Introducción al aprendizaje automático

PID_00279445

Félix José Fuentes Hurtado
Lluís Gómez Bigorda
Marc Maceira Duch
David Masip Rodó
Vicenç Torra Reventós
Carles Ventura Royo

Tiempo mínimo de dedicación recomendado: 4 horas

Universitat Oberta de Catalunya

Félix José Fuentes Hurtado **Lluís Gómez Bigorda** **Marc Maceira Duch**

David Masip Rodó **Vicenç Torra Reventós** **Carles Ventura Royo**

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por los profesores: Carles Ventura Royo, Marc Maceira Duch

Primera edición: febrero 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Félix José Fuentes Hurtado, Lluís Gómez Bigorda, Marc Maceira Duch, David Masip Rodó, Vicenç Torra Reventós, Carles Ventura Royo
Producción: FUOC

Índice

Introducción

El aprendizaje automático o aprendizaje computacional es el subconjunto de la inteligencia artificial que estudia los algoritmos informáticos que mejoran automáticamente a partir de la experiencia. Los métodos de aprendizaje automático usan modelos matemáticos basados en datos de muestra que denominamos *datos de entrenamiento*. Estos modelos matemáticos no están programados explícitamente para tomar decisiones, su comportamiento viene determinado por el aprendizaje a partir de los datos de entrenamiento. Se trata de programas capaces de generalizar comportamientos a partir del reconocimiento de patrones. Los algoritmos de aprendizaje automático se usan con éxito en una amplia variedad de aplicaciones, como por ejemplo la traducción de textos, el filtraje de correos electrónicos y la visión por ordenador. Ya en algunas asignaturas previas vimos algunos de los métodos que se consideran en esta familia de algoritmos. En este módulo repasamos los conceptos principales del aprendizaje automático, así como algunos algoritmos y casos de uso.

El módulo empieza definiendo qué entendemos por *aprendizaje automático*. Se definen los conceptos básicos y se hace una clasificación de los métodos de aprendizaje principales. En este apartado, definimos los campos más relevantes del aprendizaje automático: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

En el segundo apartado, estudiaremos los pasos necesarios antes de entrenar cualquier algoritmo de aprendizaje automático: la preparación de los datos. Veremos qué fases hacen falta antes de empezar el aprendizaje: cómo se obtienen los datos, cómo se definen los atributos a partir de los cuales se hará el aprendizaje y cómo lo podemos facilitar seleccionando y extrayendo características. Analizaremos algunos de los algoritmos lineales para llevar a cabo el proceso de extracción de características, como por ejemplo el análisis de componentes principales (PCA) o el análisis discriminante lineal (LDA).

En el tercer apartado, profundizaremos en los conceptos más importantes del aprendizaje no supervisado. Nos centraremos en uno de los métodos más utilizados para este tipo de aprendizaje: el algoritmo de categorización *k-means*.

El módulo acaba en el cuarto apartado, que introduce algunos conceptos generales de los métodos de aprendizaje automático. En este apartado recordamos los conceptos de *partición de datos*, *métricas de evaluación* y *minimización del error* que ya vimos en asignaturas previas. Finalmente, el apartado acaba planteando el problema del sesgo y la varianza (términos complementarios para la evaluación de la capacidad de generalización de un método) y la introducción de algunos conceptos que se derivan de ellos.

Objetivos

Los objetivos de este módulo didáctico son:

1. Obtener una visión general de lo que es el aprendizaje dentro de la inteligencia artificial.
2. Distinguir entre el aprendizaje supervisado y el no supervisado.
3. Conocer la diferencia entre las tareas de clasificación y las de regresión.
4. Estudiar los principales métodos de extracción de características.
5. Estudiar los principales métodos de aprendizaje no supervisado.

1. ¿Qué entendemos por aprendizaje automático?

El aprendizaje computacional o aprendizaje automático[1] es una rama de la inteligencia artificial dedicada al estudio de algoritmos informáticos que mejoran automáticamente a partir de la experiencia. Los algoritmos de aprendizaje automático construyen un modelo matemático a partir de ciertos datos de ejemplo, conocidos como *datos de entrenamiento*, para realizar tareas que son demasiado difíciles de resolver con programas convencionales fijos escritos y diseñados por seres humanos. Mitchell (1997) nos da una descripción breve y clara del concepto de *aprendizaje* en este contexto:

(1) En inglés, *machine learning*

> «Diremos que un programa informático aprende de la experiencia *E* respecto de alguna tarea *T* y de alguna medida de rendimiento *P*, si su rendimiento en *T*, medido por *P*, mejora con la experiencia *E*».
>
> T. M. Mitchell (1997). *Machine Learning*. Nueva York: McGraw-Hill.

En este módulo didáctico describiremos y veremos algunos ejemplos de tareas, medidas de rendimiento, y tipos de experiencias que se pueden usar para crear algoritmos de aprendizaje automático. De hecho, en el subapartado 1.1 veremos que estos conceptos nos sirven para crear una clasificación de las diferentes técnicas de aprendizaje.

En las asignaturas previas ya se habló de algunos métodos de aprendizaje automático, por ejemplo: los métodos de regresión lineal y regresión logística, el análisis de componentes principales, el algoritmo de categorización *k-means* o los árboles de decisión. Los objetivos de este primer módulo son, por un lado, introducir el aprendizaje automático y, por otro, repasar algunos de esos algoritmos que ya se han visto anteriormente, situándolos bajo un mismo marco común que mantendremos durante el curso.

En este primer apartado introducimos el tema del aprendizaje y presentamos una clasificación (taxonomía) de los diferentes métodos que hay.

1.1. Una clasificación de las técnicas de aprendizaje

Ya hemos comentado que hay varias técnicas de aprendizaje, y que algunas de las diferencias entre estas son debidas al hecho de aplicarlas en diferentes tareas o de utilizar diferentes tipos de experiencia para el aprendizaje. En este subapartado, damos una clasificación de los métodos de aprendizaje y describimos sus características más importantes.

En particular, uno de los aspectos más importantes a la hora de considerar estos métodos, y que está relacionado con el tipo de experiencia que usa, es su supervisión. Es decir, de qué tipo de información se dispone sobre el resultado

que tiene que dar un sistema para cada posible entrada. Así, si consideramos un sistema como una función que da resultados (la salida del sistema) a partir de cierta información (las entradas del sistema), tenemos tres tipos de métodos según lo que se pueda decir sobre la salida. Estos tipos son los siguientes: métodos con aprendizaje supervisado, aprendizaje por refuerzo y aprendizaje no supervisado. A continuación, los repasamos y damos algunos ejemplos.

Aprendizaje supervisado[2]: corresponde a la situación en que hay un conocimiento completo sobre cuál es la respuesta que se debe dar en una determinada situación. Cuando tenemos aprendizaje supervisado, el objetivo es conseguir un sistema que reproduzca la salida cuando estamos en una situación que corresponda a la entrada.

(2) En inglés, *supervised learning*

En muchos casos, el problema puede describirse formalmente como un problema de optimización o como la aproximación de una función a partir de unos ejemplos (pares entrada/salida). Cuando el problema se plantea de este modo, se habla de aprendizaje inductivo. También se habla a menudo de aprendizaje a partir de ejemplos, y el conjunto de ejemplos se denomina *conjunto de entrenamiento*[3].

(3) En inglés, *training set*

Para formalizar este tipo de aprendizaje, consideramos un conjunto de entrenamiento C formado por N ejemplos, en el que cada ejemplo es un par $(\boldsymbol{x}, y)$ en el que $\boldsymbol{x}$ es un vector de dimensión M y y es el resultado de aplicar una función f (que no conocemos) al vector $\boldsymbol{x}$. Por lo tanto, tenemos N ejemplos en un espacio de dimensión M. Es decir $X = \{x_1,..., x_y,..., x_N\}$. En caso de que haya un error en las medidas, tenemos que y es $f(\boldsymbol{x})$ más un cierto error ϵ. Es decir, $y = f(\boldsymbol{x}) + \epsilon$. A partir de esta información, quiere construirse un modelo que denotamos por M_C (usamos el subíndice porque el modelo depende del conjunto de ejemplos C). Así, el objetivo es conseguir que el modelo M_C aplicado a un elemento $\boldsymbol{x}$ dé un resultado similar a $f(\boldsymbol{x})$. Es decir, $M_C(\boldsymbol{x})$ es una aproximación de $f(\boldsymbol{x})$. Cuando no haya peligro de confusión, utilizaremos simplemente M para expresar el modelo.

A la hora de considerar el aprendizaje supervisado, conviene distinguir diferentes tipos de problemas (tareas) porque los métodos de aprendizaje que se aplican son diferentes. Hay que decir que esta división no es propia del aprendizaje supervisado, sino general, pero en los otros tipos de aprendizaje no es tan relevante. Ahora pasamos a considerar los dos tipos de problemas más comunes: problemas de regresión y problemas de clasificación.

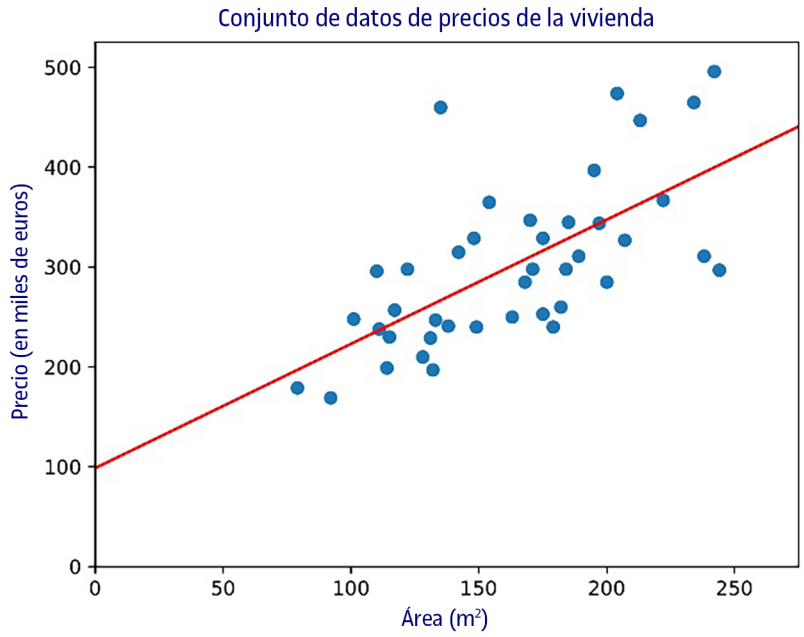
a) Problemas de regresión[4]: cada instancia, o ejemplo, incluye un atributo (consideramos la salida como un dato más de la descripción del objeto) con la solución. El objetivo de los sistemas basados en el conocimiento es reproducir este atributo. La regresión corresponde al caso de tener un atributo solución de tipo numérico.

(4) En inglés, *regression*

Dado que en las redes neurales tanto las entradas como las salidas son habitualmente numéricas, las podemos tomar como ejemplo de método de aprendizaje de esta familia. Los métodos de regresión estadística (por ejemplo, construir una recta de regresión a partir de un conjunto de puntos) también pertenecen a este grupo.

Para poner un ejemplo simple de una tarea de regresión, imaginemos que queremos crear un modelo capaz de predecir el precio de una casa a partir de los metros cuadrados que tiene. Así, la entrada a nuestro modelo es un valor numérico (los metros cuadrados) y la salida es también un valor numérico (el precio). La figura siguiente muestra un posible conjunto de datos de entrenamiento para este problema, en que cada punto representa un par entrada/salida, así como un posible modelo de regresión lineal simple que se ha ajustado a estos datos.

Figura 1. Ejemplo de una tarea de regresión. La entrada en nuestro modelo es un valor numérico (los metros cuadrados de una casa) y la salida es también un valor numérico (el precio de la casa).



b) Problemas de clasificación[5]: como en el caso de la regresión, cada instancia incluye un atributo con la descripción de la solución. En este caso, el atributo es categórico (no numérico) o bien binario. El objetivo del aprendizaje es inducir un modelo que predice de manera esmerada dicho atributo.

(5) En inglés, *classification*

A modo de ejemplo de un problema de clasificación, introducimos el conjunto de datos IRIS, que contiene 150 ejemplos de plantas de las diversas especies de iris (lirios). En concreto, disponemos de la información siguiente para cada ejemplar: anchura del pétalo (*petal width*), longitud del pétalo (*petal length*), anchura del sépalo (*sepal width*) y longitud del sépalo (*sepal length*). La siguiente tabla muestra algunos ejemplos:

Enlace de interés

Iris Data Set. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/iris.

Tabla 1. Ejemplo de un conjunto de datos para un problema de clasificación. Se muestran algunos ejemplos del conjunto de datos IRIS para la clasificación de las plantas.

Clase	*Sepal length*	*Sepal width*	*Petal length*	*Petal width*
Iris setosa	5.1	3.5	1.4	0.2
Iris setosa	4.9	3.0	1.4	0.2
Iris setosa	4.7	3.2	1.3	0.2
Iris setosa	4.6	3.1	1.5	0.2
Iris versicolor	7.0	3.2	4.7	1.4
Iris versicolor	6.4	3.2	4.5	1.5
Iris virginica	5.8	2.7	5.1	1.9
Iris virginica	7.1	3.0	5.9	2.1
Iris virginica	6.3	2.9	5.6	1.8
Iris versicolor	6.9	3.1	4.9	1.5
…	…	…	…	…

A partir de estos datos, lo que queremos es construir un modelo de clasificación que, con los valores de estas características (anchura del pétalo, longitud del pétalo, etc.), sea capaz de clasificar correctamente un nuevo ejemplo de planta por su tipo: *Iris versicolor*, *Iris setosa* o *Iris virginica*. Fijaos que en este proceso estaremos convirtiendo muestras de datos en un modelo para realizar una **tarea de clasificación**. Idealmente, el modelo resultante debe poder **generalizar** inferencias para un conjunto más amplio (potencialmente infinito) de datos, más allá de nuestro conjunto de datos de entrenamiento.

En el transcurso de la asignatura veremos varios algoritmos que pueden aplicarse en tareas de clasificación: por ejemplo, regresión logística, *naïve Bayes*, árboles de decisión, *support vector machines*, etc.

A pesar de que los problemas de regresión y clasificación son los más comunes y los que primero se estudian en aprendizaje automático, no son los únicos. Otro ejemplo son las tareas con salidas estructuradas, en que la salida del modelo es un vector (o cualquier otra estructura de datos con más de un valor)

con relaciones importantes entre sus elementos. Son ejemplos de dichas tareas las de traducción o transcripción de texto (en que tanto la entrada como la salida son secuencias de caracteres/palabras).

Aprendizaje por refuerzo[6]: en este caso el conocimiento sobre la calidad del sistema solo es parcial. No se dispone del valor que corresponde a la salida para un determinado conjunto de valores de entrada, sino solo de una gratificación o una penalización según el resultado que haya dado el sistema. Por ejemplo, si un robot planifica una trayectoria y colisiona con un objeto recibirá una penalización, pero en ningún momento hay un experto que le ofrezca una trayectoria alternativa correcta. Del mismo modo, si la trayectoria planificada por el robot no provoca ninguna colisión, el robot recibe una gratificación. Las planificaciones que el robot haga a partir de este momento tendrán en cuenta las penalizaciones y gratificaciones recibidas.

(6) En inglés, *reinforcement learning*

En el aprendizaje por refuerzo, la experiencia de los algoritmos no proviene de un conjunto fijo de datos de entrenamiento, como en el caso del aprendizaje supervisado. Por el contrario, el algoritmo interactúa con su entorno para obtener supervisión (*feedback*) de manera dinámica.

Aprendizaje no supervisado[7]: solo se dispone de información sobre las entradas, no sobre las salidas. El aprendizaje tiene que extraer conocimiento útil a partir de la información disponible. En este tipo de aprendizaje se encuentran los algoritmos de categorización (que permiten ordenar la información disponible).

(7) En inglés, *unsupervised learning*

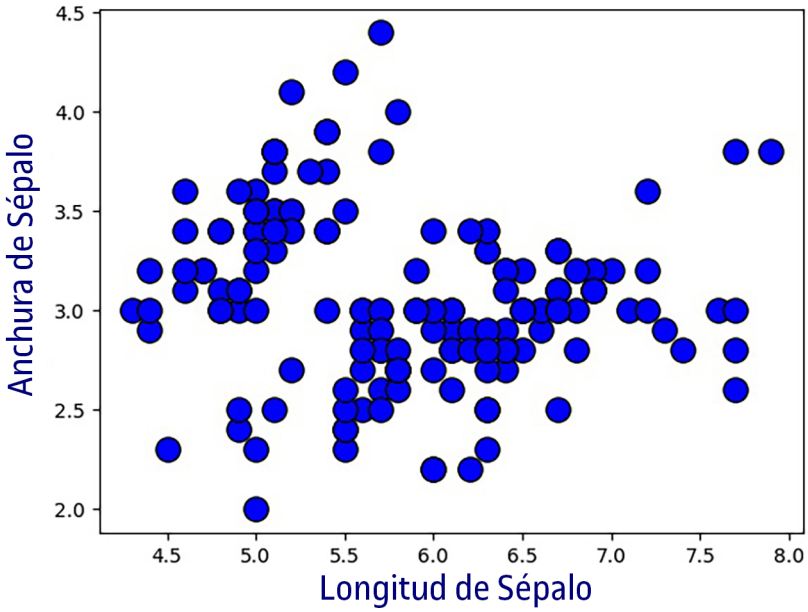
En el aprendizaje no supervisado, la experiencia de los algoritmos proviene de un conjunto de datos con varios atributos (o características) pero ahora sin una salida asignada a cada muestra, tal como teníamos en el caso de los problemas de clasificación. A partir de estos datos, el algoritmo tiene que aprender propiedades útiles de la estructura de estos datos que sirvan para hacer alguna tarea o solucionar un problema determinado.

Como ejemplo de una posible aplicación de aprendizaje no supervisado, imaginemos que tenemos el mismo conjunto de datos IRIS que hemos visto antes, pero ahora no disponemos de la información de clase (tipo de planta) de cada ejemplo.

Por lo tanto, nuestros datos de entrenamiento son como los que se muestran en la tabla 1 pero sin la primera columna (Clase). A partir de estos datos, queremos encontrar de manera automática grupos de plantas que tengan carac-

terísticas similares. La figura siguiente muestra nuestro conjunto de datos en el espacio bidimensional definido por dos de los atributos numéricos de que disponemos: la anchura del sépalo y la longitud del sépalo.

Figura 2. Visualización del conjunto de datos IRIS en el espacio bidimensional definido por los atributos *anchura de sépalo* y *longitud de sépalo*.



Fijaos que, a pesar de no disponer de la información de clase, la propia estructura de los datos en esta visualización nos revela la existencia de al menos dos grupos de plantas con características parecidas. Obviamente, la inspección visual de los datos no será una opción si trabajamos con un gran número de atributos y necesitamos algoritmos de aprendizaje no supervisado para analizar la estructura de los datos y encontrar grupos de muestras similares.

Ved también

En el apartado 3 de este módulo didáctico, veremos varios algoritmos de aprendizaje no supervisado, como por ejemplo los algoritmos de categorización o el algoritmo *k-means*.

2. Preparación de los datos

En el aprendizaje automático, antes de llevar a cabo el entrenamiento de los algoritmos necesitamos definir y preparar los datos correctamente. Un buen planteamiento del problema permitirá disponer de unos datos representativos que nos asegurarán el éxito del entrenamiento. La cantidad y la calidad de los datos marcarán cuán cuidadosos pueden ser los modelos de aprendizaje automático generados a partir de los datos.

Típicamente, el proceso de aprendizaje automático consta de las siguientes etapas:

1) Obtención de los datos: hay que determinar si se puede usar una base de datos disponible en línea. Si no es posible, hay que generar una base de datos nueva.

2) Representación de la información: Antes de realizar el entrenamiento es necesario estudiar los datos disponibles y prepararlos para que los algoritmos puedan aprender de ellos. Operaciones que se realizan en esta fase son: la transformación de los datos para tenerlos todos en formato numérico, la harmonización de los rangos dinámicos de los datos y la definición de qué categorías queremos tener.

3) Selección y extracción de características: para que el entrenamiento sea lo más provechoso posible, hay que analizar los datos de entrenamiento obtenidos en el punto anterior para eliminar redundancias y seleccionar qué información es útil para la fase de entrenamiento.

4) Aprendizaje del sistema: una vez disponemos de los datos necesarios, podemos entrenar el sistema. Tal como hemos visto en el apartado anterior, hay diferentes paradigmas de aprendizaje automático. Esta es la fase que estudiaremos más a fondo en la asignatura, pero solo es una parte de todo el proceso de aprendizaje automático.

5) Evaluación del modelo: en esta fase se comparan diferentes modelos de aprendizaje automático. La evaluación del modelo se hace con datos que no se han utilizado durante el proceso de entrenamiento. También es necesario definir las medidas de error para comparar los diferentes algoritmos.

6) Predicción: una vez tenemos el mejor modelo de aprendizaje automático, lo podemos utilizar para hacer predicciones con datos nuevos.

En este apartado veremos los dos primeros pasos del proceso de aprendizaje automático: obtención de los datos y representación de la información.

2.1. Obtención de los datos

El primer paso en cualquier problema de aprendizaje automático consiste en determinar qué datos se usarán para entrenar los modelos. Hay que tener en cuenta que el proceso de obtención de datos puede ser muy costoso, puesto que hay que generar una muestra bastante representativa a partir de la cual podamos obtener un modelo capaz de funcionar con datos nuevos.

Para obtener los datos, habrá que pasar por las siguientes etapas:

1) Determinar el problema: este paso consiste en determinar qué datos querremos utilizar en nuestro modelo. Habrá que tener un conocimiento previo del problema para poder identificar correctamente los datos necesarios. En el ejemplo de clasificación introducido en el apartado anterior, el conjunto de datos IRIS, usa la anchura del pétalo (*petal width*), la longitud del pétalo (*petal length*), la anchura del sépalo (*sepal width*) y la longitud del sépalo (*sepal length*) como datos para hacer la clasificación.

2) Obtención de los datos: hay que reunir una cantidad de datos suficiente para representar el espacio de posibilidades del problema. Suele ser un trabajo bastante tedioso, puesto que se acostumbra a trabajar con el objetivo de obtener la mayor cantidad de datos. En el ejemplo IRIS, consiste en anotar de cada flor las medidas del pétalo y del sépalo.

3) Anotación: en el caso de aprendizaje supervisado, habrá que indicar a qué clase pertenece cada muestra de la base de datos. En el ejemplo de clasificación IRIS, habrá que indicar en cada registro de flores si corresponde a *Iris versicolor*, *Iris setosa* o *Iris virginica*.

2.2. Representación de la información

2.2.1. Atributos categóricos y atributos binarios

Algunos métodos de categorización se basan en la transformación de atributos no numéricos en atributos binarios. En el recuadro adjunto, se muestra cómo puede hacerse esta transformación.

Terminología

Para diferenciar los atributos numéricos de los ordinales y nominales, a menudo se usa el término *atributos categóricos* o *discretos* para los últimos.

Transformación entre atributos categóricos y atributos binarios

La transformación distingue dos casos según si el atributo es nominal u ordinal.

a) Para atributos nominales de la forma $DOM(A_i)=\{a_{i1}, a_{i2},..., a_{iRi}\}$, podemos sustituir el atributo A_i por R_i atributos binarios $A_{i1}, A_{i2},..., A_{iRi}$, en el que A_{ij} se satisfará únicamente si el valor del objeto para aquel atributo es a_{ij}.

b) Para atributos ordinales, puede utilizarse la codificación aditiva, que consiste en sustituir de manera similar el atributo A_i pero ahora por R_i – 1 atributos binarios $A_{i2}, A_{i3},..., A_{iRi}$. Ahora tenemos que A_{ij} se satisface cuando el valor del objeto para aquel atributo es más grande o igual que A_{ij}.

Nota

Como dicen Sneath y Sokal en *Numerical Taxonomy* (W. H. Freeman, 1973, pág. 148), en el pasado también se transformaban los atributos numéricos en binarios (mediante discretización) para simplificar los cálculos.

A continuación presentamos, como ejemplo, la codificación de un atributo A que tiene un dominio de cuatro valores $DOM(A) = \{a_1, a_2, a_3, a_4\}$. Se da la codificación que tendremos cuando A sea un atributo nominal (a la izquierda) y cuando A sea un atributo ordinal (codificación aditiva, derecha).

Valor original	Codif. nominal	Codif. additiva
A	A_1 A_2 A_3 A_4	A_2 A_3 A_4
a_1	1 0 0 0	0 0 0
a_2	0 1 0 0	1 0 0
a_3	0 0 1 0	1 1 0
a_4	0 0 0 1	1 1 1

2.2.2. Definición de distancias y medidas de similitud

La aplicación práctica de los algoritmos de aprendizaje obliga a considerar medidas (cuantificaciones, habitualmente numéricas) de la similitud entre dos objetos. La medida de esta similitud viene dada por los llamados *coeficientes de similitud*.

Coeficientes de similitud

Los coeficientes de similitud se denominan en inglés *similarity coefficient* (o, no tan a menudo, *ressemblance coefficient*).

Hay muchos tipos de coeficientes –por ejemplo, basados en distancias, en asociaciones o en correlaciones, o los de similitud probabilística–, pero los más comunes son los basados en distancias.

Los coeficientes basados en distancias se basan en la definición de una distancia en un cierto espacio. Por ejemplo, la distancia euclidiana cuando los atributos son numéricos, o la distancia de Hamming para atributos binarios o categóricos que se han convertido en atributos binarios como en el ejemplo

(8) En inglés, *dissimilarity*

anterior. De hecho, en estos coeficientes tenemos que la distancia nos proporciona una disimilitud[8]. Y a partir de esta se construye la similitud como función complementaria.

Por ejemplo, partir de una distancia d, cuando esta es definida en el intervalo $[0, 1]$, se puede definir un coeficiente de similitud haciendo $s(x, y) = 1 - d(x, y)$. En este caso, la similitud siempre será positiva y es máxima e igual a 1 cuando los individuos x y y coinciden (son idénticos).

Otro aspecto que hay que considerar para calcular similitudes y distancias en problemas reales es que los dominios de los diferentes atributos pueden ser muy diferentes. En este caso, las distancias que se calculan estarán fuertemente influenciadas por los valores del dominio. Por ejemplo, si tenemos un estudio de población y disponemos para cada individuo los atributos *número de hijos* y *salario*, obtendremos que la clasificación estará condicionada, sobre todo, por el salario. La diferencia entre dos personas con el mismo número de hijos y salario diferente será normalmente más grande que la que hay entre dos personas con el mismo salario y diferente número de hijos. Esto es así porque la variabilidad del salario es más grande que la variabilidad en el número de hijos. En estos casos puede ser conveniente normalizar los datos en un rango común.

De forma análoga, otro elemento que hay que tener en cuenta es la importancia que se debe dar a los diferentes atributos. Las funciones de distancia presentadas hasta ahora consideran, por defecto, que todos los atributos son igualmente importantes. Es decir, todos los atributos tienen que afectar del mismo modo a la categorización.

A veces, esta importancia igual para todos los atributos no es conveniente. Hay veces en los que hay unos atributos que son relevantes en el proceso de categorización o de clasificación y otros, en cambio, que son irrelevantes. Para tener en cuenta esto, hay funciones de similitud y de distancia que incluyen un peso para cada atributo. Aun así, el uso de estas expresiones requiere fijar los pesos antes de empezar el proceso de categorización, tarea que no es sencilla.

2.2.3. Estructuras posibles para las categorías construidas

Una manera de distinguir los diferentes métodos de categorización es considerar la relación que se establece entre los objetos originales y las categorías construidas.

Lo más habitual es que nos encontremos con alguno de los casos siguientes:

- **Categorías disjuntas**: un objeto puede pertenecer solo a una categoría. Esta situación corresponde a los algoritmos que construyen una partición

del dominio. El algoritmo *k-means* que veremos a continuación es un algoritmo de este tipo.

- **Categorías que se sobreponen**: los límites entre las categorías no son nítidos. Este es el caso de los algoritmos que construyen una partición difusa. El *fuzzy k-means* es un algoritmo que construye este tipo de estructura.

- **Categorías organizadas en forma de árbol jerárquico**: se establece una relación entre las categorías que se expresan mediante una estructura de árbol. La estructuración de objetos en forma de árbol se denomina *dendrograma*. Los algoritmos de categorización jerárquica nos definen conceptos en diferentes niveles, y establecen que los conceptos que comparten un nodo padre son más similares entre ellos que respecto a los demás. Además, se entiende que la relación entre un nodo del árbol y su nodo padre en una jerarquía corresponde a una relación de superclase (o una relación «es uno»).

2.3. Selección y extracción de características

En este subapartado estudiaremos diferentes metodologías que permiten reducir la complejidad de los datos: por un lado, los algoritmos de selección de características y, por otro lado, la extracción de características. Veremos que el problema permite muchas aproximaciones dependiendo de la función de coste que tenemos que maximizar y aprenderemos un subconjunto de las técnicas clásicas que todavía hoy se utilizan en el área del aprendizaje automático.

En todo este subapartado, asumiremos que disponemos de un conjunto de muestras de entrenamiento que nos tienen que servir para aprender un modelo del problema que tenemos que tratar. También supondremos que las muestras no tienen valores ausentes y que son representadas mediante un vector de características numérico.

Puede ser necesario llevar a cabo un proceso de selección o extracción de características para varios factores dependiendo de la aplicación. Lo más común es disponer de grandes volúmenes de datos, pero a la vez con un alto grado de redundancia, por lo cual la extracción de la información importante se convierte en un objetivo prioritario. En este caso, los procesos de selección y extracción de características están muy ligados a los algoritmos de reducción de la dimensionalidad[9]. Los datos originales se acostumbran a representar en un espacio $\mathbb{R}^M$, en el que M es la dimensionalidad del espacio, y el objetivo es reducir el número de atributos o dimensiones en este espacio para disminuir la medida de los datos que tenemos que analizar. Estos procesos no solamente tienen un efecto positivo en el aprendizaje posterior de los datos, sino que a la vez disminuyen el coste computacional de los algoritmos posteriores y las necesidades de almacenamiento.

(9) En inglés, *dimensionality reduction*

Por otro lado, en otras aplicaciones, los datos analizados pueden tener un volumen considerable de ruido. En estos casos, los métodos de selección y extracción de características se ocuparán de filtrar la información que puede resultar perjudicial para el proceso de aprendizaje y de mantener solo la información realmente útil *a posteriori*.

La selección y la extracción de características se utilizan también para reducir el efecto de la maldición de la dimensionalidad[10]. Este concepto, recurrente en las áreas de la estadística y el aprendizaje automático, fue enunciado por primera vez por Richard Bellman. En esencia, trata la problemática del volumen de datos que tenemos que analizar cuando el espacio de características crece en dimensionalidad.

(10) En inglés, *curse of dimensionality*

Supongamos, por ejemplo, que tenemos un espacio 2D que queremos partir en una parrilla de 10 × 10 casillas. Con un mínimo de 100 muestras, ya podríamos tener cubierta esta parrilla. Ahora bien, si nuestro espacio fuera 3D, nos habrían hecho falta 1.000 muestras para tener un representante en cada cubo, y 10^{100} si nuestro espacio hubiera tenido una dimensión 100. Este crecimiento exponencial en el número de «casillas» en la partición del espacio respecto a su dimensionalidad puede hacer que también necesitemos un conjunto de muestras que crezca exponencialmente si queremos tener una caracterización adecuada del problema.

Los algoritmos de reducción de la dimensionalidad permiten reducir los efectos de la maldición de la dimensionalidad y pueden estimar mejor los parámetros de los métodos de aprendizaje en espacios más reducidos.

2.3.1. Taxonomía de los algoritmos de selección y extracción de características

Una vez definidos los objetivos de la selección y extracción de características, veremos una breve taxonomía de los diferentes algoritmos que se usan actualmente y analizaremos en detalle alguno de los más utilizados. Partiremos de que nuestros datos se nos dan como un conjunto de N muestras X, en el que $X = \{x_1, x_2, ..., x_N\}$, y cada muestra x_i consiste en un conjunto de M atributos $A = \{A_1, A_2, ..., A_M\}$, también denominados *características*.

Para abordar la problemática que hemos descrito, disponemos básicamente de dos herramientas:

1) Selección de características[11]. El objetivo es, dado el conjunto X, elegir qué subconjunto de características $A = \{A_a, A_b, ...\}$ nos resulta más beneficioso para resolver nuestro problema. Expresado de otro modo, nos interesará eliminar las características que no son útiles y preservar solo las que maximicen la función de utilidad.

(11) En inglés, *feature selection*

2) Extracción de características. El objetivo es construir un nuevo conjunto de características $A = \{A_1, A_2, ..., A_D\}$ a partir de las iniciales. Hay que observar que, en este caso, las características del nuevo conjunto no tienen que coincidir necesariamente con las del conjunto original y el resultado de la extracción puede ser una mixtura del conjunto inicial.

En lo que queda de este subapartado, nos centraremos en presentar dos técnicas de extracción de características, porque son más relevantes en el contexto del aprendizaje automático.

2.3.2. Algoritmos de extracción de características

Tal como se ha comentado, a menudo los datos pueden tener ruido, redundancia o información que no es útil para la tarea concreta que tienen que llevar a cabo. El objetivo de la extracción de características será, pues, obtener un espacio de dimensionalidad inferior, que preserve al máximo los datos útiles y elimine la información que no nos interesa.

A diferencia de la selección de características, en que solo podemos elegir algunos de los atributos de los ejemplos, en la extracción de características también podemos crear nuevos atributos a partir de los ya conocidos.

Según los objetivos para los cuales usamos la extracción de características, podemos hacer una primera taxonomía que distinga entre los tipos siguientes:

- **Extracción de características no supervisada.** Los algoritmos disponen de un conjunto de datos de entrenamiento, y el objetivo es la obtención de un espacio de dimensión reducida que minimice una función de coste determinada. Ejemplos de este tipo de técnicas son el análisis de componentes principales[12], la factorización no negativa de matrices[13] o el análisis de componentes independientes[14].

- **Extracción de características supervisada.** Los algoritmos disponen de información sobre la pertenencia de los datos a un determinado conjunto de **clases** (etiquetas asociadas a cada muestra). El objetivo es conseguir un conjunto de características que permita una separación óptima de los ejemplos de entrenamiento según este etiquetado. El ejemplo más paradigmático de esta categoría son las técnicas basadas en el análisis discriminante.

(12) En inglés, *principal component analysis*

(13) En inglés, *non-negative matrix factorization*

(14) En inglés, *independent component analysis*

En una segunda taxonomía, los algoritmos se pueden clasificar según la tipología de la función de extracción de características:

- **Extracción lineal de características**. La función que lleva a cabo la extracción de características es lineal y se puede expresar mediante un producto de matrices.

- **Extracción no lineal de características**. Su formulación utiliza métodos no lineales para hacer la mixtura de las características importantes. Hay gran cantidad de aproximaciones no lineales en la literatura. Ejemplos de dichas aproximaciones son el uso de redes neurales, los métodos basados en núcleos (*kernels*) aplicados a sistemas lineales clásicos o los métodos basados en variedades.

En este módulo nos centraremos en las técnicas lineales. Estudiaremos dos de las técnicas más importantes y utilizadas: el análisis de componentes principales (PCA), como no supervisada, y el análisis discriminante[15], respecto a las supervisadas.

(15) En inglés, *fisher linear discriminant analysis*

Más adelante, en esta asignatura, veremos los métodos basados en núcleos (*kernels*) pero aplicados a problemas de clasificación supervisada. Su extensión a la extracción de características nos permitiría definir versiones no lineales de cualquiera de los algoritmos que comentaremos en este apartado, pero su formulación queda fuera del alcance de este curso.

Análisis de componentes principales (PCA)

El análisis de componentes principales (PCA) es posiblemente la técnica de extracción de características que más se usa. Hay diferentes interpretaciones de la metodología, a pesar de que, informalmente, se puede definir como la técnica que intenta conseguir una representación de un conjunto de datos en un espacio de dimensionalidad más reducida y minimizar el error cuadrático cometido.

Nota

Ved que el resultado de la selección de características se puede simular mediante una extracción lineal de características en que la matriz de proyección tiene 0 en todas partes y 1 en las filas correspondientes a las características seleccionadas.

El descubrimiento del análisis de componentes principales se atribuye a Karl Pearson (1901), y se ha aplicado a varios campos de la ciencia, la estadística y la economía, a menudo bajo la denominación de *transformada de Karhunen-Loève* o *transformada de Hotelling*.

Para definir formalmente el algoritmo, empezaremos definiendo la notación utilizada y formularemos el problema como un proceso de minimización analítico. Consideraremos que nuestros datos son un conjunto de N ejemplos X, donde $X = \{x_1, x_2,..., x_N\}$, y cada ejemplo x_y se compone de un conjunto de M atributos $A = \{A_1, A_2,..., A_M\}$. Consideraremos que todos los ejemplos tienen el mismo número de atributos y que no tenemos atributos ausentes. Entonces,

podemos representar el conjunto de datos como una matriz de dimensión $M \times N$, en el que cada columna se corresponde con una muestra del conjunto de datos y cada fila con un atributo.

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \cdots & \cdots & \cdots \\ x_{M1} & \cdots & x_{MN} \end{pmatrix}$$

Siguiendo con esta notación, y recordando que el PCA es una técnica de extracción de características lineal, podemos expresar el resultado del proceso de extracción de características como el producto matricial:

$$S = PX$$

En el que X es nuestra matriz de datos de dimensionalidad $M \times N$; $S = \{s_1, s_2, \ldots, s_N\}$ será la matriz $D \times N$, en que tendremos los N ejemplos de dimensionalidad D (D << M); y P será la matriz de proyección, de dimensión $D \times M$, que nos permitirá extraer características. El objetivo consiste en encontrar esta matriz P que nos reduce la dimensionalidad de los datos. La única restricción impuesta es minimizar el error cuadrático cometido en este proceso de reducción de la dimensionalidad.

A continuación veremos el algoritmo que nos permite efectuar el PCA sobre un conjunto de datos X. Completaremos la descripción del PCA con una simulación bidimensional del proceso de extracción de características y una prueba constructiva del algoritmo, que nos permitirá evaluar ciertas propiedades en la medida del error de reconstrucción.

Algoritmo. Análisis de componentes principales

1) Sustraéis la media de los datos de cada columna de la matriz X. Este paso nos permite obtener un conjunto de datos centrados en el origen.

$$\hat{X} = X - Z$$

En el que Z es una matriz de $M \times N$, en la que se ha replicado N veces el vector media n de las columnas de X.

$$n = \frac{1}{N} \sum_{i=1}^{N} x_i$$

2) Calculáis la matriz de covarianza C de los datos como:

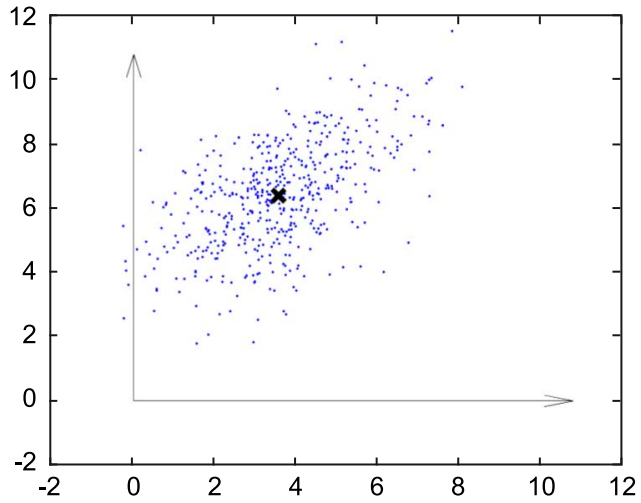
$$C = \frac{1}{N} \hat{X} \hat{X}^T$$

3) Cogéis, como matriz P, los D primeros vectores propios con más valor propio asociado de la matriz de covarianza C.

Ejemplo

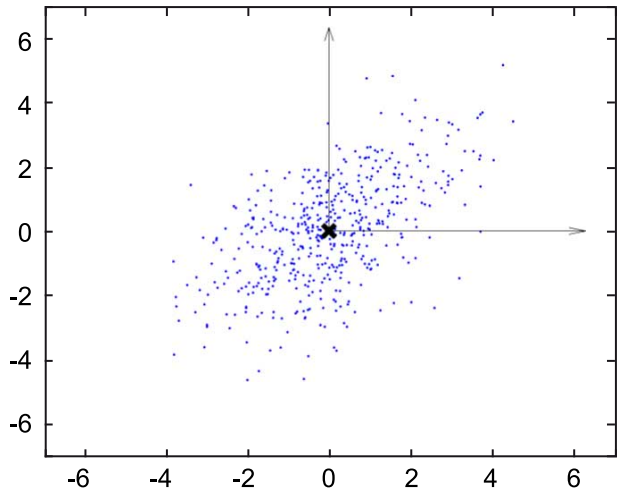
Nuestro punto de partida será un conjunto de datos de dimensión dos (y por lo tanto dibujables en un plano). Querremos efectuar una extracción de características que nos preserve solo un atributo por muestra y obtener, por lo tanto, un 50 % de compresión. Nuestro objetivo será obtener la matriz de proyección P que nos permita obtener esta nueva representación de nuestros datos minimizando el error cometido. La figura 3 muestra el ejemplo de partida que usaremos, en que se ha dibujado un punto por cada uno de los ejemplos representados por sus coordenadas 2D.

Figura 3. Ejemplo de muestras generadas aleatoriamente en un espacio en 2D.



A continuación, centraremos las muestras restando a cada punto el valor de la media del conjunto de entrenamiento (marcada con una X en la figura 4).

Figura 4. Muestras centradas (la media de las muestras coincide con el origen de coordenadas).

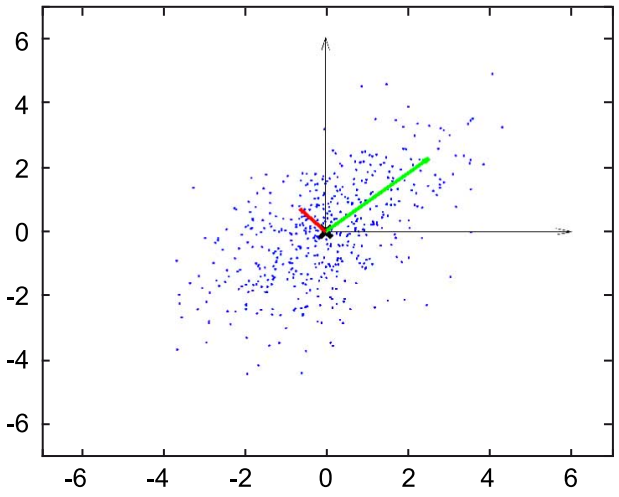


A continuación, calcularemos los vectores propios de la matriz de covarianza de los datos centrados. La figura 5 muestra gráficamente estos vectores. La longitud del vector (en color verde) indica el valor propio asociado.

Nota

Observad que el PCA coincide con los ejes de la gaussiana que cubre nuestros datos de entrenamiento e indica las direcciones donde los datos están descorrelacionados.

Figura 5. Ejes resultantes de aplicar el análisis de componentes principales.

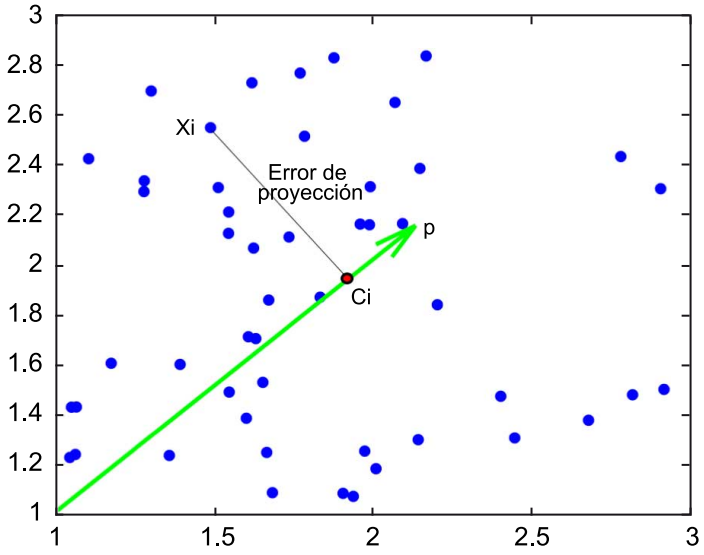


A pesar de que la demostración formal, en términos de minimización del error cuadrático, queda fuera del alcance de este módulo didáctico, es muy importante entender de manera intuitiva el principio por el cual funciona el análisis de componentes principales. Definimos el concepto de *error cuadrático* de la siguiente manera:

$$e = \sum_{i=1}^{N} \| x_i - (Px_i)P^T \|^2$$

Es decir, la suma de todos los errores individuales, entendiendo por *error individual* la diferencia entre el punto original (x_y) y su versión proyectada y antiproyectada en el espacio original ($(Px_i)P^T$). La figura 6 muestra gráficamente este error. Hay que observar que el error cometido en cada ejemplo equivale a la proyección en los componentes eliminados al hacer el PCA.

Figura 6. Error cometido al proyectar el ejemplo anterior en el componente principal de uno de los puntos.



En este ejemplo, para facilitar la visualización, consideramos que la matriz X tiene media 0 (se corresponde con unos datos centrados en su origen). En caso contrario, habría que dar un paso previo de sustracción de la media, tal como hemos comentado anteriormente. Una vez definida formalmente la función de error, vemos que proyectando cada punto x_i sobre el vector propio con mayor valor propio (p) es efectivamente la mejor manera de obtener una representación de una sola dimensión de forma que se minimice el error de reconstrucción. Para verlo todavía más claro, pensad en el caso extremo en que todos los puntos de X estuvieran en una línea pasando por el origen y con orientación igual a la del vector p. Obviamente, con el algoritmo PCA podríamos reducir la dimensionalidad de nuestro conjunto de datos a 1 sin ninguna pérdida de información.

Ronald Fisher

Ronald Fisher (1890-1962) fue uno de los estadistas y genetistas más importantes del siglo pasado. Sus trabajos más importantes, entre otros muchos, son el descubrimiento del ANOVA (análisis de varianza, o *analysis of variance*) y la estimación de parámetros mediante la máxima verosimilitud. En el año 1936 publicó el trabajo «The Use of Multiple Measurements in Taxonomic Problems» (*Annals of Eugenics*, n.º 7, págs. 179-188), en que puso las bases del análisis discriminante.

El análisis de componentes principales forma parte de la familia de técnicas de extracción de características de manera no supervisada. Como hemos visto, cuando la única fuente de información disponible son los datos mismos, la extracción de características se ve forzada a minimizar algún tipo de función de error sobre estos datos. En cambio, si disponemos de la información de pertenencia de los datos a un conjunto de clases predeterminado (aprendizaje supervisado), nos podemos plantear reducir la dimensionalidad preservando medidas de separación entre clases. El análisis discriminante lineal de Fisher que veremos a continuación representa un conjunto de métodos que abordan este problema.

Ronald Fisher

Análisis discriminante lineal (LDA)

Antes de definir formalmente el algoritmo, recordaremos la notación utilizada y añadiremos la información que nos permite hacer la tarea de manera supervisada. Consideraremos de nuevo que nuestros datos son un conjunto de N ejemplos X, en que $X = \{x_1, x_2,..., x_N\}$, y cada ejemplo x_i se compone de un

conjunto de M atributos $A = \{A_1, A_2,..., A_M\}$. Todos los ejemplos tienen el mismo número de atributos, no tenemos atributos ausentes y los representamos mediante la matriz siguiente:

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \cdots & \cdots & \cdots \\ x_{M1} & \cdots & x_{MN} \end{pmatrix}$$

Además, para cada ejemplo x_i tenemos asociada una etiqueta L_i, que nos define la pertenencia de x_i a una de las K clases que configuran el problema $L \in \{L_1, L_2,..., L_k\}$. La aparición de estas variables objetivo es lo que nos hace hablar de *aprendizaje supervisado*.

El algoritmo se basa en definir dos medidas de dispersión: la intraclase S_i y la interclase S_E. A menudo se calculan mediante matrices de dispersión específicas. La primera, S_i, nos mide lo separados que están los datos de una misma clase (de media) respecto a su punto central. La segunda, S_E, nos mide lo separados que están los centros de los datos de las clases respectivas. Con estas medidas, la función que tenemos que maximizar será, pues, la separación entre los datos que pertenecen a diferentes clases, procurando que los datos de la misma clase estén tan juntos como sea posible. Es decir, querremos encontrar el espacio de dimensión reducida que nos permite una separabilidad máxima en nuestros datos según las etiquetas que tienen asociadas. Formalmente, el criterio que estableció Fisher se puede resumir en el algoritmo que presentamos a continuación.

Algoritmo. Análisis discriminante lineal

1) En primer lugar, calcularemos la matriz de dispersión intraclase S_i de la siguiente forma:

$$S_I = \frac{1}{K} \sum_{k=1}^{K} \Sigma_k$$

En el que cada Σ_K es la matriz de covarianza de los datos de cada clase k. Esta covarianza se calcula (de manera similar a cómo hemos visto en el caso del algoritmo del PCA) mediante la fórmula:

$$\Sigma_k = \frac{1}{N_k} X_k X_k^T$$

En el que el subíndice k indica que solo se cogen los elementos de la clase k.

2) Después calcularemos la matriz de dispersión interclase S_E de la siguiente manera:

$$S_E = \frac{1}{K}\sum_{k=1}^{K}(\mu_k - \mu_0)(\mu_k - \mu_0)^T$$

Donde μ_k es la media de los datos de la clase k, y μ_0 es la media global de todos los datos de todas las clases.

3) Finalmente, los vectores que nos definen la matriz de proyección y nos extraen las características discriminantes se calculan como los vectores propios de más valor propio de la matriz:

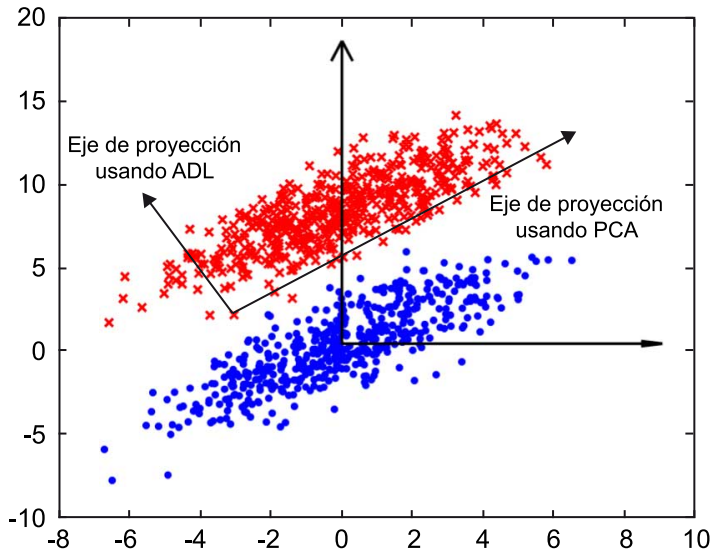
$$eig(S_E S_I^{-1})$$

Es decir, los vectores que maximizan la dispersión entre clases (numerador) manteniendo los elementos de la misma clase tan juntos como sea posible (denominador).

Ejemplo

En este ejemplo, mostramos dos nubes de puntos que pertenecen a dos clases diferentes, y buscamos el resultado de hacer una extracción de características no supervisada basada en PCA y una de supervisada basada en la LDA.

Figura 7. Conjunto de muestras de dos clases diferentes



Si se hacen los cálculos de manera adecuada, observamos que los vectores principales de proyección varían considerablemente. En el primer caso, el algoritmo de análisis de componentes principales ha considerado toda la nube de puntos conjuntamente y ha buscado el eje donde el error cometido sea en el mínimo posible. Sin embargo, este error de reconstrucción no resulta un criterio útil a la hora de clasificar posteriormente los datos del problema, puesto

que el resultado de la proyección nos muestra un espacio donde los puntos están completamente mezclados y será difícil establecer una frontera de separación.

Si observamos el comportamiento del análisis discriminante sobre el mismo problema, podemos ver que el eje que seleccionamos es precisamente el que descarta el PCA. En este eje se comete un error mucho más grande desde el punto de vista de la reconstrucción; en cambio, los datos proyectados se vuelven trivialmente separables.

Hay que hacer notar que este ejemplo sintético se ha generado para mostrar la diferencia entre la extracción de características supervisada y la no supervisada. Puede haber casos en que el análisis discriminante no solucione el problema de manera satisfactoria. De hecho, hay muchas extensiones del algoritmo ADL que permiten tratar datos heteroscedásticos[16] y no separables linealmente[17]. Estudiarlos queda, sin embargo, fuera del alcance de este curso.

[16] Datos que no tienen una distribución gaussiana uniforme en todas las clases.

[17] Las técnicas basadas en núcleos permiten tratar la no linealidad, tal como veremos en apartados posteriores de clasificación.

2.4. Conclusión

En este apartado hemos presentado los pasos que hay que hacer antes de empezar el entrenamiento de los algoritmos de aprendizaje automático. Una vez definidos los pasos necesarios para obtener los datos y definir los tipos de atributos, nos hemos centrado en la selección y la extracción de características. Hemos repasado diferentes métodos de reducción de la dimensionalidad de los datos con objeto de eliminar información redundante y ruido, y preservar solo las características útiles en procesos de aprendizaje posteriores. Los métodos que hemos visto se pueden considerar como un paso previo al aprendizaje automático, y dependerán mucho de la función objetivo que queremos maximizar.

Como hemos visto, no hay ningún método que sea universalmente mejor que otro, ni en selección de características ni en extracción. La dimensionalidad y la complejidad de los datos de entrada y la naturaleza del problema que tenemos que tratar nos pueden aconsejar usar una técnica u otra.

3. Aprendizaje no supervisado

Tal y como se ha explicado en la introducción, el aprendizaje no supervisado corresponde al caso en que disponemos de un conjunto de datos para los cuales no se conocen las variables de salida. En este caso, los métodos de aprendizaje se aplican a los datos para extraer información (conocimiento) útil.

Hay diferentes familias de métodos que pertenecen a los métodos de aprendizaje no supervisado. Algunos de los métodos son los que obtienen reglas de asociación o definen categorías. En este apartado nos centramos en los algoritmos de categorización, que pertenecen a la etapa de aprendizaje del sistema. En etapas previas, como la de extracción de características, ya hemos visto técnicas de aprendizaje no supervisado como, por ejemplo, el análisis de componentes principales (PCA).

3.1. Algoritmos de categorización

Los algoritmos de categorización[18] son uno de los métodos de aprendizaje no supervisado (y de análisis de datos) más utilizados. A partir de un conjunto de datos para una serie de objetos (datos descritos habitualmente en forma tabular), estos métodos construyen un conjunto de categorías. Cada categoría representa un subconjunto de los objetos iniciales que son, por un lado, bastante parecidos entre sí para definir una única categoría, pero, por otro lado, bastante diferentes del resto para constituir una categoría propia.

(18) En inglés, *clustering*

Las diferencias entre los métodos de categorización corresponden a diferencias en la manera de describir los datos iniciales a partir de los cuales queremos construir las categorías (los objetos se describen, por ejemplo, mediante datos numéricos o simbólicos), en cómo formalizamos los conceptos de *parecido* y *diferente* en la estructura permitida para representar el conjunto de categorías (podemos usar, por ejemplo, particiones, particiones difusas o árboles jerárquicos), en la descripción de estas categorías (algunas de las descripciones pueden ser extensionales, intermediando centroides –objeto mediano que representa los objetos que definen la categoría–, o bien mediante expresiones lógicas), etc.

Cluster analysis

A menudo también se utiliza el término *cluster analysis* para denotar los algoritmos de categorización.

Muchos algoritmos de categorización se basan en la idea de optimizar una función objetivo. Son los llamados *métodos de categorización óptima*. En este caso, el problema se puede definir como la minimización de una función objetivo (FO). A continuación repasamos uno de los métodos de categorización

más utilizados, el *k-means*. Es un método de categorización óptima en que el término *k* en el nombre del algoritmo corresponde al número de categorías que se construyen.

3.1.1. El algoritmo *k-means*

El algoritmo *k-means* pretende encontrar una partición C de k elementos $C=\{C_1,\ldots,C_k\}$ del conjunto de objetos X (es decir, $\cup C_i = X$, $C_i \cap C_j = \varnothing$ para todo $i \neq j$) de forma que se minimice la expresión siguiente:

$$\sum_{i=1}^{k}\sum_{x\in C_i} \| A(x) - \mu_i \|^2 \qquad (1)$$

Norma

En esta expresión ||u|| corresponde a la norma del vector. Así, si $u = (u_1, \ldots, u_r)$ entonces $\sqrt{u_1 \cdot u_1 + u_2 \cdot u_2 + \ldots + u_r \cdot u_r}$

Aquí, cada C_i corresponde a una categoría diferente, $A(x)$ corresponde a los atributos A del elemento x y μ_i se interpreta como el prototipo (o centroide) de la categoría C_i (podemos entender $\mu_i = A(C_i)$).

La función 1 formula el problema de cómo la distancia entre los atributos A de los elementos del conjunto de objetos X respecto a una serie de prototipo μ_i. Los prototipos μ_i tienen el mismo número de dimensiones que los atributos $A(x)$ y resumen las características de los elementos que la componen.

Podemos reescribir la función 1 en una función objetivo FO que se tiene que minimizar como:

$$FO(w,\mu) = \sum_{i=1}^{k}\sum_{m=1}^{M} w_{im} \| A(x_m) - \mu_i \|^2 \qquad (2)$$

dónde $w_{im} = 1$ si el vector de características $A(x_m)$ pertenece al cluster C_i; de lo contrario, $w_{im} = 0$. M denota el número total de elementos en el conjunto de objetos X.

El conjunto de valores w_{im} definen la partición $C=\{C_1,\ldots,C_k\}$ relacionando cada elemento x_m a un único cluster C_i de la partición C.

Con la notación introducida, podemos expresar el problema como:

$$\text{minimizar } FO(w,\mu) = \sum_{i=1}^{k}\sum_{m=1}^{M} w_{im} \| A(x_m) - \mu_i \|^2 \qquad (3)$$

sujeto a

$$w_{im} = \begin{cases} 1 & \text{si } i = \arg\min_j \| A(x_m) - \mu_j \|^2 \\ 0 & \text{altramente.} \end{cases} \qquad (4)$$

El resultado de esta minimización consistirá en un grupo de k centroides μ_i que resumirán las k categorías de la partición y el valor de los w_{im} que asociarán cada valor x con uno de estos centroides y, por lo tanto, con su correspondiente categoría.

A continuación, presentamos un algoritmo iterativo para encontrar el mínimo de esta expresión. El algoritmo, sin embargo, no asegura un mínimo global, sino que solo asegura convergencia (porque a cada paso se reduce el valor de la función objetivo) y llegar a un mínimo local.

El algoritmo sigue un esquema iterativo en el que en cada iteración se busca, en primer lugar, la mejor partición (asignación de los elementos a las categorías) a partir de unos centroides μ_i y, a continuación, se vuelven a calcular los centroides a partir de la nueva partición construida. De acuerdo con esto, para empezar, el algoritmo necesita una partición inicial o unos centroides a partir de los cuales poder calcularla. El algoritmo acaba cuando no hay ninguna modificación de la partición (ni, por lo tanto, de los centroides).

Veamos a continuación el algoritmo más en detalle:

1) Definir una partición inicial y calcular los centroides μ_i

2) Resolver iterativamente:

a) Asignar los ítems al centroide más próximo:

$$w_{im}=\begin{cases}1 & \text{si } i=\arg\min_j \| A(x_m)-\mu_j \|^2 \\ 0 & \text{altramente.}\end{cases} \qquad (5)$$

b) Recalcular de nuevo los centroides. Definimos el prototipo μ_i como la media de los elementos asignados al cluster C_i. Así:

$$\mu_i=\frac{\sum_{m=1}^{M} w_{im}A(x_m)}{\sum_{m=1}^{M} w_{im}} \qquad (6)$$

3) Parar cuando los pasos anteriores sean convergentes y, por lo tanto, las particiones no cambien en cada iteración.

Ved que, aquí, el prototipo es un vector en el espacio de los atributos. Por lo tanto, μ_i tiene un componente para cada atributo y la expresión corresponde a una media de vectores.

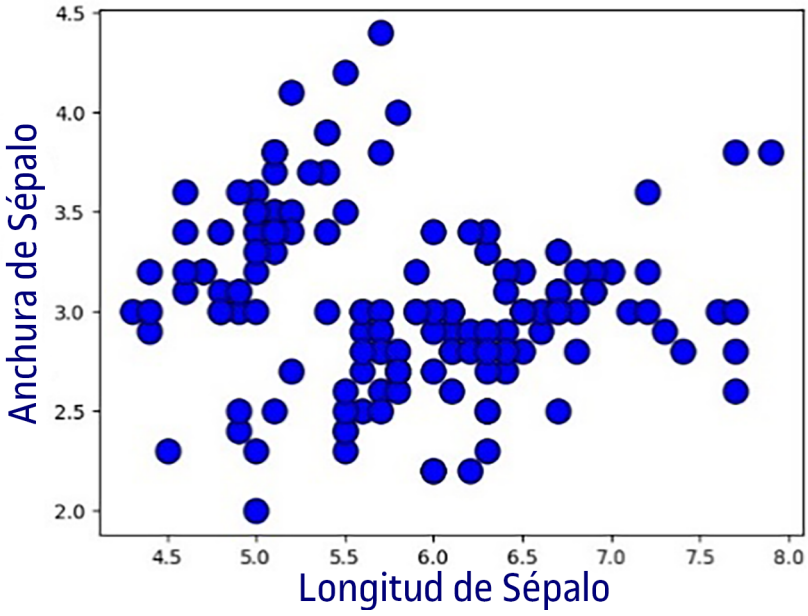
Además de la determinación de las particiones y los centroides que minimizan las expresiones en el paso 2, hay que considerar también el paso 1. Es decir, cómo definir la inicialización del problema. Una alternativa es construir una partición cualquiera (por ejemplo, de manera aleatoria) a partir de la cual encontrar los centroides. Una vez tenemos estos centroides podemos empezar a iterar en el paso 2. Otra alternativa es definir los centroides de cada categoría a partir de los objetos. La selección de un objeto para cada categoría se puede definir de una manera aleatoria. A veces, es más conveniente empezar con la asignación de objetos a centroides, porque la definición de las particiones puede provocar que, después de algunas iteraciones, queden en el paso 2 categorías sin ningún objeto asignado. Si empezamos con la selección aleatoria de k objetos que hacen de centroides iniciales, el algoritmo presentado cambia ligeramente. En el paso 1, en lugar de calcular el centroide a partir de una partición inicial aleatoria, lo que haremos es calcular la partición inicial a partir de los centroides aleatorios (asignando cada ítem al centroide más próximo). Entonces, en el paso 2, es necesario intercambiar el orden de las operaciones. Por lo tanto, primero habrá que recalcular los centroides y después habrá que asignar los ítems al centroide más próximo.

La condición de convergencia del algoritmo consistirá en ver si en dos iteraciones sucesivas las particiones que se consiguen son iguales. En algunos casos esta condición de convergencia puede ser difícil de lograr y se utiliza como criterio de finalización del algoritmo un número máximo de iteraciones preestablecido.

Ejemplo

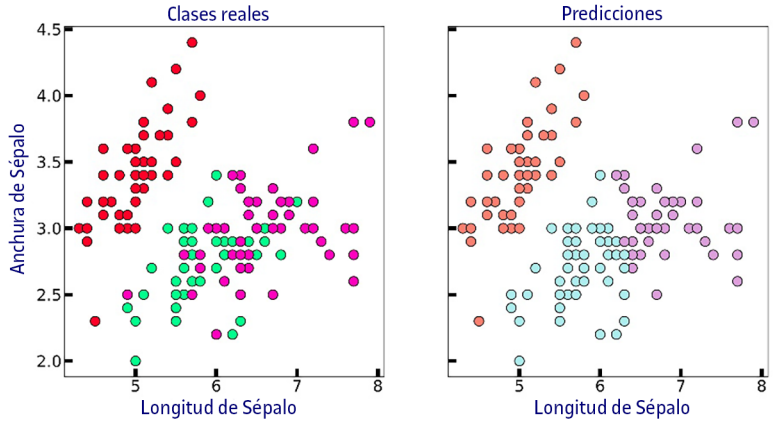
En este ejemplo usaremos el mismo conjunto de datos que en el ejemplo de problema de clasificación que hemos visto en el subapartado 1.1, el conjunto de datos IRIS. Sin embargo, en este caso, para simular una tarea de categorización utilizando aprendizaje no supervisado, descartamos la información de la clase (tipo de planta) de cada ejemplo, de forma que nuestras muestras consisten solo en atributos numéricos. Para poder visualizar los datos en dos dimensiones, utilizaremos solo dos atributos: la anchura del sépalo y la longitud del sépalo. La figura siguiente muestra nuestro conjunto de datos en este espacio bidimensional.

Figura 8. Visualización del conjunto de datos IRIS en el espacio bidimensional definido por los atributos *anchura de sépalo* y *longitud de sépalo*.



Ahora, ajustamos nuestro modelo de categorización utilizando el algoritmo *k-means* con un valor de $k = 3$. La figura 9 muestra con diferentes colores las tres clases reales del conjunto de datos y las compara con las tres categorías que ha aprendido nuestro modelo (predicción). Tal como se puede apreciar, el algoritmo *k-means* ha encontrado una partición de los datos que se ajusta bastante bien a las clases de plantas originales, pero recordad que el algoritmo no ha usado la información de clase en ningún momento.

Figura 9. Visualización del conjunto de datos IRIS con las tres clases reales (izquierda) y las tres categorías que ha aprendido nuestro modelo (derecha).



4. Conceptos generales

Acabaremos este módulo con un apartado dedicado a formalizar algunos conceptos generales del aprendizaje computacional. La mayoría de estos conceptos ya se han visto anteriormente en otras asignaturas, o incluso en apartados anteriores de este mismo módulo. Aun así, consideramos importante repasarlos aquí porque son conceptos que utilizaremos repetidamente a lo largo del curso.

4.1. Partición de datos

Ya hemos mencionado que lo primero que necesita tener un sistema de aprendizaje es un conjunto de datos de entrenamiento, puesto que es precisamente de donde los algoritmos de aprendizaje extraen la experiencia necesaria para aprender a realizar ciertas tareas.

También hemos visto en el subapartado 1.1 que, una vez tenemos un modelo que se ha ajustado (o entrenado) en este conjunto de datos de entrenamiento, nos interesará evaluar su capacidad de generalización. Es decir, necesitaremos calcular alguna medida de su rendimiento en un conjunto de datos diferente del conjunto de datos de entrenamiento.

Una manera típica de enfocar el proceso de medir la calidad de un modelo de aprendizaje computacional consiste en separar el conjunto de datos disponibles en dos grupos:

- El conjunto de entrenamiento: conjunto de los datos que realmente se utilizan para construir el modelo.

- El conjunto de prueba: conjunto de los datos que se utilizan para ver si el modelo actúa correctamente.

A la hora de hacer esta partición de los datos, debemos tener en cuenta una serie de consideraciones importantes. Antes que nada, es importante asegurarse de que los dos conjuntos de datos son disjuntos, puesto que lo que interesa precisamente es medir la calidad del modelo en muestras que no ha visto anteriormente (generalización). Por lo tanto, deberemos ir con mucho cuidado para no tener datos duplicados en los conjuntos de entrenamiento y prueba.

En segundo lugar, la partición de los datos disponibles para una determinada tarea se tiene que mantener fija para poder comparar el rendimiento de modelos diferentes de una manera justa. Obviamente, no sería justo comparar un modelo que se ha entrenado con un 50 % de los datos con otro que se ha entrenado con el 90 % de los datos disponibles, puesto que el segundo ha

tenido acceso a más información. Del mismo modo, comparar el rendimiento de dos modelos que se han evaluado en dos conjuntos de prueba diferentes (aunque tengan la misma medida) no nos permite extraer conclusiones sobre cuál de los dos generaliza mejor.

Finalmente, debemos tener en cuenta que el número de ejemplos en el conjunto de prueba debe ser lo suficientemente grande para que la medida de calidad de un modelo sea significativa. Por ejemplo, si medimos el rendimiento de un clasificador en un conjunto de solo cinco muestras elegidas aleatoriamente de entre todos los datos disponibles, tendremos una medida de calidad que variará mucho en función de cuáles fueran estas cinco muestras. Por lo tanto, cuanto mayor sea, dentro de lo posible, el conjunto de prueba, más fiable será la medida de calidad de nuestro modelo.

En caso de que el conjunto de datos disponibles sea pequeño, puede ser útil usar la validación cruzada, que consiste en repetir el proceso de entrenamiento y evaluación de la calidad del modelo varias veces utilizando diferentes particiones (normalmente aleatorias) para los subconjuntos de entrenamiento y prueba. La calidad del modelo se calcula entonces como la media de las medidas de calidad obtenidas en las diferentes evaluaciones.

4.1.1. Hiperparámetros y conjunto de validación

Casi todos los algoritmos de aprendizaje computacional tienen hiperparámetros: parámetros que modifican el comportamiento del algoritmo pero que no se ajustan de manera automática mediante el proceso de aprendizaje. Normalmente, los valores de estos hiperparámetros se tienen que ajustar de forma manual antes de ejecutar el algoritmo de aprendizaje. Por ejemplo, en el caso de los árboles de decisión, un hiperparámetro es el número máximo de niveles que queremos permitir en la construcción del árbol; otro ejemplo es el valor de k en el algoritmo *k-means*.

Como ya hemos mencionado anteriormente, es muy importante que los ejemplos del conjunto de prueba no se utilicen de ninguna forma durante el proceso de entrenamiento de un modelo, y esto incluye también el ajuste manual de los valores de los hiperparámetros. Dicho de otro modo, no está permitido utilizar el conjunto de prueba para encontrar los mejores valores para los hiperparámetros de un modelo, puesto que se considera que, al hacerlo, la medida de calidad (generalización) del modelo deja de ser fiable o realista.

Una práctica común para ajustar los valores de los hiperparámetros es utilizar un subconjunto de los datos de entrenamiento como conjunto de validación. El conjunto de datos de validación viene a ser, pues, como un conjunto de prueba pero que se utiliza únicamente para validar los hiperparámetros del modelo. Una vez se han encontrado los mejores valores para los hiperparáme-

tros utilizando este conjunto de validación, el modelo se vuelve a entrenar con todos los datos de entrenamiento (incluyendo los que se han utilizado como conjunto de validación) y se evalúa en el conjunto de prueba.

4.2. Métricas de evaluación

Una vez tenemos un modelo que ha sido ajustado a nuestro conjunto de datos de entrenamiento, necesitaremos evaluar la capacidad de generalización que tiene en el conjunto de datos de prueba.

4.2.1. Métricas de evaluación para modelos de clasificación

La tasa de error es una métrica de evaluación muy común para problemas de clasificación, que consiste en calcular el porcentaje de ejemplos del conjunto de prueba que nuestro modelo no clasifica correctamente. De manera análoga podemos definir la exactitud[19] del modelo como el porcentaje de aciertos en el conjunto de prueba:

(19) En inglés, *accuracy*

$$Acc = \frac{1}{N}\sum_{i=0}^{N} \lambda(h(x_i), c_i)$$

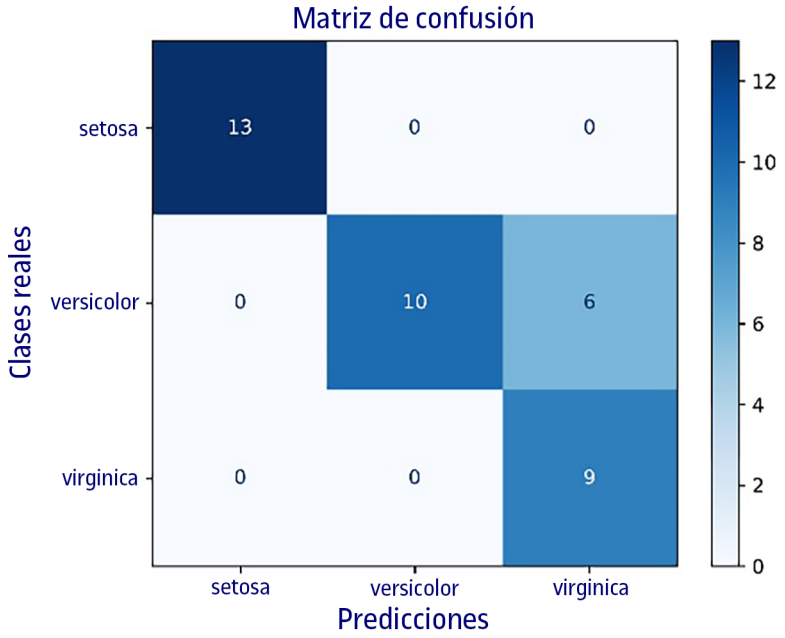
Donde N es el número de ejemplos en nuestro conjunto de prueba, h es nuestro modelo, $h(x_y)$ es la predicción de nuestro modelo para la entrada x_i del conjunto de prueba, c_i es la clase a la cual pertenece la muestra x_i, y λ es una función que devuelve 1 cuando los valores de $h(x_i)$ y c_i son iguales o cero de lo contrario.

Tanto la exactitud como la tasa de error son métricas muy utilizadas para evaluar modelos de clasificación. Una de sus ventajas principales es que resumen el rendimiento de un modelo en un único valor numérico, y por lo tanto hacen muy fácil la comparación entre diferentes modelos. Aun así, esta virtud también puede resultar un inconveniente en algunos casos, puesto que esta capacidad de resumen nos puede estar escondiendo cierta información relevante. Por ejemplo, en el caso del conjunto de datos IRIS, en que tenemos tres clases, nos puede interesar saber cuál es el rendimiento del modelo en cada una de las clases por separado. Esto puede ser especialmente relevante en casos en que las diferentes clases no estén equilibradas en nuestro conjunto de prueba, es decir, cuando tengamos un número diferente de ejemplos de cada clase. Una posible solución que nos puede ser útil en estos casos es calcular la exactitud para cada una de las clases independientemente.

También podemos utilizar una matriz de confusión para representar de forma gráfica el rendimiento del modelo. Una matriz de confusión es una tabla cuadrada con tantas filas y columnas como clases tenemos en nuestro problema de clasificación. Las columnas representan las predicciones del modelo para cada clase, mientras que las filas representan los valores reales de cada clase.

En cada celda de la tabla $c_{i,j}$ escribiremos el número de ejemplos de la clase i que se han clasificado como miembros de la clase j. La siguiente figura muestra un ejemplo de matriz de confusión para un hipotético modelo de clasificación en el conjunto de datos IRIS.

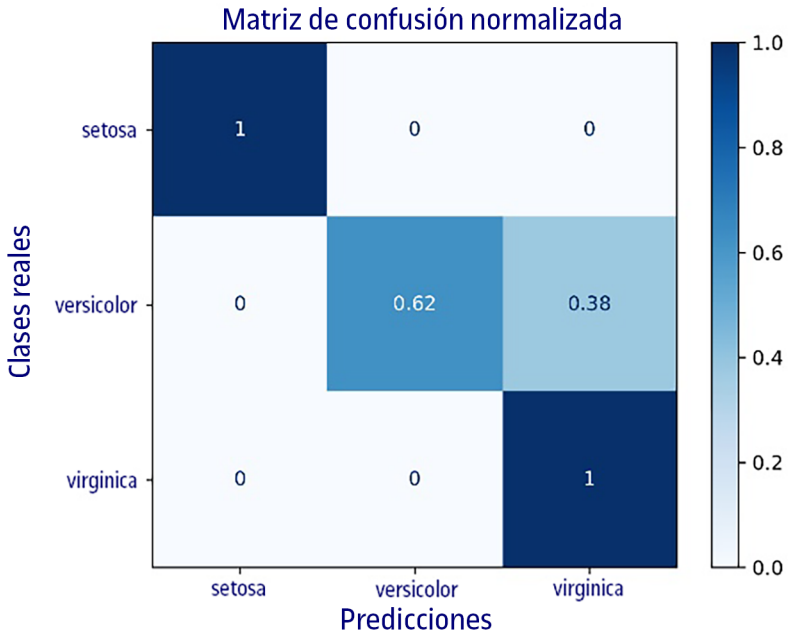
Figura 10. Matriz de confusión para un modelo de clasificación en el conjunto de datos IRIS



Fijaos que, utilizando una matriz de confusión, además de saber que de los 16 ejemplos de *Iris versicolor* que hay en el conjunto de prueba nuestro modelo ha clasificado correctamente 10 (esto es un 0.625 de exactitud), los 6 restantes los ha confundido (de ahí el nombre) con la clase *Iris virginica*.

Las matrices de confusión también pueden presentarse con valores normalizados para cada una de las clases. La figura 11 nos muestra la versión normalizada de la matriz de confusión anterior.

Figura 11. Matriz de confusión normalizada para un modelo de clasificación en el conjunto de datos IRIS



Un caso especial de matriz de confusión se da en los problemas de clasificación binaria (dos clases). En este caso se suele asignar a una de las clases el término *positivo* y al otro el término *negativo*. Pensad por ejemplo en un sistema de clasificación para pacientes de un hospital que determina si el paciente tiene una dolencia (positivo) o no (negativo) a partir de ciertos datos del paciente. Como se puede ver en la tabla siguiente, cada una de las celdas de la matriz de confusión en un problema de clasificación binario tiene nombre propio:

Tabla 2. Nomenclatura de las celdas en una matriz de confusión en un problema de clasificación con dos clases: P (positivo) y N (negativo)

		Predicció	
		T	N
Classes reals	T	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

A partir de estos conceptos, se definen la precisión (P) y la sensibilidad[20] (R) de un modelo de la manera siguiente:

[20] En inglés, *recall*

$$P = \frac{TP}{TP+FP} \qquad R = \frac{TP}{TP+FN}$$

Es decir, la precisión de un modelo es el número de ejemplos de la clase positiva que se han clasificado correctamente dividido por el número total de muestras que el modelo ha predicho como clase positiva; mientras que la sensibilidad del modelo es el número de ejemplos de la clase positiva que se han clasificado correctamente dividido por el número total de muestras de la clase positiva en el conjunto de datos de prueba.

4.2.2. Métricas de evaluación para modelos de regresión

Hasta ahora, solo hemos visto métricas de evaluación para modelos de clasificación, pero obviamente también necesitamos métricas para evaluar modelos de regresión. Como hemos visto anteriormente, un problema de regresión se diferencia de un problema de clasificación en que la salida del modelo es un valor continuo y no una clase de entre un conjunto discreto de clases posibles. Recordad el ejemplo del modelo para predecir el precio de una casa a partir de sus metros cuadrados.

Un concepto clave a la hora de validar la calidad de un modelo de regresión es el residuo, o error de una muestra, que se define de la manera siguiente:

$$\text{residuo} = y_i - y'_i$$

Donde y'_i es la predicción del modelo para la y-ésima muestra, *e* y_i es valor real (deseado) de la muestra. A partir del residuo, podemos definir el error cuadrático medio (ECM o MSE)[21] de la siguiente manera:

(21) En inglés, *mean squared error*

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N}\left(y_i - y'_i\right)^2$$

Y la raíz del error cuadrático medio (RECM)[22] como:

(22) En inglés, *root mean squared error*

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(y_i - y'_i\right)^2}$$

Donde *N* es el número de muestras en el conjunto de pruebas. Tanto el error cuadrático medio (MSE) como su raíz cuadrada (RMSE) son dos métricas de evaluación muy utilizadas para medir la calidad de los modelos de regresión. Obviamente, en los dos casos, cuanto más pequeño es el error más se ajusta el modelo a los datos. La única diferencia es que el valor del RMSE es probablemente la estadística más fácil de interpretar, puesto que tiene las mismas unidades que la cantidad representada en los datos. Por ejemplo, en el caso del modelo que predice el precio de una casa, el valor del RMSE corresponde a unidades «Euros», mientras que el valor del MSE se corresponde a euros al cuadrado.

4.3. Minimización del error

Otro concepto importante y común en todos los algoritmos de aprendizaje es el de *minimización del error*. De hecho, es el mecanismo principal por el cual los algoritmos son capaces de aprender a partir de los datos de entrenamiento. Lo hemos visto cuando hemos presentado el algoritmo PCA, que consiste esencialmente en minimizar el error de reconstrucción de los datos; y también cuando hemos introducido el algoritmo *k-means*, donde se minimiza la suma de los cuadrados de las diferencias entre cada una de las muestras y la mediana (centroide) dentro de las *k* categorías. También en el caso de los algoritmos para construir árboles de decisión se intenta minimizar, a pesar de que no de forma tan directa, el error de clasificación en el conjunto de datos de entrenamiento.

En muchos casos, la función de error que minimiza el algoritmo es la misma métrica de evaluación con que evaluaremos el modelo. Pensad, por ejemplo, en el caso de la regresión lineal, donde el error que se minimiza es el MSE que hemos introducido en el subapartado anterior como métrica de evaluación para modelos de regresión. Pero esto no tiene por qué ser siempre así. Puede haber modelos que por conveniencia minimicen una función de error diferente de la métrica de evaluación que se utiliza para evaluar el modelo. Un ejemplo de esto son las redes neuronales para clasificación, donde se minimiza la entropía cruzada[23], en vez de la tasa de error de clasificación, por sus propiedades de derivabilidad.

(23) En inglés, *cross entropy*

La intuición clave que hay que adquirir en cuanto al concepto de *minimización del error* es que si los datos de prueba y de entrenamiento provienen de un mismo proceso generador de datos (la función que queremos modelar y no conocemos), un modelo que minimice el error en los datos de entrenamiento podrá generalizar su comportamiento a los datos de prueba. Es importante tener en cuenta que para hacer esta afirmación estamos asumiendo que las muestras de nuestro conjunto de datos son independientes e idénticamente distribuidas (i. i. d.). Es decir, que cada muestra es independiente del resto, y que los conjuntos de entrenamiento y prueba tienen la misma distribución de probabilidad.

Cuando utilizamos un algoritmo de aprendizaje, hacemos uso del conjunto de datos de entrenamiento para ajustar los parámetros de nuestro modelo, de forma que se minimice el error en estos datos de entrenamiento. Por lo tanto, los factores que determinarán cuanto de bien generalizará el modelo son, por un lado, la capacidad del algoritmo para minimizar el error en el conjunto de entrenamiento y, por el otro, la diferencia que hay entre los conjuntos de datos de entrenamiento y prueba. Estos dos factores nos llevan a introducir dos conceptos fundamentales del aprendizaje computacional: la sobreespecialización o sobreajuste[24] y la subespecialización[25]. La sobreespecialización se produce cuando un modelo se ajusta muy bien a los datos de entrenamiento (con

(24) En inglés, *overfitting*

(25) En inglés, *underfitting*

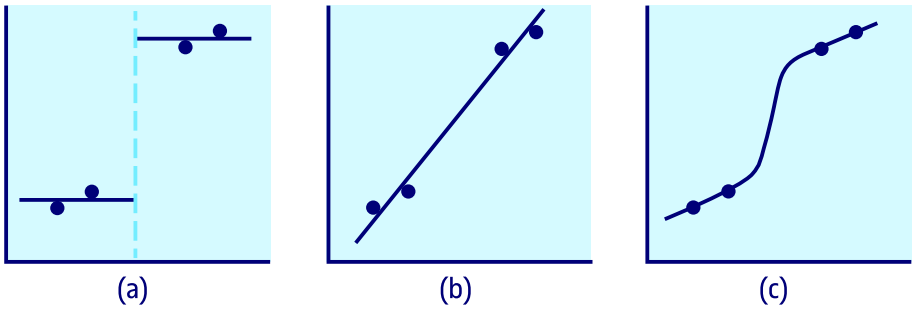
un error muy pequeño) pero no generaliza bien en los datos de prueba. En cambio, la subespecialización se produce cuando el algoritmo no es capaz de minimizar bastante el error en el conjunto de datos de entrenamiento. En el siguiente subapartado formalizaremos estos conceptos desde el punto de vista del análisis estadístico.

4.4. El sesgo y la varianza

La construcción de modelos mediante métodos de aprendizaje inductivo (utilizando el aprendizaje para clasificar o hacer regresión) requiere la capacidad de generalización a partir de los ejemplos de que se dispone. Así, por ejemplo, cuando se construye una regla de clasificación para pacientes con una determinada dolencia a partir de ejemplos, nunca tendremos ejemplos de todos los enfermos posibles, sino solo de unos cuantos. Aun así, a partir de estos ejemplos se tiene que construir una regla que clasifique no solamente a los que ya teníamos, sino también a todos los demás. Por lo tanto, el algoritmo de aprendizaje correspondiente tiene que ser capaz de generalizar para que la regla que se construya pueda tratar todos los casos de manera correcta.

En la figura 12 se ilustra el problema de la generalización y se consideran problemas con una variable de entrada y una variable de salida. La figura muestra cuatro puntos y tres generalizaciones posibles.

Figura 12. Tres generalizaciones posibles para un conjunto de cuatro puntos



En la figura hay tres posibles generalizaciones para un conjunto de cuatro puntos. La generalización aproxima los puntos (a) considerando solo rectas horizontales (con discontinuidades), la (b) usa rectas y la (c) permite considerar polinomios de grado tres. Estas consideraciones sobre el modelo corresponden al sesgo.

La generalización plantea dificultades porque a partir de un conjunto de ejemplos hay muchas generalizaciones posibles (y no todas darán el resultado correcto para los casos no incluidos en el conjunto de entrenamiento). Todos los

(26) En inglés, *inductive bias*

algoritmos incorporan de alguna manera elementos que restringen el espacio de búsqueda de las soluciones (el espacio de generalizaciones posibles). Todo aquello que restringe el espacio de búsqueda es lo que se denomina *sesgo inductivo*[26].

Nota

La limitación en el aprendizaje que provoca el sesgo encaja con la traducción de la palabra inglesa *bias* por *predisposición* (tal como aparece en la versión castellana del libro *Artificial Intelligence*, de E. Rich y K. Knight). Así, el sistema tiene predisposición para aprender ciertas cosas y no para aprender otras.

De acuerdo con lo que se ha dicho, el sesgo permite construir generalizaciones sin estar desbordado por la medida del espacio de la búsqueda. Aun así, el sesgo provoca inconvenientes porque puede darse el caso de que generalizaciones interesantes no se puedan aprender porque el conocimiento necesario no está incorporado al sistema. Así, por ejemplo, en la figura anterior puede ser interesante considerar polinomios de grado tres. En caso de que solo se permitieran rectas, no podríamos construir el modelo de la figura (c).

Recordad que la **esperanza matemática** (o **media**) de una variable X con probabilidad (función de densidad) p es:

$$E(X) = \sum_{x \in X} x \cdot p(x)$$

O bien, si los valores de la variable no están agrupados y N es el número de valores que se consideran en X, es:

$$E(X) = \frac{1}{N} \sum_{x \in X} x$$

La **varianza** de una variable X, cuando los valores no están agrupados, es:

$$s^2 = \frac{1}{N} \sum_{x \in X} (x_i - E(X))^2$$

que, para aquellos casos con $E(X) = 0$, corresponde, evidentemente, a:

$$s^2 = \frac{1}{N} \sum_{x \in X} x_i^2$$

En los métodos de aprendizaje para problemas de regresión, el sesgo se formaliza teniendo en cuenta la diferencia entre el valor estimado por nuestro modelo (si x_0 es el valor de entrada y M nuestro modelo, usamos $M(x_0)$ para denotar el valor estimado) y el valor real (el valor de la función que modelizamos, que denotamos f, aplicada al mismo valor x_0):

$$\text{Sesgo}\,(M(x_0)) = E(M(x_0)) - f(x_0)$$

En esta expresión, $E(M(x_0))$ corresponde a la esperanza de nuestro modelo para x_0 cuando consideramos diferentes conjuntos de ejemplos. De manera informal, podemos decir que si disponemos de cinco conjuntos de ejemplos C =

$\{C_1, C_2, C_3, C_4, C_5\}$, $E(M(x_0))$ corresponde a la media del resultado de construir cinco modelos diferentes M_{C1}, M_{C2}, M_{C3}, M_{C4} y M_{C5}, cada uno definido a partir de uno de los conjuntos de ejemplos: $E(M(x_0)) = (M_{C1}(x_0) + M_{C2}(x_0) + M_{C3}(x_0) + M_{C4}(x_0) + M_{C5}(x_0))/5$.

Evidentemente, tendremos que el método no tiene sesgo cuando la esperanza del modelo construido corresponde perfectamente a la función f en el punto x_0. En cambio, el método tiene sesgo si la esperanza del modelo es diferente de $f(x_0)$. Es importante ver que el hecho de no tener sesgo no quiere decir que el modelo que se construye a partir de un conjunto de ejemplos concretos aproxime bien $f(x_0)$. En el ejemplo podemos tener que $E(M(x_0)) = f(x_0)$, pero $M_{Ci}(x_0) \neq f(x_0)$ para todo M_{Ci}. Un aspecto importante de un método de aprendizaje para asegurar que aproxima bien las funciones es que sea consistente. Se dice que un método es consistente si al incrementar la medida del conjunto de ejemplos, tenemos que el modelo se acerca a la función f. Es decir, que cuando $|C_1| < |C_2| < |C_3| <...$ tenemos que para valores de γ crecientes $M_{Ci}(x_0)$ tiende hacia $f(x_0)$. Más formalmente, cuando se incrementa la medida de los ejemplos, la esperanza tiende a $f(x_0)$.

Podemos subrayar que el sesgo inductivo definido más arriba puede provocar, cuando reduce el espacio de búsqueda, que la esperanza del valor del modelo por x_0 se aleje de $f(x_0)$, e incremente el sesgo $(M(x_0))$.

El sesgo no es el único elemento que hay que tener en cuenta a la hora de evaluar un método de aprendizaje. Otro aspecto importante es la varianza que, como se sabe, corresponde a conocer la posible dispersión de los valores alrededor de la esperanza. Así, si por un modelo tenemos que para un valor x_0 se cumple que $E(M(x_0))$ es igual a $f(x_0)$, pero la varianza es muy grande, podemos tener que para el conjunto de ejemplos que tenemos en la práctica el valor obtenido de $M(x_0)$ sea muy diferente de $f(x_0)$. Por lo tanto, es importante conocer la varianza. La varianza del modelo se define como:

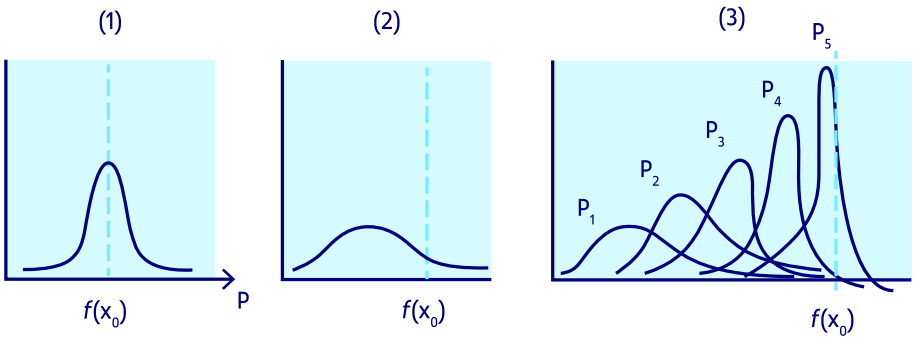
$$\text{Varianza } (M(x_0)) = E(\left[M(x_0) - E(M(x_0))\right]^2)$$

Cuando consideramos varios métodos de aprendizaje, entre los que tienen el mismo sesgo siempre interesará el que tenga una varianza más baja.

En la figura 13 se da una representación gráfica de diferentes distribuciones de probabilidad por modelos que solo dependen de un único parámetro p continuo. La distribución corresponde al valor del modelo aplicado a x_0. La primera figura corresponde a un caso de método no sesgado con varianza pequeña, mientras que la segunda corresponde a un método sesgado con varianza grande. La tercera figura representa diferentes distribuciones, cada una construida con un conjunto de ejemplos más grande que el anterior. Así, la distribución P_1 corresponde a la de los modelos con número de ejemplos igual a NE_1; P_2

a una distribución por modelos con NE_2 ejemplos, y P_i con NE_i ejemplos. Teniendo en cuenta que $NE_1 < NE_2 < NE_3 < NE_4 < NE_5$, el resultado es que la figura corresponde a un método de aprendizaje consistente porque la esperanza del modelo tiende a $f(x_0)$.

Figura 13. Distribuciones de probabilidad para métodos de aprendizaje



4.4.1. Compromiso entre el sesgo y la varianza

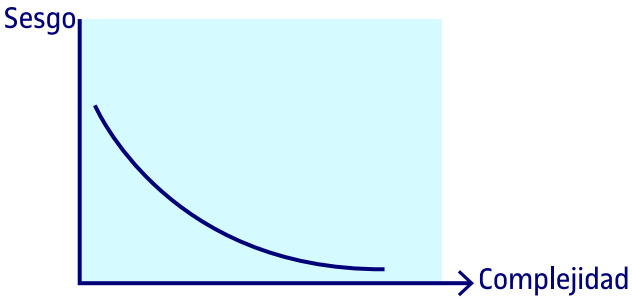
En la construcción de modelos, el sesgo decrece a medida que el modelo se complica.

Así, el sesgo sigue típicamente una curva como la de la figura 14. Con más parámetros, el modelo se va ajustando cada vez más a los ejemplos (y a los errores que los ejemplos incorporan).

Nota

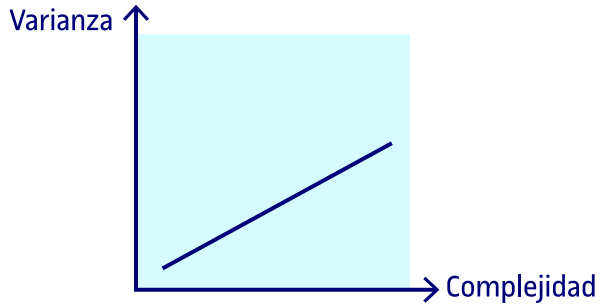
Generalmente, un modelo más complejo corresponde a un número de parámetros más elevado o a un rango más grande por los valores de los parámetros.

Figura 14. Sesgo en relación con la complejidad del modelo



La varianza, en cambio, sigue una curva como la de la figura 15. Es decir, crece la varianza al incrementar la complejidad. Esto es debido a que pequeños cambios en los ejemplos provocan grandes cambios en los modelos.

Figura 15. Varianza en relación con la complejidad del modelo

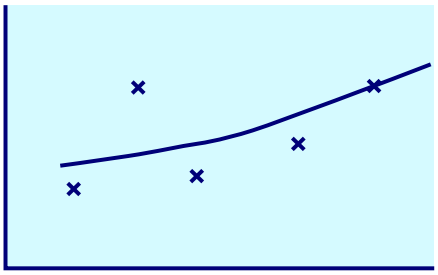


Este compromiso entre la varianza y el sesgo se puede ver cuando utilizamos polinomios para aproximar un conjunto de puntos. Es sabido que un conjunto de n puntos puede ser aproximado mediante una regresión de un polinomio de grado $m < n$. Cuanto mayor es el grado del polinomio (y más grande el número de parámetros, recordemos que un polinomio de grado m tiene $m + 1$ parámetros para determinar), más pequeña es la diferencia entre el valor estimado y el valor original por los puntos que tenemos. En este sentido el valor óptimo se encuentra cuando $m = n$ porque todos los valores son aproximados con error cero. Así, en este caso, cuando crece el grado y, por lo tanto, el número de parámetros, el sesgo tiende a cero (el valor estimado tiende al valor correcto).

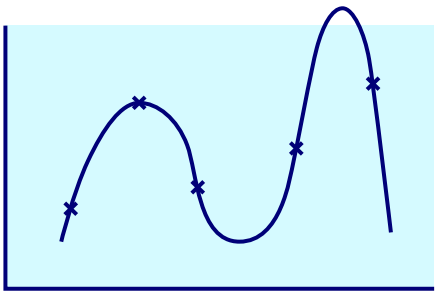
Aun así, la varianza crece con la complejidad del modelo, porque a medida que crece el grado del polinomio, variaciones muy pequeñas en la posición de los puntos pueden provocar variaciones del modelo muy grandes. Así, si a partir de doce puntos en un plano aproximamos la posición de una recta, el cambio de un punto no provocará cambios muy grandes en la posición de la recta. En cambio, si la aproximación es con un polinomio de grado once, el cambio de un punto puede provocar que la nueva curva sea completamente diferente de la anterior.

Figura 16. (1) Sesgo grande y varianza pequeña; (2) sesgo pequeño y varianza grande

(1) Sesgo grande y varianza pequeña



(2) Sesgo pequeño y varianza grande

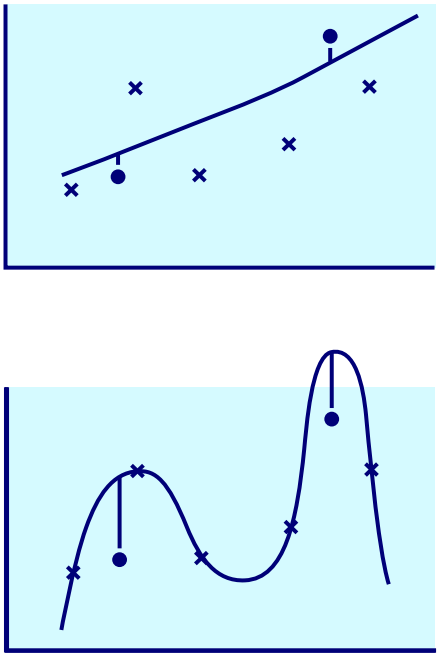


Por lo tanto, a pesar de que a primera vista puede parecer que si disponemos de unos ejemplos tenemos que preferir un sesgo pequeño a una varianza pequeña, esto no es así. Sesgo pequeño con varianza grande corresponde a situaciones en que el modelo está ajustado en exceso a los datos[27]. Por lo tanto, el modelo ha aprendido, incluso, los errores que pueda haber en los ejemplos y no tiene capacidad de generalización. Esto es el caso descrito más arriba del polinomio de grado once. La consideración de ejemplos adicionales para poner a prueba los modelos permite ilustrar este proceso. Si consideramos unos puntos adicionales en la figura 16 de los cuales conocemos el resultado, podemos ver que el modelo con menos parámetros puede aproximar mejor que el modelo ajustado en exceso (ved la figura 17).

(27) En inglés, *overfitting*

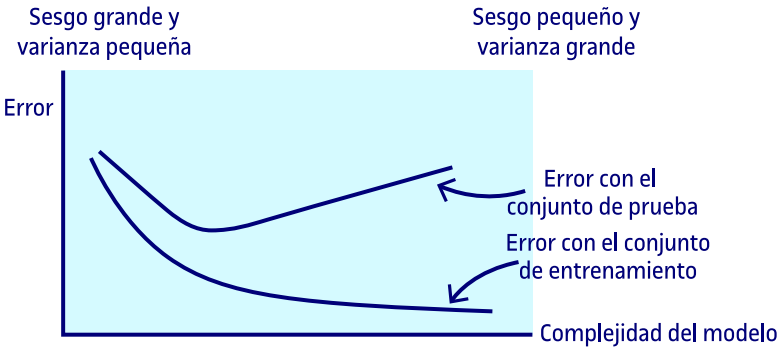
Figura 17. Consideración de ejemplos de prueba adicionales para evaluar los modelos. Los ejemplos de prueba se marcan con puntos (círculos) negros, mientras que los ejemplos de entrenamiento se marcan con cruces. El modelo con menos parámetros

(arriba) generaliza mejor que el modelo ajustado en exceso a los ejemplos de entrenamiento (abajo).



En general, la consideración de conjuntos de ejemplos para test de modelos provoca resultados como los de la figura 18. Es decir, con modelos más complejos, el error disminuye para el conjunto de ejemplos que se han utilizado para construir el modelo (el conjunto de datos de entrenamiento). Aun así, para el conjunto de ejemplos de prueba, el error solo disminuye hasta un cierto punto (cuando hay un buen compromiso entre el sesgo y la varianza, que corresponde a un buen grado de generalización), pero a partir de este momento el error aumenta (debido al exceso en el ajuste de los datos). Aun así, la elección de la complejidad adecuada para un modelo es muy difícil.

Figura 18. Error para el conjunto de entrenamiento y el conjunto de prueba en función de la complejidad del modelo



Glosario

aprendizaje inductivo *loc* Problema de aprendizaje supervisado cuando lo formulamos como el proceso de aprender una función.

aprendizaje no supervisado *loc* Aprendizaje que corresponde a la situación en que no se dispone de información sobre el resultado que tendría que dar el sistema para un ejemplo concreto.

aprendizaje por refuerzo *loc* Aprendizaje en que el conocimiento sobre la calidad del sistema solo es parcial. No se dispone del valor que corresponde a la salida para un determinado conjunto de valores de entrada, sino solo una gratificación o penalización según el resultado que ha dado el sistema.

aprendizaje supervisado *loc* Aprendizaje que corresponde a la situación en que hay un conocimiento completo sobre cuál es la respuesta que se debe dar en una determinada situación.

capacidad de un modelo *loc* Complejidad del modelo, que expresa cuán compleja puede ser la función que puede llegar a modelar.

conjunto de entrenamiento *loc* Conjunto de ejemplos que se usan en el aprendizaje.

conjunto de prueba *loc* Conjunto de ejemplos que se usan para medir la calidad (capacidad de generalización) de un modelo previamente entrenado.

conjunto de validación *loc* Conjunto de ejemplos que se usan para ajustar los hiperparámetros de un modelo.

ejemplo *m* Unidad (objeto) que permite aprender.
sin. **instancia**

generalización *f* Medida de la precisión con que un modelo es capaz de predecir valores correctos para datos que no se han visto previamente durante el entrenamiento.

hiperparámetro *m* Parámetro cuyo valor se usa para controlar el proceso de aprendizaje y, por lo tanto, no se ajusta de manera automática como el resto de parámetros durante el entrenamiento del modelo.

instancia *f* Véase **ejemplo**.

métrica de evaluación (medida de rendimiento) *loc* Métrica para evaluar (cuantificar) cuán buenos son los resultados de un modelo en un conjunto de datos determinado.

overfitting *m* (sobreajuste) Situación que se produce cuando un modelo se especializa demasiado en hacer predicciones para los datos de entrenamiento; tanto, que no es capaz de hacer predicciones correctas para ejemplos fuera de este conjunto.

problema de clasificación *loc* Problema en el que el atributo solución para los ejemplos corresponde a un valor categórico (no numérico).

problema de regresión *loc* Problema en el que el atributo solución para los ejemplos corresponde a un valor numérico.

underfitting *m* Situación que se produce cuando un modelo no es capaz de ajustarse a los datos de entrenamiento.

Bibliografía

BIBLIOGRAFÍA BÁSICA

Hastie, T.; Tibshirani, R.; Friedman, J. (2001). *The Elements of Statistical Learning. Springer*. Nueva York: Springer.

Langley, P. (1996). *Elements of Machine Learning*. Nueva York: Morgan Kaufmann.

BIBLIOGRAFÍA COMPLEMENTARIA

Burges, C. J. C. (1998). «A Tutorial on Support Vector Machines for Pattern Recognition». A:

Data Mining and Knowledge Discovery (núm. 2, págs. 121-167).

Dietterich, T.G. (1997). «Machine-Learning Research: Four Current Directions». A:*AY Magazine* (págs. 97-136).

Hernández Ovallo, J.; Ramírez Quintana, M. J.; Ferri Ramírez, C. (2004). *Introducción a la minería de datos*. Madrid: Pearson Prentice-Hall.

Miyamoto, S.; Umayahara, K. (2000). «Methods in Hard and Fuzzy Clustering». A: Liu, Z.; Miyamoto, S. (ed.). *Soft Computing and Human-Centered Machines*, Tokio: Springer-Verlag (págs. 85-129).