esta métrica indican que el grado de disimilitud entre particiones es alto, mientras que el grado de disimilitud dentro de las particiones es bajo.

Otra métrica de calidad general interesante es conocida como el índice C (*C index*). Esta métrica compara la disimilitud de los pares de instancias de una misma partición con los valores observados en el conjunto de instancias sin considerar las particiones. Se expresa de la siguiente forma:

$$CIND_\delta = \frac{\sum_{p \in P} \sum_{\substack{x_1, x_2 \in D^p \\ x_1 \neq x_2}} \delta(x_1, x_2) - \sum_{x_1, x_2 \in \Gamma_\delta^{\min}} \delta(x_1, x_2))}{\sum_{\langle x_1, x_2 \rangle \in \Gamma_\delta^{\max}} \delta(x_1, x_2) - \sum_{\langle x_1, x_2 \rangle \in \Gamma_\delta^{\min}} \delta(x_1, x_2)} \quad (5.15)$$

donde $\Gamma_\delta^{\min}$ y $\Gamma_\delta^{\max}$ son los conjuntos de mínima y máxima disimilitud, respectivamente, entre pares de instancias de D.

El valor de esta métrica se presenta en el rango $[0, 1]$, donde valores próximos a 0 indican particiones más cohesionadas en el modelo generado. Esta métrica permite comparar resultados de distintos algoritmos sobre un mismo conjunto de datos, proporcionando una métrica única para la comparación. Por contra, el valor obtenido no puede ser utilizado directamente para estimar si el número de particiones es adecuado para el problema a resolver.

5.3.3. Métricas de calidad externas

Las medidas de calidad de agrupamiento presentadas anteriormente asumen que no existe ninguna información externa disponible sobre la forma «correcta» o «deseada» de agrupar los datos. En cambio, adoptan varios enfoques para evaluar la cohesión y la separación de las particiones o su grado de coincidencia con los datos. Por el contrario, las medidas externas asumen la existencia de etiquetas de clase proporcionadas externamente, como en la tarea de clasificación, que —aunque no se utilizan para la creación de modelos— pueden utilizarse para la evaluación de modelos. Estas etiquetas de clase representan una partición de los datos en subconjuntos contra los cuales se comparan las asignaciones de particiones generadas por el modelo evaluado. La aplicación principal de las

medidas externas de calidad es la comparación entre distintos algoritmos sobre un mismo conjunto de datos, conocido habitualmente como *benchmark*, que es una tarea común de investigación.

El enfoque más directo para la evaluación del modelo de agrupamiento con medidas externas de calidad es adoptar medidas de calidad para los problemas de clasificación, vistas en la sección 5.1, como por ejemplo el error de clasificación (*misclassification error*, ERR). Esto puede hacerse tratando el modelo de agrupación evaluado como un modelo de clasificación que asocia la clase mayoritaria con cada grupo basado en el conjunto de entrenamiento y luego para cada instancia predice la clase asociada con su grupo.

Adicionalmente, podemos definir el índice Rand (*Rand index*) a partir de un conjunto de objetos o instancias y dos posibles particiones de estos objetos, la partición externa que identificamos como correcta y la partición que indica el modelo de agrupamiento evaluado. Formalmente, definimos un conjunto de n objetos o instancias $D = \{x_1, \ldots, x_n\}$ y dos particiones de D para comparar, una partición de D en r subconjuntos que indica la clase externa $c : D \to 1, \ldots, r$, y una partición de D en s subconjuntos que indica la clase predicha por el modelo de agrupamiento $h : D \to 1, \ldots, s$.

Cada par de objetos $\langle x_1, x_2 \rangle \mid x_1, x_2 \in D$ es asignado a una de las siguientes cuatro categorías, de forma similar a la matriz de confusión vista anteriormente:

- Verdaderos positivos (*True Positives*, TP), como el número de pares de elementos en D que están en el mismo subconjunto externo y en el mismo subconjunto predicho, esto es $c(x_1) = c(x_2) \wedge h(x_1) = h(x_2)$.

- Verdaderos negativos (*True Negatives*, TN), como el número de pares de elementos en D que están en el distintos subconjuntos externos y en distintos subconjuntos predichos, esto es $c(x_1) \neq c(x_2) \wedge h(x_1) \neq h(x_2)$.

- Falsos positivos (*False Positives*, FP), como el número de pares de elementos en D que están en distintos subconjuntos externos y en el mismo subconjunto predicho, esto es $c(x_1) \neq c(x_2) \wedge h(x_1) = h(x_2)$.

- Falsos negativos (*False Negatives*, FN), como el número de pares de elementos en D que están en el mismo subconjunto externo y en distintos subconjuntos predichos, esto es $c(x_1) = c(x_2) \wedge h(x_1) \neq h(x_2)$.

El índice Rand se calcula como la relación entre el número total de verdaderos positivos y verdaderos negativos respecto al número total de pares de instancias de D. Formalmente, se define como:

$$RAND = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{\binom{n}{2}} \qquad (5.16)$$

El índice Rand tiene un valor en el rango $[0, 1]$, donde 0 indica que los datos predichos no coinciden en ningún par de puntos con los datos externos, y 1 indica que los datos predichos corresponden exactamente con los datos externos.

Parte III

Extracción y selección de atributos

Capítulo 6

Extracción y selección de atributos

En el proceso de descubrimiento de conocimiento en grandes volúmenes de datos es vital escoger las variables y características más adecuadas para presentar al algoritmo de minería de datos. Este problema puede tener diferentes enfoques, entre otros: escoger los mejores atributos de los datos a partir de su análisis preliminar, eliminar los atributos redundantes o que aportan poca información al problema que se desea resolver, o reducir la dimensionalidad de los datos generando nuevos atributos a partir de atributos existentes. En cualquiera de estos casos, el objetivo es reducir el coste espacial y computacional de creación del modelo, permitiendo además la creación de un modelo de similar o mejor calidad.

La selección de atributos consiste en escoger únicamente aquellos atributos que son realmente relevantes para el problema a resolver, descartando otros que no aportan información relevante para el problema a resolver. Por otra parte, la extracción de atributos se trata de calcular nuevos atributos a

partir de los existentes, de tal forma que los nuevos atributos resuman mejor la información que contienen, capturando la naturaleza de la estructura subyacente en los datos.

Existen métodos automáticos para la selección y extracción de atributos, no obstante, ambos métodos también puede hacerse de forma manual. Es importante subrayar que la selección o extracción manual de atributos requiere de un experto en el dominio que analice y escoja los atributos más relevantes para el problema que se intenta resolver en cada caso. Este es un proceso *ad hoc* que requiere gran conocimiento del dominio del problema, así como de los datos que se utilizarán para el proceso de minería de datos.

Por ejemplo, el índice de masa corporal, que se define como el peso de una persona en kilogramos dividido por el cuadrado de su altura en metros, informa mejor del grado de obesidad de una persona que las dos variables originales por separado.

Este tipo de conocimiento proviene del contexto o dominio del problema, y no debe ser descartado. No obstante, en la mayoría de casos será necesario recurrir a los métodos automáticos para extraer características de un conjunto de datos.

En este capítulo nos centraremos en los métodos automáticos de selección y extracción de atributos.

6.1. Selección de atributos

Los métodos de selección de características o atributos (*feature selection*) permiten identificar los atributos que aportan información relevante para el proceso de minería de datos, o al revés, los atributos redundantes que no aportan información relevante a este proceso. En ambos casos el objetivo es el

mismo, elegir qué subconjunto de atributos es más beneficioso para resolver el problema en cuestión.

Dependiendo de si la selección de características usa o no información del método de clasificación posterior, podemos definir la siguiente taxonomía:

- Los algoritmos filtro (*filter*), donde los atributos o subconjuntos de atributos son evaluados de forma independiente del método de clasificación que se utilizará posteriormente.

- Los algoritmos empotrados (*wrappers*), donde el método de selección de características utiliza el clasificador que se usará posteriormente para evaluar qué característica o subconjunto de características es el más adecuado.

Los algoritmos empotrados, aunque suelen tener un buen rendimiento, pueden tener más tendencia a sobreaprender el conjunto de entrenamiento, perdiendo capacidad de generalización. Los algoritmos de selección de características han sido ampliamente estudiados, y se han desarrollado multitud de algoritmos, algunos específicos para determinados problemas [10].

En primer lugar, veremos brevemente algunos de los métodos de selección de atributos individuales, llamados algoritmos univariantes, más empleados para la selección de atributos:

- Selección de máxima relevancia (*maximum relevance selection*). Utiliza el coeficiente de correlación entre cada característica y los resultados de clasificar un determinado conjunto de entrenamiento, obteniendo una lista ordenada de las características que mejor diferencian los datos.

- Selección basada en la información mutua. Mide la información mutua entre las variables aleatorias que modelan cada característica y las etiquetas de clasificación, escogiendo las características que maximizan esta información mutua.

- Métodos basados en tests estadísticos. Aplicación de tests estadísticos de hipótesis sobre los datos, como por ejemplo el *t-statistic* o el *chi-square*.

En segundo lugar, encontramos los métodos de selección de subconjuntos de atributos, llamados algoritmos multivariantes:

- Búsqueda exhaustiva (*exhaustive search*). Consiste en definir un espacio de búsqueda y evaluar, mediante una función de coste, todas las posibles combinaciones de atributos. Solo es aplicable a problemas de dimensionalidad pequeña.

- Selección paso a paso (*stepwise selection*). Consiste en iterar un algoritmo en el cual a cada paso o bien se añade al conjunto de atributos seleccionados aquel atributo que aumenta el rendimiento global del conjunto, o bien se elimina aquel atributo que hace que el rendimiento del subconjunto empeore.

- Ramificación y poda (*branch and bound*). Consiste en aplicar la técnica de búsqueda de *branch and bound* en el espacio de las posibles combinaciones de características. Esta técnica reduce de forma muy notable la búsqueda exhaustiva de la solución.

6.2. Extracción de atributos

El objetivo de la extracción de características es obtener un espacio de dimensionalidad inferior, que preserve al máximo posible los datos útiles y elimine la información redundante. A diferencia de la selección de atributos, en la extracción de atributos se pueden crear nuevos atributos a partir de los existentes en el conjunto de datos inicial.

Las técnicas que se describen en este apartado son conocidas como técnicas de factorización matricial, puesto que descomponen una matriz de datos como el producto de matrices más simples. En cualquier caso, la factorización de una matriz no es única, y cada técnica pone de manifiesto aspectos diferentes de la información contenida en los datos originales.

6.2.1. Análisis de Componentes Principales (PCA)

El análisis de componentes principales (*Principal Component Analysis*, PCA) nos puede ayudar a solucionar problemas de reducción de dimensionalidad y extracción de características en nuestros datos de forma automática. El PCA es un algoritmo muy conocido y usado en el análisis de datos, y tiene muchas posibles aplicaciones diferentes. Informalmente, se puede definir como la técnica que intenta conseguir una representación de un conjunto de datos en un espacio de dimensionalidad más reducida, minimizando el error cuadrático cometido.

El algoritmo PCA reduce la dimensionalidad mediante una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos (es decir, realiza una rotación del espacio d-dimensional), en el cual la varianza de mayor tamaño del conjunto de datos se recoge en el primer

eje (llamado el primer componente principal), la segunda varianza más grande en el segundo eje, y así sucesivamente. Si los datos presentan alguna «inclinación» (en sentido geométrico), los ejes calculados por el PCA siguen dicha inclinación para capturar la misma varianza con un menor volumen del hiperparalelepípedo que contiene todos los datos.

Este algoritmo se basa en la matriz de covarianzas o correlaciones de los datos originales, de forma que calcula los vectores propios de esta matriz y se aplica a los datos originales para conseguir la transformación lineal. Generalmente se utiliza la matriz de correlación cuando los datos no son dimensionalmente homogéneos o el orden de magnitud de las variables aleatorias medidas no es el mismo. La matriz de covarianzas se usará cuando los datos son dimensionalmente homogéneos y presentan valores medios similares.

Podemos expresar el resultado del proceso de extracción de características como el producto matricial:

$$S_{q,n} = P_{q,m} \cdot D_{m,n} \tag{6.1}$$

donde D es el conjunto de datos original, S es la matriz de los resultados (donde $q << m$), y P es la matriz de proyección, que permite realizar la extracción de características. Es importante notar que la matriz de datos se presenta de forma traspuesta a la forma en que la hemos definido en este texto, es decir, las filas representan los atributos y las columnas las instancias.

El objetivo consiste en encontrar la matriz P que reduce la dimensionalidad de los datos, minimizando el error cuadrático cometido en este proceso de reducción de la dimensionalidad.

El proceso para el cálculo de la matriz de soluciones D es el siguiente:

1. Calcular la media de los datos de cada columna de la matriz D, para obtener un conjunto de datos centrados en su origen $\hat{D}$.

2. Calcular la matriz de covarianza C de los datos como:

$$C = \frac{1}{N}\hat{D} \times \hat{D}^T \tag{6.2}$$

3. Coger como matriz P, los q primeros vectores propios con mayor valor propio asociado de la matriz de covarianza C.

Aplicando el PCA obtenemos una serie de vectores $S = \{s_1, \ldots, s_q, \ldots, s_m\}$ donde m es el número de atributos. En un problema de reducción de dimensionalidad nos quedaríamos con los q primeros vectores que explicasen la mayor parte de la varianza, y que escogemos en el tercer paso del algoritmo detallado anteriormente. Los coeficientes de los vectores asociados a cada componente marcan la importancia de cada uno de los nuevos atributos, de forma que los valores altos serán más importantes, mientras que podemos prescindir del resto.

6.2.2. Descomposición en valores singulares (SVD)

El método de la descomposición en valores singulares (*Singular Value Descomposition*, SVD) es una herramienta de álgebra lineal que permite descomponer una matriz de datos expresándola como la suma ponderada de sus vectores propios.

El resultado de la descomposición factorial de la matriz D de tamaño $m \times n$ se representa mediante:

$$D_{m,n} = U_{m,m} \cdot S_{m,n} \cdot V_{n,n}^T \tag{6.3}$$

donde S es una matriz diagonal de tamaño $m \times n$ con componentes no negativos y U, V son matrices unitarias de tamaño $m \times m$ y $n \times n$, respectivamente.

A los componentes diagonales no nulos de la matriz S se los conoce como valores singulares, mientras que las m columnas de la matriz U y las n columnas de la matriz V se denominan vectores singulares por la izquierda y por la derecha, respectivamente. El número máximo de valores singulares diferentes de la matriz D está limitado por el rango máximo de dicha matriz, $r = min\{m, n\}$.

Aplicando este procedimiento a una matriz de datos obtendremos una representación de la información en función de unos pocos vectores singulares, y por lo tanto dispondremos de una representación de los datos en un espacio de dimensionalidad reducida.

6.2.3. Factorización de matrices no negativas

El método de factorización de matrices no negativas (*Non-Negative Matrix Factorization*, NMF o NNMF) realiza una descomposición en factores de una matriz de datos no negativa D de la siguiente forma:

$$D_{n,m} \approx W_{n,q} \cdot F_{q,m} \tag{6.4}$$

donde las matrices no negativas F y W son conocidas como matriz de características y matriz de pesos, respectivamente.

La matriz de características F, de tamaño $q \times m$, define q características y pondera cada una de ellas mediante la importancia que tiene cada uno de los m atributos. Las caracterís-

ticas corresponden a temas genéricos que han sido definidos a partir de la agrupación de atributos en diferentes instancias. La componente i, j de F representa la importancia del atributo j en la característica i.

Por otra parte, la matriz de pesos W le atribuye un peso a cada una de las características en función de su importancia en cada una de las n instancias. Sus q columnas corresponden a las características y sus n filas a las instancias del conjunto de datos analizados, de forma que la componente i, j de W indica la importancia que tiene la característica j en la instancia i.

Uno de los parámetros que deben definirse para aplicar NNMF es el número de características q que se desean. Un valor demasiado alto resulta en un etiquetado demasiado específico de las instancias, mientras que un valor excesivamente pequeño agrupa las instancias en unas pocas características demasiado genéricas y por tanto carentes de información relevante. La matriz de características F es equivalente a la matriz de vectores propios que obteníamos con otras técnicas de factorización de datos, como por ejemplo PCA. La diferencia principal entre NNMF y PCA radica en que en NNMF la matriz F es no negativa y, por tanto, la factorización de D puede interpretarse como una descomposición acumulativa de los datos originales.

Conviene precisar que la descomposición NNMF no es única, es decir, dado un número de características q hay más de una forma de descomponer los datos en la forma de la ecuación 6.4. Lo ideal es aplicar el procedimiento e interpretar los resultados para obtener información adicional de los datos. El algoritmo que se encarga de realizar esta factorización no es trivial y escapa al alcance de este libro.

Parte IV

Métodos no supervisados

Capítulo 7

Agrupamiento jerárquico

El agrupamiento jerárquico (*hierarchical clustering* o HC, en inglés) es un método de clasificación supervisada que busca construir una jerarquía de particiones o clústeres en el conjunto de datos.

Hay dos tipos de algoritmos de agrupamiento jerárquico que, aunque pueden proporcionar resultados similares, tienen enfoques inversos:

- El algoritmo aglomerativo (*Agglomerative Hirerchical Clustering, AHC*) parte de una fragmentación completa de los datos, es decir, cada dato tiene su propio grupo, y fusiona los grupos progresivamente hasta que todos los datos pertenecen a un único grupo. Aplica una estrategia *bottom-up*. En este caso hablaremos de *clustering* o agrupamiento.

- El algoritmo divisivo (*Divisive Hirerchical Clustering, DHC*) parte de un único grupo que contiene todos los datos y lo va dividiendo de forma recursiva hasta obtener un grupo para cada dato. Es decir, aplica una

estrategia *top-down*, procediendo de forma inversa a los algoritmos aglomerativos. En este caso hablaremos de segmentación.

Conceptualmente ambos tipos de algoritmos, los aglomerativos y los divisivos, son equivalentes. Sin embargo, los algoritmos aglomerativos son de más fácil construcción, dado que solo existe un modo de unir dos conjuntos, mientras que existen múltiples formas de separar un conjunto en más de dos elementos.

Por medio de los algoritmos aglomerativos, es posible construir recomendadores basados en el modelo de modo que, para producir una recomendación, solo es necesario asignar una instancia nueva a uno de los grupos generados por el modelo.

Merece la pena observar que para llevar a cabo la operación de recomendación, tan solo será necesario almacenar la descripción de los grupos generados en el modelo, y no todo el juego de datos entero, suponiendo así un ahorro notable en recursos.

El principal inconveniente de este tipo de algoritmos es que no son capaces de determinar, por sí mismos, el número idóneo de grupos a generar, sino que es necesario fijarlos de antemano, o bien definir algún criterio de parada en el proceso de construcción jerárquica.

7.1. Dendrogramas

El dendrograma es un diagrama que muestra las agrupaciones (divisiones) sucesivas que genera un algoritmo jerárquico.

Sin duda, es una forma muy intuitiva de representar el proceso de construcción de los grupos. Sin embargo, un punto débil es no ofrecer información sobre la distancia entre los

distintos objetos, aunque se puede utilizar el tamaño de las flechas o trabajar con tonalidades de un mismo color para añadir este tipo de información.

En la figura 7.1 vemos un ejemplo de dendrograma, en el que tenemos una colección de ocho letras y podemos ver cómo se agrupan progresivamente (algoritmo aglomerativo, estrategia *bottom-up*) o cómo se dividen recursivamente (algoritmo divisivo, estrategia *top-down*).

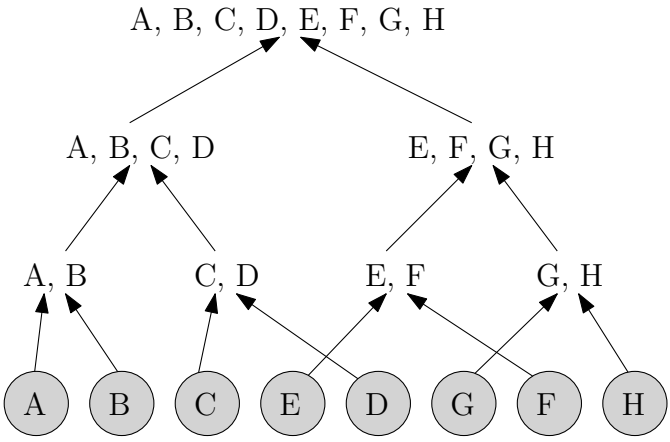


Figura 7.1. Dendrograma aglomerativo. Si las flechas fueran en sentido contrario, tendríamos un dendograma divisivo

El principal interés de los algoritmos jerárquicos es que generan diferentes niveles de agrupamiento, lo que proporciona una información igual o más útil, en algunos casos, que la obtención del agrupamiento definitivo. Es importante destacar que los algoritmos jerárquicos son voraces (*greedy*), lo que significa que buscan la mejor decisión local en cada momento, sin asegurar una solución global óptima.

7.2. Algoritmos aglomerativos

En el algoritmo 1 podemos ver el detalle del pseudocódigo del algoritmo aglomerativo. Como se puede observar, el principal punto de discusión es la medida de la distancia entre dos grupos. Hay dos criterios que afectan a esta medida:

1. ¿Cómo se calcula la distancia entre dos puntos? Para este cálculo podemos utilizar las métricas de distancia vistas en la sección 2.2, como por ejemplo la distancia euclidiana, distancia de Gauss o distancia de Mahalanobis.

2. ¿Cómo determinamos la distancia mínima entre dos grupos que contienen más de un punto? Cuando tenemos grupos formados por varios puntos, podemos determinar la distancia entre los dos grupos según diferentes expresiones, que reciben el nombre de criterios de enlace.

Para construir un dendograma aglomerativo deberemos inicialmente establecer con qué métrica de distancia desearemos trabajar y qué criterio de enlace de grupos o segmentos utilizaremos.

El siguiente paso será considerar cada instancia del juego de datos como un grupo o segmento en sí mismo y a partir de aquí empezaremos a calcular distancias entre grupos. En este punto entraremos en un proceso iterativo en el que en cada repetición fusionaremos los grupos más cercanos.

Algoritmo 1 Pseudocódigo del agrupamiento jerárquico aglomerativo (AHC)

Entrada: D (conjunto de datos individuales)
Inicializar $\ell = 0$
Inicializar el conjunto de clústeres $C_\ell = D$
mientras $|C_\ell| > 1$ **hacer**
Encontrar $c_i, c_j \in C_\ell$ tal que la distancia $d(c_i, c_j)$ sea mínima
Crear $c_* = c_i \cup c_j$ como padre de los clústeres c_i i c_j
Actualizar $C_{\ell+1} = C_\ell - \{c_i, c_j\} \cup c_*$
$\ell = \ell + 1$
fin mientras
devolver C

Algunos de los criterios de enlace más utilizados para medir la distancia entre dos grupos A y B son los siguientes:

1. Enlace simple (*simple linkage*): tomaremos como criterio la distancia mínima entre elementos de los grupos:

$$d(A, B) = \min\{d(a_i, b_j) \,|\, a_i \in A,\, b_j \in B\} \qquad (7.1)$$

   Puede ser apropiado para encontrar grupos de forma no elíptica. Sin embargo, es muy sensible al ruido en los datos y puede llegar a provocar el efecto cadena. Este consiste en el hecho de que puede llegar a forzar la unión de dos grupos que *a priori* deberían permanecer bien diferenciados, por el hecho de que estos compartan algún elemento muy próximo.

2. Enlace completo (*complete linkage*): tomaremos como criterio la distancia máxima entre elementos de los grupos:

$$d(A, B) = \max\{d(a_i, b_j) \,|\, a_i \in A,\, b_j \in B\} \qquad (7.2)$$

No produce el efecto cadena, pero es sensible a los valores *outliers*. Sin embargo, suele dar mejores resultados que el criterio simple.

3. Enlace medio (*average linkage*): tomaremos como criterio la distancia media entre elementos de los grupos:

$$d(A,B) = \frac{1}{|A||B|} \sum_{a_i \in A} \sum_{b_j \in B} d(a_i, b_j) \qquad (7.3)$$

Se trata de un criterio que trata de mitigar los inconvenientes de los dos anteriores, sin acabar de resolverlos por completo.

4. Enlace centroide (*centroid linkage*): la distancia entre dos grupos será la distancia entre sus dos centroides. Presenta la ventaja de que su coste computacional es muy inferior al de los criterios anteriores, de modo que está indicado para juegos de datos de gran volumen.

El enlace simple asume, de forma implícita, que la medida de similitud debería de ser transitiva, lo cual es solo aplicable en algunos dominios. En general, es preferible escoger el enlace completo, que promueve la formación de clústeres compactos y homogéneos, aunque pierde la propiedad de monotonicidad. El enlace medio puede representar un buen compromiso entre ambos.

7.3. Algoritmos divisivos

Aunque el agrupamiento aglomerativo representa la forma más natural de construir un dendograma, su coste computacional es alto. El agrupamiento divisivo suele ser más eficiente a nivel computacional, principalmente porque no se suele

llegar al nivel mínimo del dendograma (es decir, una clase por dato), dado que no aporta información útil. Se define un criterio de parada que determinará la profundidad o nivel máximo del árbol.

Algoritmo 2 Pseudocódigo del agrupamiento jerárquico divisivo (DHC)

Entrada: D (conjunto de datos individuales)
  Crear clúster inicial $C = \{D\}$ y marcarlo como «abierto»
  mientras (existe un clúster abierto) **hacer**
    Encontrar el clúster abierto $c^d \in C$
    si (c^d no cumple la condición de parada) **entonces**
      Dividir $c^d = c_1^d \cup c_2^d \cup \ldots \cup c_m^d$
      para $i = 1, 2, \ldots, m$ **hacer**
        Crear el clúster c_i^d como descendiente de c^d
      fin para
      Marcar c^d como «cerrado»
    si no
      Marcar c^d como «cerrado»
    fin si
  fin mientras

El pseudocódigo del algoritmo divisivo se puede ver en el algoritmo 2. Es importante destacar los dos criterios principales que afectan al diseño de este tipo de algoritmos: el criterio de parada y el criterio de división de particiones.

El criterio último de parada es cuando una partición tiene un único elemento. En este caso, el algoritmo descendería hasta el nivel de granularidad máximo. Pero generalmente este nivel de detalle no es necesario y añade mucha complejidad de cálculo al algoritmo. Existen tres condiciones de parada básicas, basadas en definir:

- La profundidad máxima del árbol resultante.
- El número máximo de instancias que deberán de tener las particiones. Las particiones con cardinalidad inferior ya no serán procesadas otra vez.
- Un valor máximo para una medida de similaridad. Las particiones que no superen este umbral, no serán divididas otra vez, ya que presentan un nivel de cohesión suficiente.

Referente a los criterios de división, es habitual utilizar algoritmos particionales (no jerárquicos) para crear las divisiones de cada partición. En algunos casos, como por ejemplo en el *k-means*, en el que debemos indicar el número de particiones a realizar, puede ser incluso una ventaja, ya que habilita la construcción de árboles del tamaño deseado (por ejemplo, árboles binarios).

7.4. Ejemplo de aplicación

Supongamos que deseamos clasificar un conjunto de 10 números aleatorios entre el 0 y el 100, por ejemplo el conjunto $\{10, 4, 20, 30, 38, 87, 82, 56, 66, 70\}$. La figura 7.2 muestra los dendogramas generados por el algoritmo aglomerativo utilizando la distancia euclídea y los criterios de enlace simple y completo. El eje vertical nos muestra la distancia entre los distintos números, dando información sobre el grado de similitud entre ellos. Como ya hemos comentado, una de las principales ventajas de los algoritmos jerárquicos es que permiten diferentes niveles de agrupación o granularidad. Por ejemplo, en la figura 7.2 podemos escoger un conjunto de tres clústeres, lo que genera los grupos marcados (marco rojo).

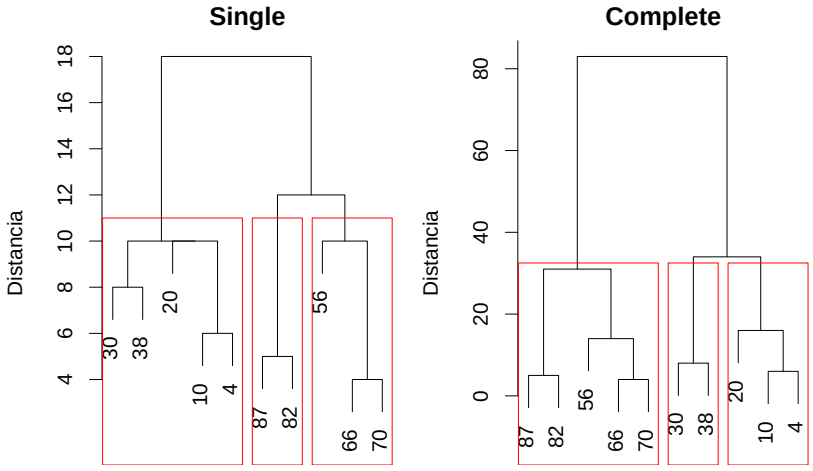


Figura 7.2. Ejemplo de dendrogramas aglomerativos utilizando la distancia euclídea y los criterios de enlace simple y completo

7.5. Resumen

Los métodos de agrupamiento jerárquico permiten no solo descubrir patrones o grupos de similitud en los datos, sino también organizar los datos para un mejor entendimiento. Además, permiten posponer la decisión del nivel de granularidad deseado, permitiendo modificarlo *a posteriori* según los resultados obtenidos.

El agrupamiento jerárquico se suele utilizar como una forma de modelado descriptivo más que predictivo. Aunque esta visión es adecuada para la mayoría de aplicaciones, es importante remarcar que el agrupamiento jerárquico también puede ser empleado como modelo predictivo, asignando las nuevas instancias de datos a las hojas o nodos del árbol de agrupación según la métrica de distancia empleada en la construcción del modelo.

Uno de sus mayores problemas es el coste computacional, aunque este puede ser reducido fijando una profundidad máxima del árbol (en los algoritmos divisivos) o partiendo de una agrupación inicial de instancias (en el caso de los algoritmos aglomerativos).

Capítulo 8

El método *k-means* y derivados

Iniciaremos este capítulo revisando uno de los métodos de clasificación supervisada más conocidos y empleados, el algoritmo *k-means*. Continuaremos, en la sección 8.2, presentando algunos de los métodos derivados del algoritmo *k-means* más importantes. Y finalizaremos este capítulo con la versión difusa del algoritmo *k-means*, conocido como *fuzzy c-means*, en la sección 8.3.

8.1. *k-means*

En esta sección veremos un algoritmo de clasificación no supervisada o segmentación de tipo particional, es decir, que genera una estructura plana. El algoritmo está basado en la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuya distancia es menor.

Algunas consideraciones importantes respecto al algoritmo *k-means* (o *k*-medias) son:

- El número de grupos o clústeres, denotado como k, debe definirse antes de ejecutar el algoritmo.
- Cada grupo o clúster está definido por un punto, generalmente identificado como el centro y llamado semilla o *centroide* del clúster, aunque puede recibir otros nombres en implementaciones concretas del algoritmo.

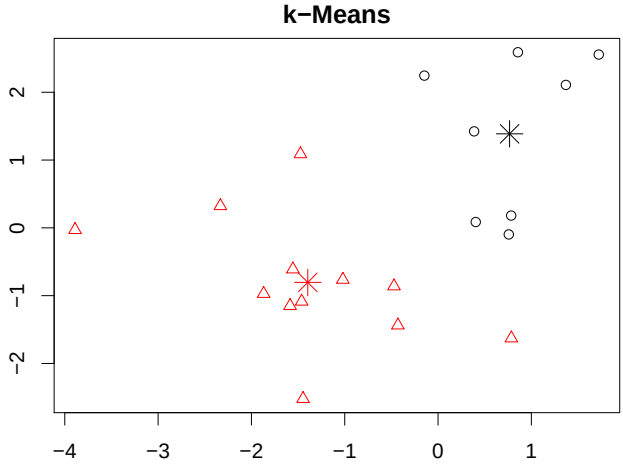


Figura 8.1. Ejemplo de visualización de los resultados de *k-means*

A grandes rasgos, el algoritmo funciona en dos fases principales, tal y como podemos ver en el pseudocódigo del algoritmo 3, que corresponden a las dos tareas siguientes:

1. En primer lugar, la fase de inicialización, identifica k puntos como *centroides* iniciales. Aunque no es necesario que sean puntos del conjunto de datos, sí es importante que sean puntos dentro del mismo intervalo.

2. La segunda fase es iterativa, y consiste en

   a) asignar a cada *centroide* los puntos del conjunto de datos más próximos, formando k grupos disjuntos,

   b) y, a continuación, recalcular los *centroides* en base a los puntos que forman parte de su grupo o partición.

   Estos dos pasos se repiten hasta que se satisface la condición de parada.

La figura 8.1 muestra el resultado de ejecutar el algoritmo sobre un conjunto de veinte puntos en un espacio bidimensional. Las cruces indican los *centroides* de cada partición.

Algoritmo 3 Pseudocódigo del algoritmo *k-means*

Entrada: D (conjunto de datos) y k (número de clústeres)
  Inicializar los clústeres vacíos $P = \{p_1, p_2, \ldots, p_k\}$
  Seleccionar los *centroides* iniciales $\zeta_1, \zeta_2, \ldots, \zeta_k$
  mientras (Condición de parada no satisfecha) **hacer**
    para todo $d_i \in D$ **hacer**
      Asignar la instancia d_i al clúster $p_j \mid d(d_i, \zeta_j)$ es mínima
    fin para
    para todo $p_i \in P$ **hacer**
      Recalcular ζ_i según los valores $d_i \in p_i$
    fin para
  fin mientras
  devolver El conjunto de clústeres P

De los pasos anteriores se desprenden cuatro criterios importantes para definir el comportamiento del algoritmo:

- ¿Cómo podemos inicializar los *centroides*?
- ¿Cómo podemos calcular la distancia entre cada punto d_i y los *centroides*?
- ¿Cómo podemos recalcular los *centroides* a partir de las instancias que forman su grupo o partición?
- ¿Cómo podemos establecer la condición de parada del algoritmo?

8.1.1. Inicialiación de los *centroides*

Un primer enfoque, simple y ampliamente usado, consiste en inicializar los *centroides* con k puntos aleatorios del conjunto de datos. Es decir, $\zeta_i = rand(d_j) \mid d_j \in D, i = \{1, \ldots, k\}$.

Una de las principales consecuencias derivadas de este tipo de inicialización de *centroides* es que implica que el proceso de *clustering* será no determinista. Es decir, diferentes ejecuciones del algoritmo pueden conducir a diferentes modelos y, lógicamente, a diferentes niveles de calidad del resultado. Esta característica, que en principio podría parecer un inconveniente importante, puede ser superada gracias a la simplicidad y efectividad del método, que permite ejecutar el algoritmo múltiples veces con distintos *centroides* y escoger el mejor resultado.

El algoritmo *k-means* utiliza la inicialización de *centroides* aleatoria, pero existen otras aproximaciones más complejas que son utilizadas por métodos derivados del *k-means*. Por ejemplo, se pueden seleccionar los *centroides* iniciales a partir de la distribución de probabilidades de las instancias de D, intentando cubrir todo el rango de valores de los datos, especialmente las zonas con mayor concentración de instancias.

8.1.2. Cálculo de la distancia

El algoritmo *k-means*, generalmente, utiliza la distancia euclídea. Aun así, es posible utilizar algún otro tipo de métrica de similitud (consultar las métricas en la sección 2.2).

8.1.3. Recálculo de los *centroides*

El valor de cada *centroide* es calculado como la media (*mean*) de todos los puntos que pertenecen a este segmento (de aquí el nombre del algoritmo, *k-means*). Por lo tanto, este algoritmo solo es aplicable a atributos continuos. En caso de tener atributos no continuos en el conjunto de datos, deberemos aplicar una transformación previa.

8.1.4. Criterio de parada

La convergencia del algoritmo es la condición de parada natural y última. Esta se alcanza cuando no hay cambios en el recálculo de los *centroides* durante una iteración completa, provocando que no haya alteraciones en la distribución de las instancias en las distintas particiones o clústeres. Es decir, se llega a una situación de estabilidad en la distribución de las instancias. Esta condición se puede garantizar después de un número finito de iteraciones, dependiendo de la métrica empleada en el cálculo de la distancia y el recálculo de los *centroides*.

En la práctica, no es necesario que el método converja, y generalmente es suficiente cuando se tiene pequeños cambios en la pertenencia de las instancias en los distintos grupos o particiones, es decir, cuando estamos próximos a una situación de convergencia.

8.1.5. Criterios para seleccionar un valor k

Una de las principales características de *k-means* es que se le deben indicar el número de particiones o clústeres a identificar. Aunque esto le permite mantener un nivel de simplicidad y eficiencia elevados, puede ser considerado un inconveniente importante. Identificar el número correcto de particiones (k) no es una tarea elemental, e incluso en muchas ocasiones no se puede extraer directamente del conocimiento del dominio. Aun así, en muchos casos el rango de valores de k a considerar no es muy grande, con lo cual se pueden probar diferentes valores y seleccionar el que proporcione mejores resultados.

Un criterio es minimizar la suma de residuos cuadrados (*residual sum of squares*, RSS), es decir, la suma de las distancias de cualquier vector o instancia a su *centroide* más cercano. Este criterio busca la creación de segmentos lo más compactos posibles.

Otro criterio podría ser el de maximizar la suma de distancias entre segmentos, por ejemplo entre sus centros. En este caso estaríamos priorizando tener segmentos los más alejados posible entre sí, es decir, tener segmentos lo más diferenciados posible.

Como sucede en muchos problemas de maximización o minimización, es fácil intuir que el algoritmo no siempre alcanzará un óptimo global, puede darse el caso de que antes encuentre un óptimo local. Además, también es posible que el algoritmo no sea capaz de encontrar ningún óptimo, bien porque simplemente el juego de datos no presente ninguna estructura de segmentos, bien porque no se haya escogido correctamente el número k de segmentos a construir.

Los criterios anteriores (minimización de distancias intragrupo o maximización de distancias intergrupo) pueden usarse

para establecer un valor adecuado para el parámetro k. Valores k para los que ya no se consiguen mejoras significativas en la homogeneidad interna de los segmentos o la heterogeneidad entre segmentos distintos, deberían descartarse.

Por ejemplo, en la figura 8.2 podemos ver cómo evoluciona la métrica de calidad respecto al número de particiones creadas. En concreto, observamos cómo a partir de cinco segmentos, la mejora que se produce en la distancia interna de los segmentos ya es insignificante. Este hecho debería ser indicativo de que cinco segmentos es un valor adecuado para k.

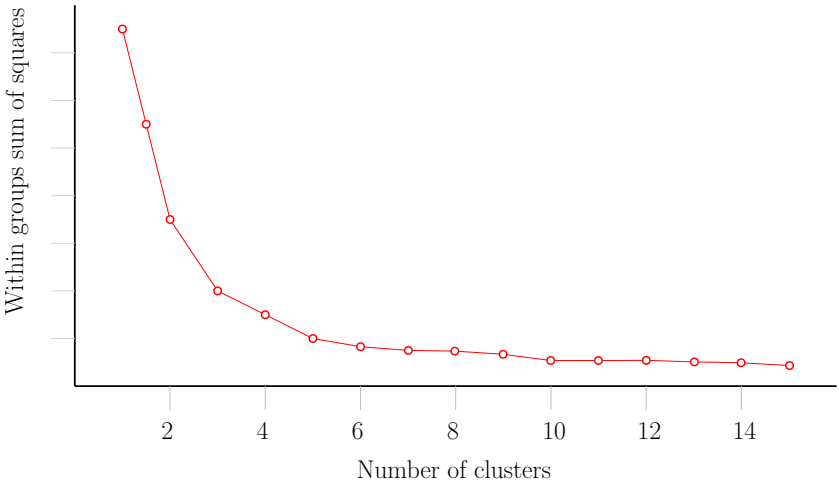


Figura 8.2. Criterio para seleccionar un valor k

Otra aproximación para solucionar este problema consiste en permitir que el número k se modifique durante la ejecución del algoritmo. En este caso, se inicia el algoritmo con un valor proporcionado de k, pero este se puede modificar (incrementar o disminuir) según ciertos criterios. Por ejemplo:

- Si existen dos particiones con los centros muy juntos, quizás es mejor unir ambas particiones y reducir el número de particiones.
- Si existe una partición con un nivel de diversidad muy elevado, se puede dividir esta en dos nuevas particiones, incrementando por tanto el valor de k.

8.1.6. Resumen

El algoritmo *k-means* es, en esencia, un algoritmo de clasificación o segmentación, lo que permite identificar estructuras dentro de un conjunto de datos sin etiquetas o clases. Aun así, también es ampliamente usado para tareas de predicción, donde se utilizan los *centroides* para clasificar nuevas instancias que no estaban en el conjunto de datos original. Una vez construido el modelo, debe preservarse el valor de los *centroides*, que serán imprescindibles para poder realizar el análisis predictivo.

El algoritmo *k-means* es muy usado como primer intento de clasificación en conjuntos de datos continuos. Por el contrario, no es una buena opción en conjuntos de datos categóricos.

8.2. Métodos derivados de *k-means*

El uso de la distancia media para el cálculo de los *centroides* proporciona un modelo simple y eficiente, y además proporciona unos resultados muy fácilmente interpretables. Aun así, tiene sus limitaciones. Por ejemplo, cuando los datos están distribuidos de forma asimétrica y contienen valores extremos (*outliers*) en algunos atributos.

8.2.1. *k-medians*

El algoritmo k-mediana (*k-medians*) es una variación del método anterior, donde se sustituye el cálculo basado en el valor medio, por el valor de la mediana. Aunque la obtención del valor de la mediana requiere mayor complejidad computacional, es preferible en determinados contextos.

Al igual que en el caso del *k-means*, este método puede implementarse mediante distintas métricas de similaridad, aunque existe el riesgo de perder la garantía de convergencia. Una de las métricas de distancia más empleadas en este algoritmo es la distancia de Manhattan (ver sección 2.2).

8.2.2. *k-medoids*

El método *k-medians* presenta mejores resultados que el *k-means* cuando las distribuciones son asimétricas o existen valores *outliers*. Aun así, *k-medians* no garantiza que los centros sean similares a alguna instancia del conjunto de datos. Esto se debe a que, a menos que los atributos sean independientes, los vectores de atributos medianos pueden no parecerse a ninguna instancia existente del conjunto de datos ni del dominio completo.

El algoritmo *k-medoids* propone el recálculo de los centros a partir de instancias que presentan un valor de disimilitud mínimo respecto a las demás instancias de la partición o clúster. La identificación de estos puntos, conocidos como *medoids*, es computacionalmente más costosa que las aproximaciones vistas anteriormente.

Aunque esta aproximación sea considerablemente más costosa, a nivel de cálculo, que *k-means* o *k-medians*, es preferible en algunos dominios donde puede ser interesante que los pun-

tos de centro sean puntos del conjunto de datos. Además, este método es particularmente robusto al ruido y a los valores *outliers*.

8.3. *fuzzy c-means*

El algoritmo *c*-medios difuso (*fuzzy c-means*) puede ser interpretado como una generalización de los algoritmos vistos anteriormente. La diferencia fundamental, desde el punto de vista operacional, es que este algoritmo construye una partición difusa, mientras que los algoritmos vistos anteriormente construyen una partición nítida o disjunta del conjunto de datos.

La idea que subyace en los métodos difusos se basa en la idea de que no todas las instancias tienen la misma relación de pertenencia con el grupo al que pertenecen. Existen puntos que se encuentran muy próximos al centro de una partición, de los cuales podemos decir que están fuertemente asociados con la partición en cuestión. Otros, por el contrario, se encuentran más próximos al borde que separa dos o más particiones y, por lo tanto, podemos decir que su grado de asociación es más débil que en el caso anterior. Resultaría lógico poder decir que estos segundos mantienen cierta relación con múltiples particiones, aunque pueda haber una partición principal. Esta es la idea principal subyacente en los métodos de segmentación o *clustering* difuso.

8.3.1. Funcionamiento del método

En este caso, el algoritmo es prácticamente idéntico al algoritmo 3 visto anteriormente, pero con algunas diferencias que detallamos a continuación:

- Cada instancia d_i tiene asociado un vector de k valores reales que indican el grado de pertenencia de esta instancia a cada uno de los k grupos o particiones. El grado de pertenencia dependerá de la distancia que lo separa con el *centroide* de dicha partición. En general, se normaliza dicho valor para que la suma de todos los valores sea igual a 1. De esta forma, se puede ver el valor como la probabilidad de pertenencia a cada una de las k particiones.

- Al inicio del algoritmo, se asigna el grado de pertenencia de cada instancia a cada grupo aleatoriamente, y después se calculan los *centroides* a partir de estos datos.

- El recálculo de los centroides está ponderado por el grado de pertenencia de cada punto a la partición correspondiente.

La figura 8.3 muestra una posible visualización de los resultados de aplicar el algoritmo *fuzzy c-means* sobre un conjunto de veinte puntos en el espacio bidimensional. Para cada punto (eje horizontal) se muestra la probabilidad de pertenencia a cada una de las particiones. Se han ordenado los datos según la probabilidad de pertenencia al primer grupo. Es interesante ver que, aunque se pueden diferenciar las dos clases claramente, algunos datos están fuertemente asociados con una única partición, mientras que otros mantienen relaciones de pertinencia un poco más débiles con ambas clases.

8.3.2. Resumen

Esta variante, o generalización, presenta las mismas ventajas e inconvenientes que el algoritmo *k-means*. Por un lado,

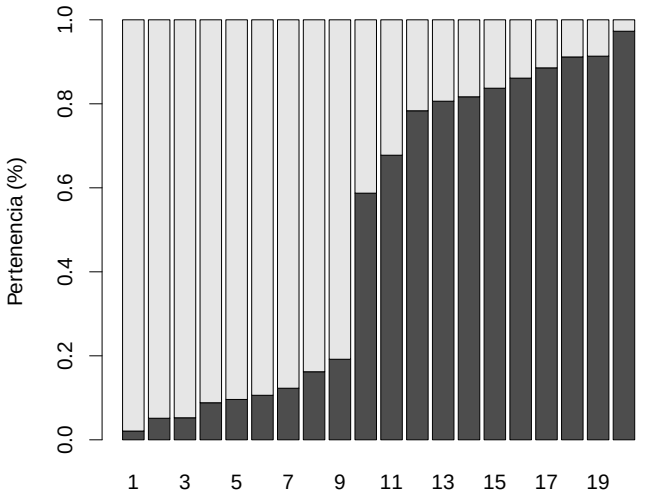


Figura 8.3. Ejemplo de visualización de los resultados de *fuzzy c-means*

la simplicidad y eficiencia son sus principales puntos fuertes. Por el otro, el no determinismo y la excesiva dependencia de la métrica utilizada para calcular las distancias son sus posibles limitaciones o inconvenientes.

Capítulo 9

Algoritmo de agrupamiento Canopy

El algoritmo de agrupamiento Canopy (*Canopy clustering algorithm*) [14] es un método no supervisado de *preclustering*. El *preclustering* se puede entender como el conjunto de tareas de procesamiento previo a un proceso de agrupamiento o *clustering*, con el objetivo de acelerar el proceso de *clustering*, especialmente con grandes conjuntos de datos.

9.1. El concepto de *preclustering*

Este algoritmo es utilizado, generalmente, como procesamiento previo a algoritmos de *clustering*, como por ejemplo el *clustering* jerárquico aglomerativo (AHC) o el *k-means*. En conjuntos grandes de datos puede resultar impracticable aplicar directamente los algoritmos de *clustering*, especialmente en el caso de agrupamiento jerárquico aglomerativo.

La idea que subyace a esta técnica es que podemos reducir drásticamente el número de cálculos requeridos en el proceso

de *clustering* posterior, introduciendo un proceso previo de generación de grupos superpuestos (*canopies*) a partir de una métrica más sencilla de calcular (*cheapest metric*). De esta forma, solo calcularemos distancias con la métrica inicial, más estricta y pesada en cálculos, para los puntos que pertenecen al mismo *canopy*.

Podríamos resumirlo diciendo que previamente, mediante una métrica simple, decidimos qué puntos están definitivamente lejos y, en consecuencia, para estos puntos alejados ya no valdrá la pena malgastar más cálculos con una métrica más exigente.

9.2. Funcionamiento del método

El método de agrupamiento Canopy divide el proceso de segmentación en dos etapas:

- En una primera etapa, usaremos una métrica sencilla en cálculos, con el objetivo de generar los *canopies* o subgrupos superpuestos de instancias. Cada instancia pueda pertenecer a más de un *canopy* y, a su vez, todas las instancias tienen que pertenecer, al menos, a un *canopy*.

- En una segunda etapa, utilizaremos un método de segmentación tradicional, como por ejemplo el agrupamiento jerárquico aglomerativo (AHC) o el método *k-means*, pero lo haremos con la restricción de no calcular la distancia entre los puntos que no pertenecen al mismo *canopy*.

Algoritmo 4 Pseudocódigo del algoritmo Canopy

Entrada: D (conjunto de datos), $T_1, T_2 \mid T_1 > T_2$ (distancias)
  mientras ($D <> \emptyset$) **hacer**
    Elegir un d_i aleatoriamente e inicializar nuevo *canopy* $\omega_v = \{d_i\}$
    Eliminar d_i del conjunto de datos $D = D - \{d_i\}$
    para todo $d_j \in D$ **hacer**
      Calcular la distancia $\Delta_{i,j} = d(d_i, d_j)$ con la métrica sencilla
      si ($\Delta_{i,j} < T_1$) **entonces**
        Añadir d_j al *canopy*, esto es $\omega_v = \omega_v \cup \{d_j\}$
      fin si
      si ($\Delta_{i,j} < T_2$) **entonces**
        Eliminar d_j del conjunto de datos $D = D - \{d_j\}$
      fin si
    fin para
  fin mientras
  devolver El conjunto de *canopies*

El método, descrito en el algoritmo 4, parte del conjunto de datos y dos valores límite (*threshold*): T_1 que indica la distancia máxima de la periferia (*the loose distance*) y T_2 que indica la distancia máxima del núcleo (*the tight distance*), donde $T_1 > T_2$. A partir del conjunto inicial de instancias o puntos, se escoge un punto (aleatoriamente) para formar un nuevo *canopy*. A continuación, se calcula la distancia entre este punto y los demás puntos del conjunto de datos, utilizando una métrica más sencilla de calcular —que hemos llamado *cheapest metric*. Se incluyen en el mismo *canopy* todos los puntos que tengan una distancia inferior al *threshold* T_1. Además, los puntos que tengan una distancia inferior a T_2, son

eliminados del conjunto de datos. De esta forma, se excluye a estos puntos para formar un nuevo *canopy* y ser el centro de este. Este proceso se repite de forma iterativa hasta que no quedan puntos en el conjunto de datos.

Para facilitar la comprensión, vamos a situarnos en los dos casos extremos:

1. Supongamos, como consecuencia de la primera etapa, que nuestro universo de puntos cae por completo en un solo *canopy*. Entonces el método de segmentación por *canopies* sería exactamente igual al del método de segmentación tradicional seleccionado, es decir, AHC o *k-means*.

2. Supongamos que como resultado de la primera etapa, generamos *canopies* relativamente pequeños y con muy poca superposición. En este caso, al aplicar la técnica tradicional solo dentro de cada *canopy*, habremos ahorrado un gran número de cálculos.

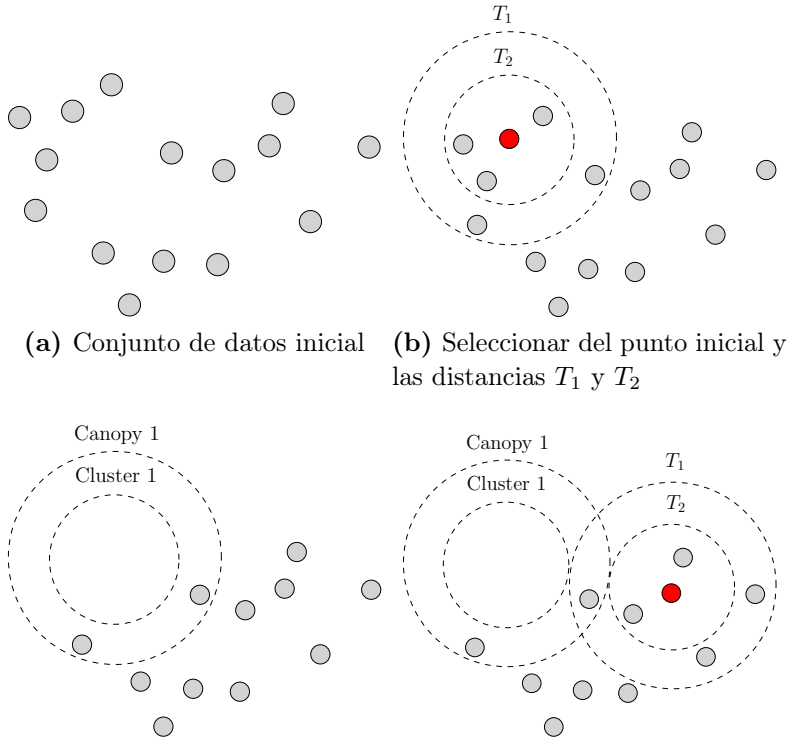
9.3. Ejemplo de aplicación

Veamos de una forma gráfica cómo funciona el algoritmo:

1. En primer lugar, debemos obtener el conjunto de datos D y establecer una valores límite (*threshold*) T_1 y T_2 adecuados para el conjunto de datos con el que estamos trabajado. La figura 9.1a muestra un pequeño conjunto de datos, que utilizaremos para ejemplificar.

2. A continuación escogemos un punto (aletoriamente) $d_i \in D$ y lo definimos como el centro del primer *canopy*. Calcularemos la distancia a todos los demás puntos del

conjunto de datos D, utilizando la métrica más sencilla de calcular (*cheapest metric*). Por lo tanto, podremos fácilmente definir los puntos que están a distancia menor que T_2, los que están entre T_2 y T_1 y los que están más allá de T_1, tal y como muestra la figura 9.1b.



(a) Conjunto de datos inicial

(b) Seleccionar del punto inicial y las distancias T_1 y T_2

(c) Eliminar los puntos con distancia inferior a T_2

(d) Seleccionar otro punto inicial y repetir el proceso

Figura 9.1. Ejemplo de funcionamiento del algoritmo Canopy

3. Eliminaremos del conjunto de datos D los puntos que están a distancia menor que T_2, ya que estos no podrán

formar ningún *canopy* nuevo ni formar parte del centro (figura 9.1c).

4. Repetiremos este proceso forma iterativa hasta que no queden puntos en el conjunto de datos. Es decir, seleccionamos otro punto para formar un nuevo *canopy* y volvemos a aplicar los mismos pasos, manteniendo fijo el valor de T_1 y T_2 (figura 9.1d).

9.4. Resumen

El algoritmo de agrupamiento Canopy, o en general los métodos de *preclustering*, presentan la principal ventaja de reducir el número de casos de datos de entrenamiento que se deben comparar en cada paso del método de agrupamiento que se utilice, ya sea *k-means*, agrupamiento jerárquico u otros métodos de agrupamiento no supervisado.

Estos métodos son especialmente indicados en el caso de lidiar con grandes conjuntos de datos, incluyendo conjuntos con millares o millones de instancias y con un número elevado de atributos a considerar.

Aunque se pueda pensar que el uso de este tipo de estrategias de *preclustering* puedan reducir el rendimiento posterior de los métodos de agrupamiento, no se observa un deterioro en los clústeres resultantes e incluso, en algunos casos, los resultados mejoran con el proceso de *preclustering*.

Parte V

Métodos supervisados

Capítulo 10

Algoritmo k-NN

El k-NN o «k vecinos más cercanos» (en inglés, k *nearest neighbours*) es un algoritmo de aprendizaje supervisado de clasificación, de modo que a partir de un juego de datos de entrenamiento su objetivo será clasificar correctamente todas las instancias nuevas. El juego de datos típico de este tipo de algoritmos está formado por varios atributos descriptivos y un solo atributo objetivo, también llamado *clase*.

10.1. Funcionamiento del método

En contraste con otros algoritmos de aprendizaje supervisado, el algoritmo k-NN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se pide clasificar una nueva instancia. A este tipo de algoritmos se les llama métodos de aprendizaje perezoso o *lazy learning methods*, en inglés.

El funcionamiento del algoritmo es muy simple. Para cada nueva instancia a clasificar, se calcula la distancia con todas las instancias de entrenamiento y se seleccionan las k instan-

cias más cercanas. La clase de la nueva instancia se determina como la clase mayoritaria de sus k instancias más cercanas.

El funcionamiento del método se detalla en el algoritmo 5 y se describe a continuación:

1. Fijamos una valor para k, habitualmente pequeño.

2. Dada una nueva instancia x del juego de datos de prueba, el algoritmo selecciona las k instancias del juego de datos de entrenamiento más cercanas, de acuerdo con la métrica de similitud utilizada.

3. Finalmente, se asigna la instancia x a la clase más frecuente de entre las k instancias seleccionadas como más cercanas.

Algoritmo 5 Pseudocódigo del algoritmo k-NN

Entrada: k (número de vecinos) y x (nueva instancia a clasificar)
para todo $d_i \in D$ **hacer**
Calcular la distancia $d(d_i, x)$
fin para
Inicializar I con las clases c_i de las k instancias de entrenamiento más próximas a x.
devolver La clase mayoritaria en I

Sin duda se trata de un algoritmo tremendamente simple y, a la par, con un nivel de efectividad similar a otros algoritmos más complejos y elaborados.

Veamos gráficamente, mediante la figura 10.1, las distintas clasificaciones que se obtienen al variar el valor de k. Supongamos que debemos clasificar una nueva instancia, representada

aquí como una bola blanca, utilizando un conjunto de datos de entrenamiento binario, representado por bolas de color gris o negro.

Según el valor de k, la nueva instancia será clasificada como:

- Para $k = 1$ el algoritmo clasificará la nueva instancia como clase «gris».
- Para $k = 2$ el algoritmo no tiene criterio para clasificar la nueva instancia.
- Para $k = 3$ el algoritmo clasificará la nueva instancia como clase «negra».

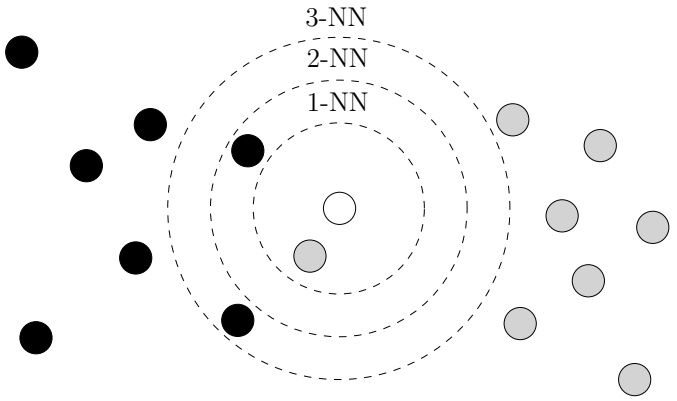


Figura 10.1. Ejemplo de valores de k en el algoritmo k-NN

Su mayor debilidad es la lentitud en el proceso de clasificación, puesto que su objetivo no es obtener un modelo optimizado, sino que cada instancia de test es comparada con todo el juego de datos de entrenamiento. Será la bondad de los resultados lo que determinará el ajuste de aspectos del algoritmo como el propio valor k, el criterio de selección de instancias

para formar parte del juego de datos D de entrenamiento o la propia métrica de medida de similitud.

10.2. Detalles del método

Este método presenta dos variables importantes que pueden determinar la bondad del método. Estas son, la métrica de similitud escogida y al valor de la propia k.

La métrica de similitud utilizada debería tener en cuenta la importancia relativa de cada atributo, puesto que esta influirá fuertemente en la relaciones de cercanía que se irán estableciendo en el proceso de construcción del algoritmo. La métrica de distancia puede llegar a contener pesos que nos ayudarán a calibrar el algoritmo de clasificación, convirtiéndola de hecho en una métrica personalizada. Además, debería ser eficiente computacionalmente, ya que deberemos ejecutar el cálculo de similitud multitud de veces durante el proceso de clasificación de nuevas instancias.

Una de las distancias más utilizadas es la euclídea, especialmente en el caso de tratar con atributos numéricos:

$$de(x, d_i) = \sqrt{\sum_{j=1}^{m} (x_j - d_{ij})^2} \tag{10.1}$$

En el caso de tratar con atributos nominales o binarios, una de las más utilizadas es la distancia de Hamming:

$$dh(x, d_i) = \sum_{j=1}^{m} \delta(x_j, d_{ij}) \tag{10.2}$$

donde $\delta(x_j, d_{ij})$ toma el valor 0 si $x_j = d_{ij}$ y 1 en caso contrario.

El valor de la variable k es muy importante en la ejecución de este método, de modo que con valores distintos de k podemos obtener resultados también muy distintos. Este valor suele fijarse tras un proceso de pruebas con varias instancias. Se suele escoger un número impar o primo para minimizar la posibilidad de empates en el momento de decidir la clase de una nueva instancia. Aun así, cuando se produce un empate se debe decidir cómo clasificar la instancia. Algunas alternativas pueden ser, por ejemplo, no dar predicción o dar la clase más frecuente en el conjunto de aprendizaje de las clases que han generado el empate.

Otro factor importante que se debe considerar antes de aplicar el algoritmo k-NN es el rango u orden de los datos. Es importante expresar los distintos atributos en valores que sean «comparables». Por ejemplo, si modificamos un atributo que originalmente estaba expresado en kilogramos y lo transformamos en gramos, el resultado de las métricas de distancias pueden cambiar drásticamente. En este sentido, existen transformaciones, como por ejemplo la unidad tipificada (*standard score* o *z-score* en inglés) que resta a cada valor el valor medio del atributo y lo divide por su respectiva desviación estándar.

Merece la pena comentar que puede llegar a producirse el efecto del sobreentrenamiento del modelo (*overfitting*). El sobreentrenamiento provoca que el modelo que se construye, en lugar de describir las estructuras subyacentes del juego de datos, lo que acaba describiendo sea exclusivamente el juego de datos de entrenamiento, no siendo extrapolable a otros juegos de datos.

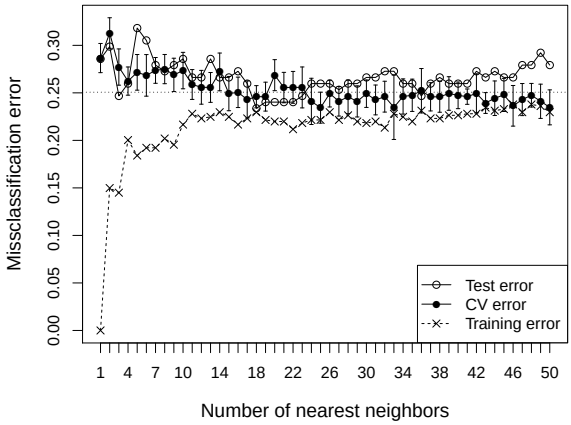
10.3. Ejemplo de selección empírica del valor de k

Veamos un ejemplo de cómo seleccionar el valor de la k utilizando el conjunto de datos Pima Indians Diabetes[1]. Este conjunto de datos presenta 768 instancias y 9 atributos utilizados para determinar si un paciente tiene diabetes. El 35 % de las instancias pertenecen a la clase de pacientes positivos, es decir, que finalmente se les diagnosticó diabetes.

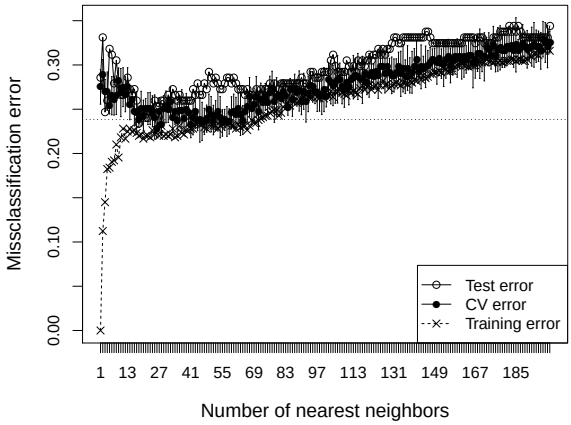
La figura 10.2a muestra los resultados de un conjunto de pruebas con valores de $k = \{1, \ldots, 50\}$ (eje horizontal) utilizando el método *10-fold cross validation*. La figura muestra tres tipos de errores (eje vertical), que se corresponden con el error del conjunto de test (*test error*), el error en las pruebas de validación cruzada (*CV error*) y el error en el conjunto de entrenamiento (*training error*). Como se puede apreciar, a partir de un valor de $k = 15$, el error de test y de validación cruzada no experimenta cambios importantes. Por lo tanto, un valor entre 18 y 26 parece adecuado. Es interesante notar que el error de entrenamiento es muy bajo para valores de $1 \leq k \leq 5$, pero considerablemente por encima de la media en el caso de considerar un conjunto de test o una estimación mediante validación cruzada.

Cuando el valor del parámetro k aumenta demasiado, aparte del coste computacional, estaremos considerando instancias próximas y no tan próximas para determinar la clase de un individuo concreto, lo que deriva en problemas de subentrenamiento (*underfitting*). La figura 10.2b muestra un proceso similar al visto en la figura 10.2a, pero donde anali-

[1] `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`



(a) Valores de k en el rango $\{1, \ldots, 50\}$



(b) Valores de k en el rango $\{1, \ldots, 200\}$

Figura 10.2. Evolución del error en función del parámetro k

zamos el comportamiento del algoritmo en un intervalo de $k = \{1, \ldots, 200\}$. Se puede apreciar claramente cómo el comportamiento del algoritmo se degrada de forma considerable para un valor de $k \geq 70$.

10.4. Resumen

Los principales puntos fuertes de este método de clasificación supervisado son la simplicidad y la robustez ante el ruido presente en los datos de entrenamiento. Además, el hecho de no «crear» específicamente un modelo para procesos de predicción puede ser visto como una ventaja importante en ciertos problemas o dominios.

Por el contrario, algunas de las principales limitaciones de este método también están relacionadas con el hecho de que no se cree un modelo específico durante el entrenamiento. Esto provoca que cada nueva instancia deba ser comparada con los demás para encontrar los k vecinos más próximos e identificar la clase de la nueva instancia. Por lo tanto, el proceso de clasificación de cada nueva instancia no es trivial a nivel de coste computacional, y crece al aumentar el conjunto de entrenamiento etiquetado.

En este sentido, es muy importante escoger una implementación del algoritmo que permita una ejecución óptima de cálculo y memoria. Además, el rendimiento se puede ver seriamente afectado cuando se trata con conjuntos con muchos atributos, como por ejemplo en el caso de imágenes que pueden presentar cientos o miles de dimensiones, problema que se conoce como la *maldición de la dimensión* (*the curse of dimensionality*).

Capítulo 11

Máquinas de soporte vectorial

Las máquinas de soporte vectorial (en inglés, *Support Vector Machines* o SVM) es el nombre con el cual se conoce un algoritmo de aprendizaje supervisado capaz de resolver problemas de clasificación, tanto lineales como no lineales. Actualmente está considerado como uno de los algoritmos más potentes en reconocimiento de patrones.

Su eficiencia y los buenos resultados obtenidos en comparación con otros algoritmos han convertido esta técnica en la más utilizada en campos como el reconocimiento de textos y habla, predicción de series temporales y estudios sobre bases de datos de marketing, entre otros, pero también en otros campos como la secuenciación de proteínas y el diagnóstico de varios tipos de cáncer.

«*No existe nada más práctico que una buena teoría.*»
Vladimir Vapnik

Con esta frase, Vladimir Vapnik [29] daba a entender el porqué de los reconocidos resultados del método de las máquinas de soporte vectorial, desarrollado en los años noventa fruto de sus trabajos sobre aprendizaje estadístico. La gran aportación de Vapnik radica en que construye un método que tiene por objetivo producir predicciones en las que se puede tener mucha confianza, en lugar de lo que se ha hecho tradicionalmente, que consiste en construir hipótesis que cometan pocos errores.

La hipótesis tradicional se basa en lo que se conoce como minimización del riesgo empírico (*empirical risk minimization*) mientras que el enfoque de las SVM se basa en la minimización del riesgo estructural (*structural risk minimization*), de modo que lo que se busca es construir modelos que estructuralmente tengan poco riesgo de cometer errores ante clasificaciones futuras. El concepto de minimización del riesgo estructural fue introducido en 1974 por Vapnik y Chervonenkis [28].

Para entender mejor la teoría de Vapnik y Chervonenkis aplicada a las SVM, vamos a simplificar y focalizarnos en el problema de la clasificación binaria, en la que a partir del conjunto de datos de entrenamiento se construye un hiperplano (separador lineal) capaz de dividir los puntos en dos grupos.

La idea detrás de las máquinas de soporte vectorial es muy intuitiva. Si la frontera definida entre dos regiones con elementos de clases diferentes es compleja, en lugar de construir un clasificador complejo que reproduzca dicha frontera, lo que se intenta es «doblar» el espacio de datos en un espacio de mayor dimensionalidad de forma que con un único corte (es decir, un clasificador muy sencillo) se puedan separar fácilmente ambas regiones.

11.1. Definición de margen e hiperplano separador

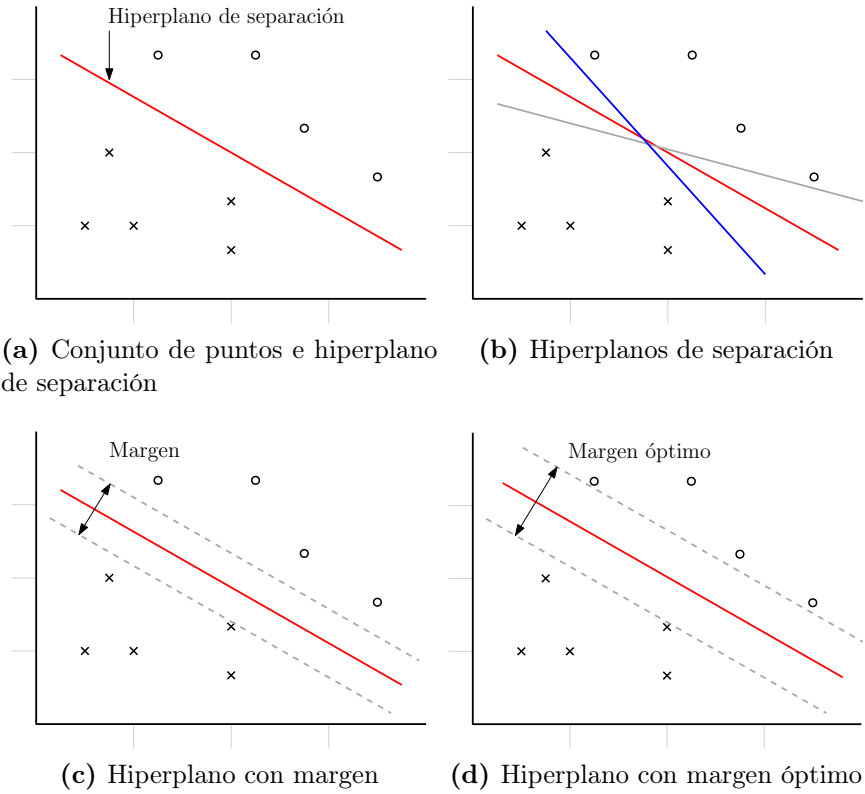
Definiremos el margen como la zona de separación entre los distintos grupos a clasificar. Esta zona de separación quedará delimitada a través de hiperplanos.

Uno de los factores diferenciadores de este algoritmo respecto de otros clasificadores es su gestión del concepto de margen. Las SVM buscan maximizar el margen entre los puntos pertenecientes a los distintos grupos a clasificar. Maximizar el margen implica que el hiperplano de decisión o separación esté diseñado de tal forma que el máximo número de futuros puntos queden bien clasificados. Por este motivo, como veremos más adelante, las SVM utilizan las técnicas de optimización cuadrática propuestas por el matemático italiano Giuseppe Luigi Lagrange.

El objetivo de las SVM es encontrar el hiperplano óptimo que maximiza el margen entre clases (variable objetivo) del juego de datos de entrenamiento.

Veamos en la figura 11.1a un gráfico de dispersión del juego de datos de entrenamiento, donde vemos las dos clases presentes en el conjunto de datos. A partir de esta figura podemos observar que a partir de un hiperplano podríamos separar las dos clases de datos de forma clara. Diremos que esta recta de separación es en realidad un hiperplano de separación.

Debemos tener en cuenta que el hecho de disponer de un hiperplano separador no nos garantiza, en absoluto, que este sea el mejor de todos los posibles hiperplanos separadores. Vemos en la figura 11.1b que en nuestro ejemplo podemos tener infinitos hiperplanos separadores, todos ellos perfectamente



(a) Conjunto de puntos e hiperplano de separación

(b) Hiperplanos de separación

(c) Hiperplano con margen

(d) Hiperplano con margen óptimo

Figura 11.1. Ejemplo de datos binarios para ilustrar el concepto de margen e hiperplano

válidos para separar nuestro juego de datos en las dos clases propuestas.

Ante la pregunta de si hay o no un hiperplano separador mejor que otro, pensemos, por ejemplo, que aquellos hiperplanos que estén demasiado cerca de los puntos del gráfico, muy probablemente no serán demasiado buenos para clasificar nuevos puntos, ya que correremos mucho riesgo de acabar teniendo puntos en el lado no deseado del hiperplano. Por lo

tanto, nuestra intuición nos dice que sería conveniente fijar un margen de seguridad capaz de establecer una distancia entre los puntos de cada clase de puntos a clasificar.

Como puede observarse en la figura 11.1c, el margen es un «terreno de nadie», donde no deberíamos encontrar ningún punto del juego de datos de entrenamiento. Para construir el margen asociado a un hiperplano, procederemos a calcular la distancia entre el hiperplano y el punto más cercano a este. Una vez tenemos esta distancia, la doblamos y este será nuestro margen. De este modo, cuanto más cerca esté un hiperplano de los puntos, menor será su margen. Por contra, cuanto más alejado esté un hiperplano de los puntos, mayor será su margen y, en consecuencia, menor será su riesgo de clasificar incorrectamente nuevos puntos del juego de datos.

Así, el hiperplano separador óptimo se define como el hiperplano que tenga mayor margen posible, tal y como podemos ver en la figura 11.1d. El objetivo de las SVM es encontrar el hiperplano separador óptimo que maximice el margen del juego de datos de entrenamiento.

11.2. Cálculo del margen y del hiperplano separador óptimo

En esta sección usaremos el cálculo vectorial para poder operar con vectores y estudiar las propiedades de los mismos, que nos van a permitir obtener la ecuación del hiperplano separador con el margen óptimo. Para ello, seguiremos la didáctica forma de explicar las SVM propuesta por Adam Berger [2] mediante un sencillo ejemplo.

11.2.1. La ecuación del hiperplano

La expresión más habitual para representar una recta en el plano es $y = ax+b$. Si generalizamos este caso a un espacio de más dimensiones, por cuestiones de practicidad se suele usar la expresión $w^T \cdot x = 0$, donde la equivalencia con la expresión anterior se muestra a continuación:

$$w = \begin{pmatrix} -b \\ -a \\ 1 \end{pmatrix}, x = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix} \tag{11.1}$$

A partir de la notación, es fácil operar y ver que estas dos expresiones son equivalentes:

$$w^T \cdot x = (-b)(1) + (-a)x + (1)y = y - ax - b \tag{11.2}$$

Aunque ambas ecuaciones son equivalentes, generalmente es más cómodo trabajar con la expresión $w^T \cdot x$ por los siguientes motivos:

- Es más adecuada para más de dos dimensiones.
- El vector w siempre será perpendicular al hiperplano, por definición.

Un cálculo que se repite constantemente en las SVM es la distancia entre un punto x_0 y un hiperplano definido por su vector normal w. Este cálculo se efectúa mediante una proyección del vector definido entre el punto x_0 y el hiperplano en cuestión sobre el vector normal al hiperplano. De forma resumida:

$$d = \frac{|x_0 \cdot w|}{\|w\|} \tag{11.3}$$

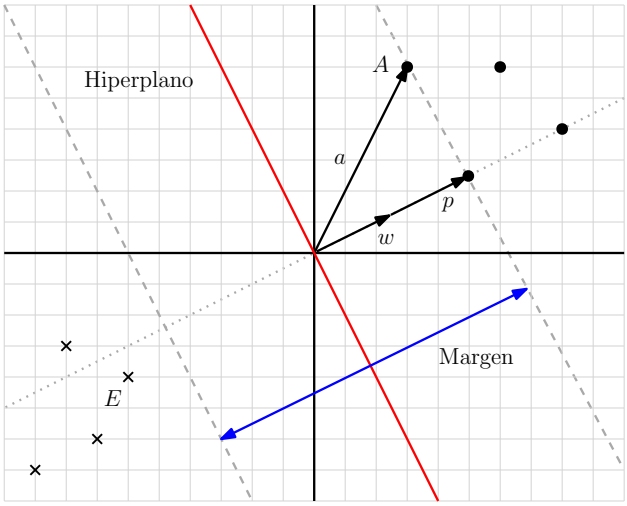
Entonces, si x_0 era el punto más cercano al hiperplano, podemos definir el margen como el doble de la distancia calculada (extendiendo dicha distancia a cada lado del hiperplano), dado que en ese margen no existe ningún otro punto.

En la figura 11.2a podemos apreciar cómo el margen calculado para el punto A no es el óptimo, ya que podríamos conseguir un margen mayor hasta alcanzar el punto E, haciendo retroceder el hiperplano en cuestión. Sin embargo, en la figura 11.2b vemos cómo el margen alcanza los puntos A y E que marcan la frontera entre los dos grupos de puntos. Diremos entonces que el margen e hiperplano generados son los óptimos.

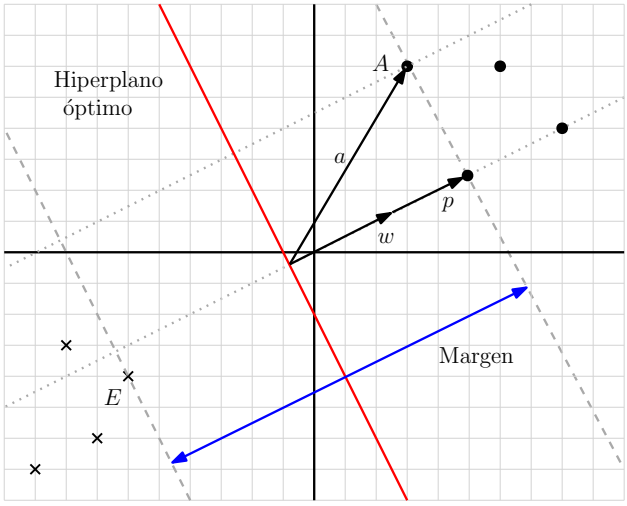
Más adelante veremos cómo el algoritmo SVM llama a los puntos A y E, vectores de soporte, es decir, vectores definidos por puntos que marcarán la frontera que nos servirá para calcular el margen y el hiperplano separador.

Obviamente, los conceptos hiperplano y margen están intrínsecamente relacionados, puesto que fijado un margen podemos obtener el hiperplano que pasa justo por su centro y al revés, fijado un hiperplano podemos obtener el margen a partir de la distancia de los puntos frontera con el hiperplano. Por lo tanto, podemos concluir que determinar el mayor margen posible es equivalente a encontrar el hiperplano óptimo.

El proceso de optimización se completa con el método de los multiplicadores de Lagrange, adecuado para encontrar máximos y mínimos de funciones continuas de múltiples variables y sujetas a restricciones. En dos dimensiones, los multiplicadores de Lagrange permiten resolver el problema de maximizar una función $f(x, y)$ sujeta a las restricciones $g(x, y) = 0$, con la única condición de que ambas tengan derivadas parciales continuas, es decir, que su forma sea «suave» en todas sus variables. De este modo, Lagrange optimizará la función



(a) Hiperplano y margen para el punto A



(b) Hiperplano óptimo entre los puntos A y E

Figura 11.2. Ejemplo de datos bidimensionales para ilustrar el cálculo del margen óptimo

$f(x, y) - \lambda \cdot g(x, y)$, donde λ es el coste de la restricción. Esto es generalizable a cualquier número de dimensiones.

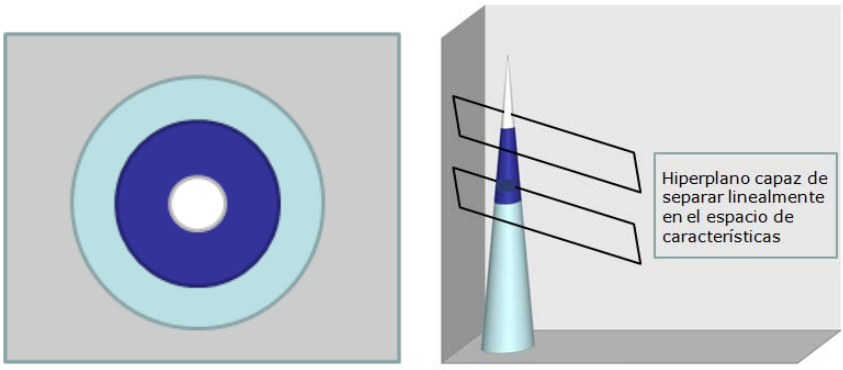
En el caso de las SVM, el objetivo es encontrar aquel hiperplano que maximiza el margen (la distancia del punto más cercano a dicho hiperplano) y que satisface que todos los puntos de una clase estén a un lado del hiperplano y todos los de la otra clase estén al otro lado. Cuando esto es posible se habla de *hard-margin*. Si esto no es posible (es decir, el conjunto de datos no es linealmente separable), se habla de *soft-margin* y es necesario añadir una condición más de forma que el hiperplano óptimo minimice el número de elementos de cada clase que se encuentran en el lado equivocado, teniendo en cuenta también su distancia.

11.3. Funciones *kernel*

Estrictamente hablando, las SVM se ocupan tan solo de resolver problemas de clasificación lineal y se apoyan en las funciones *kernel* para transformar un problema no lineal en el espacio original X en un problema lineal en el espacio transformado F. Las funciones *kernel* son las que «doblan» el espacio de entrada para poder realizar cortes complejos con un simple hiperplano.

11.3.1. ¿Cómo pueden ayudarnos las funciones *kernel*?

La figura 11.3a nos muestra un problema de clasificación con tres zonas de distintos colores. Claramente se trata de un problema de clasificación no lineal puesto que visualmente observamos cómo solo dos circunferencias concéntricas son capaces de realizar una correcta clasificación.



(a) Clasificación radial **(b)** Función *kernel* radial

Figura 11.3. Ejemplo de clasificación y función *kernel* radial

Supongamos que disponemos de una función *kernel* capaz de transportar la figura anterior a un espacio de tres dimensiones, tal y como podemos ver en la figura 11.3b. De hecho, podemos pensar en la figura 11.3a como un plano en picado de un cono dibujado, tal y como vemos en la figura 11.3b.

En el espacio transformado por la función *kernel* (también llamado espacio de características) sí que podemos separar los tres colores por planos, es decir, tenemos un problema de clasificación lineal. En este nuevo espacio podríamos identificar el margen óptimo de separación para clasificar las tres zonas de colores diferentes.

11.3.2. Función *kernel*

Una función *kernel* es aquella que a cada par de vectores de un espacio origen $X \subseteq \mathbb{R}^n$, asigna un número real, cumpliendo con las propiedades del producto escalar.

$$k : X \times X \to \mathbb{R}, \text{ donde } X \subseteq \mathbb{R}^n \tag{11.4}$$

Las propiedades que debe cumplir una función *kernel* son:

- La propiedad distributiva.
- La propiedad conmutativa.
- La propiedad semidefinida positiva.

En términos algebraicos el producto escalar es la suma de los productos de los correspondientes componentes de cada vector. Así mismo, en términos geométricos podemos pensar el producto escalar como la proyección de un vector sobre otro, tal y como hemos estudiado en la sección anterior.

En definitiva, una función *kernel* no es más que una versión «modificada» de la definición clásica de producto escalar. Este tipo de funciones reciben el nombre de funciones *kernel* porque se construyen a partir de una asignación de pesos a los distintos componentes de cada vector, tal y como podemos apreciar en la ecuación 11.5.

11.3.3. Algoritmo SVM

Para entenderlo mejor, veamos un caso de clasificación binaria, es decir, cada punto de nuestro espacio de características, $x_i \in X$, estará asociado a una clase binaria, esto es $c_i \in \{-1, 1\}$.

Podríamos pensarlo como la medida de similitud que nos permita determinar si un nuevo punto se encuentra a la derecha o a la izquierda de un hiperplano de separación.

En este caso, nuestra función de clasificación podría consistir en calcular la suma de distancias alteradas por un peso w, como se muestra a continuación:

$$h(z) = signo\Big(\sum_{i=1}^{n} w_i \cdot c_i \cdot k(x_i, z)\Big) \tag{11.5}$$

donde:

- n es el número de entradas del juego de datos de entrenamiento.
- z es el nuevo punto a clasificar.
- $h(z) \in \{-1, 1\}$ es la función de clasificación.
- $k : X \times X \to \mathbb{R}$ es la función *kernel* que mide la similitud entre puntos (producto escalar, distancia, etc.).
- El conjunto de pares $\{(x_i, c_i)\}_{i=1}^{n}$ son los puntos etiquetados del juego de datos. Es decir, constituyen el juego de datos de entrenamiento, donde $c_i \in \{-1, 1\}$ es la etiqueta o clase del punto x_i.
- $w_i \in \mathbb{R}$ son los pesos que el algoritmo ha determinado para el juego de datos de entrenamiento.
- $signo()$ es la función que nos devuelve el signo de un número, esto es + o −.

El trabajo del algoritmo SVM será, precisamente, determinar los valores w y b óptimos en términos de maximización del margen, con el objetivo de encontrar el mejor hiperplano de separación.

11.3.4. El método *kernel*

En ocasiones puede ser interesante añadir a la ecuación 11.4 una función adicional de transformación del espacio original X en un nuevo espacio intermedio V:

$$\phi : X \to V \text{ donde } X \subseteq \mathbb{R}^n; V \subseteq \mathbb{R}^m \tag{11.6}$$

con $m > n$, de tal modo que:

$$k : X \times X \to V \times V \to F \text{ donde } X \subseteq \mathbb{R}^n; V \subseteq \mathbb{R}^m; F \subseteq \mathbb{R} \tag{11.7}$$

La figura 11.4 muestra el detalle del proceso comentado hasta ahora, donde nuevamente la operación $<,>$ deberá cumplir la condición de ser producto escalar.

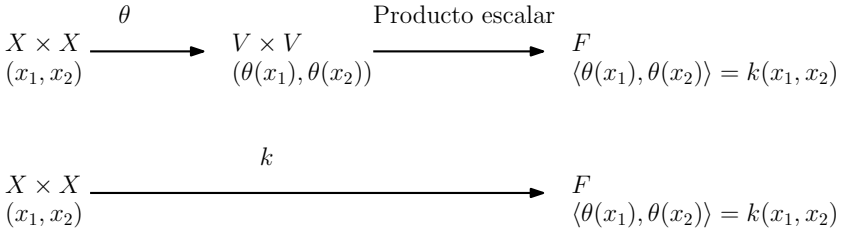


Figura 11.4. Función *kernel* como producto escalar

De este modo, nuestra función *kernel* $k(x_1, x_2)$ podría pensarse como un producto escalar $< \phi(x_1), \phi(x_2) >$, mucho más fácil de operar ya que podemos aplicar las propiedades algebraicas y geométricas del producto escalar.

Esta transformación del espacio de características, que consiste en hacer que nuestra función *kernel* opere en un espacio de más dimensiones, podréis encontrarla en la literatura inglesa como *kernel trick* o método *kernel*.

Gracias a este tipo de transformaciones, conseguimos que juegos de datos que en el espacio $\mathbb{R}^n$ no son linealmente separables, sí que lo sean en un espacio de mayor dimensionalidad, esto es $\mathbb{R}^m$ donde $m > n$.

La principal potencia de este método es que en realidad no necesitamos trabajar explícitamente en el espacio $\mathbb{R}^m$ ya que simplemente necesitamos operar el producto escalar del espacio transformado. A continuación veremos algunos ejemplos de transformaciones.

Transformación cuadrática en un espacio de más dimensiones

La transformación:

$$[u_1, u_2] = [u_1, u_2, u_1^2 + u_2^2] \tag{11.8}$$

es la función $\phi : \Re^2 \rightarrow \Re^3$ que usaremos para una función *kernel* lineal (producto escalar clásico), de modo que:

$$\begin{aligned} k(u, v) = u^T \cdot v &=< \phi(u_1, u_2), \phi(v_1, v_2) > \\ &=< (u_1, u_2, u_1^2 + u_2^2), (v_1, v_2, v_1^2 + v_2^2) > \end{aligned} \tag{11.9}$$

En la figura 11.5 vemos muy gráficamente el beneficio de nuestra transformación, donde añadiendo adecuadamente una dimensión más, conseguimos trabajar con un juego de datos que sí es linealmente separable.

Transformación cuadrática manteniendo las dimensiones

Imaginemos una distribución concéntrica como la que apreciamos en la figura 11.6a.

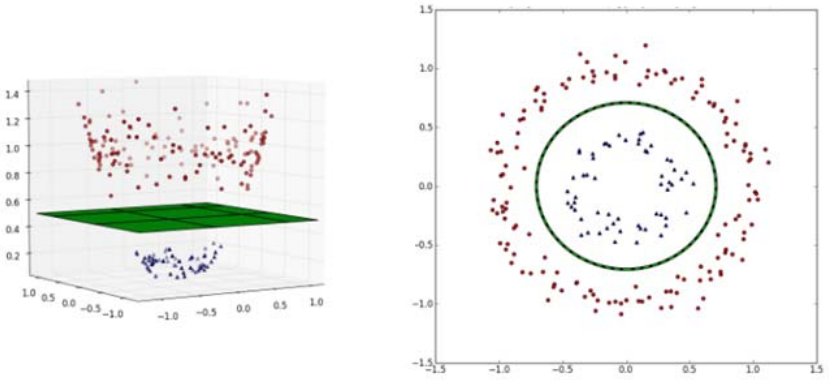


Figura 11.5. Ejemplo de *kernel trick*

Lo que quisiéramos es una función que transforme el espacio original en un espacio de características en el que la distribución de puntos sea linealmente separable, es decir, que sea separable a través de un hiperplano.

Una función *kernel* que nos ayudará a realizar esta transformación es la función cuadrática que simplemente consiste en considerar el cuadrado de cada uno de los dos componentes de cada punto, manteniendo el número de dimensiones. El espacio de características que conseguiríamos sería el que apreciamos en la figura 11.6b.

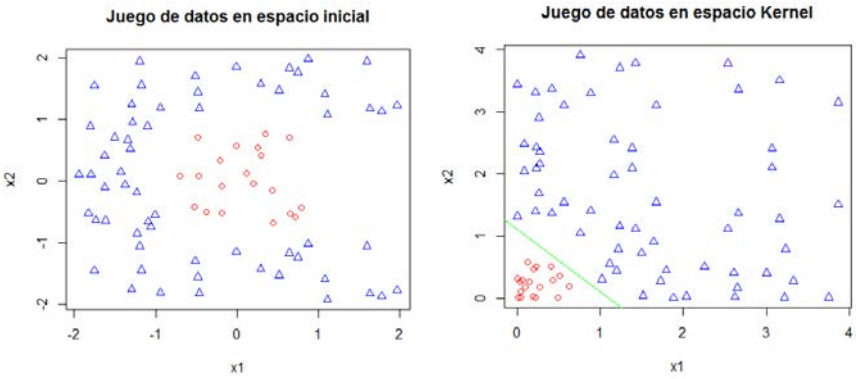
La función *kernel* de nuestro ejemplo podría tener dos expresiones equivalentes:

$$k(u, v) = u_1^2 v_1^2 + u_2^2 v_2^2 \tag{11.10}$$

donde $u, v \in X \subseteq \mathbb{R}^2$, y

$$k(u, v) =< (u_1^2, u_2^2), (v_1^2, v_2^2) >=< \phi(u), \phi(v) > \tag{11.11}$$

donde $u, v \in X \subseteq \mathbb{R}^2$.



(a) Distribución de puntos en el espacio original

(b) Distribución de puntos en el espacio de características

Figura 11.6. Ejemplo de transformación cuadrática manteniendo las dimensiones

Probablemente, las matemáticas nos ayudan más si lo pensamos como la ecuación 11.11, dado que a partir de la definición de producto escalar, podemos encontrar el margen óptimo de separación.

Transformación polinomial

La transformación:

$$[u_1, u_2] = [u_1^2, u_1 \cdot u_2\sqrt{2}, u_1 \cdot \sqrt{2b}, u_2 \cdot \sqrt{2b}, b] \qquad (11.12)$$

es la función $\phi : \Re^2 \rightarrow \mathbb{R}^5$ que usaremos para una función *kernel* polinomial

$$k(u, v) = (u^T \cdot v + b)^2 \qquad (11.13)$$

Esta técnica es muy eficiente en términos de computación porque en realidad siempre estamos computando un producto

escalar. En un espacio con más dimensiones, pero al final siempre computamos un producto escalar para medir la distancia de nuestro punto o vector al hiperplano de separación.

11.4. Tipos de funciones *kernel*

Una de las tareas que realiza el algoritmo SVM es identificar los vectores de soporte, es decir, aquellos vectores que marcaran la trayectoria del hiperplano separador.

Los vectores de soporte marcan la frontera. Sería como una línea de seguridad que una vez sobrepasada deja de ofrecer garantías de fiabilidad en la predicción, de modo que a la hora de valorar un modelo predictivo basado en las SVM podríamos plantearnos el cuantificar, por ejemplo, qué porcentaje de predicciones se encuentran alejadas de los vectores de soporte y qué porcentaje de predicciones se encuentran en sus aledaños.

Veremos a continuación que cada estrategia de clasificación (lineal, polinomial, radial y sigmoidal) tiene su propia función *kernel*. Tal y como menciona el autor Jean Philippe Vert [30] la idea que hay detrás de todas ellas es un concepto equivalente al del producto escalar que persigue medir el grado de similitud entre dos puntos (vistos como vectores) de nuestro juego de datos.

11.4.1. *Kernel* lineal

La función *kernel* utilizada para la SVM de clasificación lineal es:

$$k(u, v) = u^T \cdot v \tag{11.14}$$

donde $u, v \in X$.

Observamos que simplemente se trata del producto escalar de dos vectores y no supone ninguna transformación del espacio de características. En este caso la función *kernel* responde enteramente a lo que nuestra intuición entiende por producto escalar. La figura 11.7a muestra un ejemplo de la superficie de predicción creada por un modelo lineal.

11.4.2. *Kernel* polinomial

La función *kernel* polinomial es:

$$k(u, v) = (\gamma u^T \cdot v + b)^p \tag{11.15}$$

donde $u, v \in X$ y $\gamma > 0$, $b \geq 0$ y $p > 0$ son parámetros.

Podríamos pensar el *kernel* polinomial como una primera generalización del concepto de producto escalar, en el que tenemos la opción de trabajar en un espacio transformado de más dimensiones que el espacio original.

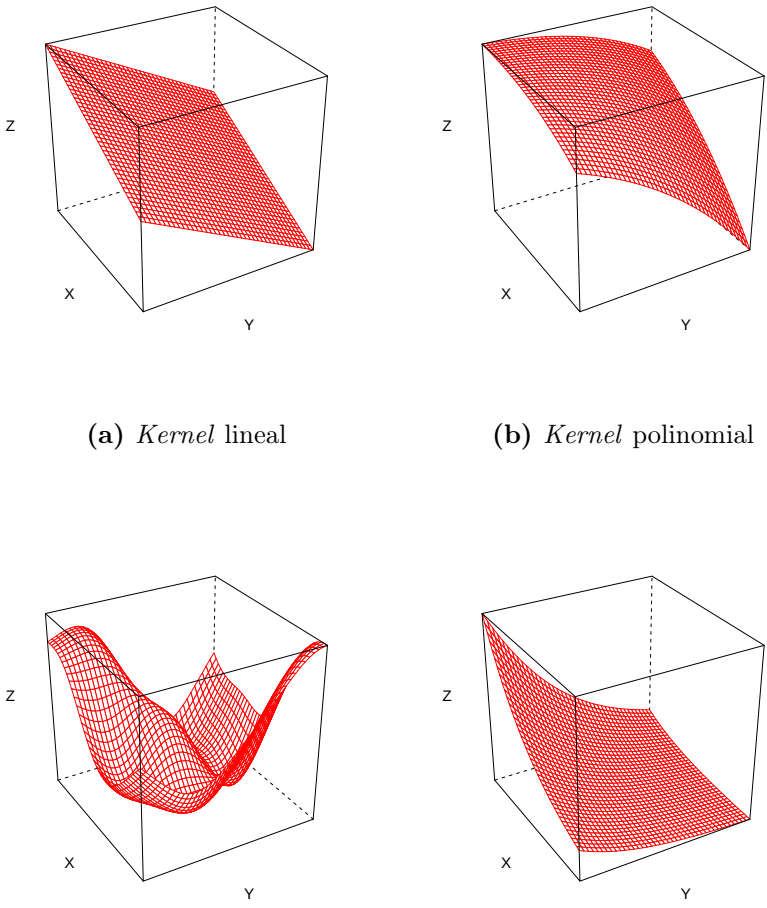
Los parámetros γ, b, p nos permiten modelar en cierto grado la morfología de la transformación del espacio original. El calibraje de estos parámetros y la influencia que cada uno de ellos ejerce sobre la transformación del juego de datos, forman parte del estudio necesario para poder usar de forma eficiente esta función *kernel*. La figura 11.7b muestra un ejemplo de la superficie de predicción creada por un modelo polinomial.

11.4.3. *Kernel* radial

La función *kernel* radial es:

$$k(u, v) = e^{-\gamma||u-v||^2} \tag{11.16}$$

donde $u, v \in X$ y $\gamma > 0$ es un parámetro.



(a) *Kernel* lineal

(b) *Kernel* polinomial

(c) *Kernel* radial

(d) *Kernel* sigmoidal

Figura 11.7. Ejemplo de las distintas formas en el espacio transformado que podemos gestionar con las funciones *kernel*.

También conocido como *kernel* de Gauss, generaliza el concepto de distancia entre vectores en lugar de su producto escalar. Está especialmente indicada para modelar relaciones complejas y no lineales, como podemos ver en el ejemplo de la superficie de predicción de la figura 11.7c. Esta función es muy dependiente del parámetro γ, de modo que será importante tenerlo en cuenta en aspectos como el posible sobreentrenamiento del juego de datos.

11.4.4. *Kernel* sigmoidal

La función *kernel* sigmoidal es:

$$k(u, v) = \tanh(\gamma u^T \cdot v + b) \tag{11.17}$$

donde $u, v \in X$ y $\gamma > 0$, $b \geq 0$ son parámetros.

Se trata de la tangente hiperbólica del producto escalar en el espacio original, de modo que sin duda se trata de otro caso de generalización del producto escalar.

Es capaz de modelar relaciones complejas similares al *kernel* radial. La tangente hiperbólica es una función muy utilizada como función de activación en las redes neuronales y por este motivo también tiene un papel importante en el mundo de las SVM. La figura 11.7d muestra un ejemplo de la superficie de predicción creada por un modelo sigmoidal.

Sin embargo, tiene el riesgo de que para ciertos valores de sus parámetros γ, b puede llegar a representar una función *kernel* no válida, es decir, que no cumpla con las condiciones que se le exigen a una función *kernel* para ser considerada como tal.

11.5. Resumen

La apuesta de las máquinas de soporte vectorial por maximizar las opciones de clasificar correctamente las nuevas entradas del juego de datos, convierten este algoritmo en un clasificador muy robusto cuando los datos son claramente separables (mediante un hiperplano), sin embargo, cuando hay ruido, sobre todo en la franja de separación, puede dar algunos problemas.

Las SVM son muy efectivas en juegos de datos con muchas dimensiones o atributos, sacando provecho de sus propiedades geométricas. Además, gracias al recurso de los vectores de soporte que funcionan como puntos de referencia, se consigue una gran eficiencia computacional.

Como bien apunta el autor Jeffrey Strickland [27], las SVM actualmente son un algoritmo muy utilizado en tareas de clasificación de documentos o también en el ámbito de la salud, como por ejemplo en la prevención y detección de enfermedades de diagnóstico complejo.

En definitiva y a pesar de que también pueden usarse para tareas de regresión, las máquinas de soporte vectorial son un algoritmo muy eficiente y ampliamente usado en tareas de clasificación. El uso que hacen de las funciones *kernel* y su visión puramente vectorial del problema de clasificación le confieren una potencia matemática que explica claramente su buen rendimiento ante otras tipologías de algoritmos.

El principal problema de las SVM es la dificultad para entender el hiperplano de corte en el espacio de mayor dimension donde se realiza la separación de elementos de clases diferentes, dado que puede no resultar trivial entender el «pliegue» del espacio de entrada que realiza la función *kernel*, funcionando entonces como un modelo de caja negra.

Capítulo 12

Redes neuronales

Las redes neuronales artificiales (*Artificial Neural Networks* o ANN) [1, 11] son un conjunto de algoritmos inspirados en el mecanismo de comunicación de la neurona biológica. Han demostrado ser una buena aproximación a problemas donde el conocimiento es impreciso o variante en el tiempo. Su capacidad de aprender convierte a las redes neuronales en algoritmos adaptativos y elaborados a la vez.

El inicio de las redes neuronales se remonta a los años cuarenta, cuando McCulloch y Pitts [15] propusieron un modelo para intentar explicar el funcionamiento del cerebro humano. Una década después, en los años cincuenta, se empezaron a implementar modelos de cálculo basados en el concepto del perceptrón [25]. Posteriormente, Marvin Minsky demostró las limitaciones teóricas de los modelos existentes [18], provocando un estancamiento de la investigación y aplicaciones relacionadas con las redes neuronales durante los años setenta y ochenta. Hasta que en el año 1982 John Hopfield propuso el modelo de la «propagación hacia atrás» (*backpropagation*) [12], que permitió superar las limitaciones del perceptrón y,

además, facilitó en gran medida la fase de entrenamiento de las redes.

La redes neuronales artificiales permiten realizar tareas de clasificación en juegos de datos etiquetados y tareas de regresión en juegos de datos continuos, aunque también permiten realizar tareas de segmentación en juegos de datos no etiquetados a partir del establecimiento de similitudes entre los datos de entrada.

En general las redes neuronales relacionan entradas con salidas, es decir, el juego de datos inicial con la salida o resultado del algoritmo aplicado sobre los mismos. Algunos autores se refieren a las redes neuronales como «aproximadores universales», porque son capaces de aprender y aproximar con precisión la función $f(x) = y$ donde x se refiere a los datos de entrada y $f(x)$ se refiere al resultado del algoritmo.

12.1. Las neuronas

Una red neuronal artificial consiste en la interconexión de un conjunto de unidades elementales llamadas neuronas. Cada una de las neuronas de la red aplica una función determinada a los valores de sus entradas procedentes de las conexiones con otras neuronas, y así se obtiene un valor nuevo que se convierte en la salida de la neurona.

La figura 12.1 presenta el esquema básico de una neurona artificial. Cada neurona tiene una conjunto de entradas, denotadas como $X = \{x_1, x_2, \ldots, x_n\}$. Cada una de estas entradas está ponderada por el conjunto de valores $W^i = \{w_1^i, w_2^i, \ldots, w_n^i\}$. Debemos interpretar el valor w_j^i como el peso o la importancia del valor de entrada x_j que llega a la neurona i procedente de la neurona j.

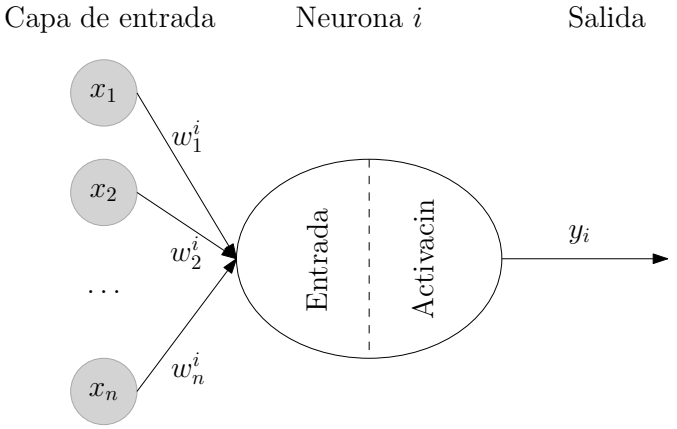


Figura 12.1. Nodo de una red neuronal

Cada neurona combina los valores de entrada, aplicando sobre ellos una función de entrada o combinación. El valor resultante es procesado por una función de activación, que modula el valor de las entradas para generar el valor de salida y_i. Este valor, generalmente se propaga a las conexiones de la neurona i con otras neuronas o bien es empleado como valor de salida de la red.

12.1.1. Función de entrada o combinación

Como hemos visto, cada conexión de entrada x_j tiene un peso determinado w_j^i que refleja su importancia o estado. El objetivo de la función de entrada es combinar las distintas entradas con su peso y agregar los valores obtenidos de todas las conexiones de entrada para obtener un único valor.

Para un conjunto de n conexiones de entrada en el que cada una tiene un peso w_j^i, las funciones más utilizadas para combinar los valores de entrada son las siguientes:

- La función suma ponderada:

$$z(x) = \sum_{j=1}^{n} x_j w_j^i \tag{12.1}$$

- La función máximo:

$$z(x) = \max(x_1 w_1^i, \ldots, x_n w_n^i) \tag{12.2}$$

- La función mínimo:

$$z(x) = \min(x_1 w_1^i, \ldots, x_n w_n^i) \tag{12.3}$$

- La función lógica AND ($\wedge$) o OR ($\vee$), aplicable solo en el caso de entradas binarias:

$$z(x) = (x_1 w_1^i \wedge \ldots \wedge x_n w_n^i) \tag{12.4}$$

$$z(x) = (x_1 w_1^i \vee \ldots \vee x_n w_n^i) \tag{12.5}$$

La utilización de una función u otra está relacionada con el problema y los datos concretos con los que se trabaja. Sin embargo, generalmente, la suma ponderada suele ser la más utilizada.

12.1.2. Función de activación o transferencia

Merece la pena detenernos en este punto para plantear la siguiente reflexión. Si la red neuronal consistiera simplemente en pasar de nodo en nodo distintas combinaciones lineales de sus respectivos datos de entrada, sucedería que a medida que incrementamos el valor X también incrementaríamos el valor

resultante Y, de modo que la red neuronal solo sería capaz de llevar a cabo aproximaciones lineales.

Las funciones de activación o transferencia toman el valor calculado por la función de combinación y lo modifican antes de pasarlo a la salida.

Algunas de las funciones de transferencia más utilizadas son:

- La función escalón, que se muestra en la figura 12.2a y cuyo comportamiento es:

$$y(x) = \begin{cases} 1 & \text{si } x \geq \alpha \\ -1 & \text{si } x \leq \alpha \end{cases} \tag{12.6}$$

  donde α es el valor umbral de la función de activación. La salida binaria se puede establecer en los valores $\{-1, 1\}$, pero también se utilizan a menudo los valores $\{0, 1\}$.

- La función lineal, que permite generar combinaciones lineales de las entradas. En su forma más básica se puede ver en la figura 12.2b.

$$y(x) = \beta x \tag{12.7}$$

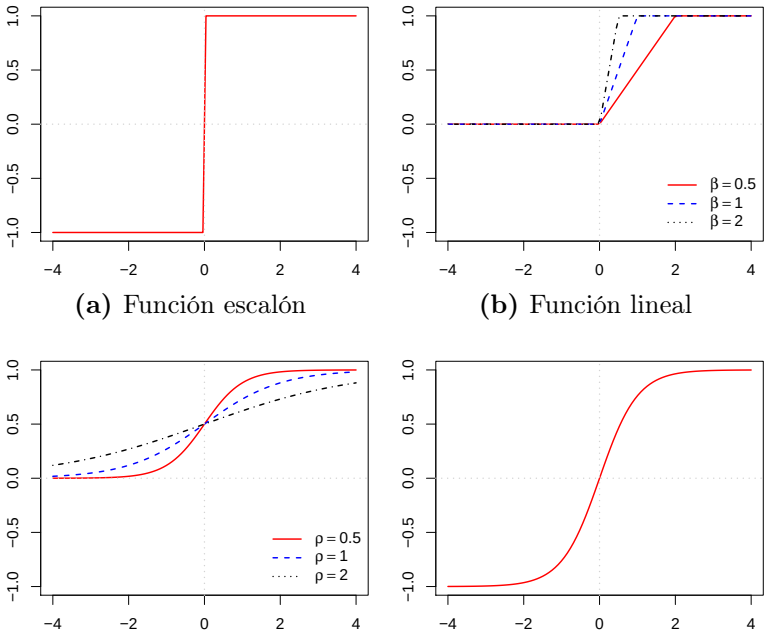
- La función sigmoide o función logística, que se muestra en la figura 12.2c y cuya fórmula incluye un parámetro (ρ) para determinar la forma de la curva, pudiendo actuar como un separador más o menos «suave» de las salidas.

$$y(x) = \frac{1}{1 + e^{\frac{-x}{\rho}}} \tag{12.8}$$

- La tangente hiperbólica, que describimos a continuación y podemos ver en la figura 12.2d.

$$y(x) = tanh(x) = \frac{senh(x)}{cosh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{12.9}$$

Las funciones sigmoides e hiperbólicas satisfacen los criterios de ser diferenciables y monótonas. Además, otra propiedad importante es que su razón de cambio es mayor para valores intermedios y menor para valores extremos.



(a) Función escalón **(b)** Función lineal

(c) Función sigmoide **(d)** Tangente hiperbólica

Figura 12.2. Representación de las funciones escalón, lineal, sigmoide e hiperbólica

También es importante destacar que las funciones sigmoides e hiperbólicas presentan un comportamiento no lineal. Por lo tanto, cuando se utilizan estas funciones en una red neuronal, el conjunto de la red se comporta como una función no lineal compleja. Si los pesos que se aplican a las entradas se consideran los coeficientes de esta función, entonces el proceso de aprendizaje se convierte en el ajuste de los coeficientes de esta función no lineal, de manera que se aproxime a los datos que aparecen en el conjunto de las entradas.

12.1.3. Casos concretos de neuronas

Como hemos visto, la neurona es una unidad que se puede parametrizar en base a dos funciones principales: la función de entrada o combinación y la función de activación o transferencia. Fijando estas dos funciones obtenemos algunas neuronas que son especialmente útiles y cuyo funcionamiento merece la pena conocer.

El perceptrón

El perceptrón (*perceptron*) es una estructura propuesta por Rosenblatt [25] caracterizada por las siguientes funciones:

- La función de entrada de la neurona i es la suma ponderada de las entradas (x_j) y los pesos (w_j^i).

$$f(x) = \sum_{j=1}^{n} x_j w_j^i \tag{12.10}$$

- La función de activación se representa mediante la función escalón, presentada en la ecuación 12.6. En consecuencia, la salida de un perceptrón es un valor binario.

Si analizamos un poco el comportamiento de un perceptrón vemos que la salida tomará el valor 0 o 1 dependiendo de:

$$y_i = \begin{cases} 0 & \text{si } x_1 w_1^i + \ldots + x_n w_n^i - \alpha \leq 0 \\ 1 & \text{si } x_1 w_1^i + \ldots + x_n w_n^i - \alpha > 0 \end{cases} \tag{12.11}$$

donde el parámetro α es el umbral o sesgo empleado en la función escalón. En general, hablamos de umbral (*threshold*) cuando se encuentra en la parte derecha de la desigualdad; mientras que hablamos de sesgo (*bias*) cuando lo incorporamos en la parte izquierda de la desigualdad y se puede ver como un valor de entrada fijo. Para simplificar, y haciendo uso de la notación vectorial, sustituiremos la expresión $x_1 w_1^i + \ldots + x_n w_n^i - \alpha$ por wx. Es interesante notar que hemos añadido el parámetro de sesgo en el mismo vector que los pesos, lo que se conoce como notación extendida. Podemos ver el sesgo como un peso más del vector de pesos asociado a una entrada que siempre es igual a 1.

La neurona sigmoide

La neurona sigmoide (*sigmoid neuron*) es una de las más utilizadas en la actualidad, y se caracteriza por:

- La función de entrada de la neurona i es la suma ponderada de las entradas (x_j) y los pesos (w_j^i).

$$f(x) = \sum_{j=1}^{n} x_j w_j^i \tag{12.12}$$

- La función de activación emplea la función sigmoide (ver ecuación 12.8). Por lo tanto, su salida no es binaria, sino un valor continuo en el rango $[0, 1]$.

La salida de la neurona sigmoide puede parecer muy similar a la de un perceptrón, pero con una salida más «suave» que permite valores intermedios. Precisamente, la suavidad de la función sigmoide es crucial, ya que significa que los cambios pequeños Δw_j en los pesos y en el sesgo $\Delta\alpha$ producirán un cambio pequeño Δy en la salida de la neurona. Matemáticamente, se puede expresar de esta forma:

$$\Delta y = \sum_{j=1}^{n} \frac{\partial y}{\partial w_j}\Delta w_j + \frac{\partial y}{\partial \alpha}\Delta\alpha \tag{12.13}$$

donde $\frac{\partial y}{\partial w_j}$ denota la derivada parcial de la salida respecto a w_j y $\frac{\partial y}{\partial \alpha}$ representa la derivada parcial de la salida respecto a α.

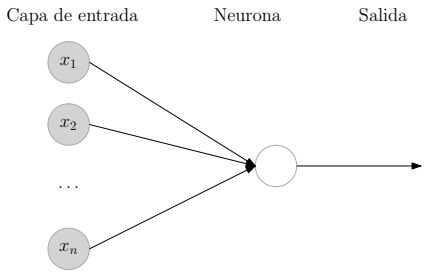
Así, aunque las neuronas sigmoides tienen cierta similitud de comportamiento cualitativo con los perceptrones, las primeras permiten que sea mucho más fácil definir cómo afectará a la salida el cambio en los pesos y sesgos.

12.2. Arquitectura de una red neuronal

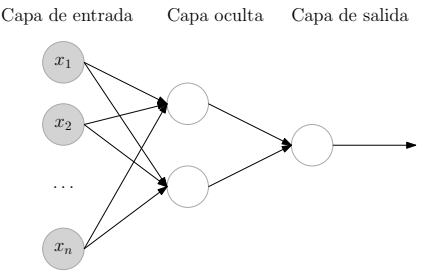
La arquitectura o topología de una red neuronal se define como la organización de las neuronas en distintas capas, así como los parámetros que afectan a la configuración de las neuronas, tales como las funciones de entrada o activación. Cada arquitectura o topología puede ser válida para algunos tipos de problemas, presentando diferentes niveles de calidad de los resultados y, también, diferentes niveles de coste computacional.

La red neuronal más simple está formada por una única neurona, conectada a todas las entradas disponibles y con una única salida. La arquitectura de esta red se muestra en

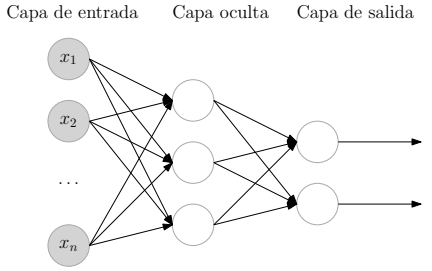
la figura 12.3a. Este tipo de redes permite efectuar funciones relativamente sencillas, equivalentes a la técnica estadística de regresión no lineal.



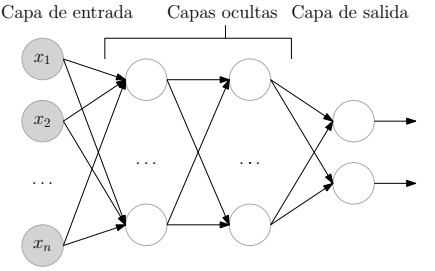
(a) Red con una neurona



(b) Red monocapa con una salida



(c) Red monocapa con múltiples salidas



(d) Red multicapa con múltiples salidas

Figura 12.3. Ejemplos de arquitecturas de redes neuronales

La segunda arquitectura más simple de redes neuronales la forman las redes monocapa. Estas redes presentan la capa de entrada; una capa oculta de procesamiento, formada por un conjunto variable de neuronas, y finalmente la capa de salida con una o más neuronas. La figura 12.3b muestra la arquitectura monocapa con una única salida, mientras que la figura 12.3c presenta una red monocapa con múltiples salidas (dos en este caso concreto). Este tipo de redes son capaces de clasifi-

car patrones de entrada más complejos o realizar predicciones sobre dominios de dimensionalidad más alta. El número de neuronas de la capa oculta está relacionado con la capacidad de procesamiento y clasificación, pero un número demasiado grande de neuronas en esta capa aumenta el riesgo de sobreespecialización en el proceso de entrenamiento. Finalmente, el número de neuronas de la capa de salida depende, en gran medida, del problema concreto que vamos a tratar y de la codificación empleada. Por ejemplo, en casos de clasificación binaria, una única neurona de salida suele ser suficiente, pero en casos de clasificación en n grupos se suele emplear n neuronas, donde cada una indica la pertenencia a una determinada clase.

Finalmente, se puede añadir un número indeterminado de capas ocultas, produciendo lo que se conoce como redes multicapa, como se puede ver en la figura 12.3d. Cuando se añaden neuronas y capas a una red se aumenta, generalmente, su poder de predicción, la calidad de dicha predicción y la capacidad de separación. Aunque también se aumenta su tendencia a la sobreespecialización y se aumenta el coste computacional y temporal de entrenamiento.

Hasta ahora, hemos estado discutiendo redes neuronales donde la salida de una capa se utiliza como entrada a la siguiente capa. Tales redes se llaman redes neuronales prealimentadas (*Feedforward Neural Networks*, FNN). Esto significa que no hay bucles en la red —la información siempre avanza, nunca se retroalimenta.

Sin embargo, hay otros modelos de redes neuronales artificiales en las que los bucles de retroalimentación son posibles. Estos modelos se llaman redes neuronales recurrentes (*Recurrent Neural Networks*, RNN). La idea en estos modelos

es tener neuronas que se activan durante un tiempo limitado. Esta activación puede estimular otras neuronas, que se pueden activar un poco más tarde, también por una duración limitada. Es decir, la redes recurrentes reutilizan todas o parte de las salidas de la capa i como entradas en la capa $i - q \mid q \geq 1$. Algunos ejemplos interesantes son las redes de Hopfield [6], que tienen conexiones simétricas, o las máquinas de Boltzmann [6].

12.2.1. Dimensiones de una red neuronal

A continuación discutiremos la importancia y parametrización de las dimensiones de la red neuronal artificial. En este sentido, debemos considerar:

- la dimensión de la capa de entrada
- la dimensión de la capa de salida
- la topología y dimensiones de las capas ocultas

La dimensión de la capa de entrada se fija inicialmente con el número de atributos que se consideran del conjunto de datos. Es relevante recordar que las estrategias de selección de atributos o reducción de dimensionalidad permiten reducir el número de elementos a considerar sin una pérdida excesiva en la capacidad de predicción del modelo.

La capa de salida de una red neuronal se define a partir del número de clases y de la codificación empleada para su representación. Generalmente, en un problema de clasificación, se utiliza una neurona en la capa de salida para cada clase a representar, de modo que solo una neurona debería de «activarse» para cada instancia. Alternativamente, cada neurona

en la capa de salida nos puede proporcionar el valor de pertenencia a una determinada clase en problemas de clasificación difusa.

La topología y dimensión de la capa oculta no es un problema trivial, y no existe una solución única y óptima *a priori* para este problema. Existe literatura específica dedicada a la obtención de la arquitectura óptima para una red neuronal artificial [16].

En general, añadir nuevas neuronas en las capas ocultas implica:

1. Aumentar el poder predictivo de la red, es decir, la capacidad de reconocimiento y predicción de la red. Pero también aumenta el peligro de sobreajuste a los datos de entrenamiento.

2. Disminuir la posibilidad de caer en un mínimo local [6].

3. Alterar el tiempo de aprendizaje. Este varía de forma inversa al número de neuronas de las capas ocultas [22].

Por el contrario, una red con menos neuronas en las capas ocultas permite:

1. Reducir el riesgo de sobreespecilización a los datos de entrenamiento, ya que al disponer de menos nodos se obtiene un modelo más general.

En la práctica, se suele utilizar un número de neuronas en las capas ocultas que se encuentre entre una y dos veces el número de entradas de la red. Si se detecta que la red está sobreajustando, debemos reducir el número de neuronas en las capas ocultas. Por el contrario, si la evaluación del modelo no es satisfactoria, debemos aumentar el número de neuronas en las capas ocultas.

12.3. Entrenamiento de una red neuronal

En esta sección discutiremos los principales métodos de entrenamiento de una red neuronal. En primer lugar, analizaremos el funcionamiento de una neurona tipo perceptrón. En la sección 12.3.2 veremos las bases teóricas del método del descenso del gradiente, que plantea las bases necesarias para el método de entrenamiento más utilizado en la actualidad, conocido como el método de retropropagación, que analizaremos en la sección 12.3.3.

12.3.1. Funcionamiento y entrenamiento de una neurona

Como hemos visto en las secciones anteriores, el funcionamiento genérico de una neurona viene determinado por la función:

$$y = \sigma(wx - \alpha) \tag{12.14}$$

donde σ representa la función de activación de la neurona, por ejemplo la función escalón en el caso de un perceptrón.

El proceso de aprendizaje de las neuronas se basa en la optimización de dos grupos de variables:

- El conjunto de pesos de la entrada de cada neurona $W^i = \{w_1^i, w_2^i, \ldots, w_n^i\}$.
- El valor umbral de la función de activación, también llamado sesgo, que denotaremos por α.

En el caso de un perceptrón, la salida del mismo viene determinada por:

$$y = \begin{cases} 0 & \text{si } wx - \alpha \leq 0 \\ 1 & \text{si } wx - \alpha > 0 \end{cases} \tag{12.15}$$

En consecuencia, el borde de separación entre estas dos regiones viene dado por la expresión $wx - \alpha = 0$. Limitándonos en el espacio bidimensional, obtenemos una recta que separa las dos clases. Los puntos por encima de la línea corresponden a la clase 1, mientras que por debajo corresponden a la clase 0. Podemos ver un ejemplo de esta recta en la figura 12.4 y su expresión matemática en la ecuación 12.16. La figura muestra distintos valores para los parámetros α, w_1 y w_2. Si se extiende a más dimensiones, se obtiene el hiperplano separador. Por lo tanto, podemos concluir que una neurona simple con función de entrada basada en suma ponderada y función de activación escalón es un separador lineal, es decir, permite clasificar instancias dentro de regiones linealmente separables.

$$x_2 = \frac{\alpha - x_1 w_1}{w_2} \tag{12.16}$$

12.3.2. Método del descenso del gradiente

Definimos el error que comete una red neuronal como la diferencia entre el resultado esperado en una instancia de entrenamiento y el resultado obtenido. Hay varias formas de cuantificarlo, como por ejemplo, utilizando el error cuadrático:

$$\varepsilon = \frac{1}{2}(c - y)^2 \tag{12.17}$$

donde c es el valor de clase de la instancia de entrenamiento e y la salida de la red asociada a esta misma instancia.

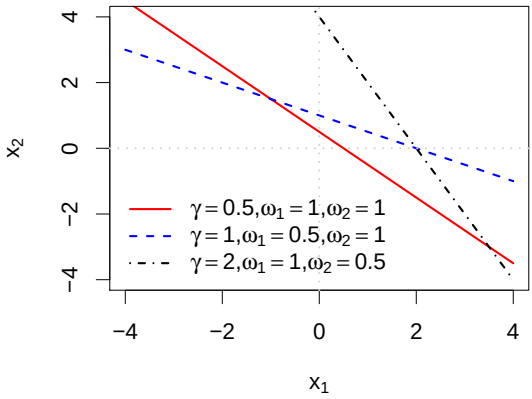


Figura 12.4. Espacio de clasificación de un perceptrón

Cada vez que aplicamos una instancia de entrenamiento al perceptrón, comparamos la salida obtenida con la esperada y, en el caso de diferir, deberemos modificar los pesos y el sesgo de las neuronas para que la red «aprenda» la función de salida deseada.

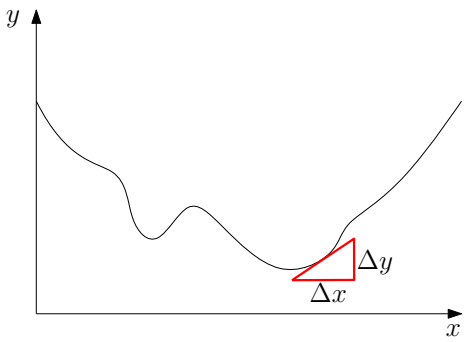


Figura 12.5. Caracterización del mínimo de una función

La idea principal es representar el error como una función continua de los pesos e intentar llevar la red hacia una configuración de valores de activación que nos permita encontrar el

mínimo de esta función. Para determinar la magnitud y dirección del cambio que debemos introducir en el vector de pesos para reducir el error que está cometiendo, utilizaremos el concepto de *pendiente*, ilustrado en la figura 12.5 y expresado por la ecuación:

$$\frac{\Delta y}{\Delta x} \tag{12.18}$$

que indica que la pendiente de un punto dado es el gradiente de la tangente a la curva de la función en el punto dado. La pendiente es cero cuando nos encontramos en un punto mínimo (o máximo) de la función, donde no hay pendiente.

La función que nos permite modificar el vector de pesos w a partir de una instancia de entrenamiento $d \in D$ se conoce como la regla delta y se expresa de la siguiente forma:

$$\Delta w = \eta(c - y)d \tag{12.19}$$

donde c es el valor que indica la clase en el ejemplo de entrenamiento, y el valor de la función de salida para la instancia d, y η la tasa o velocidad de aprendizaje (*learning rate*).

El método utilizado para entrenar este tipo de redes, y basado en la regla delta, se muestra en el algoritmo 6.

Es importante destacar que para que el algoritmo converja es necesario que las clases de los datos sean linealmente separables.

Funciones de activación no continuas

En el caso de que la función de activación neuronal no sea continua, como por ejemplo en el perceptrón u otras neuronas con función escalón, la función de salida de la red no es conti-

Algoritmo 6 Pseudocódigo del método del descenso del gradiente

Entrada: W (conjunto de vectores de pesos) y D (conjunto de instancias de entrenamiento)
  mientras $(y \not\approx c_i \forall (d_i, c_i) \in D)$ **hacer**
    para todo $((d_i, c_i) \in D)$ **hacer**
      Calcular la salida y de la red cuando la entrada es d_i
      si $(y \not\approx c_i)$ **entonces**
        Modificar el vector de pesos $w' = w + \eta(c - y)d_i$
      fin si
    fin para
  fin mientras
  devolver El conjunto de vectores de pesos W

nua. Por lo tanto, no es posible aplicar el método de descenso del gradiente sobre los valores de salida de la red, pero sí es posible aplicarlo sobre los valores de activación. Este técnica se conoce como *elementos adaptativos lineales* (*Adaptative Linear Elements* o ADALINE).

Por lo tanto, modificamos la ecuación anterior del error para adaptarla a las funciones no continuas:

$$\varepsilon = \frac{1}{2}(c - z)^2 \tag{12.20}$$

donde z es el valor de activación, resultado de la función de entrada de la neurona.

Aplicando el descenso del gradiente en esta función de error, obtenemos la siguiente regla delta:

$$\Delta w = \eta(c - z)d \tag{12.21}$$

El método utilizado para entrenar este tipo de redes, y basado en la regla delta, se muestra en el algoritmo 7.

Algoritmo 7 Pseudocódigo del método ADALINE

Entrada: W (conjunto de vectores de pesos) y D (conjunto de instancias de entrenamiento)
  mientras $(y \not\approx c_i \forall (d_i, c_i) \in D)$ **hacer**
    para todo $((d_i, c_i) \in D)$ **hacer**
      Calcular el valor de activación z de la red cuando la entrada es d_i
      si $(y \not\approx c_i)$ **entonces**
        Modificar el vector de pesos $w' = w - \eta(c - z)d_i$
      fin si
    fin para
  fin mientras
  devolver El conjunto de vectores de pesos W

12.3.3. Método de retropropagación

En las redes multicapa no podemos aplicar el algoritmo de entrenamiento visto en la sección anterior. El problema aparece con los nodos de las capas ocultas: no podemos saber *a priori* cuáles son los valores de salida correctos.

En el caso de una neurona j con función sigmoide, la regla delta es:

$$\Delta w_i^j = \eta \sigma'(z^j)(c^j - y^j)x_i^j \tag{12.22}$$

donde:

- $\sigma'(z^j)$ indica la pendiente (derivada) de la función sigmoide, que representa el factor con que el nodo j puede afectar al error. Si el valor de $\sigma'(z^j)$ es pequeño, nos

encontramos en los extremos de la función, donde los cambios no afectan demasiado a la salida (figura 12.2c). Por el contrario, si el valor es grande, nos encontramos en el centro de la función, donde pequeñas variaciones pueden alterar considerablemente la salida.

- $(c^j - y^j)$ representa la medida del error que se produce en la neurona j.
- x_i^j indica la responsabilidad de la entrada i de la neurona j en el error. Cuando este valor es igual a cero, no se modifica el peso, mientras que si es superior a cero, se modifica proporcionalmente a este valor.

La delta que corresponde a la neurona j puede expresarse de forma general para toda neurona, simplificando la notación anterior:

$$\delta^j = \sigma'(z^j)(c^j - y^j) \tag{12.23}$$

$$\Delta w_i^j = \eta \delta^j x_i^j \tag{12.24}$$

A partir de aquí debemos determinar qué parte del error total se asigna a cada una de las neuronas de las capas ocultas. Es decir, debemos definir cómo modificar los pesos y tasas de aprendizaje de las neuronas de las capas ocultas a partir del error observado en la capa de salida.

El método de retropropagación (*backpropagation*) se basa en un esquema general de dos pasos:

1. Propagación hacia adelante (*feedforward*), que consiste en introducir una instancia de entrenamiento y obtener la salida de la red neuronal.
2. Propagación hacia atrás (*backpropagation*), que consiste en calcular el error cometido en la capa de salida y

propagarlo hacia atrás para calcular los valores delta de las neuronas de las capas ocultas.

El algoritmo 8 muestra el detalle del método de retropropagación. No discutiremos aquí todo el proceso y derivación matemática de la regla de retropropagación, pero se puede consultar en [6].

Algoritmo 8 Pseudocódigo del método de retropropagación

Entrada: W (conjunto de vectores de pesos) y D (conjunto de instancias de entrenamiento)

mientras (error de la red $>\varepsilon$) **hacer**
  para todo $((d_i, c_i) \in D)$ **hacer**
    Calcular el valor de salida de la red para la entrada d_i
    para todo (neurona j en la capa de salida) **hacer**
      Calcular el valor δ^j para esta neurona: $\delta^j = \sigma'(z^j)(c^j - y^j)$
      Modificar los pesos de la neurona siguiendo el método del gradiente: $\Delta w_i^j = \eta \delta^j x_i^j$
    fin para
    para todo (neurona k en las capas ocultas) **hacer**
      Calcular el valor δ^k para esta neurona: $\delta^k = \sigma'(z^k) \sum_{j \in S_k} \delta^k w_k^j$
      Modificar los pesos de la neurona siguiendo el método del gradiente: $\Delta w_i^k = \eta \delta^k x_i^k$
    fin para
  fin para
fin mientras

El entrenamiento del algoritmo se realiza ejemplo a ejemplo, es decir, una instancia de entrenamiento en cada iteración. Para cada una de estas instancias se realizan los siguientes pasos:

1. El primer paso, que es la propagación hacia adelante, consiste en aplicar el ejemplo a la red y obtener los valores de salida (línea 3).

2. A continuación el método inicia la propagación hacia atrás, empezando por la capa de salida. Para cada neurona j de la capa de salida:

   a) Se calcula, en primera instancia, el valor δ^j basado en el valor de salida de la red para la neurona j (y^j), el valor de la clase de la instancia (c^j) y la derivada de la función sigmoide ($\sigma'(z^j)$) (línea 5).

   b) A continuación, se modifica el vector de pesos de la neurona de la capa de salida, a partir de la tasa de aprendizaje (η), el valor delta de la neurona calculado en el paso anterior (δ^j) y el factor x_i^j que indica la responsabilidad de la entrada i de la neurona j en el error (línea 6).

3. Finalmente, la propagación hacia atrás se aplica a las capas ocultas de la red. Para cada neurona k de las capas ocultas:

   a) En primer lugar, se calcula el valor δ^k basado en la derivada de la función sigmoide ($\sigma'(z^k)$) y el sumatorio del producto de la delta calculada en el paso anterior (δ^k) por el valor w_k^j, que indica el peso de la conexión entre la neurona k y la neurona

j (línea 9). El conjunto S_k está formado por todos los nodos de salida a los que está conectada la neurona k.

b) En el último paso de la iteración, se modifica el vector de pesos de la neurona k, a partir de la tasa de aprendizaje (η), el valor delta de la neurona calculado en el paso anterior (δ^k) y el factor x_i^k que indica la responsabilidad de la entrada k de la neurona j en el error (línea 10).

La idea que subyace a este algoritmo es relativamente sencilla, y se basa en propagar el error de forma proporcional a la influencia que ha tenido cada nodo de las capas ocultas en el error final producido por cada una de las neuronas de la capa de salida.

12.4. Optimización del proceso de aprendizaje

En esta sección veremos una serie de técnicas que pueden ser usadas para mejorar los resultados de entrenamiento de las redes neuronales. En concreto, veremos cuatro técnicas comúnmente utilizadas, de entre una gran multitud de herramientas para mejorar y optimizar el proceso de aprendizaje:

- Una alternativa para la función de coste que hemos visto anteriormente, conocida como la función de coste de la entropía cruzada (*cross-entropy cost function*).
- Un nuevo tipo de neurona, conocido como *softmax*, que permite trabajar con particiones difusas o probabilidades en los resultados de la red.

- Varios métodos de «regularización» (*regularization*), que permiten a las redes neuronales generalizar más allá de los datos de entrenamiento, reduciendo de esta forma el sobreentrenamiento (*overfitting*).
- Un método alternativo para inicializar los pesos en la red.
- Y un conjunto de heurísticas que nos pueden ayudar a elegir los parámetros de configuración de la red.

12.4.1. Función de entropía cruzada

En algunos casos, el proceso de aprendizaje es muy lento en las primeras etapas de entrenamiento. Esto no significa que la red no esté aprendiendo, pero lo hace de forma muy lenta en las primeras etapas de entrenamiento, hasta alcanzar un punto donde el aprendizaje se acelera de forma considerable. Cuando usamos la función de coste cuadrática, el aprendizaje es más lento cuando la neurona produce resultados muy dispares con los resultados esperados. Esto se debe a que la neurona sigmoide de la capa de salida se encuentra saturada en los valores extremos, ya sea en el valor 0 o 1 y, por lo tanto, los valores de sus derivadas en estos puntos son muy pequeños, produciendo pequeños cambios en la red que generan un aprendizaje lento.

En estos casos, es preferible utilizar otra función de coste, que permita que las neuronas de la capa de salida puedan aprender más rápido, aun estando en un estado de saturación. Definimos la función de coste de entropía cruzada (*cross-entropy cost function*) para una neurona como:

$$C = -\frac{1}{n}\sum_{d}[c\ ln(y) + (1-c)ln(1-y)] \qquad (12.25)$$

donde n es el número total de instancias de entrenamiento, d es cada una de las instancias de entrenamiento, c es la salida deseada correspondiente a la entrada d e y es la salida obtenida.

Generalizando la función para una red de múltiples neuronas y capas, obtenemos la siguiente expresión para la función de coste:

$$C = -\frac{1}{n}\sum_{d}\sum_{j}[c_j\ ln(y_j^L) + (1-c_j)ln(1-y_j^L)] \qquad (12.26)$$

donde $c = c_1, c_2, \ldots$ son los valores deseados e $y_1^L, y_2^L, \ldots$ son los valores actuales en las neuronas de la capa de salida.

Utilizando la función de coste de entropía cruzada, el aprendizaje es más rápido cuando la neurona produce valores alejados de los valores esperados. Este comportamiento no está relacionado con la tasa de aprendizaje, con lo cual no se resuelve modificando este valor.

En general, utilizar la función de coste basada en la entropía cruzada suele ser la mejor opción, siempre que las neuronas de salida sean neuronas sigmoides, dado que en la inicialización de pesos puede producir la saturación de neuronas de salida cerca de 1, cuando deberían ser 0, o viceversa. En estos casos, si se utiliza la función de coste cuadrático el proceso de entrenamiento será más lento, aunque no nulo. Pero, obviamente, es deseable un proceso de entrenamiento más rápido en las primeras etapas.

12.4.2. *Softmax*

La idea de *softmax* es definir un nuevo tipo de neurona para la capa de salida de una red neuronal.

La función de entrada de una neurona *softmax* es la misma que en el caso de una neurona sigmoide, es decir, para la neurona j de la capa L el valor de entrada es:

$$z_j^L = \sum_k w_{jk}^L \, y_k^{L-1} + \alpha_j^L \tag{12.27}$$

donde w_{jk}^L representa el peso asignado a la conexión entre la neurona k de la capa $L-1$ y la neurona j de la capa L; y_k^{L-1} indica el valor de salida de la neurona k de la capa $L-1$ y α_j^L es el valor de sesgo de la neurona j de la capa L.

En este caso, en lugar de aplicar la función sigmoide al valor de entrada, esto es $\sigma(z_j^L)$, se aplica la denominada función *softmax*:

$$y_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{12.28}$$

donde en el denominador se suma sobre todas las neuronas de salida.

Los valores de salida de la capa de *softmax* tienen dos propiedades interesantes:

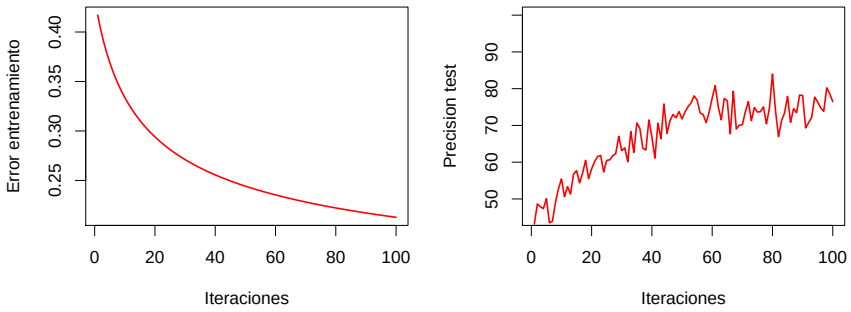
- Los valores de activación de salida son todos positivos, debido a que la función exponencial es siempre positiva.
- El conjunto de los valores de salida siempre suma 1, es decir, $\sum_j y_j^L = 1$.

En otras palabras, la salida de la capa *softmax* puede ser interpretada como una distribución de probabilidades. En mu-

chos problemas es conveniente interpretar la activación de salida y_j^L como la estimación de la probabilidad de que la salida correcta sea j.

12.4.3. Sobreentrenamiento y regularización

Supongamos que tenemos una red y que analizamos el error en el conjunto de entrenamiento durante las cien primeras etapas del proceso (*epochs*). La figura 12.6a muestra un posible resultado, que *a priori* parece bueno, ya que el error de la red desciende de forma gradual conforme avanza el proceso de entrenamiento. Pero si también analizamos la precisión del conjunto de test asociado a cada etapa del proceso de entrenamiento, podemos obtener unos datos similares a los presentados por la figura 12.6b. En este caso vemos que la precisión de la red mejora hasta llegar al *epoch=50*, donde empieza a fluctuar pero no se aprecia una mejora significativa en la precisión de la red.



(a) Error en el conjunto de entrenamiento **(b)** Precisión en el conjunto de test

Figura 12.6. Ejemplo de sobreentrenamiento

Por lo tanto, nos encontramos ante un ejemplo donde el error de entrenamiento nos dice que la red está mejorando a cada etapa de entrenamiento, mientas que el conjunto de test nos dice que a partir de cierto punto no hay mejora alguna en la precisión de la red. Nos encontramos ante un caso de sobreentrenamiento (*overfitting* o *overtraining*) de la red.

La red reduce el error de entrenamiento porque se va adaptando a los datos del conjunto de entrenamiento, pero ya no es capaz de generalizar correctamente para datos nuevos, en este caso el conjunto de test, y los resultados que produce no mejoran con los nuevos datos.

El sobreentrenamiento es un problema importante en las redes neuronales. Esto es especialmente cierto en las redes modernas, que a menudo tienen un gran número de pesos y sesgos. Para entrenar con eficacia, necesitamos una forma de detectar cuándo se está sobreentrenando.

Aumentar la cantidad de datos de entrenamiento es una forma de reducir este problema. Otro enfoque posible es reducir el tamaño de la red. Sin embargo, las redes grandes tienen el potencial de ser más poderosas que las redes pequeñas. Afortunadamente, existen otras técnicas que pueden reducir el sobreentrenamiento, incluso cuando tenemos una red fija y datos de entrenamiento fijos. Estas son conocidas como técnicas de regularización.

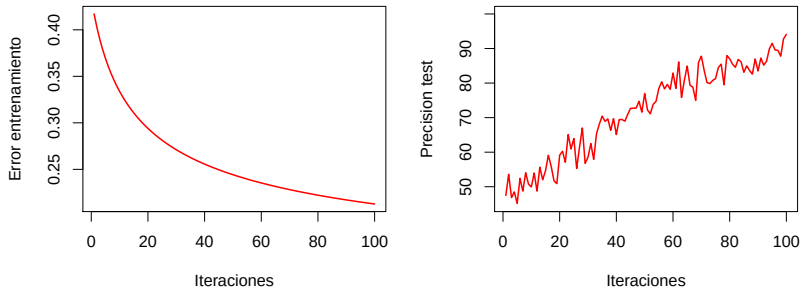
A continuación veremos una de las técnicas de regularización más utilizadas, conocida como regularización L2 (*L2 regularization* o *weight decay*).

La idea de la regularización L2 es añadir un término extra a la función de coste, llamado el término de regularización:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{12.29}$$

donde C_0 es la función de coste original, esto es, no regularizada.

El objetivo de la regularización es que la modificación de pesos en la red se realice de forma gradual. Es decir, la regularización puede ser vista como una forma de equilibrio entre encontrar pesos pequeños y minimizar la función de coste. La importancia relativa de los dos elementos del equilibrio depende del valor de λ: cuando este es pequeño, se minimiza la función de coste original; en caso contrario, se opta por premiar los pesos pequeños.



(a) Error en el conjunto de entrenamiento

(b) Precisión en el conjunto de test

Figura 12.7. Ejemplo de regularización

La figura 12.7 muestra, empíricamente, que la regularización está ayudando a que la red generalice mejor y reduzca, por lo tanto, los efectos del sobreentrenamiento. Como se puede ver en la figura, la precisión en el conjunto de test continúa mejorando junto con el error en el conjunto de entrenamiento, signo de que no se está produciendo sorbreentrenamiento. Se ha demostrado, aunque solo de forma empírica, que las redes neuronales regularizadas suelen generalizar mejor que las redes no regularizadas.

Existen muchas otras técnicas de regularización distintas de la regularización L2. Otros enfoques para reducir el sobreentrenamiento pueden ser: la regularización L1 (*L1 regularization*), la técnica conocida como «abandono» (*dropout*) o la técnica basada en aumentar artificialmente el tamaño del conjunto de entrenamiento (*artificially expanding the training data*) [21], entre muchos otros.

12.4.4. Inicialización de los pesos de la red

Es habitual escoger aleatoriamente los pesos y el sesgo de las neuronas de forma aleatoria, generalmente utilizando valores aleatorios independientes de una distribución gaussiana con media 0 y la desviación estándar 1.

Una inicialización de los pesos más eficientes puede ayudar a la red, en especial a las neuronas de las capas ocultas, a reducir el efecto de aprendizaje lento, de forma similar a como el uso de la función de entropía cruzada ayuda a las neuronas de la capa de salida.

Una de las opciones más utilizadas consiste en inicializar los pesos de las neuronas en las capas ocultas de la red como variables aleatorias gaussianas con media 0 y desviación estándar $\frac{1}{\sqrt{n_{in}}}$, donde n_{in} es el número de entradas de la neurona. Con esto conseguiremos reducir la probabilidad de que las neuronas de las capas ocultas queden saturadas durante el proceso de entrenamiento, mejorando por lo tanto la velocidad en dicho proceso.

Es importante subrayar que la inicialización de pesos nos puede permitir un aprendizaje más rápido, pero no implica necesariamente una mejora en el rendimiento final del proceso de entrenamiento.

12.4.5. Elección de los parámetros de la red

Más allá de las optimizaciones vistas hasta este punto, debemos calibrar algunos parámetros, como por ejemplo la tasa de aprendizaje (η) o el parámetro de regularización (λ).

La velocidad de aprendizaje (determinada por el parámetro η) es un factor relevante sobre el proceso de aprendizaje de la red: valores muy altos de este parámetro pueden provocar que la red se vuelva inestable, es decir, se aproxima a valores mínimos, pero no es capaz de estabilizarse entorno a estos valores. En general, se recomienda empezar el proceso con velocidades de aprendizaje altas, e ir reduciéndolas para estabilizar la red.

Una buena estrategia, aunque no la única ni necesariamente la mejor, es empezar por la calibración de la tasa de aprendizaje. En este sentido es recomendable realizar un conjunto de pruebas con distintos valores de η, que sean sensiblemente diferentes entre ellos. Por ejemplo, la figura 12.8 muestra el error en la función de coste durante el entrenamiento utilizando tres valores muy distintos para el parámetro η. Con el valor $\eta = 10$ podemos observar que el valor de error en el coste realiza considerables oscilaciones desde el inicio del proceso de entrenamiento, sin una mejora aparente. El segundo valor testado, $\eta = 1$, reduce el error de la función de coste en las primeras etapas del entrenamiento, pero a partir de cierto punto oscila suavemente de forma aleatoria. Finalmente, en el caso de utilizar $\eta = 0{,}1$ vemos que los valores de error decrecen de forma progresiva y suave durante todo el proceso de prueba. Se suele llamar *umbral de aprendizaje* al valor máximo de η que produce un decrecimiento durante las primeras etapas del proceso de aprendizaje. Una vez determinado este valor de umbral, es aconsejable seleccionar valores de η inferiores,

en general, una o dos magnitudes inferiores, para asegurar un proceso de entrenamiento adecuado.

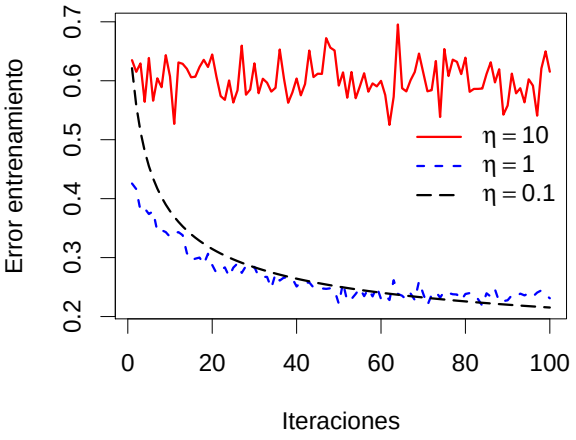


Figura 12.8. Comparación del error de entrenamiento con múltiples valores de η

Una vez se ha determinado la tasa de aprendizaje, esta puede permanecer fija durante todo el proceso. Aun así, tiene sentido reducir su valor para realizar ajustes más finos en las etapas finales del proceso de entrenamiento. En este sentido, se puede establecer una tasa de aprendizaje variable, que decrezca de forma proporcional al error de la función de coste.

El parámetro de regularización (λ) se suele desactivar para realizar el calibrado de la tasa de aprendizaje, esto es $\lambda = 0$. Una vez se ha establecido la tasa de aprendizaje, podemos activar la regularización con un valor de $\lambda = 1$ e ir incrementando y decrementando el valor según las mejoras observadas en el rendimiento de la red.

Otro parámetro importante se refiere al número de iteraciones del proceso de aprendizaje. Se puede utilizar un número fijo, aunque no es sencillo determinar cuál es el valor óptimo.

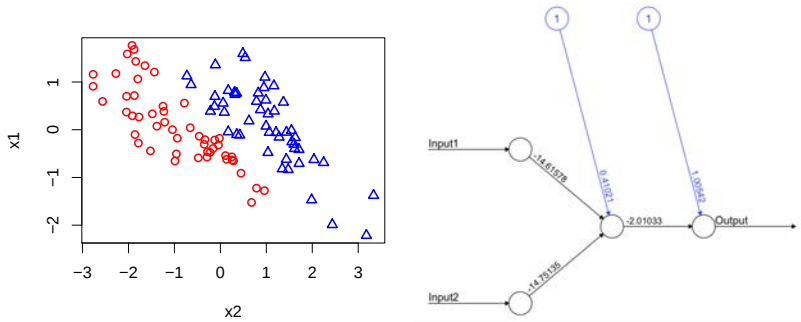
Otra estrategia más simple y eficaz consiste en terminar el proceso de aprendizaje de forma automática, por ejemplo cuando no se producen mejoras durante un número seguido de iteraciones. Es importante tener en cuenta que en algunos casos la mejora del entrenamiento puede «estancarse» durante unas cuantas iteraciones, para volver a mejorar a continuación. Es importante definir este error de forma correcta, que sea proporcional a los valores empleados en la salida y al cálculo del error de cada instancia de entrenamiento. Lógicamente, valores de umbral del error muy pequeños dificultarán el proceso de estabilización de la red, mientras que valores demasiado grandes producirán modelos poco precisos.

Existen distintas técnicas para automatizar este proceso, como por ejemplo la técnica conocida como *grid search* [26]. Además, este es un campo activo de investigación, donde aparecen nuevas técnicas y mejoras sobre las anteriores.

12.5. Ejemplo de aplicación

En primer lugar, veremos un ejemplo sencillo con un conjunto de datos linealmente separables (figura 12.9a). Como ya hemos comentado anteriormente, este tipo de conjuntos puede ser clasificado correctamente con un modelo de red neuronal basado en una única neurona, que «aprende» la función necesaria para separar ambas clases. En este caso concreto, estamos hablando de un espacio de dos dimensiones, con lo cual el modelo genera una recta que divide el espacio de datos en dos partes, una para cada clase. También hemos comentado que el modelo generado por una red neuronal, una vez entrenado, incluye la arquitectura concreta (número de neuronas y capas), el tipo de función de entrada y activación, y

los pesos correspondientes a todas las conexiones de la red. En este ejemplo, el modelo generado para la clasificación de estos puede verse en la figura 12.9b. Este modelo clasifica correctamente todas las instancias del conjunto de entrenamiento y test.



(a) Ejemplo de datos linealmente separables **(b)** Red monocapa con una salida

Figura 12.9. Ejemplos basados en datos lineales

A continuación veremos un ejemplo utilizando datos en un espacio bidimensional pero, a diferencia del caso anterior, estos presentan una estructura claramente radial y no pueden, por lo tanto, ser clasificados correctamente por una red como la vista en el caso anterior.

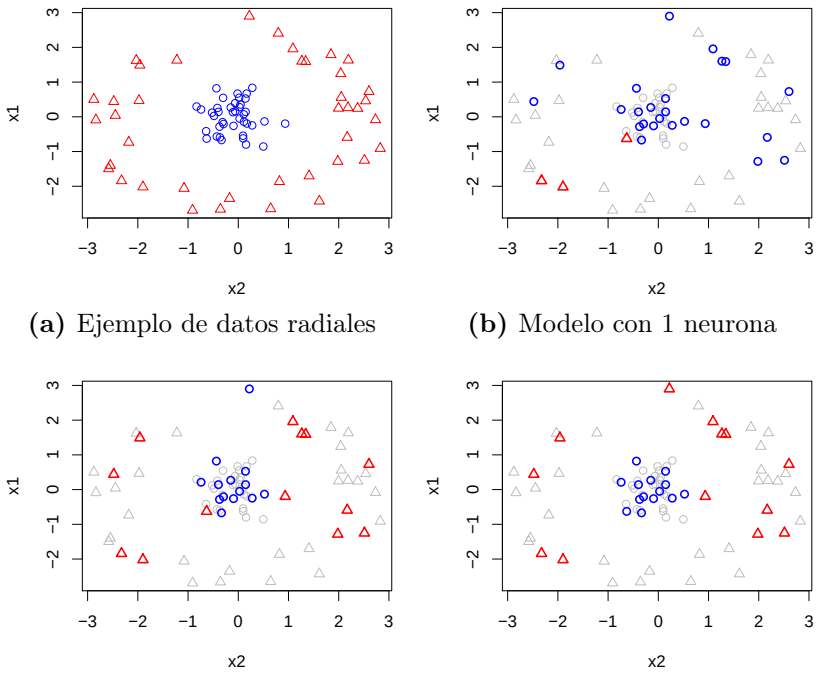
Los datos del ejemplo se pueden ver en la figura 12.10a. La clase del centro se muestra con círculos azules (clase 1) y la otra mediante triángulos rojos (clase 2). Aunque ya hemos comentado que una única neurona no es capaz de separar datos que no sean linealmente separables, vamos a ver el resultado que obtendría una arquitectura similar a la utilizada en el ejemplo anterior, basada en una única neurona en la capa oculta. En este caso, la precisión del modelo se reduce a 0,592

es decir, 59,2 % de instancias correctamente clasificadas. La figura 12.10b muestra la clasificación propuesta por el modelo de las instancias de test. Las instancias de entrenamiento se muestran en color gris, para facilitar la visualización de los resultados. Podemos ver que se han clasificado muchas instancias como clase 1 cuando realmente pertenecen a la clase 2.

Si ampliamos a una red con dos neuronas en la capa oculta, el conjunto de la red será capaz de usar dos rectas separadoras para la clasificación de los datos. En este caso la precisión sube hasta 0,888 añadiendo una sola neurona en la capa oculta. La visualización de los resultados obtenidos sobre las instancias de test (figura 12.10c) permite identificar de forma aproximada los dos hiperplanos (en este caso concreto, rectas en el espacio de dos dimensiones) que se han utilizado para la clasificación. Ampliando hasta diez neuronas en una única capa oculta nos permite mejorar hasta una precisión de 0,963; es decir, solo una instancia de test incorrectamente clasificada. Los resultados se pueden ver en la figura 12.10d. El resultado es exactamente el mismo que si creamos una red de dos capas, con arquitectura $(3, 2)$, es decir, tres neuronas en la primera capa oculta y dos en la segunda.

12.6. Redes neuronales profundas

Muchos de los problemas existentes pueden ser resueltos mediante redes neuronales con una sola capa oculta, más la capa de entrada y la capa de salida. Aun así, se espera que las redes con un número mayor de capas ocultas puedan «aprender» conceptos más abstractos, y por lo tanto, pueden aproximar problemas más complejos o mejorar las soluciones obtenidas con arquitecturas basadas en una única capa oculta.



(a) Ejemplo de datos radiales **(b)** Modelo con 1 neurona

(c) Modelo con 2 neuronas **(d)** Modelo con 10 neuronas

Figura 12.10. Ejemplo basado en datos radiales

En esta sección veremos los problemas que han propiciado la aparición de las redes neuronales profundas (*deep neural networks*).

12.6.1. El problema de la desaparición del gradiente

Cuando intentamos entrenar redes con múltiples capas ocultas empleando los algoritmos de entrenamiento vistos hasta el momento encontramos el problema conocido como la desaparición del gradiente (*the vanishing gradient problem*).

El gradiente, empleado en el algoritmo de entrenamiento que hemos visto, se vuelve inestable en las primeras capas de neuronas cuando tratamos con redes con múltiples capas ocultas, produciendo una explosión del aprendizaje (*exploding gradient problem*) o una desaparición del mismo (*vanishing gradient problem*). En ambos casos, dificulta enormemente el aprendizaje de las neuronas en las primeras capas de la red. Esta inestabilidad es un problema fundamental para el aprendizaje basado en gradientes en redes neuronales profundas, es decir, en redes neuronales con múltiples capas ocultas.

El problema fundamental es que el gradiente en las primeras capas se calcula a partir de una función basada en el producto de términos de todas las capas posteriores. Cuando hay muchas capas, produce una situación intrínsecamente inestable. La única manera de que todas las capas puedan aprender a la misma velocidad es equilibrar todos los productos de términos que se emplean en el aprendizaje de las distintas capas.

En resumen, el problema es que las redes neuronales sufren de un problema de gradiente inestable, ya sea por desaparición o explosión del mismo. Como resultado, si usamos técnicas de aprendizaje basadas en gradiente estándar, las diferentes capas de la red tenderán a aprender a velocidades muy diferentes, dificultando el proceso de aprendizaje de la red.

12.6.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (*convolutional neural networks*) son especialmente útiles cuando se trabaja con imágenes. Estas redes utilizan una arquitectura especial que está particularmente adaptada para clasificar imágenes. El uso de una arquitectura especial hace que las redes convolucionales

sean rápidas para entrenar, facilitando el entrenamiento de redes profundas de muchas capas, que son muy buenas en la clasificación de imágenes. Hoy en día, las redes convolucionales profundas, o alguna variante cercana, se utilizan en la mayoría de las redes neuronales para el reconocimiento de imágenes.

Las redes neuronales convolucionales usan tres ideas básicas que las diferencian de las redes vistas hasta ahora: los campos receptivos locales, los pesos compartidos y las capas de agrupación.

Campos receptivos locales (*local receptive fields*)

En los problemas de clasificación de imágenes, cada uno de los píxeles de la imagen se asocia a una entrada de la capa de entrada. Las redes convolucionales proponen una arquitectura que intenta aprovechar las propiedades espaciales de las imágenes, donde la posición de una característica dentro de la imagen es relevante.

Las redes vistas hasta ahora proponen una arquitectura donde las capas de red adyacentes están completamente conectadas entre sí. Es decir, cada neurona de la red está conectada a todas las neuronas de las capas adyacentes (anterior y posterior). Este tipo de redes reciben el nombre de redes totalmente conectadas (*fully connected networks*).

En las redes convolucionales, las neuronas de una capa se conectan solo a un subconjunto de las neuronas de la capa siguiente, y asimismo están conectadas solo a un subconjunto de neuronas contiguas de la capa anterior. La figura 12.11 muestra un ejemplo de esta arquitectura, donde cada grupo de 3×3 neuronas de la capa i se conectan con una neurona de la capa $i + 1$.

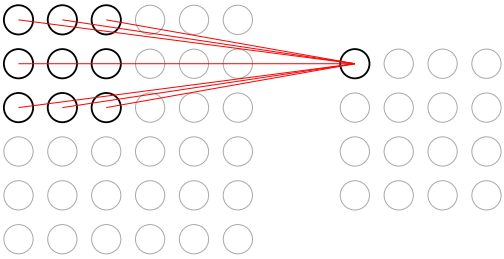


Figura 12.11. Ejemplo de campos receptivos locales de una red convolucional

Pesos compartidos (*shared weights*)

Según la estructura presentada, cada neurona oculta tiene un sesgo y un conjunto de pesos que depende del tamaño de su campo receptivo local. A diferencia de lo visto anteriormente, en este tipo de redes se usan los mismos pesos y sesgos para todas las neuronas ocultas de una misma capa. En otras palabras, para la neurona oculta, la salida es:

$$\sigma\Big(b + \sum_{l=0}^{n-1}\sum_{m=0}^{n-1} w_{l,m} a_{j+l,k+m}\Big) \tag{12.30}$$

donde σ representa la función de activación, b es el valor compartido de sesgo, $w_{l,m}$ es una matriz de $n \times n$ que contiene los pesos compartidos de las neuronas de una misma capa, y $a_{x,y}$ es el valor de entrada de la posición (x, y).

Esto significa que todas las neuronas de la capa oculta detectan exactamente la misma característica, solo que realizan esta función en diferentes ubicaciones de la imagen de entrada. Supongamos que los pesos y sesgos son tales que la neurona oculta puede distinguir, digamos, un borde vertical en un campo receptivo local particular. Esta misma habilidad es probable que sea útil en otros lugares de la imagen.

Por lo tanto, es útil aplicar el mismo detector de características en todas partes de la imagen. Para ponerlo en términos un poco más abstractos, las redes convolucionales están bien adaptadas a la invariancia de las características a detectar.

Generalmente, se llama mapa de características (*feature map*) a la relación entre las neuronas de la capa de entrada a la capa oculta. Los pesos compartidos (*shared weights*) y el sesgo compartido son los pesos y sesgo que definen el mapa de características. A menudo se dice que los pesos compartidos y el sesgo definen un *kernel* o *filtro*.

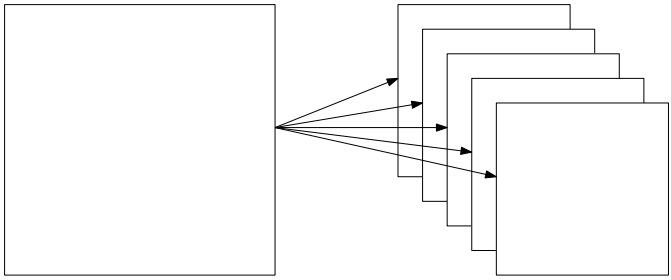


Figura 12.12. Ejemplo de estructura con cinco mapas de características en la capa oculta

Las redes convolucionales deberán utilizar múltiples mapas de características según el número de patrones a detectar en los datos en entrada. Por ejemplo, la figura 12.12 muestra una estructura con una capa de entrada conectada a cinco mapas de características que forman la primera capa oculta de neuronas.

Una gran ventaja de compartir pesos y sesgos es que reduce en gran medida el número de parámetros involucrados en una red convolucional. Esto, a su vez, resultará en un entrenamiento más rápido para el modelo convolucional, y en última

instancia, nos ayudará a construir redes profundas usando capas convolucionales.

Capas de agrupación (*pooling layers*)

Las capas de agrupamiento (*pooling layers*) se suelen usan inmediatamente después de las capas convolucionales. Su principal función es simplificar la información en la salida de la capa convolucional. En concreto, la capa de agrupación recibe las salidas de todos los mapas de características de la capa convolucional y prepara un mapa de características condensado.

Una de las agrupaciones más conocidas y usadas es la función *max-pooling*, que simplemente selecciona el valor máximo de conjunto de entradas. El factor de reducción del número de neuronas entre la capa de convolución y la capa de agrupación se determina según la región de entrada para cada neurona de la capa de agrupación. La figura 12.13 muestra un ejemplo donde una capa de convolución de 6×6 neuronas se convierte en una capa de agrupación 3×3 utilizando una región de entrada de 2×2.

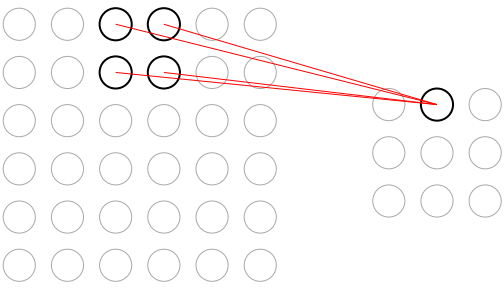


Figura 12.13. Ejemplo de capa de agrupación *max-pooling* con región de entrada 2×2

La función *max-pooling* aplicada a una capa de agrupación permite que la red sepa si una característica dada se encuentra en cualquier parte de una región de la imagen, ignorando la posición exacta dentro de la imagen. Intuitivamente, podemos decir que es más importante conocer si existe una característica dentro de la imagen y su posición relativa u otras características, que la posición exacta de esta. La principal ventaja es la reducción del número de parámetros necesarios en capas posteriores.

Otra función de agrupación común se conoce como agrupación de L2 (*L2 pooling*). Esta función calcula la raíz cuadrada de la suma de los cuadrados de los valores de activación. Aunque los detalles son diferentes entre las dos funciones, la idea principal es similar: ambas funciones condensan la información de la capa convolucional. En la práctica, ambas técnicas han sido ampliamente utilizadas.

Resumiendo todo lo visto hasta ahora, el formato final de una red convolucional puede ser similar al que muestra la figura 12.14. La red comienza con las 28×28 neuronas de entrada, que se utilizan para codificar las intensidades de píxeles de la imagen. Esto es seguido por una capa convolucional usando un campo receptivo local de 5×5 y tres mapas de características. El resultado es una capa de características de $3 \times 24 \times 24$ neuronas ocultas. El siguiente paso es una capa de agrupación *max-pooling*, aplicada a regiones 2×2, en cada uno de los mapas de características. El resultado es una capa de agrupación de $3 \times 12 \times 12$ neuronas ocultas. Finalmente, la capa final de conexiones en la red es una capa totalmente conectada. Es decir, esta capa conecta cada neurona de la capa de agrupación a cada una de las neuronas de salida.

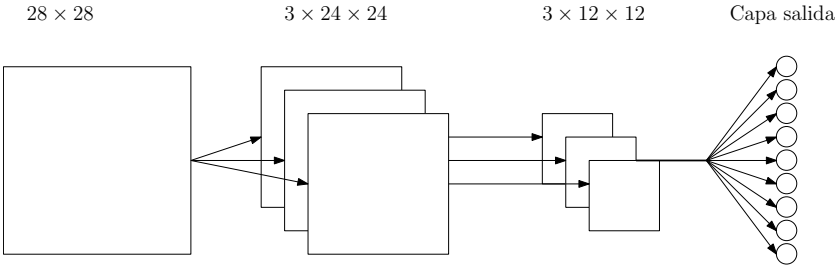


Figura 12.14. Ejemplo de red convolucional completo para procesamiento de imágenes de 28×28 píxeles

12.7. Resumen

Las redes neuronales artificiales permiten resolver problemas relacionados con la clasificación y predicción. Una de sus principales ventajas es que son capaces de lidiar con problemas de alta dimensionalidad y encontrar soluciones basadas en hiperplanos no lineales.

Por el contrario, dos de sus principales problemas o inconvenientes son: en primer lugar, el conocimiento está «oculto» en las redes, y en determinados casos puede ser complejo explicitar este conocimiento de forma clara. En segundo lugar, la preparación de los datos de entrada y salida no es una tarea trivial. Es recomendable asegurarse que los datos de entrada se encuentren en el intervalo $[0, 1]$, lo cual implica unos procesos previos de transformación de los datos.

Las redes neuronales se utilizan muy a menudo para problemas de clasificación relacionados con imágenes, así como otros problemas en los cuales tenemos datos de muy alta dimensionalidad.

Capítulo 13

Árboles de decisión

Los árboles de decisión son uno de los modelos de minería de datos más comunes y estudiados, y no precisamente por su capacidad predictiva, superada generalmente por otros modelos más complejos, sino por su alta capacidad explicativa y la facilidad para interpretar el modelo generado. Por ejemplo, son ampliamente utilizados en sectores como el bancario y las compañías aseguradoras para tomar decisiones respecto a la concesión de créditos o el cálculo de las pólizas, respectivamente, dado que permiten determinar qué características relativas a los usuarios son las que comportan mayor o menor riesgo, siendo posible segmentar a los usuarios en función de dichas características.

Como modelos supervisados que son, a los árboles de decisión que clasifican los datos del conjunto de entrada en función de una variable clasificadora categórica (es decir, que toma un conjunto finito de valores) se los llama *árboles de clasificación*. Si la variable clasificadora es continua, hablaríamos de árboles de regresión. Ambos tipos de árboles de decisión dan el nombre común CART (*Classification and Regression Trees*) según

el trabajo original descrito por Breiman, Friedman, Stone y Olhsen en 1984 [4].

Básicamente, los árboles de decisión son un modelo de minería de datos que intenta subdividir el espacio de datos de entrada para generar regiones disjuntas, de forma que todos los elementos que pertenezcan a una misma región sean de la misma clase, la cual es utilizada como representante o clase de dicha región. Si una región contiene datos de diferentes clases es subdividida en regiones más pequeñas siguiendo el mismo criterio, hasta particionar todo el espacio de entrada en regiones disjuntas que solamente contienen elementos de una misma clase. Así, un árbol de decisión se llama completo o puro si es posible generar una partición del espacio donde cada subregión solo contenga elementos de una misma clase. Esto siempre será posible, excepto si en el conjunto de datos de entrada existen elementos idénticos etiquetados con clases diferentes. En este caso será necesario decidir si se trata de *outliers* o bien se trata de casos que deben ser tratados de forma separada.

Respecto a su estructura, un árbol de decisión consta de nodos hoja o terminales, que representan regiones etiquetadas o clasificadas de acuerdo a una clase, y nodos internos o *splits* que representan condiciones que permiten decidir a qué subregión va cada elemento que llega a dicho nodo, es decir, la partición realizada. Cuando se presenta un nuevo dato a un árbol de decisión, se empieza por el nodo raíz que contiene una condición la cual determina por qué rama del árbol debe descenderse (es decir, a qué subregión pertenece el dato) hasta alcanzar la siguiente condición o bien una hoja terminal, en cuyo caso se determina qué clase corresponde al nuevo dato. Contrariamente a la idea intuitiva ligada al concepto de

árbol, la raíz es el elemento superior y las hojas se sitúan en posiciones más profundas. Así, la profundidad máxima de un árbol de decisión es el máximo número de condiciones que es necesario resolver para llegar a una hoja, siendo posible medir también la profundidad media, ponderando la profundidad de cada hoja con respecto al número de elementos del conjunto de entrenamiento que contiene.

Por lo tanto, un árbol de decisión es una secuencia de condiciones que son interrogadas con respecto a los datos de entrada, tomando una decisión parcial que lleva hacia una rama u otra, repitiendo este proceso hasta llegar a una hoja donde se toma una decisión final. Por ejemplo, en el hundimiento del Titanic, mostrado en la figura 13.1, de las 2201 personas a bordo solamente sobrevivieron 711 (un 32,3 %). Si se tuviera que tomar una decisión sobre todos los embarcados, esta sería «no sobrevivieron», dado que es correcta en el 67,7 % de los casos. Pero si analizamos la supervivencia con respecto a otras variables disponibles, la primera condición es el sexo, dado que las mujeres a bordo del Titanic (344 de 470, un 73,2 %) sobrevivieron en mayor proporción que los hombres (367 de 1731, un 21,2 %). Esto genera una partición del conjunto de entrada en dos conjuntos disjuntos; tras la primera condición (el nodo raíz), por una rama del árbol solo quedarán hombres y por la otra mujeres. La decisión con respecto a las mujeres sería «sobrevivieron», dado que lo hicieron en un 73,2 %. En cambio, en el caso de los hombres, la decisión sería «no sobrevivieron», la cual sería cierta en un 78,8 % de los casos. Se puede observar que la precisión de la predicción ha aumentado para ambas hojas con respecto a la primera predicción. Esto siempre sucede al menos para una hoja, dado que el procedimiento seguido intenta mejorar la predicción anterior. De la misma manera, en el caso de los hombres, la siguiente condición es la edad, dado que los niños sobrevivieron en una proporción

mayor que los adultos. En cambio, en el caso de las mujeres, la siguiente condición es la clase en la cual viajaban, dado que las mujeres que lo hacían en primera clase sobrevivieron más que las que lo hacían en segunda y mucho más que las que lo hacían en tercera clase. De hecho, las mujeres de tercera clase tuvieron una tasa de supervivencia inferior al 50 %, mientras que el resto sobrevivió en más de un 90 %. Nótese que cada rama puede contener una secuencia de condiciones diferentes, lo cual aporta información sobre el problema a resolver. De hecho, este árbol de decisión representa razonablemente bien la típica frase «las mujeres (no pobres) y los niños primero» usada en catástrofes de este tipo.

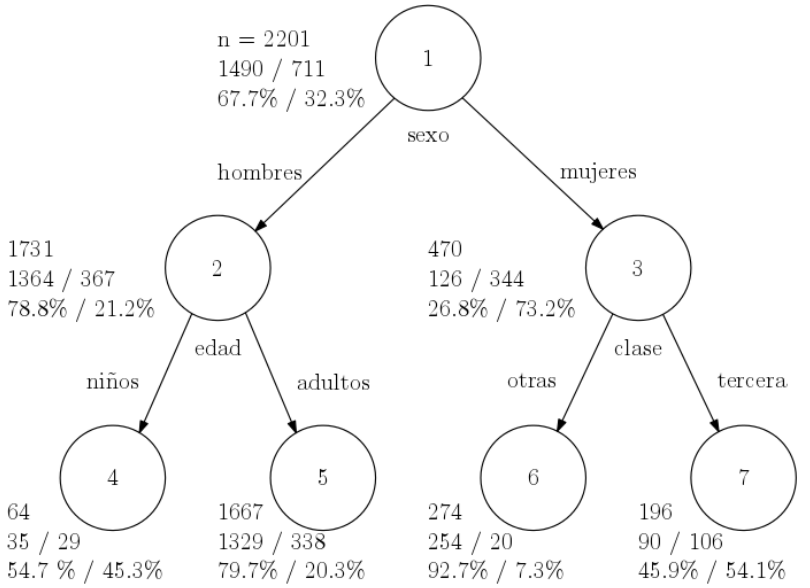


Figura 13.1. Supervivencia de los embarcados en el Titanic

La figura 13.2 muestra un ejemplo de árbol de decisión y una representación de la partición del espacio que genera, in-

tentando separar elementos de dos clases diferentes en un espacio de datos de entrada de dos dimensiones. En este caso, la profundidad máxima del árbol es dos. La primera condición (C1) crea dos regiones, una de las cuales (R1) solamente contiene elementos de una clase, por lo que no es necesario continuar dividiéndola. La otra región es dividida de acuerdo a una segunda condición (C2) creando dos nuevas regiones (R2 y R3). La región R2 es también pura y no es necesario dividirla de nuevo, mientras que la región R3 aún contiene elementos de las dos clases, por lo que el algoritmo debería ejecutarse hasta conseguir un árbol completo.

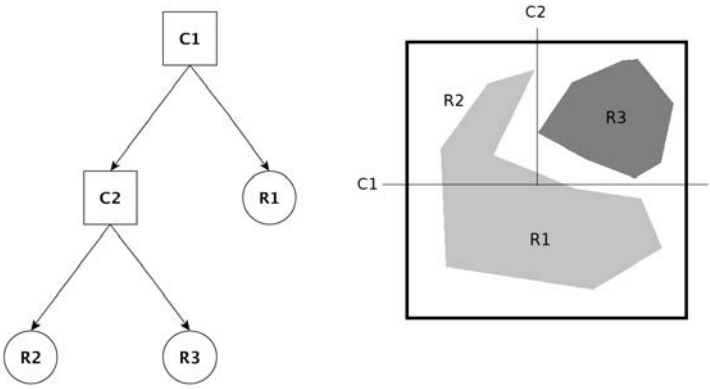


Figura 13.2. Ejemplo de árbol de decisión y la partición del espacio que genera

El algoritmo genérico para construir un árbol de decisión utiliza los siguientes criterios:

- C_p o criterio de parada, determina en qué momento se deja de seguir seleccionando nodos para ser subdivididos. El más habitual es generar un árbol completo, lo cual quiere decir que no quedan nodos que particionar.

- C_s o criterio de selección, determina qué nodo es seleccionado para ser particionado en dos o más subnodos. Existen diferentes criterios de selección, pero si se generan árboles de decisión completos el criterio utilizado no es relevante.

- C_c o criterio de clasificación, determina qué clase se asigna a un nodo hoja. Normalmente se trata de la clase que minimiza el error de clasificación o el valor numérico que minimiza el error de regresión.

- C_d o criterio de partición, determina cómo se particiona un nodo en dos o más subnodos. Normalmente los árboles de decisión son binarios (es decir, $P = 2$), indicando que cada región se subdivide en dos regiones disjuntas. Esto simplifica el criterio de partición y mejora la interpretabilidad del árbol a costa de generar árboles más profundos.

El algoritmo genérico para construir un árbol de decisión o, lo que es equivalente, una partición del espacio de entrada, se muestra en el algoritmo 9.

Los dos criterios más importantes son los de clasificación (C_c) y el de partición (C_d), dado que determinan el árbol construido a partir del conjunto de datos de entrada, especialmente cuando se construyen árboles completos.

13.1. Detalles del método

Como ya se ha comentado, el árbol de decisión construido depende de los cuatro criterios anteriormente mencionados: parada, selección, clasificación y partición. Los cuatro crite-

rios están relacionados entre sí y pueden ser establecidos independientemente.

Algoritmo 9 Pseudocódigo para construir un árbol de decisión

Entrada: conjunto de datos a clasificar D
  $T = \{D\}$ // El árbol inicial es un solo nodo hoja
  Etiquetar T de acuerdo a C_c
  $p = \{T\}$ // Lista de nodos pendientes (hojas)
  mientras no se deba parar según C_p **hacer**
    Seleccionar un nodo q de acuerdo a C_s
    si es posible particionar q según C_d **entonces**
      Particionar q en $q_1, \ldots, q_P$
      Etiquetar $q_1, \ldots, q_P$ según C_c
      Añadir $q_1, \ldots, q_P$ a p
      Sustituir q en T por un nodo interno
    fin si
    Eliminar q de p
  fin mientras
  devolver T

13.1.1. El criterio de parada

El criterio de parada determina cuándo se dejan de subdividir nodos para decidir que ya se ha construido un árbol suficientemente adecuado para los datos de entrada a clasificar. Este criterio suele usar aspectos relativos a la estructura del árbol y a su rendimiento como clasificador. Por ejemplo, puede decidirse parar cuando el árbol alcanza una cierta profundidad máxima o bien cuando la profundidad media (ponderada para todas las hojas) alcanza cierto valor. Otra posibilidad

es medir el error cometido para los datos de entrada y parar cuando dicho error está por debajo de un cierto nivel. Obviamente, si no hay ninguna hoja que pueda ser particionada, el algoritmo también se detiene. En este sentido, si el objetivo es construir un árbol completo, el criterio de parada es simplemente no parar mientras exista alguna hoja que pueda ser particionada.

13.1.2. El criterio de selección

El criterio de selección determina qué nodo hoja es escogido para ser particionado. Obviamente, una hoja que solamente contenga elementos de una misma clase no debe ser particionada, ya que no mejora la capacidad predictiva del árbol, así que este criterio combina aspectos relativos a la impureza de las hojas, combinando el error cometido en dicha hoja y otras características como el número de elementos de conjunto de entrada que contiene o su profundidad en el árbol. Si el objetivo es construir un árbol completo, todas las hojas impuras deberán ser particionadas, sea en el orden que sea, por lo que el criterio puede ser tan sencillo como seleccionar el siguiente nodo en la lista de nodos pendientes. No obstante, el criterio habitual es seleccionar el nodo más impuro, es decir, aquel que contiene una mayor mezcla de elementos de diferentes clases. Habitualmente esto puede medirse mediante la entropía, que mide el grado de desorden de una distribución de elementos de k clases diferentes:

$$H = \sum_{i=1}^{k} p_i \log p_i \tag{13.1}$$

La entropía es cero si todos los elementos son de una misma clase c, ya que entonces $p_c = 1$ y el resto de $p_i = 0$ ($\forall i \neq c$) (se asume que $0 \log 0 = 0$). Cada p_i se calcula como el número de elementos de la clase i presentes en la hoja t dividido por el total de elementos en dicha hoja n_t, de la manera siguiente:

$$p_i(t) = n_i(t)/n_t \tag{13.2}$$

13.1.3. El criterio de clasificación

El criterio de clasificación determina qué clase se asigna a una región u hoja. Este criterio determina el error cometido en aquella hoja y también el error global que comete el árbol de decisión. Obviamente, si una hoja solamente contiene elementos de una clase, dicha clase es la elegida como representante de la región, minimizando el error cometido en la hoja, el cual es cero. En el caso de que en una hoja existan elementos de diferentes clases, se escogerá aquella clase que minimiza el error cometido, normalmente la más poblada. En el improbable caso de empate entre diferentes clases, puede tomarse una al azar. En general, la clase escogida es aquella que satisface la siguiente ecuación:

$$c(t) = \arg\min_j \sum_{i=1}^{k} p_i(t)\, C_{i,j} \tag{13.3}$$

Es decir, se trata de escoger aquella clase que minimiza el error cometido, teniendo en cuenta el posible diferente coste $C_{i,j}$ de cometer un error al escoger una clase i en lugar de la otra j, es decir, con costes asimétricos ($C_{i,j} \neq C_{j,i}$). Obviamente, $C_{i,i} = 0\ \forall i$. Por ejemplo, en un árbol de decisión que intenta determinar mediante la resonancia magnética de una muestra de tejido si esta se trata de un tumor o no, es mejor

cometer un error y obtener falsos positivos (es decir, indicar que hay un tumor cuando no es así) que no falsos negativos (es decir, indicar que no se trata de un tumor cuando realmente sí lo es).

El error total cometido por un árbol T es la suma ponderada del error cometido en cada hoja, teniendo en cuenta el tamaño de la región definido por cada hoja, estimado a través del número de elementos que la componen:

$$e(T) = \frac{1}{n} \sum_{t \in T} n_t \sum_{i=1}^{k} p_i(t)\, C_{i,c(t)} \tag{13.4}$$

que es equivalente a:

$$e(T) = \frac{1}{n} \sum_{t \in T} \sum_{i=1}^{k} n_i(t)\, C_{i,c(t)} \tag{13.5}$$

13.1.4. El criterio de partición

El criterio de partición determina cómo se divide una hoja en dos o más regiones. De hecho, el número de regiones y del criterio utilizado determinan qué tipo de árbol de decisión se está construyendo, dado que existen diferentes algoritmos que intentan aprovechar la naturaleza de los datos de entrada. Lo habitual es trabajar con árboles binarios, donde la región a dividir se particiona en dos regiones disjuntas. Se puede pensar como si un hiperplano cortara el espacio de entrada en dos. En el ejemplo de la figura 13.2, cada una de las condiciones C1 y C2 representa este hiperplano.

El criterio de partición combina, de hecho, dos criterios diferentes: uno determina cómo se construye el hiperplano y el

otro determina en qué posición del espacio se sitúa el hiperplano para realizar la partición.

En el primer caso, lo más habitual es utilizar hiperplanos ortogonales a cada uno de los ejes (variables) del espacio de entrada. Es decir, cada hiperplano es, en realidad, una condición que particiona los elementos de una hoja en dos subconjuntos en función de un valor dado. Es decir, en cada nodo interno solo se utiliza una variable para decidir la clasificación de un elemento. Otra opción es buscar un hiperplano oblicuo el cual seguramente mejorará los resultados obtenidos con el método anterior, aunque la obtención de dicho hiperplano puede ser muy costosa, complicará su interpretación posterior y, además, puede provocar problemas de mala generalización para datos nunca vistos. En el caso mostrado en la figura 13.3, un único hiperlano oblicuo puede separar perfectamente las dos clases presentes en el conjunto de datos, mientras que si se usaran hiperplanos ortogonales serían necesarios al menos cuatro, intentando reproducir el contorno de cada una de las regiones. El lector interesado en la construcción de hiperplanos oblicuos puede consultar [19].

En el segundo caso, determinar la posición del hiperplano o corte para todos los hiperplanos posibles, lo que se hace es medir cómo mejoraría el árbol si se aplicara la partición definida por cada hiperplano, maximizando o minimizando algún criterio. Una opción que parece *a priori* interesante es utilizar el hiperplano que maximiza la reducción del error cometido por el árbol de decisión, tal y como vemos en la siguiente ecuación. Sea T el árbol antes de realizar la partición de una hoja t y sea T_h el árbol resultante de particionar t en $t_{h\leq}$ y $t_{h>}$ mediante el hiperplano h. Se escogería, por lo tanto, el hiperplano tal que:

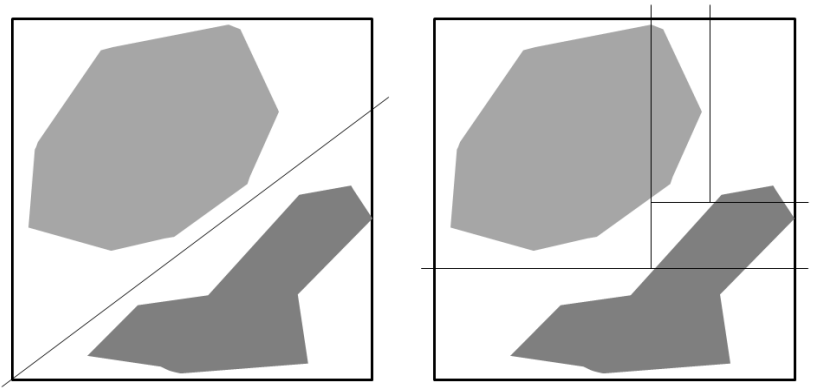


Figura 13.3. Ejemplo de hiperplano oblicuo

$$h = \arg\max_{h} \left(e(T) - e(T_h)\right) \tag{13.6}$$

No obstante, el uso del error de clasificación para los criterios de selección y partición está desaconsejado, siendo mejor usar otros criterios relacionados con la impureza del árbol, como la anteriormente mencionada entropía. El criterio más habitual es el llamado índice de Gini o I_G, definido a partir de la probabilidad $p_i(t)$ de cada clase i en una hoja t como:

$$I_G(t) = \sum_{i=1}^{k} p_i(t)(1 - p_i(t)) \tag{13.7}$$

Entonces, para todos los hiperplanos posibles h que particionan la hoja t, se escogería aquel que maximiza la reducción en impureza:

$$h = \arg\max_{h} \left(I_G(t) - I_G(t_h)\right) \tag{13.8}$$

Existen otros criterios para decidir cómo particionar una hoja. El lector interesado puede referirse al trabajo de Rokach and Maimon [24].

Selección del hiperplano

La selección del hiperplano o corte se realiza por fuerza bruta para cada una de las variables, atendiendo a su tipo. Si se trata de una variable numérica u ordinal, lo que se hace es ordenar el conjunto de datos de acuerdo a dicha variable a_j y recorrer todos los posibles hiperplanos h, seleccionando aquel corte $a_j \leq h$, es decir, variando los valores de h entre el mínimo y el máximo encontrados al ordenar todos los valores posibles. Este proceso es costoso para conjuntos de entrenamiento grandes pero, conforme se va construyendo el árbol, el número de hiperplanos a comprobar se va reduciendo considerablemente.

En el caso de una variable categórica, donde no existe un orden implícito entre los posibles valores que puede tomar, es necesario generar todos los subconjuntos posibles, de forma que el hiperplano es en realidad una comparación de si un valor pertenece a un subconjunto o no. El número de subconjuntos posibles (sin incluir el subconjunto vacío o el subconjunto trivial conteniendo todos los elementos) para una variable categórica que toma c valores diferentes es $2^c - 2$, y el número de combinaciones que es necesario probar es de $2^{c-1} - 1$ (la mitad), número que crece exponencialmente con c.

13.2. Poda del árbol

Uno de los principales problemas de los árboles de decisión es que están orientados a obtener una partición del conjun-

to de entrenamiento «completa», es decir, intentan clasificar correctamente todos los elementos del conjunto de entrenamiento, aun cuando ello implique hacer crecer el árbol hasta profundidades muy elevadas, creando árboles enormes con cientos o miles (o incluso millones) de nodos. El problema es que dicho árbol es muy específico para el conjunto de entrenamiento, y seguramente cometerá muchos errores para datos nuevos, causando sobreentrenamiento. Es como si se diseñara un mueble para almacenar una colección muy específica de objetos; si se presenta un objeto nuevo nunca visto anteriormente, posiblemente no encajará bien en ninguna parte, ya que el mueble es demasiado específico para la colección para la cual fue construido.

Para resolver este problema existe una segunda fase en el proceso de creación de un árbol de decisión llamada poda, dado que consiste en eliminar aquellas hojas que son las que causan el sobreentrenamiento. Para ello se calcula cuál es la partición (de la cual cuelgan las hojas a eliminar o, en general, un subárbol) que aporta menor ratio entre el incremento de profundidad media del árbol y el decremento del error global de clasificación. Es decir, se eliminan aquellas hojas (o subárboles) que menos ayudan a mejorar el árbol durante el proceso de creación.

Desafortunadamente, esto no puede ser realizado durante el propio proceso de creación, ya que exigiría comprobar todos los subárboles existentes, número que crece exponencialmente a cada paso del algoritmo de crecimiento. Sin embargo, una vez se ha alcanzado el árbol puro que clasifica perfectamente el conjunto de entrenamiento, sí que es posible encontrar de forma eficiente la secuencia de subárboles óptimos, que son aquellos que proporcionan el menor error de clasificación en

el conjunto de entrenamiento para cada una de las posibles profundidades medias.

Sea n_t el número de elementos de una hoja t y n el número total de elementos del conjunto de datos de entrenamiento. Sea $l(t)$ la profundidad de una hoja t. Entonces, definimos la profundidad media $l(T)$ de un árbol T como:

$$l(T) = \frac{1}{n} \sum_{t \in T} n_t l(t) \tag{13.9}$$

La figura 13.4 muestra la idea detrás del algoritmo de poda. Cada nodo o circunferencia representa un posible árbol, siendo T_0 el árbol inicial constituido por una sola hoja que contiene todos los elementos del conjunto de entrenamiento.

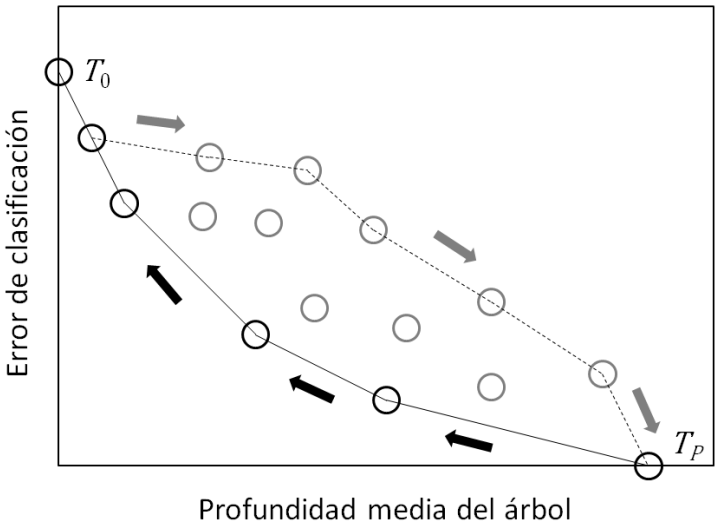


Figura 13.4. Procesos de creación y poda de un árbol

Durante el proceso de creación del árbol, indicado por las flechas grises, se toman decisiones que generan una secuencia de árboles, unidos por la línea de puntos, los cuales aumentan la profundidad media y van reduciendo el error de clasificación. Si otras hojas hubieran sido seleccionadas para ser particionadas durante dicho proceso, se hubiera generado otra secuencia de árboles posibles, representados por los nodos en gris. Cuando se alcanza el árbol completo T_P, el algoritmo de poda selecciona, indicado por las flechas negras, aquellos subárboles que son óptimos, es decir, aquellos que minimizan el error de clasificación para una profundidad media dada.

Es entonces cuando entra en juego el conjunto de datos de test no usado para crear el árbol de decisión. Para cada uno

de los subárboles óptimos encontrados por el proceso de poda, se mide el error de clasificación cometido en el conjunto de datos de test. Empezando por T_0, lo más habitual es que, conforme crece $l(T)$ para cada uno de los subárboles óptimos encontrados por el algoritmo de poda, el error cometido en el conjunto de test (indicado por la línea de puntos) también vaya descendiendo, como sucede con el conjunto de entrenamiento. Sin embargo, es posible que, a partir de cierto punto, el error en el conjunto de test se estabilice o incluso repunte, indicando que árboles más grandes son demasiado específicos y sufren de sobreentrenamiento. Así, tal como muestra la figura 13.5, el resultado del proceso de creación de un árbol de decisión sería aquel subárbol T_T que minimiza el error cometido en el conjunto de test.

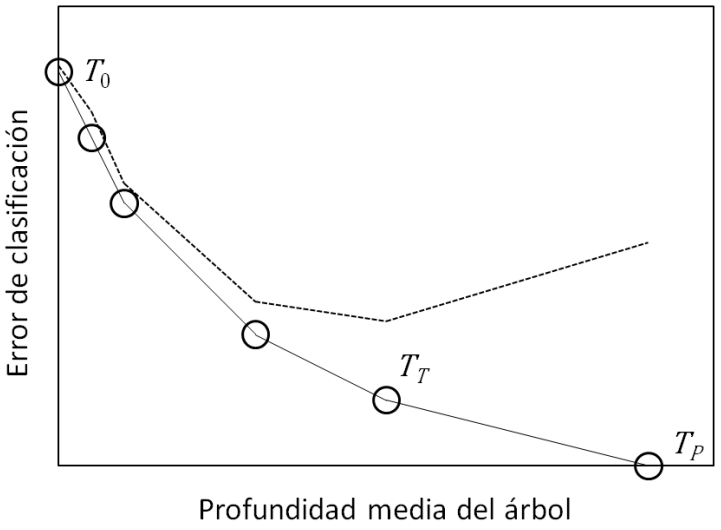


Figura 13.5. Selección del subárbol óptimo

Se recuerda al lector que en el material adicional que acompaña este libro puede encontrar ejemplos de creación y poda de árboles de decisión usando Jupyter y R.

13.3. Resumen

Existen muchas razones por las cuales los árboles de decisión son uno de los modelos más utilizados en la práctica:

- Su construcción es sencilla, aunque puede ser costosa computacionalmente para conjuntos de entrenamiento muy grandes.
- El resultado obtenido es directamente interpretable, siendo también posible evaluar la importancia de cada variable utilizada en la construcción del modelo.
- Pueden combinar variables numéricas y categóricas en el mismo modelo, siendo invariantes a traslaciones y escalados de los datos, por lo que no es necesario normalizar los datos (aun cuando esto sea recomendable en general).
- Pueden trabajar con valores perdidos, utilizando condiciones alternativas (conocidas como *surrogate splits*).
- Su implementación práctica se reduce a una serie de reglas que pueden ser fácilmente escritas como un conjunto the sentencias *if-then-else*.

Por otra parte, los árboles de decisión presentan una serie de problemas que, aun siendo conocidos, no son sencillos de resolver sin una inspección del árbol construido (por otra parte, siempre necesaria):

- Fragmentación: a veces una hoja es particionada en dos de forma muy desequilibrada, generando dos nuevas hojas, una con pocos elementos y la otra con la mayoría. Este proceso puede generar ramas muy largas, incrementando la profundidad media del árbol y fragmentando el espacio de datos de entrada en regiones muy pequeñas, seguramente no representativas. La única solución consiste en imponer unos requisitos mínimos a la hora de particionar una hoja, de forma que las dos nuevas hojas estén equilibradas.

- Repetición: este fenómeno aparece cuando la secuencia de condiciones tomada en cada nodo interno hasta llegar a una hoja repite la misma variable, cambiando el valor por el cual se toma la decisión. Esto indica que dicha variable tiene una relación claramente no lineal con respecto a la variable objetivo, por lo que se puede intentar realizar una transformación no lineal de dicha variable a partir de un análisis de su distribución.

- Replicación: en muchos casos, los datos de entrada pueden presentar una estructura interna que es imposible de capturar si se usan hiperplanos o condiciones ortogonales, es decir, que solo utilizan una variable en cada decisión o nodo interno. Lo que suele ocurrir es que los subárboles generados reproducen la estructura de variables, indicando dicho problema. La única solución posible es introducir el uso de hiperplanos oblicuos o bien realizar algún procedimiento de extracción de características en los datos originales de forma que se reduzca la colinealidad entre variables y se elimine dicha estructura interna, capturándola en una nueva variable.

Los árboles de decisión son muy sensibles al conjunto de datos de entrenamiento, por lo que la presencia de *outliers* puede generar ramas profundas que no generalicen bien para nuevos datos. Aunque el proceso de poda seguramente eliminará dichas ramas, es mejor construir el árbol de decisión una vez se han detectado y eliminado los *outliers* del conjunto de entrenamiento.

Capítulo 14

Métodos probabilísticos

En este capítulo veremos los métodos estadísticos, que se basan en las probabilidades condicionadas de cada clase o variable objetivo dadas las propiedades de un ejemplo, expresado en forma de atributos. El objetivo de este tipo de métodos es maximizar las probabilidades de clasificación de nuevos ejemplos a partir de los ejemplos del conjunto de entrenamiento.

En concreto, discutiremos el método Naïve Bayes, también conocido como el clasificador bayesiano ingenuo, en la sección 14.1, y el clasificador basado en la máxima entropía, en la sección 14.2.

14.1. Naïve Bayes

El algoritmo de clasificación de Naïve Bayes [7] se basa en el concepto de probabilidad condicional y busca maximizar la verosimilitud del modelo, es decir, otorgar mayor importancia a aquellos eventos que son realmente relevantes en el juego de datos, como bien apunta el autor Jeffrey Strickland [27].

En general, los métodos estadísticos suelen estimar un conjunto de parámetros probabilísticos, que expresan la probabilidad condicionada de cada clase dadas las propiedades de un ejemplo (descrito en forma de atributos). A partir de aquí, estos parámetros pueden ser combinados para asignar las clases que maximizan sus probabilidades a nuevos ejemplos.

14.1.1. Probabilidad condicional

Un aspecto central en el algoritmo de Naïve Bayes es el concepto de probabilidad condicionada. Para entenderlo bien, fijémonos en el cuadro 14.1.

Clase	Color	Textura	Calibre	Hidratación
Primera	Intenso	Suave	Grande	Alta
Primera	Intenso	Suave	Grande	Alta
Primera	Intenso	Rugosa	Grande	Alta
Primera	Intenso	Suave	Grande	Alta
Primera	Intenso	Suave	Medio	Alta
Primera	Pálido	Suave	Medio	Baja
Segunda	Pálido	Rugosa	Medio	Baja
Segunda	Pálido	Rugosa	Grande	Alta
Segunda	Irregular	Rugosa	Grande	Alta
Segunda	Irregular	Suave	Pequeño	Alta
Tercera	Intenso	Rugosa	Pequeño	Baja
Tercera	Irregular	Rugosa	Pequeño	Baja

Cuadro 14.1. Ejemplo de instancias del conjunto de cerezas

Se trata de medidas tomadas sobre un conjunto de doce cerezas con el objetivo de clasificarlas según su categoría o clase a partir de atributos como: color, textura, calibre e hidratación.

Es fácil observar que la probabilidad de clase primera es del 50 %. Para expresarlo en términos probabilísticos lo pensaremos del siguiente modo: $P(clase = primera) = \frac{6}{12}$.

Del mismo modo podemos ver cómo la probabilidad de que tenga un color intenso también es del 50 %, esto es $P(color = intenso) = \frac{6}{12}$.

La probabilidad condicional $P(A|B)$ es la probabilidad de que ocurra un evento A, sabiendo que también sucede otro evento B, y se expresa con la siguiente ecuación:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{14.1}$$

Continuando con el ejemplo presentado, expresaremos una probabilidad condicional del siguiente modo: de entre las cerezas que se han clasificado como clase primera, ¿cuántas tienen una textura suave?

Intuitivamente, observamos que de entre las seis clasificadas como clase primera, solo cinco tienen una textura suave, de modo que diremos que $P(suave|primera) = \frac{5}{6}$.

Aplicando la fórmula de la ecuación 14.1, obtenemos la siguiente expresión:

$$P(suave|primera) = \frac{P(suave \cap primera)}{P(primera)} = \frac{\frac{5}{12}}{\frac{6}{12}} = \frac{5}{6}$$

Se puede observar que ambos resultados son coherentes.

14.1.2. Algoritmo Naïve Bayes

El algoritmo Naïve Bayes clasifica nuevos ejemplos $d = (d_1, \ldots, d_m)$ asignándole la clase c que maximiza la proba-

bilidad condicional de la clase, dada la secuencia observada de atributos del ejemplo. Es decir:

$$\begin{aligned} &\arg\max_c \ P(c|d_1,\ldots,d_m) \\ &= \arg\max_c \ \frac{P(d_1,\ldots,d_m|c)P(c)}{P(d_1,\ldots,d_m)} \\ &\approx \arg\max_c \ P(c)\prod_{i=1}^{m} P(d_i|c) \end{aligned} \tag{14.2}$$

donde $P(c)$ y $P(d_i|c)$ se estiman a partir del conjunto de entrenamiento, utilizando las frecuencias relativas (estimación de la máxima verosimilitud, en inglés *maximum likelihood estimation*).

En el caso de tener un conjunto de datos que incluya atributos numéricos continuos es necesario algún tipo de preproceso antes de poder aplicar el método descrito. Existen dos alternativas principales para lidiar con los atributos continuos:

- La forma más simple consiste en categorizar los atributos continuos, de forma que sean convertidos en intervalos discretos y se puedan tratar de igual forma que los atributos nominales.

- Otra opción consiste en asumir que los valores de cada clase siguen una distribución gaussiana, y aplicar la siguiente fórmula:

$$P(d=v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} exp\Big(-\frac{(v-\mu_c)^2}{2\sigma_c^2}\Big) \tag{14.3}$$

  donde d corresponde al atributo, v a su valor, c a la clase, μ_c al promedio de valores de la clase c, y σ_c^2 a su desviación estándar.

Es importante destacar un problema relacionado con este algoritmo de clasificación, que se produce cuando encontramos en el conjunto de test una pareja <atributo, valor>que no ha aparecido en el conjunto de entrenamiento. En este caso, no dispondremos del valor $P(d_i|c)$.

Para solucionar este problema se suele aplicar alguna técnica de suavizado [20] (*smoothing*), como por ejemplo asignar como probabilidad condicional el valor $\frac{P(c)}{n}$, donde n indica el número de instancias de entrenamiento.

14.1.3. Ejemplo de Naïve Bayes

En el cuadro 14.2 podemos ver nuestro ejemplo organizado por $\langle atributo, valor \rangle$ para que sea más fácil apreciar las probabilidades condicionales.

Atributo - Valor	Primera	Segunda	Tercera
Color - Intenso	5/6	0/4	1/2
Color - Pálido	1/6	2/4	0/2
Color - Irregular	0/6	2/4	1/2
Textura - Suave	5/6	1/4	0/2
Textura - Rugosa	1/6	3/4	2/2
Calibre - Grande	4/6	2/4	0/2
Calibre - Medio	2/6	1/4	0/2
Calibre - Pequeño	0/6	1/4	2/2
Hidratación - Alta	5/6	3/4	0/2
Hidratación - Baja	1/6	1/4	2/2

Cuadro 14.2. Ejemplo de Naïve Bayes

Simularemos el razonamiento del algoritmo con el objetivo de clasificar una entrada nueva en el juego de datos, es decir, queremos clasificar una nueva cereza conociendo sus atributos de color, textura, calibre e hidratación.

Empezaremos por calcular las probabilidades del atributo objetivo o clase:

- $P(c = primera) = \frac{6}{12}$
- $P(c = segunda) = \frac{4}{12}$
- $P(c = tercera) = \frac{2}{12}$

El siguiente paso será calcular las probabilidades condicionadas $P(d_i|c)$, veamos algunos ejemplos que se desprenden del cuadro 14.2:

- $P(intenso|primera) = \frac{5}{6}$
- $P(palido|primera) = \frac{1}{6}$
- $P(irregular|primera) = \frac{0}{6}$
- $P(suave|segunda) = \frac{1}{4}$
- $P(rugosa|segunda) = \frac{3}{4}$
- $P(alta|tercera) = \frac{0}{2}$
- $P(baja|tercera) = \frac{2}{2}$

Con esto habremos terminado el proceso de entrenamiento. A partir de aquí trataremos de clasificar una entrada nueva, presentada en el cuadro 14.3.

	Clase	Color	Textura	Calibre	Hidratación	
		$P(d_i\|c)$	$P(d_i\|c)$	$P(d_i\|c)$	$P(d_i\|c)$	
$P(c)$	?	Pálido	Suave	Grande	Alta	
6/12	Primera	1/6	5/6	4/6	5/6	$= 0,0386$
4/12	Segunda	2/4	1/4	2/4	3/4	$= 0,0156$
2/12	Tercera	0/2	0/2	0/2	0/2	$= 0,0000$

Cuadro 14.3. Clasificación con Naïve Bayes

Queremos saber si una cereza con atributos de color pálido, textura suave, calibre grande e hidratación alta deberíamos clasificarla como clase primera, segunda o tercera. Para ello aplicaremos el criterio de Naïve Bayes:

$$\arg\max_{c} P(c) \prod_{i=1}^{m} P(d_i|c) \tag{14.4}$$

y lo haremos del siguiente modo:

- Para la clase «primera», la probabilidad es: $\frac{6}{12}(\frac{1}{6}\frac{5}{6}\frac{4}{6}\frac{5}{6}) = 0,0386$
- Para la clase «segunda», la probabilidad es: $\frac{4}{12}(\frac{2}{4}\frac{1}{4}\frac{2}{4}\frac{3}{4}) = 0,0156$
- Para la clase «tercera», la probabilidad es: $\frac{2}{12}(\frac{0}{2}\frac{0}{2}\frac{0}{2}\frac{0}{2}) = 0$

Por lo tanto, clasificaremos la nueva cereza con clase «primera», dado que presenta la mayor verosimilitud de acuerdo con el método Naïve Bayes.

14.2. Máxima entropía

Para entender cómo funciona este algoritmo de clasificación, empezaremos viendo el concepto de distribución uniforme, para luego pasar a presentar el principio basado en la máxima entropía.

14.2.1. Distribución uniforme

Imaginemos tres recipientes que contienen cada uno de ellos diez bolas y diez cruces, como puede apreciarse en la figura 14.1.

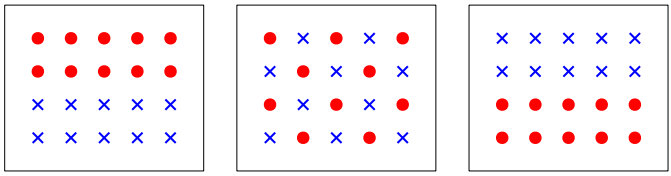


Figura 14.1. Entropía máxima o distribución uniforme

Observamos que tanto el recipiente (a) como el (c) tienen perfectamente ordenadas las bolas y los cruces. Diferente criterio para ordenar en ambos casos, pero en cualquier caso ordenados. Este orden provoca que no dispongamos de una distribución uniforme de bolas y de cruces ya que en el primer recipiente las bolas se concentran en la parte superior, mientras que en el tercer recipiente las bolas se concentran en la parte inferior.

El recipiente (b), sin embargo, tiene una distribución uniforme de bolas y cruces, es decir, tenemos una distribución desordenada o aleatoria. En esta circunstancia diremos que tenemos máxima entropía.

Alcanzaremos una situación de máxima entropía cuando tengamos una distribución uniforme o una situación de aleatoriedad.

14.2.2. Descripción del problema

Supongamos que queremos realizar un estudio sobre las posibles traducciones de la palabra inglesa *in* al español, con el objetivo de construir un traductor automático.

Nos encontraríamos con que *in* puede tomar valores $Y \in \{\text{en}, \text{dentro de}, \text{dentro}, \text{adentro}, \text{sobre}, \text{de cada}\}$. Bajo un punto de vista de distribución de probabilidades, podríamos expresarlo del siguiente modo:

$$P(\text{en}) + P(\text{dentro de}) + P(\text{dentro}) + P(\text{adentro}) \\ + P(\text{sobre}) + P(\text{de cada}) = 1$$

La asignación de un valor concreto de la variable y puede estar condicionada por palabras que forman parte de un contexto. Llamaremos X al conjunto de palabras que conforman este espacio contextual que ayudan al traductor a determinar el valor correcto de y.

En realidad hay infinitas distribuciones de probabilidad que satisfacen la ecuación anterior, pero nuestra intuición nos dice que asignar una probabilidad $\frac{1}{6}$ a cada uno de los valores, es decir, $P(\text{en}) = \frac{1}{6}$, $P(\text{dentro de}) = \frac{1}{6}$, ... parece lo más razonable. Esta aproximación sería razonable por ser uniforme y respetar el principio de entropía máxima cuando no tenemos más información. Llamaremos a esta distribución o modelo $P(y|x)$.

Supongamos ahora que en el 90 % de los casos el traductor se decanta por las opciones *en* y *dentro*.

$$P(en) + P(dentro) = 9/10$$

Nuevamente, añadiendo este condicionante, hay infinidad de posibles distribuciones de probabilidad o infinitos modelos que satisfacen las dos ecuaciones anteriores de forma simultánea, pero nuestra intuición nos dice, una vez más, que en ausencia de más información una distribución de probabilidades lo más uniforme posible sería lo más acertado.

- $P(\text{en}) = \frac{9}{20}$
- $P(\text{dentro}) = \frac{9}{20}$
- $P(\text{dentro de}) = \frac{1}{40}$
- $P(\text{adentro}) = \frac{1}{40}$
- $P(\text{sobre}) = \frac{1}{40}$
- $P(\text{de cada}) = \frac{1}{40}$

Es fácil observar que añadiendo conocimiento sobre diferentes traducciones en diferentes contextos (funciones f_i), podemos llegar a construir un sistema de ecuaciones condicionadas realmente extenso y complejo. De modo que surge la necesidad de expresar de un modo formal cómo encontrar la distribución más uniforme posible.

Formalizaremos nuestro problema partiendo de un juego de datos de entrenamiento que representaremos con pares $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ donde la variable x representa palabras de frases que acompañan la palabra *in*. Mientras que la variable y contiene la traducción escogida en español.

Representaremos la distribución empírica basada en nuestro juego de datos de entrenamiento como $\tilde{p}(x, y)$, que indica la cantidad de veces que (x, y) se da en el juego de datos.

Supongamos ahora que observamos que el 60 % de casos en que la palabra *in* se encuentra entre dos números, esta se traduce como *de cada*. Diríamos entonces que $P(\text{de cada}|\text{entre dos números}) = \frac{6}{10}$.

Para dar cabida a estas relaciones condicionadas, utilizaremos una función de activación del siguiente modo:

$$f(x, y) = \begin{cases} 1 & \text{si } y = \text{“de cada” y “in” entre dos números} \\ 0 & \text{en caso contrario} \end{cases} \tag{14.5}$$

El valor esperado de f con respecto a la distribución empírica en nuestro juego de datos de entrenamiento $\tilde{p}(x, y)$ será:

$$\tilde{p}(f) = \sum_{x,y} \tilde{p}(x, y) f(x, y) \tag{14.6}$$

En un abuso de notación consideraremos la función f como la función que recoge no solo uno sino todos los condicionantes que añaden conocimiento al proceso de traducción de la variable y.

El valor esperado de f con respecto al modelo $p(y|x)$ es:

$$p(f) = \sum_{x,y} \tilde{p}(x) p(y|x) f(x, y) \tag{14.7}$$

Lo que buscaremos es que el valor esperado de la función f con respecto al modelo $p(y|x)$ y el valor esperado de la función f en el juego de datos $\tilde{p}(x, y)$ coincidan, es decir:

$$p(f) = \tilde{p}(f) \tag{14.8}$$

Si sustituimos, tendremos que:

$$\sum_{x,y} \tilde{p}(x)p(y|x)f(x,y) = \sum_{x,y} \tilde{p}(x,y)f(x,y) \tag{14.9}$$

Esta restricción nos permite representar de un modo formal el hecho de que estamos interesados en aquellos modelos que coinciden con el juego de datos en la frecuencia en la que se cumple la función o característica f.

Es importante distinguir bien entre funciones o características y restricciones ya que son términos muy utilizados en el ámbito de conocimiento de la máxima entropía.

Una característica es una función binaria $f(x, y)$ que expresa el evento en el juego de datos de que consideraremos y cuando se dé x.

Por otro lado, una restricción es una ecuación entre el valor esperado de una característica en un modelo y su valor esperado en el juego de datos.

14.2.3. Principio de máxima entropía

Supongamos que sobre un juego de datos X disponemos de n características con una relación de propiedades f_i que consideramos importantes para modelar nuestro juego de datos. Queremos, entonces, considerar modelos p que cumplan estas características. Es decir:

$$C = p \in P | p(f_i) = \tilde{p}(f_i) \forall i \in \{1, 2, \ldots, n\} \tag{14.10}$$

Consideremos $p \in P$ como un modelo en el espacio de modelos sin ningún tipo de características o propiedades conocidas.

Lo que buscamos es que nuestro modelo p cumpla estas características, de modo que diremos que queremos que se cumpla $p \in C$, donde $C = C_1 \cap C_2 \cap \ldots \cap C_n$ es el espacio de

todos los posibles modelos que cumplen con las características f_i para $i \in \{1, 2, 3, \ldots, n\}$.

En de la figura 14.2, podemos apreciar cuatro cuadros que representan el espacio de modelos en función de su cumplimiento de las características f_i:

- El cuadro (a) representa el espacio P formado por todos los modelos sin característica alguna.
- El cuadro (b) contiene en la zona blanca el espacio de modelos C_1 que consideran la característica f_1.
- El cuadro (c) contiene en la zona blanca el espacio de modelos que consideran las características f_1, f_3. Observamos que se trata de un espacio imposible ya que no presenta coincidencias.
- El cuadro (d) contiene en la zona blanca un espacio de modelos que consideran las características f_1, f_2.

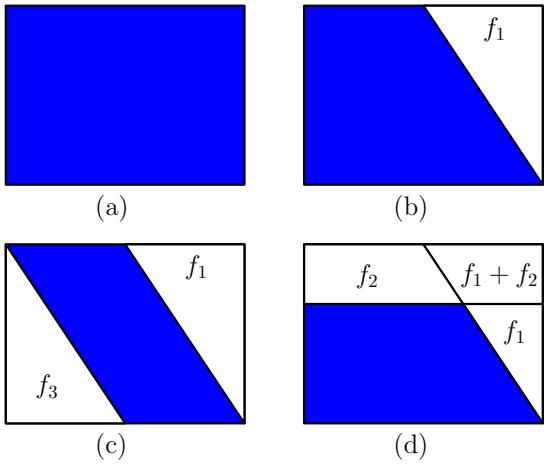


Figura 14.2. Principio de entropía máxima

En este punto, el problema que se nos plantea es que el espacio C formado por todos los modelos que cumplen con las n características posibles, contiene infinitos modelos y queremos saber cuál es el más adecuado.

De acuerdo con nuestro razonamiento al inicio de esta sección, diremos que el mejor modelo será aquel que presente una distribución uniforme, es decir, una entropía máxima.

El principio de máxima entropía nos garantiza que dentro del espacio de modelos que cumplen con las características establecidas, existe un único modelo que posee una distribución uniforme o entropía máxima.

Sin embargo, esta afirmación nos plantea el problema de cómo formalizar y concretar esta uniformidad.

Una medida matemática de la uniformidad de una distribución condicional $P(y|x)$ se obtiene a partir de la entropía condicional:

$$H(p) = -\sum_{x,y} \tilde{p}(x)p(y|x)log(p(y|x)) \tag{14.11}$$

donde $\tilde{p}(x)$ corresponde a la distribución de probabilidad empírica.

De este modo vemos que la entropía tiene como valor mínimo cero, que se correspondería con un modelo sin ninguna incertidumbre. En el otro extremo encontraremos su valor máximo en $log|Y|$ que se corresponde con la entropía de la distribución uniforme de todos los posibles valores de y.

Con los conceptos vistos hasta ahora, podemos presentar el enunciado del principio de máxima entropía: para seleccionar un modelo de entre un grupo C de modelos que cumplen con las propiedades que permitimos de distribución de probabi-

lidades, escogeremos aquel modelo p^* con máxima entropía $H(p)$.

$$p^* = arg\ \max_{p \in C} H(p) \tag{14.12}$$

Es posible demostrar que nuestro modelo p^* es único.

En definitiva, el principio de máxima entropía nos dice que el mejor modelo es aquel que permite máxima incertidumbre en el juego de datos.

La distribución de probabilidades a usar debería asignar una probabilidad distinta de cero y no sesgada a cualquier evento desconocido, de este modo, la información tanto conocida como no conocida será la base para construir el modelo.

14.2.4. Ejemplo de máxima entropía

Hasta ahora hemos visto que el principio de entropía máxima coincide con nuestra intuición en términos de construcción de un modelo que sea lo más representativo posible del juego de datos que trata de modelar.

Veamos mediante un ejemplo concreto cómo intuición y teoría son coherentes.

Supongamos que queremos modelar un proceso de segmentación de clientes totalmente aleatorio que puede darnos tres posibles resultados: cliente tipo 1, cliente tipo 2 y cliente tipo 3.

Tomemos sus respectivas probabilidades como:

- q_1 = probabilidad de cliente tipo 1
- q_2 = probabilidad de cliente tipo 2
- q_3 = probabilidad de cliente tipo 3

donde $q_1 + q_2 + q_3 = 1$.

De este modo, la entropía de nuestro proceso de segmentación será:

$$H(x) = -q_1 log_2(q_1) - q_2 log_2(q_2) - q_3 q_1 log_2(q_3)$$

Nuestra intuición nos dice claramente que si no tenemos más información sobre nuestro proceso de segmentación, es decir, si realmente es totalmente aleatorio, entonces lo más razonable sería asignar una probabilidad de $\frac{1}{3}$ a cada una de las tres probabilidades ya que de este modo conseguiremos tener una entropía máxima.

Podemos verlo en la figura 14.3 donde la función entropía dibuja una superficie que alcanza su máximo con $q_1 = q_2 = q_3 = \frac{1}{3}$

Convirtamos nuestro proceso de segmentación en algo un poco más real. Pongamos que no es del todo aleatorio y que disponemos de información adicional sobre los clientes tipo 2 de modo que sabemos que su probabilidad es de $\frac{1}{2}$.

Nuestra intuición nos dice en este caso que si no disponemos de más información sobre los clientes tipo 1 y tipo 3, deberíamos asignar máxima entropía a estos dos conjuntos. De este modo tendríamos:

- $q_1 = \frac{1}{4}$
- $q_2 = \frac{1}{2}$
- $q_3 = \frac{1}{4}$

Veamos cómo el principio de máxima entropía va a llegar a la misma conclusión.

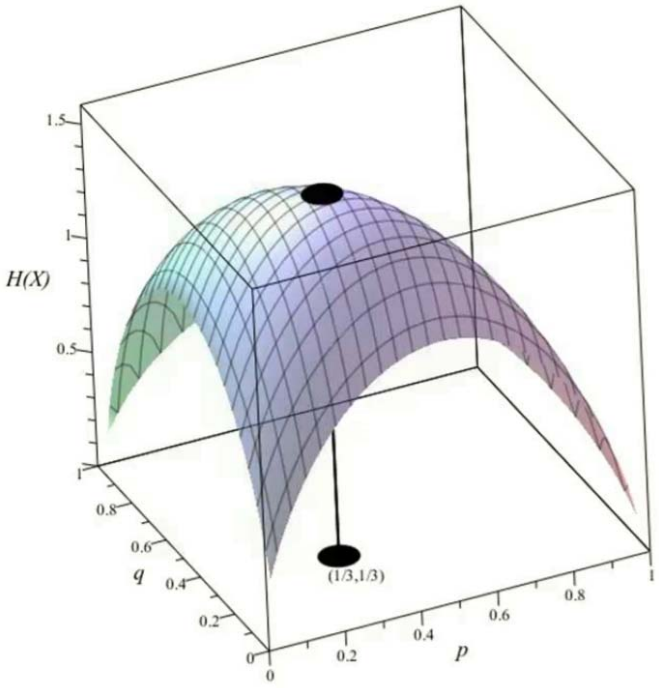


Figura 14.3. Ejemplo de entropía máxima

Sabemos que $q_2 = \frac{1}{2}$, por lo tanto tenemos que la entropía para nuestro proceso de segmentación será ahora:

$$H(x) = \frac{-1}{2} log_2(\frac{1}{2}) - q_1 log_2(q_1) - (\frac{1}{2} - q_1) log_2(\frac{1}{2} - q_1)$$

Para encontrar su máximo exigiremos que su primera derivada sea cero y su segunda derivada sea negativa.

La primera derivada de la función entropía de nuestro proceso de segmentación es:

$$\frac{dH}{dq_1} = -log_2(q_1) - \frac{q_1}{q_1 ln(2)} + \frac{1/2 - q_1}{(1/2 - q_1) ln(2)} + log_2(\frac{1}{2} - q_1)$$

Observamos que los dos componentes centrales se cancelan entre sí, de modo que podemos simplificar nuestra ecuación.

$$\frac{dH}{dq_1} = -log_2(q_1) + log_2(\frac{1}{2} - q_1) \tag{14.13}$$

Es obvio que tendremos derivada cero solo cuando $q_1 = \frac{1}{2} - q_1$, de modo que $q_1 = \frac{1}{4}$.

La segunda derivada de nuestra función entropía es:

$$\frac{d^2H}{dq_1^2} = \frac{-1}{q_1} + \frac{-1}{\frac{1}{2} - q_1} \tag{14.14}$$

Esta será negativa cuando $0 < q_1 < \frac{1}{2}$.

De modo que $H(x)$ toma su máximo en $q_1 = \frac{1}{4}$, que coincide con nuestra intuición.

14.3. Resumen

Los principales puntos fuertes del algoritmo de clasificación de Naïve Bayes son su simplicidad y eficiencia computacional. A pesar de su enorme simplicidad, sigue ofreciendo resultados comparables a algoritmos mucho más elaborados y sofisticados. Es especialmente adecuado para juegos de datos con pocos registros, ya que no requiere de un gran entrenamiento para empezar a ser eficaz. En realidad, Naïve Bayes se aprovecha del hecho de que en muchas ocasiones la clase correcta en un juego de datos es también la clase más probable.

Una de la principales limitaciones o críticas que ha recibido el algoritmo Naïve Bayes es que para su correcto funcionamiento asume la independencia de los diferentes atributos que representan un ejemplo, y esto no siempre es cierto en los conjuntos de datos reales. No obstante, utilizando técni-

cas de extracción de características puede intentarse forzar o favorecer dicha condición.

Otra limitación del clasificador se produce cuando nos encontramos ante conjuntos de datos de entrenamiento muy poco balanceados, es decir, cuando el conjunto de datos de entrenamiento no tiene un número de instancias similar para cada una de las clases o variables objetivo. En este caso, el algoritmo tiende a clasificar las nuevas instancias como si fueran de las clases con mayor número de instancias en el conjunto de entrenamiento.

Efectivamente los detalles importan y en muchas ocasiones marcan el camino hacia la excelencia. Modelar un juego de datos consiguiendo fijar y enmarcar aquello que conocemos bien de él, y para aquello que nos es desconocido mantener todas las opciones abiertas es sencillamente una idea que responde plenamente al sentido común, pero encierra una complejidad considerable llevarla a cabo de un modo tan brillante y eficaz como se ha conseguido plasmar con el método basado en la máxima entropía.

Seguramente porque el lenguaje humano también está lleno de detalles y matices, los modelos que consideran el principio de máxima entropía tienen tan buenos resultados en este campo.

Parte VI

Combinación de clasificadores

Capítulo 15

Combinación de clasificadores

En la actualidad, el estado del arte en minería de datos combina dos aproximaciones diferentes. La primera consiste en el desarrollo de nuevos algoritmos (lo cual ocurre muy ocasionalmente), o bien en el ajuste fino de los parámetros de algún algoritmo conocido, habitualmente para adaptarlo a la naturaleza concreta de unos datos o del problema a resolver. En este sentido, los avances en el área de minería de datos están asegurados por el llamado *no free lunch theorem* [31], el cual postula que no existe un algoritmo *a priori* que sea superior al resto para cualquier conjunto de datos.

La segunda aproximación para crear mejores modelos consiste en combinar clasificadores más o menos sencillos para crear uno mucho más complejo, de forma que la decisión tomada sea una combinación de cientos o miles de decisiones parciales. Obviamente, los clasificadores usados como base para construir el clasificador combinado deben ser lo más diversos posible, con la esperanza de que los errores cometidos por

un clasificador base concreto sean minoritarios con respecto al resto, de forma que los errores puntuales no alteren una decisión basada en la opinión correcta de la mayoría. Una revisión reciente de los trabajos en esta área de investigación puede encontrarse en [32].

Existen dos opciones para la construcción de clasificadores combinados. La primera, que combina clasificadores trabajando en paralelo, es cuando todos los clasificadores base utilizan el mismo modelo o algoritmo (p. ej. árboles de decisión), así que será necesario manipular el conjunto de entrenamiento original para generar diferentes clasificadores base y poder tomar una decisión conjunta a partir de la decisión parcial de cada uno de ellos. La segunda opción consiste en combinar clasificadores base muy diferentes (p. ej. árboles de decisión y redes neuronales) de forma secuencial, de forma que cada clasificador utilice los resultados de un clasificador anterior, intentando capturar alguna característica clave de la naturaleza de los datos o del problema a resolver.

15.1. Combinación paralela de clasificadores base similares

Como se ha comentado, una opción es generar una gran cantidad de clasificadores base construidos a partir de la alteración del conjunto de entrenamiento original. Todos estos clasificadores base son muy parecidos, al estar basados en ligeras variaciones del conjunto de entrenamiento, pero no son idénticos, proporcionando diversidad al clasificador combinado, el cual combina (mediante un esquema de votación) las clasificaciones parciales para tomar una decisión.

Existen dos técnicas básicas para la creación del clasificador combinado, en función de cómo se genera el conjunto de entrenamiento de cada clasificador parcial y del peso asignado a cada uno de ellos en la votación final, llamadas *bagging* y *boosting*.

15.1.1. Bagging

La idea básica del *bagging* es utilizar el conjunto de entrenamiento original para generar centenares o miles de conjuntos similares usando muestreo con reemplazo. Es decir, de un conjunto de N elementos se pueden escoger $N' \leq N$ al azar, existiendo la posibilidad de escoger un mismo elemento más de una vez (de ahí «con reemplazo»). Normalmente $N' = N$, por lo que existirán elementos repetidos.

Aunque no es tan habitual, los conjuntos generados también pueden usar una dimensionalidad $d' \leq d$, de forma que no todas las mismas variables disponibles existan en los conjuntos generados. Esto puede ayudar a reducir el grado de colinealidad entre variables, haciendo emerger variables relevantes que pueden quedar siempre descartadas en frente de otra variable.

Una vez se han construido los conjuntos de entrenamiento parciales, se construye un clasificador para cada uno de ellos, siendo los árboles de decisión la opción más típica. De hecho, teniendo en cuenta la naturaleza del clasificador combinado, no es necesario crear clasificadores base muy precisos, ya que el objetivo es que los errores cometidos por cada clasificador base queden minimizados en frente de la decisión de la mayoría. Por lo tanto, en el caso de árboles de decisión, lo que es habitual es crear clasificadores más pequeños (es decir, limitados en

profundidad) reduciendo el coste computacional de todo el conjunto.

En el caso del *bagging*, la decisión final se toma por mayoría, dando el mismo peso a todas las decisiones parciales. Es decir, la clase resultante del clasificador combinado es aquella que aparece más veces entre las decisiones parciales tomadas por los clasificadores base. La figura 15.1 representa el proceso de creación del clasificador combinado.

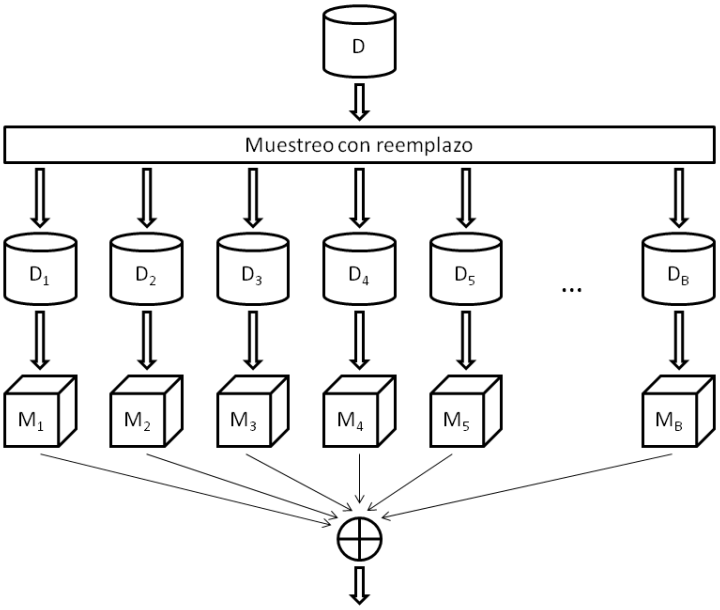


Figura 15.1. Diagrama de un clasificador combinado basado en *bagging*

Una manera de medir el error cometido por el clasificador combinado se conoce como *out-of-bag*, dado que el error estimado es el promedio de todos los errores parciales cometidos por cada clasificador base para todos aquellos elementos no usados en el conjunto de entrenamiento usado para construir-

lo. De esta manera no es necesario separar los datos de entrada en un conjunto de entrenamiento y otro de test, sino que se usan todos para construir el clasificador combinado. No obstante, si se dispone de suficientes datos, es siempre recomendable utilizar un conjunto de test para validar el clasificador combinado.

Random forests

Cuando los clasificadores base son árboles de decisión y se utiliza un muestreo tanto de los elementos del conjunto original de entrenamiento como de sus variables, el clasificador combinado se conoce como *Random forest* [3], dado que se trata precisamente de un conjunto (o *bosque*) de árboles que han sido creados mediante un proceso aleatorio (por lo que respecta al conjunto de entrenamiento usado para cada uno de ellos).

La práctica habitual consiste en generar versiones diferentes del conjunto de entrenamiento usando muestreo con reemplazo, tal y como define el método de *bagging*. Entonces, durante el proceso de construcción de cada árbol de decisión se selecciona aleatoriamente un subconjunto de las variables del conjunto de datos, dando opciones a variables que normalmente quedarían eclipsadas por otras que tuvieran mayor relevancia. Este procedimiento permite medir la importancia relativa de cada variable, estimando el error cometido por el clasificador combinado cuando se altera dicha variable, permutando aleatoriamente sus valores en el conjunto de test. Este error se mide para cada uno de los clasificadores parciales, promediando el error cometido en todos ellos para cada una de las variables. El porcentaje de error se compara al estimado con el conjunto de test sin dicha permutación aleatoria, de forma

que es posible medir el impacto de dicha variable, dado que si el error aumenta cuando se permuta una variable, esto quiere decir que la variable es relevante para el problema que se está resolviendo.

Se recuerda al lector que en el material adicional a este libro se puede encontrar un ejemplo completo donde se muestra el proceso de creación de un *random forest* usando Jupyter y R.

15.1.2. Boosting

La idea del *boosting* es ligeramente diferente. Se parte también de clasificadores base muy sencillos (también llamados débiles) de los cuales se supone que, al menos, cometen menos errores que aciertos, es decir, que funcionan ligeramente mejor que un clasificador aleatorio.

Se empieza construyendo un primer clasificador base usando el conjunto de entrenamiento original. Como el clasificador es débil (por ejemplo, un árbol de decisión de profundidad limitada), se cometen unos cuantos errores para el conjunto de entrenamiento. Entonces, para construir el siguiente clasificador base, lo que se hace es ponderar los elementos del conjunto de entrenamiento, de forma que aquellos que han sido clasificados erróneamente por el clasificador anterior tengan más peso y, de una manera u otra, tengan mayor peso también en el proceso de creación del siguiente clasificador base. El clasificador combinado es la combinación de la predicción del primer clasificador base más la del segundo, ponderadas de acuerdo a algún esquema de pesos que tenga en cuenta la precisión de cada clasificador. Este clasificador combinado también cometerá unos errores que se usarán para dar más peso a aquellos elementos erróneamente clasificados, construyendo un tercer clasificador combinado, etc.

Es decir, en cada etapa se construye un clasificador nuevo que combina una serie de decisiones sucesivas que tienden a enfocarse en aquellos elementos más difíciles de clasificar, intentando que cada etapa corrija los errores de la anterior, pero sin estropear las decisiones correctas ya tomadas. Para ello, la secuencia de pesos de los sucesivos clasificadores parciales suele ser descendiente, de forma que el procedimiento se para cuando el siguiente clasificador ya no aporta nada respecto al anterior. Este esquema, mostrado en la figura 15.2, permite múltiples variaciones, siendo la más conocida la descrita por Freund and Schapire en 1996 [8], llamada AdaBoost.

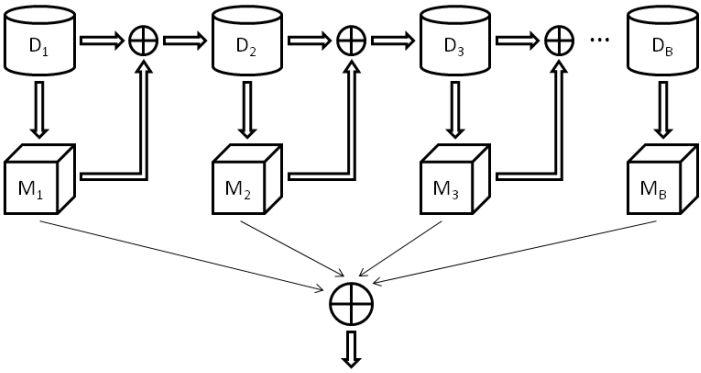


Figura 15.2. Diagrama de un clasificador combinado basado en *boosting*

Es importante destacar que el proceso de creación del clasificador combinado es secuencial, dado que para realizar una nueva iteración y obtener un nuevo clasificador combinado es necesario haber evaluado los clasificadores base anteriores. No obstante, una vez se ha alcanzado el clasificador combinado final, la evaluación se realiza también en paralelo, ya que en una primera etapa se genera la decisión parcial para cada uno

de los clasificadores base y en la segunda se obtiene la clasificación final ponderando todas la decisiones parciales.

Uno de los problemas de *boosting* es la posibilidad de sobreentrenar el clasificador combinado, por lo que es necesario disponer de un conjunto de test e ir evaluando en cada etapa el resultado obtenido sobre el mismo, deteniendo el proceso cuando el error en el conjunto de test empieza a aumentar.

En el material adicional a este libro se puede encontrar un ejemplo completo donde se muestra el proceso de aplicación de *boosting* con árboles de decisión usando Jupyter y R.

15.2. Combinación secuencial de clasificadores base diferentes

En este caso el objetivo es construir un clasificador que combina clasificadores base muy diferentes, con el objetivo de incrementar la diversidad de predicciones e intentar aprovechar todo el conocimiento explícito que se tenga sobre el problema a resolver o los datos disponibles. Por ejemplo, si se utiliza un árbol de decisión como clasificador, se pueden construir unos cuantos clasificadores que funcionen como una etapa previa, de forma que alimenten el árbol de decisión con información extraída de los datos u otras decisiones parciales.

Se trata, entonces, de un proceso secuencial, dado que se generan unas cuantas decisiones parciales que posteriormente son usadas para tomar la decisión final. En función de la información que recibe el clasificador combinado de los clasificadores parciales, se distinguen dos métodos, llamados *stacking* y *cascading*.

Estas técnicas son adecuadas cuando la naturaleza del problema a resolver también presenta una estructura secuencial.

Por ejemplo, en el diagnóstico no invasivo de tumores cerebrales mediante el uso de espectroscopia de resonancia magnética [17], en lugar de intentar predecir el tipo de tumor detectado directamente (lo cual es muy complicado porque existen muchos tipos de tumores diferentes y de diferente grado de malignidad), lo habitual es establecer una secuencia de decisiones parciales intentando atacar el problema desde una primera decisión muy sencilla (p. ej. establecer simplemente si se trata de un tumor o no), seguida de una segunda decisión que determina si se trata de un tumor benigno o maligno, seguida de una tercera que determina el grado de malignidad. Es decir, en cada paso se utiliza la decisión tomada anteriormente, empezando por una primera decisión sencilla que se va refinando en una secuencia de decisiones cada vez más específicas.

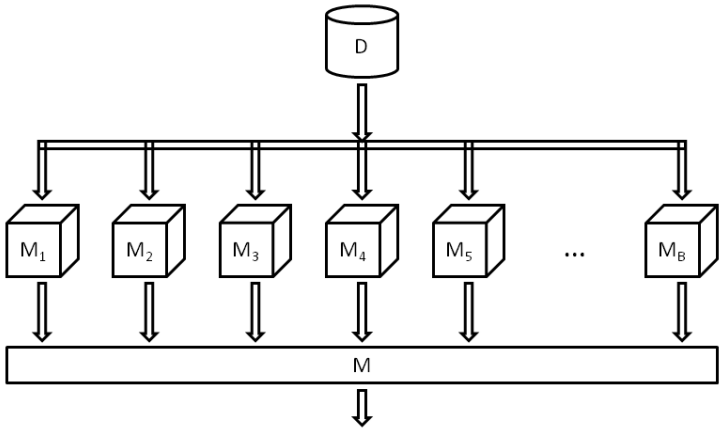


Figura 15.3. Diagrama general de un clasificador combinado basado en *cascading* y *stacking*

En general, se puede pensar en la combinación secuencial como una manera de abordar el problema, intentando resolver primero los casos más sencillos con un clasificador también

sencillo, dejando el resto a una secuencia de clasificadores cada vez más complejos y específicos.

15.2.1. Stacking

La idea básica de *stacking* es construir diferentes clasificadores base de forma que cada uno de ellos genere una decisión parcial para cada elemento de entrada del conjunto de entrenamiento. Entonces se construye un nuevo clasificador usando como datos de entrada todas las predicciones parciales, en lugar de los datos originales de entrada. Este segundo clasificador suele ser un árbol de decisión, una red neuronal sencilla o una regresión logística (en el caso de que el número de clases sea dos).

Aunque no es habitual, esta estructura puede repetirse en diferentes niveles, combinando decisiones de otros clasificadores combinados, de ahí la idea de apilamiento o *stacking*. El problema de siempre es la tendencia a crear un clasificador combinado demasiado específico para el conjunto de entrenamiento que no generalice bien ante nuevos datos.

15.2.2. Cascading

El caso de *cascading* es parecido al de *stacking* pero utilizando no solamente las predicciones parciales de los clasificadores base, sino también los datos originales e incluso otros datos que se hayan podido generar durante la toma de decisiones. La idea básica es alimentar al clasificador combinado con decisiones parciales, así como los motivos que han llevado a tomar dichas decisiones.

Por ejemplo, cuando un árbol de decisión ha asignado una clase a un elemento del conjunto de entrada, dicha clase es el

resultado de una serie de decisiones internas que llevan a una hoja, la cual tiene una profundidad en el árbol, representa una región que contiene un porcentaje de los datos de entrada y se conoce el error cometido al asignar dicha clase como representante de todos los datos de entrada contenidos en la región que representa dicha hoja. O por ejemplo, si uno de los clasificadores base es un algoritmo de *clustering*, se puede usar como información adicional a la clase o clúster asignado a un elemento del conjunto de entrada cada una de las distancias a cada uno de los centroides de los clústeres, como medidas de la fuerza que tiene dicha decisión.

15.3. Resumen

El uso de clasificadores combinados permite, en muchas ocasiones, mejorar la capacidad predictiva de un modelo, siendo posible también la inclusión de conocimiento relativo a la naturaleza del problema a resolver, especialmente con los clasificadores que operan secuencialmente. Su construcción es sencilla, utilizando en muchas ocasiones los mismos algoritmos y técnicas (reemplazo con muestreo, sistemas de votación por mayoría simple, etc.), siendo el caso de *boosting* el más complejo, aunque existen diferentes algoritmos que lo implementan.

No obstante, los clasificadores combinados también presentan una serie de problemas, algunos ya mencionados a lo largo de este libro. El primer punto a tener en cuenta es el coste computacional que puede tener el clasificador combinado, tanto por lo que respecta a su creación (especialmente) como a su ejecución. No obstante, es posible aprovechar la estructura paralela de los clasificadores combinados en caso de disponer de infraestructura tecnológica que permita la paralelización

de procesos, de forma que cada clasificador base se ejecuta en un nodo en paralelo (a la vez) a todos los otros clasificadores base, reduciendo así el tiempo de ejecución necesario.

Un aspecto a tener en cuenta es que no por aumentar el número de clasificadores base involucrados se va a conseguir siempre una mejora de la precisión. Es mucho más importante la diversidad de los clasificadores base que su número. Como la diversidad depende del conjunto de entrenamiento usado para cada clasificador base, es importante que el conjunto de entrenamiento original sea suficientemente grande y diverso y que el proceso de muestreo asegure un buen grado de aleatoriedad.

Por otra parte, la interpretabilidad de los resultados es mucho más complicada, ya que exige la interpretación de cientos o miles de clasificadores base parciales y su posterior combinación. No obstante, es posible medir la importancia relativa de cada variable en el conjunto mediante diferentes técnicas (p. ej. contando cuántas veces aparece cada variable en cada clasificador base con su peso en el clasificador combinado), lo que puede proporcionar cierto conocimiento al respecto.

Parte VII

Apéndices

Apéndice A. Notación

Generalmente, en los métodos supervisados partiremos de un conjunto de datos correctamente etiquetados, D, en el que distinguimos la siguiente estructura:

$$D_{n,m} = \begin{pmatrix} d_1 = & a_{1,1} & a_{1,2} & \cdots & a_{1,m} & c_1 \\ d_2 = & a_{2,1} & a_{2,2} & \cdots & a_{2,m} & c_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ d_n = & a_{n,1} & a_{n,2} & \cdots & a_{n,m} & c_n \end{pmatrix}$$

donde:

- n es número de elementos o instancias,
- m el número de atributos del conjunto de datos o también su dimensionalidad,
- d_i indica cada una de las instancias del juego de datos,
- a_i indica cada uno de los atributos descriptivos,
- c_i indica el atributo objetivo o clase a predecir para cada elemento.

En el caso de los métodos no supervisados, el conjunto de datos sigue el mismo patrón y notación, excepto para el atributo de clase, c_i, que no existe en el caso de los conjuntos no etiquetados.

El cuadro 15.1 muestra un resumen de la notación empleada en los distintos capítulos de este libro.

Símbolo	Descripción
$D_{n,m}$	Conjunto de datos
n	Número de instancias
m	Número de atributos
k	Número de clases
d_i	Instancia i del conjunto de datos D
a_i	Conjunto del atributo i en D
c_i	Clase de la instancia d_i
$p_i(t)$	Proporción de elementos de la clase i en la hoja t

Cuadro 15.1. Resumen de la notación empleada en el libro.

Apéndice B. Ejemplos

En los distintos capítulos de este libro se describen algunos ejemplos sencillos, con el objetivo de que puedan ser interpretados de forma fácil y ayuden en la comprensión de los detalles teóricos de los métodos descritos. Sin embargo, no se incluye el código asociado a estos ejemplos, para evitar que pueda quedar rápidamente desfasado debido a cambios en las funciones o estructuras de las librerías empleadas de minería de datos.

En su lugar, se proporciona una página web que permite la descarga de los ejemplos actualizados, ya sea algunos de los descritos en el libro, como también otros ejemplos más complejos que permiten ver la aplicación real de los algoritmos presentados en este texto.

Los ejemplos se presentan en dos lenguajes principales, que representan estándares *de facto* de la industria y la academia actualmente, como son los lenguajes R y Python.

Los ejemplos están agrupados según el capítulo al que pertenecen, y se encuentran en un formato de *notebook*, concretamente empleando el Jupyter Notebook.

Jupyter Notebook es un entorno interactivo web de ejecución de código que permite mezclar código con texto de explicaciones, incluyendo imágenes y otros recursos que faciliten la comprensión del código. Así, por ejemplo, puedes incluir

gráficas que ayuden en el análisis y explicación de tus datos. Los *notebooks*, en general, son utilizados para facilitar la explicación y reproducción de estudios y análisis.

Para trabajar con ellos se realiza directamente desde el cliente o navegador web. Se puede encontrar más información y los detalles para su instalación en la documentación de la página web `http://jupyter.org/`.

Finalmente, podemos indicar que la página web que contiene los ejemplos se puede encontrar en `http://oer.uoc.edu/libroMD/`.

Bibliografía

[1] Beale, R. y T. Jackson: *Neural Computing: An Introduction*. IOP Publishing Ltd., Bristol, UK, UK, 1990, ISBN 0-85274-262-2.

[2] Berger, Adam: *A brief maxent tutorial*. `http://www.cs.cmu.edu/afs/cs/user/aberger/www/html/tutorial/tutorial.html`, July 1996. (Accessed on 01/03/2017).

[3] Breiman, Leo: *Random forests*. Machine learning, 45(1):5–32, 2001.

[4] Breiman, Leo, Jerome Friedman, Charles J Stone y Richard A Olshen: *Classification and regression trees*. CRC press, 1984.

[5] Cha, Sung Hyuk: *Comprehensive survey on distance/similarity measures between probability density functions*. City, 1(2):1, 2007.

[6] David E. Rumelhart, James L. McClelland: *Parallel Distributed Processing*. MIT press, 1986.

[7] Duda, Richard O. y Peter E. Hart: *Pattern Classification and Scene Analysis*. A Wiley Interscience Publication. Wiley, 1973.

[8] Freund, Yoav y Robert E. Schapire: *Experiments with a new boosting algorithm*. En *Proceedings of the 13th International Conference on Machine Learning*, volumen 96, páginas 148–156, 1996.

[9] Friedman, Jerome, Trevor Hastie y Robert Tibshirani: *The elements of statistical learning*, volumen 1. Springer series in statistics Springer, Berlin, 2001.

[10] Guyon, Isabelle y André Elisseeff: *An Introduction to Variable and Feature Selection*. J. Mach. Learn. Res., 3:1157–1182, Marzo 2003, ISSN 1532-4435. `http://dl.acm.org/citation.cfm?id=944919.944968`.

[11] Haykin, Simon O.: *Neural Networks and Learning Machines, 3rd Edition*. Pearson, 2009.

[12] Hopfield, J. J.: *Neurons with graded response have collective computational properties like those of two-state neurons*. Proceedings of the National Academy of Sciences, 81(10):3088–3092, 1984.

[13] Kohavi, Ron *y cols.*: *A study of cross-validation and bootstrap for accuracy estimation and model selection*. En *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, volumen 14, páginas 1137–1145. Stanford, CA, 1995.

[14] McCallum, Andrew, Kamal Nigam y Lyle H. Ungar: *Efficient Clustering of High-dimensional Data Sets with*

Application to Reference Matching. En *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, páginas 169–178, New York, NY, USA, 2000. ACM, ISBN 1-58113-233-6. http://doi.acm.org/10.1145/347090.347123.

[15] McCulloch, Warren S. y Walter Pitts: *Neurocomputing: Foundations of Research.* capítulo A Logical Calculus of the Ideas Immanent in Nervous Activity, páginas 15–27. MIT Press, 1988, ISBN 0-262-01097-6.

[16] Mezard, M y Jean P Nadal: *Learning in feedforward layered networks: the tiling algorithm.* Journal of Physics A: Mathematical and General, 22(12):2191, 1989.

[17] Minguillón, Julià, Anne Rosemary Tate, Carles Arús y John R. Griffiths: *Classifier Combination for In Vivo Magnetic Resonance Spectra of Brain Tumours*, páginas 282–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[18] Minsky, Marvin y Seymour Papert: *Perceptrons. An Introduction to Computational Geometry.* 1969.

[19] Murthy, Sreerama K., Simon Kasif y Steven Salzberg: *A System for Induction of Oblique Decision Trees.* Journal of Artificial Intelligence Research, 2(1):1–32, Agosto 1994, ISSN 1076-9757. http://dl.acm.org/citation.cfm?id=1622826.1622827.

[20] Ng, Hwee Tou: *Exemplar-Based Word Sense Disambiguation: Some Recent Improvements*, 1997.

[21] Nielsen, Michael: *Neural Networks and Deep Learning.*

[22] Plaut, David C. y Geoffrey E. Hinton: *Learning sets of filters using back-propagation.* Computer Speech & Language, 2(1):35 – 61, 1987, ISSN 0885-2308.

[23] Raudys, Sarunas J, Anil K Jain *y cols.*: *Small sample size effects in statistical pattern recognition: Recommendations for practitioners.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(3):252–264, 1991.

[24] Rokach, L. y O. Maimon: *Top-down induction of decision trees classifiers - A survey.* IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 35(4):476–487, 2005.

[25] Rosenblattd, Frank: *Principles of neurodynamics; perceptrons and the theory of brain mechanisms.* Spartan Books, 1962.

[26] Snoek, Jasper, Hugo Larochelle y Ryan P. Adams: *Practical Bayesian Optimization of Machine Learning Algorithms.* En *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, páginas 2951–2959, USA, 2012. Curran Associates Inc.

[27] Strickland, Jeffrey: *Predictive Analytics using R.* Lulu, inc, 2014.

[28] Vapnik, V. N. y A. Ya. Chervonenkis: *On the method of ordered risk minimization. I.* Autom. Remote Control, 35(8):1226–1235, 1974.

[29] Vapnik, Vladimir N.: *Statistical Learning Theory.* A Wiley Interscience Publication. Wiley, 1998.

[30] Vert, Jean Philippe: *kernel methods in Computational Biology. Cambridge,Masachissetts.* The MIT Press, 2004.

[31] Wolpert, David H: *The lack of a priori distinctions between learning algorithms.* Neural computation, 8(7):1341–1390, 1996.

[32] Woźniak, Michał, Manuel Graña y Emilio Corchado: *A survey of multiple classifier systems as hybrid systems.* Information Fusion, 16:3 – 17, 2014, ISSN 1566-2535. `http://www.sciencedirect.com/science/article/pii/S156625351300047X`, Special Issue on Information Fusion in Hybrid Intelligent Fusion Systems.

JORDI GIRONÉS ROIG
JORDI CASAS ROMA
JULIÀ MINGUILLÓN ALFONSO
RAMON CAIHUELAS QUILES

TECNOLOGÍA

MINERÍA DE DATOS

MODELOS Y ALGORITMOS

En este libro se introducen los conceptos fundamentales de la minería de datos (*data mining*) y del aprendizaje automático (*machine learning*). El lector podrá encontrar una revisión completa de las técnicas avanzadas más usadas en estos campos con un enfoque claramente descriptivo para que entienda los conceptos e ideas básicos ocultos detrás de cada algoritmo o técnica.

Las páginas de este libro abordan desde las etapas previas de preparación de los datos —los métodos de reducción de la dimensionalidad y extracción de características (PCA, SVD, NNMF), métodos de aprendizaje no supervisado (agrupamiento jerárquico, *k-means*, *canopy*), métodos de aprendizaje supervisado (*k*-NN, SVM, redes neuronales, árboles de decisión, métodos probabilísticos)—, hasta los diferentes métodos de combinación de clasificadores.

Con este libro aprenderás sobre:

✓ minería de datos; ✓ aprendizaje automático; ✓ inteligencia artificial; ✓ ciencia de datos; ✓ *data mining*; ✓ *machine learning*; ✓ *data science*

EDITORIAL UOC