

CAPÍTULO 4. CÁLCULO NUMÉRICO

dónde es la **tasa de aprendizaje**, un escalar positivo que determina el tamaño del paso. Podemos elegir de varias maneras diferentes. Un enfoque popular es establecer a una pequeña constante. A veces, podemos resolver el tamaño de paso que hace que la derivada direccional desaparezca. Otro enfoque es evaluar $F(x - \eta \nabla F(X))$ para varios valores de η y elija la que resulte en el menor valor de la función objetivo. Esta última estrategia se llama **búsqueda de línea**.

El descenso más pronunciado converge cuando todos los elementos del gradiente son cero (o, en la práctica, muy cerca de cero). En algunos casos, podemos evitar ejecutar este algoritmo iterativo y simplemente saltar directamente al punto crítico resolviendo la ecuación $\nabla F(X) = 0$ para X .

Aunque el descenso de gradiente se limita a la optimización en espacios continuos, el concepto general de hacer repetidamente un pequeño movimiento (que es aproximadamente el mejor movimiento pequeño) hacia mejores configuraciones puede generalizarse a espacios discretos. El ascenso de una función objetivo de parámetros discretos se llama **Montañismo** (Russell y Norvig, 2003).

4.3.1 Más allá del gradiente: matrices jacobianas y hessianas

A veces necesitamos encontrar todas las derivadas parciales de una función cuya entrada y salida son ambos vectores. La matriz que contiene todas estas derivadas parciales se conoce como **matriz jacobiana**. Específicamente, si tenemos una función $F: \mathbb{R}_{\text{metro}} \rightarrow \mathbb{R}_{\text{norte}}$, entonces la matriz jacobiana $J \in \mathbb{R}_{\text{norte} \times \text{metro}}$ de F se define tal que $j_{yo, j} = \frac{\partial F(X)}{\partial X_j}$.

A veces también nos interesa una derivada de una derivada. Esto se conoce como **segunda derivada**. Por ejemplo, para una función $F: \mathbb{R}_{\text{norte}} \rightarrow \mathbb{R}$, la derivada con respecto a X_i de la derivada de F con respecto a X_j se denota como $\frac{\partial^2}{\partial X_i \partial X_j} F$.

En una sola dimensión, podemos denotar $\frac{d}{dx} F$ por $F'(X)$. La segunda derivada dice cómo cambiará la primera derivada a medida que varíamos la entrada. Esto es importante porque nos dice si un paso de gradiente causará una mejora tan grande como esperaríamos basándonos solo en el gradiente. Podemos pensar que la segunda derivada mide **curvatura**. Supongamos que tenemos una función cuadrática (muchas funciones que surgen en la práctica no son cuadráticas pero pueden逼近arse bien como cuadráticas, al menos localmente). Si tal función tiene una segunda derivada de cero, entonces no hay curvatura. Es una línea perfectamente plana, y su valor se puede predecir usando solo el gradiente. Si el gradiente es 1, entonces podemos hacer un paso de tamaño -1 a lo largo del gradiente negativo, y la función de costo disminuirá en -. Si la segunda derivada es negativa, la función se curva hacia abajo, por lo que la función de costo en realidad disminuirá en más de -. Finalmente, si la segunda derivada es positiva, la función se curva hacia arriba, por lo que la función de costo puede disminuir menos de -. Ver

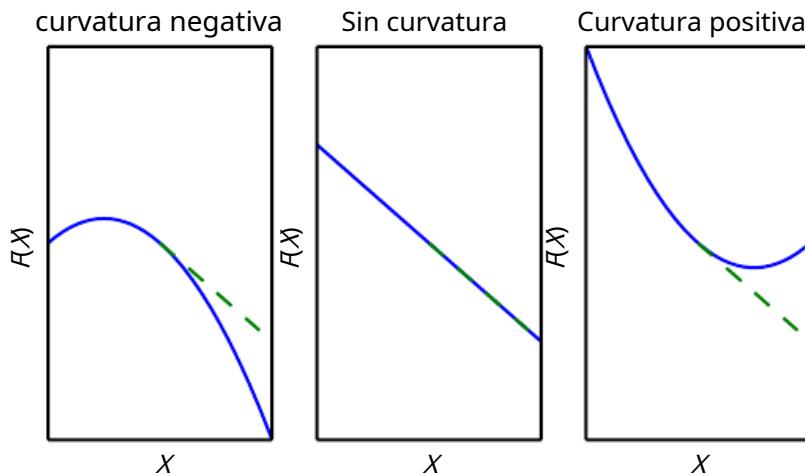


Figura 4.4: La segunda derivada determina la curvatura de una función. Aquí mostramos funciones cuadráticas con varias curvaturas. La línea punteada indica el valor de la función de costo que esperaríamos basándonos solo en la información del gradiente a medida que avanzamos cuesta abajo. En el caso de curvatura negativa, la función de costo en realidad disminuye más rápido de lo que predice el gradiente. En el caso de que no haya curvatura, el gradiente predice correctamente la disminución. En el caso de una curvatura positiva, la función disminuye más lentamente de lo esperado y finalmente comienza a aumentar, por lo que los pasos que son demasiado grandes pueden aumentar la función sin darse cuenta.

cifra 4.4 para ver cómo las diferentes formas de curvatura afectan la relación entre el valor de la función de costo predicha por el gradiente y el valor real.

Cuando nuestra función tiene múltiples dimensiones de entrada, hay muchas segundas derivadas. Estos derivados se pueden reunir en una matriz llamada **matriz Hessiana**. La matriz de Hesse $H(F)(X)$ se define tal que

$$H(F)(X)_{yo,j} = \frac{\partial^2}{\partial x_i \partial x_j} F(X). \quad (4.6)$$

De manera equivalente, el hessiano es el jacobiano del gradiente.

Siempre que las segundas derivadas parciales sean continuas, los operadores diferenciales son comutativos, es decir, su orden puede intercambiarse:

$$\frac{\partial^2}{\partial x_i \partial x_j} F(X) = \frac{\partial^2}{\partial x_j \partial x_i} F(X). \quad (4.7)$$

Esto implica que $H_{yo,j} = H_{ji}$, por lo que la matriz hessiana es simétrica en dichos puntos. La mayoría de las funciones que encontramos en el contexto del aprendizaje profundo tienen un hessiano simétrico en casi todas partes. Como la matriz hessiana es real y simétrica, podemos descomponerla en un conjunto de valores propios reales y una base ortogonal de

vectores propios. La segunda derivada en una dirección específica representada por un vector unitario d es dado por *d-alta definición*. Cuando d es un vector propio de H , la segunda derivada en esa dirección viene dada por el valor propio correspondiente. Para otras direcciones de d , la segunda derivada direccional es un promedio ponderado de todos los valores propios, con pesos entre 0 y 1, y vectores propios que tienen un ángulo menor con d recibiendo más peso. El valor propio máximo determina la segunda derivada máxima y el valor propio mínimo determina la segunda derivada mínima.

La segunda derivada (direccional) nos dice qué tan bien podemos esperar que funcione un paso de descenso de gradiente. Podemos hacer una aproximación de la serie de Taylor de segundo orden a la función $F(X)$ alrededor del punto actual $X(0)$:

$$F(X) \approx F(X(0)) + (x - X(0)) \cdot \text{gramo} + \left(x - \frac{1}{2}X(0)\right) \cdot H(x - X(0)). \quad (4.8)$$

dónde gramo es el gradiente y H es la arpilla en $X(0)$. Si usamos una tasa de aprendizaje de α , entonces el nuevo punto X será dado por $X(0) - \alpha \cdot \text{gramo}$. Sustituyendo esto en nuestra aproximación, obtenemos

$$F(X(0) - \alpha \cdot \text{gramo}) \approx F(X(0)) - \text{gramo} \cdot \text{gramo} + \frac{1}{2} \text{gramo} \cdot H \cdot \text{gramo}. \quad (4.9)$$

Aquí hay tres términos: el valor original de la función, la mejora esperada debido a la pendiente de la función y la corrección que debemos aplicar para dar cuenta de la curvatura de la función. Cuando este último término es demasiado grande, el escalón de descenso del gradiente puede moverse cuesta arriba. Cuando $\text{gramo} \cdot H$ es cero o negativo, la aproximación de la serie de Taylor predice que al aumentar α siempre disminuirá F para siempre. En la práctica, es poco probable que la serie de Taylor siga siendo precisa para grandes α , por lo que uno debe recurrir a opciones más heurísticas de α en este caso. Cuando $\text{gramo} \cdot H$ es positivo, resolviendo el tamaño de paso óptimo que disminuye la aproximación de la serie de Taylor de la función que más produce

$$\alpha^* = \frac{\text{gramo} \cdot \text{gramo}}{\text{gramo} \cdot H \cdot \text{gramo}}. \quad (4.10)$$

En el peor de los casos, cuando gramo se alinea con el vector propio de H correspondiente a la valor propio máximo λ_{\max} , entonces este tamaño de paso óptimo viene dado por $\frac{1}{\lambda_{\max}}$. Hacia la medida en que la función que minimizamos se puede aproximar bien mediante una función cuadrática, los valores propios de la hessiana determinan la escala de la tasa de aprendizaje.

La segunda derivada se puede utilizar para determinar si un punto crítico es un máximo local, un mínimo local o un punto de silla. Recuérdese que en un punto crítico, $F'(X) = 0$. Cuando la segunda derivada $F''(X) > 0$, la primera derivada $F'(X)$ aumenta a medida que avanzamos hacia la derecha y disminuye a medida que avanzamos hacia la izquierda. Esto significa

$F(x - \Delta) < F(x)$ para lo suficientemente pequeño Δ . En otras palabras, a medida que nos movemos a la derecha, la pendiente comienza a apuntar hacia arriba a la derecha, y a medida que nos movemos a la izquierda, la pendiente comienza a apuntar hacia arriba a la izquierda. Así, cuando $F'(x) = 0$ y $F''(x) > 0$, podemos concluir que x es un mínimo local. Del mismo modo, cuando $F'(x) = 0$ y $F''(x) < 0$, podemos concluir que x es un máximo local. Esto se conoce como **el prueba de la segunda derivada**.

Desafortunadamente, cuando $F''(x) = 0$, la prueba no es concluyente. En este caso x puede ser un punto de silla, o una parte de una región plana.

En múltiples dimensiones, necesitamos examinar todas las segundas derivadas de la función. Usando la descomposición propia de la matriz hessiana, podemos generalizar la prueba de la segunda derivada a múltiples dimensiones. En un punto crítico, donde $\nabla F(x) = 0$, podemos examinar los valores propios de la arpillería para determinar si el punto crítico es un máximo local, un mínimo local o un punto de silla. Cuando la arpillería es definida positiva (todos sus valores propios son positivos), el punto es un mínimo local. Esto se puede ver observando que la segunda derivada direccional en cualquier dirección debe ser positiva y haciendo referencia a la prueba de la segunda derivada univariada. Asimismo, cuando la arpillería es definida negativa (todos sus valores propios son negativos), el punto es un máximo local. En múltiples dimensiones, en realidad es posible encontrar evidencia positiva de puntos de silla en algunos casos. Cuando al menos un valor propio es positivo y al menos un valor propio es negativo, sabemos que x es un máximo local en una sección transversal de F pero un mínimo local en otra sección transversal. Ver figura 4.5 para un ejemplo.

Finalmente, la prueba multidimensional de la segunda derivada puede no ser concluyente, al igual que la versión univariante. La prueba no es concluyente cuando todos los valores propios distintos de cero tienen el mismo signo, pero al menos un valor propio es cero. Esto se debe a que la prueba de la segunda derivada univariante no es concluyente en la sección transversal correspondiente al valor propio cero.

En múltiples dimensiones, hay una segunda derivada diferente para cada dirección en un solo punto. El número de condición del hessiano en este punto mide cuánto difieren las segundas derivadas entre sí. Cuando el Hessian tiene un número de condición pobre, el descenso de gradiente funciona mal. Esto se debe a que en una dirección, la derivada aumenta rápidamente, mientras que en otra dirección aumenta lentamente. Gradient Descent no es consciente de este cambio en la derivada, por lo que no sabe que necesita explorar preferentemente en la dirección en la que la derivada permanece negativa durante más tiempo. También hace que sea difícil elegir un buen tamaño de paso. El tamaño del paso debe ser lo suficientemente pequeño para evitar sobrepasar el mínimo e ir cuesta arriba en direcciones con fuerte curvatura positiva. Esto generalmente significa que el tamaño del paso es demasiado pequeño para lograr un progreso significativo en otras direcciones con menos curvatura. Ver figura 4.6 para un ejemplo.

Este problema se puede resolver usando información de la matriz Hessian para guiar

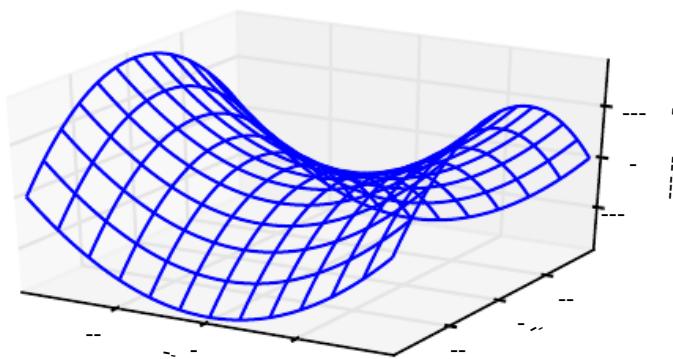


Figura 4.5: Un punto de silla que contiene curvatura tanto positiva como negativa. La función en este ejemplo es $f(x) = x_1 - x_2^2$. A lo largo del eje correspondiente a x_1 , la función curva hacia arriba. Este eje es un vector propio de la arpillera y tiene un valor propio positivo. A lo largo del eje correspondiente a x_2 , la función se curva hacia abajo. Esta dirección es un vector propio de la arpillera con valor propio negativo. El nombre "punto de silla de montar" se deriva de la forma de silla de montar de esta función. Este es el ejemplo por excelencia de una función con un punto silla. En más de una dimensión, no es necesario tener un valor propio de 0 para obtener un punto de silla: solo es necesario tener valores propios tanto positivos como negativos. Podemos pensar en un punto de silla con ambos signos de valores propios como un máximo local dentro de una sección transversal y un mínimo local dentro de otra sección transversal.

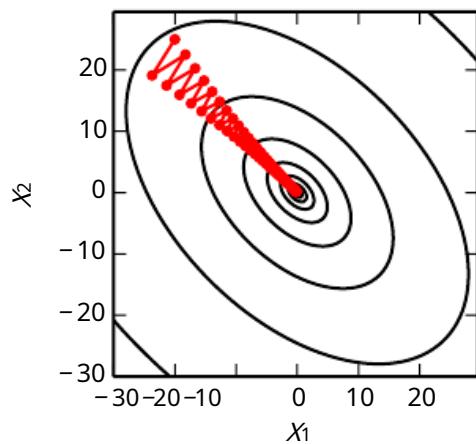


Figura 4.6: El descenso de gradiente no aprovecha la información de curvatura contenida en la matriz Hessiana. Aquí usamos el descenso de gradiente para minimizar una función cuadrática $f(X)$ cuya matriz hessiana tiene la condición número 5. Esto significa que la dirección de mayor curvatura tiene cinco veces más curvatura que la dirección de menor curvatura. En este caso, la mayor curvatura está en la dirección $[1, 1]$ y la menor curvatura está en la dirección $[1, -1]$. Las líneas rojas indican el camino seguido por el descenso del gradiente. Esta función cuadrática muy alargada se asemeja a un largo cañón. El descenso en pendiente hace perder el tiempo descendiendo repetidamente por las paredes del cañón, porque son la característica más empinada. Debido a que el tamaño del paso es demasiado grande, tiende a sobrepasar la parte inferior de la función y, por lo tanto, necesita descender por la pared opuesta del cañón en la siguiente iteración. El gran valor propio positivo de Hessian correspondiente al vector propio apuntado en esta dirección indica que esta derivada direccional está aumentando rápidamente, por lo que un algoritmo de optimización basado en Hessian podría predecir que la dirección más inclinada no es realmente una dirección de búsqueda prometedora en este contexto.

la búsqueda. El método más simple para hacerlo se conoce como **método de newton**. El método de Newton se basa en el uso de una expansión de la serie de Taylor de segundo orden para aproximar $F(X)$ cerca de algún punto $X(0)$:

$$F(X) \approx F(X(0)) + (x - X(0)) \cdot \nabla_x F(X(0)) + \frac{1}{2} (x - X(0)) \cdot H(F)(X(0))(x - X(0)). \quad (4.11)$$

Si luego resolvemos para el punto crítico de esta función, obtenemos:

$$X = X(0) - H(F)(X(0))^{-1} \nabla_x F(X(0)). \quad (4.12)$$

Cuando F es una función cuadrática definida positiva, el método de Newton consiste en aplicar la ecuación 4.12 una vez para saltar directamente al mínimo de la función. Cuando F no es verdaderamente cuadrática pero puede aproximarse localmente como una cuadrática definida positiva, el método de Newton consiste en aplicar la ecuación 4.12 varias veces. La actualización iterativa de la aproximación y el salto al mínimo de la aproximación pueden alcanzar el punto crítico mucho más rápido que el descenso de gradiente. Esta es una propiedad útil cerca de un mínimo local, pero puede ser una propiedad dañina cerca de un punto de silla. Como se discutió en la sección 8.2.3, el método de Newton solo es apropiado cuando el punto crítico cercano es un mínimo (todos los valores propios de la arpillera son positivos), mientras que el descenso del gradiente no es atraído por los puntos de silla a menos que el gradiente apunte hacia ellos.

Los algoritmos de optimización que usan solo el gradiente, como el descenso de gradiente, se denominan **algoritmos de optimización de primer orden**. Los algoritmos de optimización que también utilizan la matriz hessiana, como el método de Newton, se denominan **algoritmos de optimización de segundo orden** (Nocedal y Wright, 2006).

Los algoritmos de optimización empleados en la mayoría de los contextos de este libro son aplicables a una amplia variedad de funciones, pero casi no tienen garantías. Los algoritmos de aprendizaje profundo tienden a carecer de garantías porque la familia de funciones utilizadas en el aprendizaje profundo es bastante complicada. En muchos otros campos, el enfoque dominante de la optimización es diseñar algoritmos de optimización para una familia limitada de funciones.

En el contexto del aprendizaje profundo, a veces obtenemos algunas garantías al restringirnos a funciones que son **Lipschitz continuo** tener derivados continuos de Lipschitz. Una función continua de Lipschitz es una función F cuya tasa de cambio está limitada por un **Constante de Lipschitz** L :

$$\forall X, \forall y, |F(X) - F(y)| \leq L \|x - y\|_2. \quad (4.13)$$

Esta propiedad es útil porque nos permite cuantificar nuestra suposición de que un pequeño cambio en la entrada realizada por un algoritmo como el descenso de gradiente tendrá

un pequeño cambio en la salida. La continuidad de Lipschitz también es una restricción bastante débil, y muchos problemas de optimización en el aprendizaje profundo pueden convertirse en continuos de Lipschitz con modificaciones relativamente menores.

Quizás el campo más exitoso de la optimización especializada es **optimización convexa**. Los algoritmos de optimización convexos pueden proporcionar muchas más garantías al establecer restricciones más estrictas. Los algoritmos de optimización convexa son aplicables solo a funciones convexas, funciones para las cuales la matriz de covarianza es semidefinida positiva en todas partes. Tales funciones se comportan bien porque carecen de puntos de silla y todos sus mínimos locales son necesariamente mínimos globales. Sin embargo, la mayoría de los problemas del aprendizaje profundo son difíciles de expresar en términos de optimización convexa. La optimización convexa se usa solo como una subrutina de algunos algoritmos de aprendizaje profundo. Las ideas del análisis de los algoritmos de optimización convexa pueden ser útiles para probar la convergencia de los algoritmos de aprendizaje profundo. Sin embargo, en general, la importancia de la optimización convexa disminuye considerablemente en el contexto del aprendizaje profundo. Para obtener más información sobre la optimización convexa, consulte [Boyd y Vandenberghe\(2004\)](#) o [Rockafellar\(1997\)](#).

4.4 Optimización restringida

A veces deseamos no solo maximizar o minimizar una función $f(X)$ sobre todos los valores posibles de X . En su lugar, podemos desear encontrar el valor máximo o mínimo de $f(X)$ para valores de X en algún conjunto S . Esto se conoce como **optimización con restricciones**. Puntos X que se encuentran dentro del conjunto S son llamados **factible** puntos en la terminología de optimización restringida.

A menudo deseamos encontrar una solución que sea pequeña en algún sentido. Un enfoque común en tales situaciones es imponer una restricción normativa, como $\|x\| \leq 1$.

Un enfoque simple para la optimización restringida es simplemente modificar el descenso del gradiente teniendo en cuenta la restricción. Si usamos un pequeño tamaño de paso constante, podemos hacer pasos de descenso de gradiente, luego proyectar el resultado de nuevo en S . Si usamos una búsqueda de línea, podemos buscar solo sobre tamaños de paso que producen nuevos X puntos que son factibles, o podemos proyectar cada punto en la línea de regreso a la región de restricción. Cuando sea posible, este método se puede hacer más eficiente proyectando el gradiente en el espacio tangente de la región factible antes de dar el paso o comenzar la búsqueda de la línea ([Rosa, 1960](#)).

Un enfoque más sofisticado es diseñar un problema de optimización sin restricciones diferente cuya solución se pueda convertir en una solución al problema de optimización restringido original. Por ejemplo, si queremos minimizar $f(X)$ para

$X \in \mathbb{R}^n$ con X restringido a tener exactamente la unidad L_2 norma, podemos en su lugar minimizar $gramo(\theta) = F[\text{porque } \theta, \text{pecado } \theta]$ con respecto a θ , luego regresa $[\text{porque } \theta, \text{pecado } \theta]$ como solución al problema original. Este enfoque requiere creatividad; la transformación entre problemas de optimización debe diseñarse específicamente para cada caso que nos encontramos.

El **Karush–Kuhn–Tucker** (KKT) enfoque¹ proporciona una solución muy general para la optimización restringida. Con el enfoque KKT, presentamos una nueva función llamada **Lagrangiano generalizado** o **función de Lagrange generalizada**.

Para definir el Lagrangiano, primero necesitamos describir en términos de ecuaciones y desigualdades. Queremos una descripción de los términos de F y las funciones $gramo(\cdot)$ y las funciones $h(\cdot)$ de modo que $S = \{x \mid \forall i \quad G_i(x) = 0 \text{ y } h_i(x) \leq 0\}$. Las ecuaciones que involucran $gramo(\cdot)$ se llaman **restricciones de igualdad** y las desigualdades que implican $h(\cdot)$ son llamados **restricciones de desigualdad**.

Introducimos nuevas variables λ y a para cada restricción, estos se denominan multiplicadores KKT. El lagrangiano generalizado se define entonces como

$$L(x, \lambda, a) = F(x) + \sum_i \lambda_i g_i(x) + \sum_j a_j h_j(x). \quad (4.14)$$

Ahora podemos resolver un problema de minimización con restricciones utilizando la optimización sin restricciones del Lagrangiano generalizado. Observe que, siempre que exista al menos un punto factible y $F(x)$ no se permite tener valor ∞ , entonces

$$\min_x \max_{\lambda, a \geq 0} L(x, \lambda, a). \quad (4.15)$$

tiene el mismo valor óptimo de función objetivo y conjunto de puntos óptimos x como

$$\min_{x \in S} F(x). \quad (4.16)$$

Esto se deduce porque cada vez que se satisfacen las restricciones,

$$\max_{\lambda, a \geq 0} \max_x L(x, \lambda, a) = F(x), \quad (4.17)$$

mientras que cada vez que se viola una restricción,

$$\max_{\lambda, a \geq 0} \max_x L(x, \lambda, a) = \infty. \quad (4.18)$$

¹El enfoque KKT generaliza el método de **Multiplicadores de Lagrange** que permite restricciones de igualdad pero no restricciones de desigualdad.

Estas propiedades garantizan que ningún punto no factible puede ser óptimo y que el óptimo dentro de los puntos factibles no cambia.

Para realizar la maximización restringida, podemos construir la función de Lagrange generalizada de $-F(X)$, lo que conduce a este problema de optimización:

$$\min_{X} \max_{\lambda} \max_{a, a \geq 0} -F(X) + \sum_i \lambda_i h_i(X) + \sum_j a_j h_j(X). \quad (4.19)$$

También podemos convertir esto en un problema con la maximización en el ciclo externo:

$$\max_{X} \max_{\lambda} \min_{a, a \geq 0} -F(X) - \sum_i \lambda_i h_i(X) - \sum_j a_j h_j(X). \quad (4.20)$$

El signo del término para las restricciones de igualdad no importa; podemos definirlo con sumas o restas como queramos, porque la optimización es libre de elegir cualquier signo para cada λ_i .

Las restricciones de desigualdad son particularmente interesantes. Decimos que una restricción $h_i(X)$ es **activo** si $h_i(X^*) = 0$. Si una restricción no está activa, entonces la solución al problema encontrado usando esa restricción seguiría siendo al menos una solución local si se eliminara esa restricción. Es posible que una restricción inactiva excluya otras soluciones. Por ejemplo, un problema convexo con una región completa de puntos globalmente óptimos (una región ancha y plana de igual costo) podría tener un subconjunto de esta región eliminado por restricciones, o un problema no convexo podría tener mejores puntos estacionarios locales excluidos por una restricción que está inactiva en la convergencia. Sin embargo, el punto encontrado en la convergencia sigue siendo un punto estacionario ya sea que se incluyan o no las restricciones inactivas. Porque un inactivo h_i tiene valor negativo, entonces la solución deminimáximo $a, a \geq 0$ de $L(x, \lambda, a)$ tendrá $a=0$. Así podemos observar que en la solución, $a \cdot h_i(X) = 0$. En otras palabras, para todos i , sabemos que al menos una de las restricciones $a_i \geq 0$ y $h_i(X) \leq 0$ debe estar activa en la solución. Para ganar algo de intuición sobre esta idea, podemos decir que la solución está en el límite impuesto por la desigualdad y debemos usar su multiplicador KKT para influir en la solución a X , o la desigualdad no tiene influencia en la solución y representamos esto poniendo a cero su multiplicador KKT.

Un conjunto simple de propiedades describe los puntos óptimos de los problemas de optimización con restricciones. Estas propiedades se denominan condiciones de Karush-Kuhn-Tucker (KKT) ([Karush, 1939](#); [Kuhn y Tucker, 1951](#)). Son condiciones necesarias, pero no siempre condiciones suficientes, para que un punto sea óptimo. Las condiciones son:

- El gradiente del Lagrangiano generalizado es cero.
- Todas las restricciones en ambos X y los multiplicadores KKT están satisfechos.

- Las restricciones de desigualdad exhiben "holgura complementaria": $a \cdot h(X) = 0$.

Para obtener más información sobre el enfoque KKT, consulte [Nocedal y Wright\(2006\)](#).

4.5 Ejemplo: Mínimos cuadrados lineales

Supongamos que queremos encontrar el valor de X que minimiza

$$F(X) = \frac{1}{2} \|Hacha - b\|_2^2. \quad (4.21)$$

Existen algoritmos de álgebra lineal especializados que pueden resolver este problema de manera eficiente. Sin embargo, también podemos explorar cómo resolverlo utilizando la optimización basada en gradientes como un ejemplo simple de cómo funcionan estas técnicas.

Primero, necesitamos obtener el gradiente:

$$\nabla_X F(X) = A(Hacha - b) = A \cdot Hacha - A \cdot b. \quad (4.22)$$

Luego podemos seguir este gradiente cuesta abajo, dando pequeños pasos. Ver algoritmo [4.1](#) para detalles.

Algoritmo 4.1 Un algoritmo para minimizar $F(X) = \frac{1}{2} \|Hacha - b\|_2^2$ con respecto a X utilizando descenso de gradiente, a partir de un valor arbitrario de X .

Establezca el tamaño del paso (α) y tolerancia (δ) a números pequeños y positivos.

mientras $\|A \cdot X - b\|_2 > \delta$ **hacer**
 $X \leftarrow X - \alpha \cdot A \cdot Hacha - A \cdot b$

terminar mientras

También se puede resolver este problema usando el método de Newton. En este caso, debido a que la verdadera función es cuadrática, la aproximación cuadrática empleada por el método de Newton es exacta y el algoritmo converge al mínimo global en un solo paso.

Supongamos ahora que deseamos minimizar la misma función, pero sujeta a la restricción $X \cdot X \leq 1$. Para ello, introducimos el Lagrangiano

$$L(x, \lambda) = F(X) + \lambda(X \cdot x - 1). \quad (4.23)$$

Ahora podemos resolver el problema.

$$\begin{array}{l} \text{mínimo máximo } L(x, \lambda), \\ x, \lambda \geq 0 \end{array} \quad (4.24)$$

La solución de norma más pequeña para el problema de mínimos cuadrados sin restricciones se puede encontrar utilizando el pseudoinverso de Moore-Penrose: $X = A + b$. Si este punto es factible, entonces es la solución al problema restringido. De lo contrario, debemos encontrar una solución donde la restricción esté activa. Al diferenciar el Lagrangiano con respecto a X , obtenemos la ecuación

$$A \cdot H \cdot a - A \cdot b + 2\lambda x = 0. \quad (4.25)$$

Esto nos dice que la solución tomará la forma

$$X = (A \cdot A + 2\lambda I)^{-1} A \cdot b. \quad (4.26)$$

La magnitud de λ debe elegirse de manera que el resultado obedezca la restricción. Podemos encontrar este valor realizando un ascenso de gradiente en λ . Para ello, observa

$$\frac{\partial}{\partial \lambda} L(x, \lambda) = X \cdot x - 1. \quad (4.27)$$

Cuando la norma de X excede 1, esta derivada es positiva, por lo que para seguir la derivada cuesta arriba y aumentar el Lagrangiano con respecto a λ , aumentamos λ . Porque el coeficiente de la $X \cdot X$ la penalización ha aumentado, resolviendo la ecuación lineal para X ahora producirá una solución con una norma más pequeña. El proceso de resolver la ecuación lineal y ajustar λ continúa hasta que X tiene la norma correcta y la derivada es 0.

Esto concluye los preliminares matemáticos que usamos para desarrollar algoritmos de aprendizaje automático. Ahora estamos listos para construir y analizar algunos sistemas de aprendizaje completos.

Capítulo 5

Conceptos básicos de aprendizaje automático

El aprendizaje profundo es un tipo específico de aprendizaje automático. Para comprender bien el aprendizaje profundo, uno debe tener una comprensión sólida de los principios básicos del aprendizaje automático. Este capítulo proporciona un curso breve sobre los principios generales más importantes que se aplicarán a lo largo del resto del libro. Se recomienda a los lectores novatos o aquellos que desean una perspectiva más amplia que consideren libros de texto de aprendizaje automático con una cobertura más completa de los fundamentos, como [Murphy \(2012\)](#) o [Bishop \(2006\)](#). Si ya está familiarizado con los conceptos básicos de aprendizaje automático, no dude en pasar directamente a la sección [5.11](#). Esa sección cubre algunas perspectivas sobre las técnicas tradicionales de aprendizaje automático que han influido fuertemente en el desarrollo de algoritmos de aprendizaje profundo.

Comenzamos con una definición de lo que es un algoritmo de aprendizaje y presentamos un ejemplo: el algoritmo de regresión lineal. Luego procedemos a describir cómo el desafío de ajustar los datos de entrenamiento difiere del desafío de encontrar patrones que se generalicen a nuevos datos. La mayoría de los algoritmos de aprendizaje automático tienen configuraciones denominadas hiperparámetros que deben determinarse de forma externa al propio algoritmo de aprendizaje; discutimos cómo configurarlos usando datos adicionales. El aprendizaje automático es esencialmente una forma de estadística aplicada con un mayor énfasis en el uso de computadoras para estimar estadísticamente funciones complicadas y un menor énfasis en probar intervalos de confianza en torno a estas funciones; por lo tanto, presentamos los dos enfoques centrales de la estadística: los estimadores frecuentistas y la inferencia bayesiana. La mayoría de los algoritmos de aprendizaje automático se pueden dividir en las categorías de aprendizaje supervisado y aprendizaje no supervisado; describimos estas categorías y damos algunos ejemplos de algoritmos de aprendizaje simples de cada categoría. La mayoría de los algoritmos de aprendizaje profundo se basan en un algoritmo de optimización llamado descenso de gradiente estocástico. Describimos cómo combinar varios componentes del algoritmo, como

un algoritmo de optimización, una función de costo, un modelo y un conjunto de datos para construir un algoritmo de aprendizaje automático. Finalmente, en la sección 5.11, describimos algunos de los factores que han limitado la capacidad de generalización del aprendizaje automático tradicional. Estos desafíos han motivado el desarrollo de algoritmos de aprendizaje profundo que superan estos obstáculos.

5.1 Algoritmos de aprendizaje

Un algoritmo de aprendizaje automático es un algoritmo que puede aprender de los datos. Pero, ¿qué entendemos por aprender? Mitchell(1997) proporciona la definición “Se dice que un programa de computadora aprende de la experiencia m_i con respecto a alguna clase de tareas T y medida de desempeño PAG , si su desempeño en tareas en T , medido por PAG , mejora con la experiencia m_i .” Uno puede imaginar una variedad muy amplia de experiencias m_i , tareas T y medidas de desempeño PAG , y no hacemos ningún intento en este libro de proporcionar una definición formal de lo que puede usarse para cada una de estas entidades. En cambio, las siguientes secciones brindan descripciones intuitivas y ejemplos de los diferentes tipos de tareas, medidas de rendimiento y experiencias que se pueden usar para construir algoritmos de aprendizaje automático.

5.1.1 La Tarea, T

El aprendizaje automático nos permite abordar tareas que son demasiado difíciles de resolver con programas fijos escritos y diseñados por seres humanos. Desde un punto de vista científico y filosófico, el aprendizaje automático es interesante porque desarrollar nuestra comprensión del aprendizaje automático implica desarrollar nuestra comprensión de los principios que subyacen a la inteligencia.

En esta definición relativamente formal de la palabra “tarea”, el proceso de aprendizaje en sí mismo no es la tarea. El aprendizaje es nuestro medio para lograr la capacidad de realizar la tarea. Por ejemplo, si queremos que un robot pueda caminar, la tarea es caminar. Podríamos programar el robot para que aprenda a caminar, o podríamos intentar escribir directamente un programa que especifique cómo caminar manualmente.

Las tareas de aprendizaje automático generalmente se describen en términos de cómo el sistema de aprendizaje automático debe procesar **un ejemplo**. Un ejemplo es una colección de **características** que se han medido cuantitativamente a partir de algún objeto o evento que queremos que procese el sistema de aprendizaje automático. Normalmente representamos un ejemplo como un vector $X \in \mathbb{R}^n$ donde cada entrada x_i del vector es otra característica. Por ejemplo, las características de una imagen suelen ser los valores de los píxeles de la imagen.

Muchos tipos de tareas se pueden resolver con el aprendizaje automático. Algunas de las tareas de aprendizaje automático más comunes incluyen las siguientes:

- **Clasificación:** En este tipo de tarea, se le pide al programa de computadora que especifique cuál de k categorías a las que pertenece alguna entrada. Para resolver esta tarea, generalmente se le pide al algoritmo de aprendizaje que produzca una función $f: \mathcal{X} \rightarrow \{1, \dots, k\}$. Cuando $y = f(X)$, el modelo asigna una entrada descrita por el vector X a una categoría identificada por un código numérico y . Hay otras variantes de la tarea de clasificación, por ejemplo, donde f genera una distribución de probabilidad sobre las clases. Un ejemplo de una tarea de clasificación es el reconocimiento de objetos, donde la entrada es una imagen (generalmente descrita como un conjunto de valores de brillo de píxeles) y la salida es un código numérico que identifica el objeto en la imagen. Por ejemplo, el robot Willow Garage PR2 puede actuar como un camarero que puede reconocer diferentes tipos de bebidas y entregárselas a las personas que se lo ordenen ([Buen compañero et al., 2010](#)). El reconocimiento de objetos moderno se logra mejor con el aprendizaje profundo ([Krizhevskiet al., 2012](#); [Ioffe y Szegedy, 2015](#)). El reconocimiento de objetos es la misma tecnología básica que permite a las computadoras reconocer rostros ([Taigmanet al., 2014](#)), que se puede usar para etiquetar automáticamente a personas en colecciones de fotos y permitir que las computadoras interactúen de manera más natural con sus usuarios.
- **Clasificación con entradas faltantes:** La clasificación se vuelve más desafiante si no se garantiza que el programa de computadora siempre proporcionará todas las medidas en su vector de entrada. Para resolver la tarea de clasificación, el algoritmo de aprendizaje sólo tiene que definir un *solver* mapeo de funciones desde una entrada vectorial a una salida categórica. Cuando algunas de las entradas pueden faltar, en lugar de proporcionar una única función de clasificación, el algoritmo de aprendizaje debe aprender una *colección* de funciones. Cada función corresponde a clasificar X con un subconjunto diferente de sus entradas faltantes. Este tipo de situación se presenta con frecuencia en el diagnóstico médico, debido a que muchos tipos de pruebas médicas son costosas o invasivas. Una forma de definir eficientemente un conjunto tan grande de funciones es aprender una distribución de probabilidad sobre todas las variables relevantes y luego resolver la tarea de clasificación marginando las variables que faltan. Con n variables de entrada, ahora podemos obtener todas 2^n necesitan diferentes funciones de clasificación para cada posible conjunto de entradas faltantes, pero solo necesitamos aprender una única función que describa la distribución de probabilidad conjunta. Ver [Buen compañero et al. \(2013b\)](#) para ver un ejemplo de un modelo probabilístico profundo aplicado a tal tarea de esta manera. Muchas de las otras tareas descritas en esta sección también se pueden generalizar para trabajar con entradas faltantes; la clasificación con entradas faltantes es solo un ejemplo de lo que puede hacer el aprendizaje automático.

- **Regresión:** En este tipo de tarea, se le pide al programa de computadora que prediga un valor numérico dada alguna entrada. Para resolver esta tarea, se le pide al algoritmo de aprendizaje que genere una función $F: R_{norte} \rightarrow r$. Este tipo de tarea es similar a la clasificación, excepto que el formato de salida es diferente. Un ejemplo de una tarea de regresión es la predicción del monto esperado de la reclamación que realizará una persona asegurada (que se utiliza para establecer las primas de seguros) o la predicción de los precios futuros de los valores. Este tipo de predicciones también se utilizan para el comercio algorítmico.
- **Transcripción:** En este tipo de tarea, se le pide al sistema de aprendizaje automático que observe una representación relativamente no estructurada de algún tipo de datos y la transcriba a una forma textual discreta. Por ejemplo, en el reconocimiento óptico de caracteres, al programa informático se le muestra una fotografía que contiene una imagen de texto y se le pide que devuelva este texto en forma de una secuencia de caracteres (por ejemplo, en formato ASCII o Unicode). Google Street View utiliza el aprendizaje profundo para procesar números de direcciones de esta manera ([Buen compañero et al., 2014d](#)). Otro ejemplo es el reconocimiento de voz, donde el programa de computadora recibe una forma de onda de audio y emite una secuencia de caracteres o códigos de identificación de palabras que describen las palabras que se pronunciaron en la grabación de audio. El aprendizaje profundo es un componente crucial de los sistemas modernos de reconocimiento de voz utilizados en las principales empresas, incluidas Microsoft, IBM y Google ([Hinton et al., 2012b](#)).
- **Máquina traductora:** En una tarea de traducción automática, la entrada ya consiste en una secuencia de símbolos en algún idioma, y el programa de computadora debe convertir esto en una secuencia de símbolos en otro idioma. Esto se aplica comúnmente a los lenguajes naturales, como la traducción del inglés al francés. El aprendizaje profundo ha comenzado recientemente a tener un impacto importante en este tipo de tareas ([Sutskever et al., 2014; Bahdanau et al., 2015](#)).
- **Salida estructurada:** Las tareas de salida estructurada involucran cualquier tarea donde la salida es un vector (u otra estructura de datos que contiene múltiples valores) con relaciones importantes entre los diferentes elementos. Esta es una categoría amplia y comprende las tareas de transcripción y traducción descritas anteriormente, pero también muchas otras tareas. Un ejemplo es el análisis: mapear una oración de lenguaje natural en un árbol que describe su estructura gramatical y etiquetar los nodos de los árboles como verbos, sustantivos o adverbios, etc. Ver [coloberto\(2011\)](#) para ver un ejemplo de aprendizaje profundo aplicado a una tarea de análisis. Otro ejemplo es la segmentación de imágenes por píxeles, en la que el programa informático asigna cada píxel de una imagen a una categoría específica. Para

Por ejemplo, el aprendizaje profundo se puede utilizar para anotar las ubicaciones de las carreteras en fotografías aéreas (Mnih y Hinton, 2010). La salida no necesita tener su forma reflejando la estructura de la entrada tan fielmente como en estas tareas de estilo de anotación. Por ejemplo, en el subtítulo de imágenes, el programa de computadora observa una imagen y genera una oración en lenguaje natural que describe la imagen (kirós *et al.*, 2014a,b; mao *et al.*, 2015; vinilo *et al.*, 2015b; Donahue *et al.*, 2014; Karpatía y Li, 2015; Colmillo *et al.*, 2015; xu *et al.*, 2015). Estas tareas se denominan tareas de salida estructurada porque el programa debe generar varios valores que están estrechamente relacionados entre sí. Por ejemplo, las palabras producidas por un programa de subtítulos de imágenes deben formar una oración válida.

- **Detección de anomalías:** En este tipo de tarea, el programa de computadora filtra un conjunto de eventos u objetos y marca algunos de ellos como inusuales o atípicos. Un ejemplo de una tarea de detección de anomalías es la detección de fraudes con tarjetas de crédito. Al modelar sus hábitos de compra, una compañía de tarjetas de crédito puede detectar el uso indebido de sus tarjetas. Si un ladrón roba su tarjeta de crédito o la información de su tarjeta de crédito, las compras del ladrón a menudo provendrán de una distribución de probabilidad sobre los tipos de compra diferente a la suya. La compañía de la tarjeta de crédito puede prevenir el fraude reteniendo una cuenta tan pronto como la tarjeta haya sido utilizada para una compra inusual. VerChandola *et al.*(2009) para un estudio de los métodos de detección de anomalías.
- **Síntesis y muestreo:** En este tipo de tarea, se le pide al algoritmo de aprendizaje automático que genere nuevos ejemplos que sean similares a los de los datos de entrenamiento. La síntesis y el muestreo a través del aprendizaje automático pueden ser útiles para aplicaciones de medios donde puede ser costoso o aburrido para un artista generar grandes volúmenes de contenido a mano. Por ejemplo, los videojuegos pueden generar automáticamente texturas para objetos grandes o paisajes, en lugar de requerir que un artista etiquete manualmente cada píxel (Luo *et al.*, 2013). En algunos casos, queremos que el procedimiento de muestreo o síntesis genere algún tipo específico de salida dada la entrada. Por ejemplo, en una tarea de síntesis de voz, proporcionamos una oración escrita y le pedimos al programa que emita una forma de onda de audio que contenga una versión hablada de esa oración. Este es un tipo de tarea de salida estructurada, pero con la calificación adicional de que no hay una única salida correcta para cada entrada, y deseamos explícitamente una gran cantidad de variación en la salida, para que la salida parezca más natural y realista.
- **Imputación de valores faltantes:** En este tipo de tareas, el algoritmo de aprendizaje automático recibe un nuevo ejemplo $X \in \mathbb{R}^n$, pero con algunas entradas X_i de X desaparecido. El algoritmo debe proporcionar una predicción de los valores de las entradas que faltan.

- **eliminación de ruido:** En este tipo de tarea, el algoritmo de aprendizaje automático se da en la entrada a *ejemplo corrupto* $\tilde{x} \in R_{norte}$ obtenido por un proceso de corrupción desconocido de un *ejemplo limpio* $x \in R_{norte}$. El alumno debe predecir el ejemplo limpio. \tilde{x} de su versión corrupta x , o más generalmente predecir la distribución de probabilidad condicional $p_{\text{ag}}(x / \tilde{x})$.
- **Estimación de densidad/estimación de la función de masa de probabilidad:** En el problema de estimación de densidad, se le pide al algoritmo de aprendizaje automático que aprenda una función $p_{\text{ag}}_{\text{modelo}}: R_{norte} \rightarrow R$, donde $p_{\text{ag}}_{\text{modelo}}(x)$ puede interpretarse como una función de densidad de probabilidad (si x es continua) o una función de masa de probabilidad (si x es discreto) en el espacio del que se extrajeron los ejemplos. Para hacer bien esa tarea (especifiquemos exactamente lo que eso significa cuando discutamos las medidas de desempeño PAG), el algoritmo necesita aprender la estructura de los datos que ha visto. Debe saber dónde se agrupan estrechamente los ejemplos y dónde es poco probable que ocurran. La mayoría de las tareas descritas anteriormente requieren que el algoritmo de aprendizaje capture al menos implícitamente la estructura de la distribución de probabilidad. La estimación de densidad nos permite capturar explícitamente esa distribución. En principio, podemos realizar cálculos en esa distribución para resolver también las otras tareas. Por ejemplo, si hemos realizado una estimación de densidad para obtener una distribución de probabilidad $p_{\text{ag}}(x)$, podemos usar esa distribución para resolver la tarea de imputación del valor faltante. Si un valor x_i falta y todos los demás valores, denotados x_{-i} , están dadas, entonces sabemos que la distribución sobre ella está dada por $p_{\text{ag}}(x_i | x_{-i})$. En la práctica, la estimación de la densidad no siempre nos permite resolver todas estas tareas relacionadas, porque en muchos casos las operaciones requeridas en $p_{\text{ag}}(x)$ son computacionalmente intratables.

Por supuesto, muchas otras tareas y tipos de tareas son posibles. Los tipos de tareas que enumeramos aquí solo pretenden proporcionar ejemplos de lo que puede hacer el aprendizaje automático, no definir una taxonomía rígida de tareas.

5.1.2 La Medida de Desempeño, PAG

Para evaluar las capacidades de un algoritmo de aprendizaje automático, debemos diseñar una medida cuantitativa de su rendimiento. Por lo general, esta medida de desempeño PAG es específico para la tarea T siendo llevado a cabo por el sistema.

Para tareas como clasificación, clasificación con entradas faltantes y transcripción, a menudo medimos la **exactitud** del modelo. La precisión es solo la proporción de ejemplos para los cuales el modelo produce el resultado correcto. Podemos

también obtener información equivalente midiendo la **Tasa de error**, la proporción de ejemplos para los que el modelo produce una salida incorrecta. A menudo nos referimos a la tasa de error como la pérdida esperada de 0-1. La pérdida de 0-1 en un ejemplo particular es 0 si está correctamente clasificado y 1 si no lo está. Para tareas como la estimación de la densidad, no tiene sentido medir la precisión, la tasa de error o cualquier otro tipo de pérdida 0-1. En su lugar, debemos utilizar una métrica de rendimiento diferente que proporcione al modelo una puntuación de valor continuo para cada ejemplo. El enfoque más común es reportar la probabilidad logarítmica promedio que el modelo asigna a algunos ejemplos.

Por lo general, nos interesa qué tan bien se desempeña el algoritmo de aprendizaje automático con datos que no ha visto antes, ya que esto determina qué tan bien funcionará cuando se implemente en el mundo real. Por lo tanto, evaluamos estas medidas de desempeño usando un **equipo de prueba** de datos que está separado de los datos utilizados para entrenar el sistema de aprendizaje automático.

La elección de la medida de rendimiento puede parecer sencilla y objetiva, pero a menudo es difícil elegir una medida de rendimiento que se corresponda bien con el comportamiento deseado del sistema.

En algunos casos, esto se debe a que es difícil decidir qué se debe medir. Por ejemplo, al realizar una tarea de transcripción, ¿deberíamos medir la precisión del sistema al transcribir secuencias completas, o deberíamos usar una medida de rendimiento más detallada que dé crédito parcial por corregir algunos elementos de la secuencia? Al realizar una tarea de regresión, ¿debemos penalizar más al sistema si comete errores medianos con frecuencia o si rara vez comete errores muy grandes? Este tipo de opciones de diseño dependen de la aplicación.

En otros casos, sabemos qué cantidad nos gustaría medir idealmente, pero medirla no es práctico. Por ejemplo, esto surge con frecuencia en el contexto de la estimación de la densidad. Muchos de los mejores modelos probabilísticos representan distribuciones de probabilidad solo de manera implícita. Calcular el valor de probabilidad real asignado a un punto específico en el espacio en muchos de estos modelos es intratable. En estos casos, se debe diseñar un criterio alternativo que aún corresponda a los objetivos de diseño, o diseñar una buena aproximación al criterio deseado.

5.1.3 La Experiencia, *mi*

Los algoritmos de aprendizaje automático se pueden clasificar en términos generales como **sin supervisión** o **supervisado** por el tipo de experiencia que se les permite tener durante el proceso de aprendizaje.

La mayoría de los algoritmos de aprendizaje en este libro pueden entenderse como permitidos para experimentar un **conjunto de datos**. Un conjunto de datos es una colección de muchos ejemplos, como

definido en la sección 5.1.1. A veces también llamaremos ejemplos **puntos de datos**.

Uno de los conjuntos de datos más antiguos estudiados por estadísticos e investigadores de aprendizaje automático es el conjunto de datos Iris ([Pescador, 1936](#)). Es una colección de medidas de diferentes partes de 150 plantas de iris. Cada planta individual corresponde a un ejemplo. Las características dentro de cada ejemplo son las medidas de cada una de las partes de la planta: la longitud del sépalo, el ancho del sépalo, la longitud del pétalo y el ancho del pétalo. El conjunto de datos también registra a qué especie pertenecía cada planta. Tres especies diferentes están representadas en el conjunto de datos.

Algoritmos de aprendizaje no supervisado experimenta un conjunto de datos que contiene muchas características, luego aprende propiedades útiles de la estructura de este conjunto de datos. En el contexto del aprendizaje profundo, por lo general queremos aprender toda la distribución de probabilidad que generó un conjunto de datos, ya sea explícitamente como en la estimación de densidad o implícitamente para tareas como síntesis o eliminación de ruido. Algunos otros algoritmos de aprendizaje no supervisados desempeñan otras funciones, como el agrupamiento, que consiste en dividir el conjunto de datos en grupos de ejemplos similares.

Algoritmos de aprendizaje supervisado experimentar un conjunto de datos que contiene características, pero cada ejemplo también está asociado con una **etiqueta o objetivo**. Por ejemplo, el conjunto de datos de Iris se anota con la especie de cada planta de iris. Un algoritmo de aprendizaje supervisado puede estudiar el conjunto de datos de Iris y aprender a clasificar las plantas de iris en tres especies diferentes según sus medidas.

En términos generales, el aprendizaje no supervisado implica observar varios ejemplos de un vector aleatorio X , e intentar implícita o explícitamente aprender la distribución de probabilidad $pag(X)$, o algunas propiedades interesantes de esa distribución, mientras que el aprendizaje supervisado implica observar varios ejemplos de un vector aleatorio X y un valor asociado o vector y , aprender a predecir y de X , generalmente estimando $pag(y/X)$. El término **aprendizaje supervisado** se origina en la vista del objetivo y proporcionado por un instructor o maestro que le muestra al sistema de aprendizaje automático qué hacer. En el aprendizaje no supervisado, no hay instructor ni maestro, y el algoritmo debe aprender a dar sentido a los datos sin esta guía.

El aprendizaje no supervisado y el aprendizaje supervisado no son términos formalmente definidos. Las líneas entre ellos a menudo son borrosas. Se pueden utilizar muchas tecnologías de aprendizaje automático para realizar ambas tareas. Por ejemplo, la regla de la cadena de probabilidad establece que para un vector $X \in \mathbb{R}^n$, la distribución conjunta se puede descomponer como

$$pag(x) = \prod_{i=1}^{n_{\text{norte}}} pag(X_i/X_1, \dots, X_{i-1}). \quad (5.1)$$

Esta descomposición significa que podemos resolver el problema aparentemente no supervisado de modelar $pag(X)$ al dividirlo en n problemas de aprendizaje supervisado. Alternativamente, nosotros

puede resolver el problema de aprendizaje supervisado del aprendizaje $pag(y/X)$ mediante el uso de tecnologías tradicionales de aprendizaje no supervisado para aprender la distribución conjunta $pag(X,y)$ e inferir

$$pag(y/x) = \frac{pag(x,y)}{y \cdot pag(x,y)}. \quad (5.2)$$

Aunque el aprendizaje no supervisado y el aprendizaje supervisado no son conceptos completamente formales o distintos, ayudan a categorizar aproximadamente algunas de las cosas que hacemos con los algoritmos de aprendizaje automático. Tradicionalmente, la gente se refiere a los problemas de regresión, clasificación y salida estructurada como aprendizaje supervisado. La estimación de densidad en apoyo de otras tareas generalmente se considera aprendizaje no supervisado.

Son posibles otras variantes del paradigma de aprendizaje. Por ejemplo, en el aprendizaje semisupervisado, algunos ejemplos incluyen un objetivo de supervisión pero otros no. En el aprendizaje de instancias múltiples, una colección completa de ejemplos se etiqueta como que contiene o no contiene un ejemplo de una clase, pero los miembros individuales de la colección no están etiquetados. Para ver un ejemplo reciente de aprendizaje de varias instancias con modelos profundos, consulte [Kotzias et al. \(2015\)](#).

Algunos algoritmos de aprendizaje automático no solo experimentan un conjunto de datos fijo. Por ejemplo, **aprendizaje reforzado** los algoritmos interactúan con un entorno, por lo que existe un ciclo de retroalimentación entre el sistema de aprendizaje y sus experiencias. Tales algoritmos están más allá del alcance de este libro. Por favor mira [Sutton y Barto \(1998\)](#) o [Bertsekas y Tsitsiklis \(1996\)](#) para obtener información sobre el aprendizaje por refuerzo, y [Mnih et al. \(2013\)](#) para el enfoque de aprendizaje profundo para el aprendizaje por refuerzo.

La mayoría de los algoritmos de aprendizaje automático simplemente experimentan un conjunto de datos. Un conjunto de datos se puede describir de muchas maneras. En todos los casos, un conjunto de datos es una colección de ejemplos, que a su vez son colecciones de características.

Una forma común de describir un conjunto de datos es con un **matriz de diseño**. Una matriz de diseño es una matriz que contiene un ejemplo diferente en cada fila. Cada columna de la matriz corresponde a una característica diferente. Por ejemplo, el conjunto de datos de Iris contiene 150 ejemplos con cuatro características para cada ejemplo. Esto significa que podemos representar el conjunto de datos con una matriz de diseño. $X \in \mathbb{R}^{150 \times 4}$, donde $X_{i,1}$ es la longitud del sépalo de la planta i , $X_{i,2}$ es el ancho del sépalo de la planta i , etc. Describiremos la mayoría de los algoritmos de aprendizaje en este libro en términos de cómo operan en conjuntos de datos de matriz de diseño.

Por supuesto, para describir un conjunto de datos como una matriz de diseño, debe ser posible describir cada ejemplo como un vector, y cada uno de estos vectores debe tener el mismo tamaño. Esto no siempre es posible. Por ejemplo, si tiene una colección de fotografías con diferentes anchos y altos, las diferentes fotografías contendrán diferentes números de píxeles, por lo que es posible que no todas las fotografías se describan con la misma longitud de vector. Sección [9.7](#) y capítulo [10](#) describir cómo manejar diferentes

tipos de datos tan heterogéneos. En casos como estos, en lugar de describir el conjunto de datos como una matriz con $metro$ filas, lo describiremos como un conjunto que contiene $metro$ elementos: $\{X(1), X(2), \dots, X(metro)\}$. Esta notación no implica que dos vectores de ejemplo cualesquiera $X(i)$ y $X(j)$ tienen el mismo tamaño.

En el caso del aprendizaje supervisado, el ejemplo contiene una etiqueta o un objetivo, así como una colección de características. Por ejemplo, si queremos utilizar un algoritmo de aprendizaje para realizar el reconocimiento de objetos a partir de fotografías, debemos especificar qué objeto aparece en cada una de las fotografías. Podríamos hacer esto con un código numérico, donde 0 significa una persona, 1 significa un automóvil, 2 significa un gato, etc. A menudo, cuando se trabaja con un conjunto de datos que contiene una matriz de diseño de observaciones de características X , también proporcionamos un vector de etiquetas y , con y proporcionando la etiqueta, por ejemplo i .

Por supuesto, a veces la etiqueta puede ser más que un solo número. Por ejemplo, si queremos entrenar un sistema de reconocimiento de voz para transcribir oraciones completas, entonces la etiqueta para cada oración de ejemplo es una secuencia de palabras.

Así como no existe una definición formal de aprendizaje supervisado y no supervisado, no existe una taxonomía rígida de conjuntos de datos o experiencias. Las estructuras descritas aquí cubren la mayoría de los casos, pero siempre es posible diseñar otras nuevas para nuevas aplicaciones.

5.1.4 Ejemplo: regresión lineal

Nuestra definición de un algoritmo de aprendizaje automático como un algoritmo que es capaz de mejorar el rendimiento de un programa de computadora en alguna tarea a través de la experiencia es algo abstracta. Para hacer esto más concreto, presentamos un ejemplo de un algoritmo de aprendizaje automático simple: **regresión lineal**. Volveremos a este ejemplo repetidamente a medida que presentemos más conceptos de aprendizaje automático que ayuden a comprender su comportamiento.

Como su nombre lo indica, la regresión lineal resuelve un problema de regresión. En otras palabras, el objetivo es construir un sistema que pueda tomar un vector $X \in \mathbb{R}^{norte}$ como entrada y predecir el valor de un escalar $y \in \mathbb{R}$ como su salida. En el caso de la regresión lineal, la salida es una función lineal de la entrada. Dejar \hat{y} sea el valor que predice nuestro modelo y debería asumir. Definimos la salida como

$$\hat{y} = w \cdot X \quad (5.3)$$

dónde $w \in \mathbb{R}^{norte}$ es un vector de **parámetros**.

Los parámetros son valores que controlan el comportamiento del sistema. En este caso, w es el coeficiente que multiplicamos por característica X antes de resumir las contribuciones de todas las características. podemos pensar en w como un conjunto de **pesos** que determinan cómo cada característica afecta la predicción. Si una característica X recibe un peso positivo w_i ,

luego, aumentar el valor de esa característica aumenta el valor de nuestra predicción \hat{y} . Si una característica recibe un peso negativo, al aumentar el valor de esa característica disminuye el valor de nuestra predicción. Si el peso de una característica es grande en magnitud, entonces tiene un gran efecto en la predicción. Si el peso de una característica es cero, no tiene efecto en la predicción.

Así tenemos una definición de nuestra tarea. T : predecir y de X dando salida $\hat{y} = w \cdot X$. A continuación, necesitamos una definición de nuestra medida de rendimiento, PAG .

Supongamos que tenemos una matriz de diseño de $metro$ entradas de ejemplo que no usaremos para el entrenamiento, solo para evaluar qué tan bien se desempeña el modelo. También tenemos un vector de objetivos de regresión que proporciona el valor correcto de y para cada uno de estos ejemplos. Debido a que este conjunto de datos solo se utilizará para la evaluación, lo llamamos el **equipo de prueba**. Nos referimos a la matriz de diseño de entradas como $X_{(prueba)}$ y el vector de objetivos de regresión como $y_{(prueba)}$.

Una forma de medir el rendimiento del modelo es calcular el **error medio cuadrado** del modelo en el conjunto de prueba. Si $\hat{y}_{(prueba)}$ da las predicciones del modelo en el conjunto de prueba, entonces el error cuadrático medio viene dado por

$$MSE_{prueba} = \frac{1}{metro} \sum_i (\hat{y}_{(prueba)} - y_{(prueba)})^2. \quad (5.4)$$

Intuitivamente, se puede ver que esta medida de error disminuye a 0 cuando $\hat{y}_{(prueba)} = y_{(prueba)}$. También podemos ver que

$$MSE_{prueba} = \frac{1}{metro} \| \hat{y}_{(prueba)} - y_{(prueba)} \|_2^2, \quad (5.5)$$

por lo que el error aumenta cada vez que aumenta la distancia euclídea entre las predicciones y los objetivos.

Para hacer un algoritmo de aprendizaje automático, necesitamos diseñar un algoritmo que mejore los pesos w de una manera que reduce el MSE_{prueba} cuando se permite que el algoritmo gane experiencia observando un conjunto de entrenamiento $(X_{(tren)}, y_{(tren)})$. Una forma intuitiva de hacer esto (que justificaremos más adelante, en la sección 5.5.1) es solo para minimizar el error cuadrático medio en el conjunto de entrenamiento, MSE_{tren} .

Para minimizar MSE_{tren} , podemos simplemente resolver dónde está su gradiente 0:

$$\nabla_w MSE_{tren} = 0 \quad (5.6)$$

$$\Rightarrow \nabla_w \frac{1}{metro} \| \hat{y}_{(tren)} - y_{(tren)} \|_2^2 = 0 \quad (5.7)$$

$$\Rightarrow \nabla_w \frac{1}{metro} \| X_{(tren)} w - y_{(tren)} \|_2^2 = 0 \quad (5.8)$$

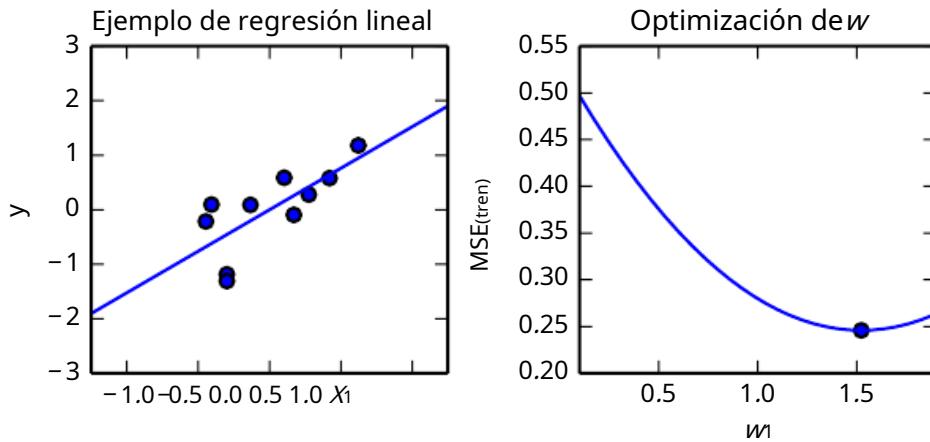


Figura 5.1: Un problema de regresión lineal, con un conjunto de entrenamiento que consta de diez puntos de datos, cada uno con una característica. Debido a que solo hay una característica, el vector de peso w contiene un solo parámetro para aprender, w . (Izquierda) Observe que la regresión lineal aprende a establecer w tal que la línea $y = w \cdot X_1$ se acerque lo más posible a pasar por todos los puntos de entrenamiento. (Bueno) El punto trazado indica el valor de w encontrado por las ecuaciones normales, que podemos ver minimiza el error cuadrático medio en el conjunto de entrenamiento.

$$\Rightarrow \nabla_w X_{\text{(tren)}}^T w - y_{\text{(tren)}} = 0 \quad (5.9)$$

$$\Rightarrow \nabla_w w^T X_{\text{(tren)}} - y_{\text{(tren)}} = 0 \quad (5.10)$$

$$\Rightarrow 2w^T X_{\text{(tren)}} - 2y_{\text{(tren)}} = 0 \quad (5.11)$$

$$\Rightarrow w = X_{\text{(tren)}}^{-1} y_{\text{(tren)}} \quad (5.12)$$

El sistema de ecuaciones cuya solución viene dada por la ecuación 5.12 es conocido como el **ecuaciones normales**. Evaluando ecuación 5.12 constituye un algoritmo de aprendizaje simple. Para ver un ejemplo del algoritmo de aprendizaje de regresión lineal en acción, consulte la figura 5.1.

Vale la pena señalar que el término **regresión lineal** se usa a menudo para referirse a un modelo un poco más sofisticado con un parámetro adicional: un término de intersección b , en este modelo

$$\hat{y} = w \cdot X + b \quad (5.13)$$

por lo tanto, el mapeo de parámetros a predicciones sigue siendo una función lineal, pero el mapeo de características a predicciones ahora es una función afín. Esta extensión a las funciones afines significa que la gráfica de las predicciones del modelo aún parece una línea, pero no es necesario que pase por el origen. En lugar de agregar el parámetro de sesgo

b , uno puede continuar usando el modelo con solo pesos pero aumentar X con una entrada adicional que siempre se establece en 1. El peso correspondiente al extra 1 entrada juega el papel del parámetro de sesgo. Usaremos con frecuencia el término “lineal” al referirnos a funciones afines a lo largo de este libro.

El término intercepto b a menudo se le llama el **parámetro de la transformación afín**. Esta terminología se deriva del punto de vista de que el resultado de la transformación está sesgado hacia ser *ben* en ausencia de cualquier entrada. Este término es diferente de la idea de un sesgo estadístico, en el que la estimación esperada de una cantidad de un algoritmo de estimación estadística no es igual a la cantidad real.

La regresión lineal es, por supuesto, un algoritmo de aprendizaje extremadamente simple y limitado, pero proporciona un ejemplo de cómo puede funcionar un algoritmo de aprendizaje. En las secciones siguientes, describiremos algunos de los principios básicos que subyacen al diseño de algoritmos de aprendizaje y demostraremos cómo se pueden utilizar estos principios para construir algoritmos de aprendizaje más complicados.

5.2 Capacidad, sobreadaptación y subadecuación

El desafío central en el aprendizaje automático es que debemos tener un buen desempeño en *nuevas*, *nunca antes visto* entradas, no solo aquellas en las que se entrenó nuestro modelo. La capacidad de desempeñarse bien en entradas previamente no observadas se llama **generalización**.

Por lo general, cuando entrenamos un modelo de aprendizaje automático, tenemos acceso a un conjunto de entrenamiento, podemos calcular alguna medida de error en el conjunto de entrenamiento llamado **error de entrenamiento**, y reducimos este error de entrenamiento. Hasta ahora, lo que hemos descrito es simplemente un problema de optimización. Lo que separa el aprendizaje automático de la optimización es que queremos que el **error de generalización**, también llamado **error de prueba**, sea bajo también. El error de generalización se define como el valor esperado del error en una nueva entrada. Aquí la expectativa se toma a través de diferentes entradas posibles, extraídas de la distribución de entradas que esperamos que el sistema encuentre en la práctica.

Por lo general, estimamos el error de generalización de un modelo de aprendizaje automático midiendo su rendimiento en un **equipo de prueba** de ejemplos que se recopilaron por separado del conjunto de entrenamiento.

En nuestro ejemplo de regresión lineal, entrenamos el modelo minimizando el error de entrenamiento,

$$\frac{1}{\text{metros(tren)}} / \|X(\text{tren})W - Y(\text{tren})\|_2^2, \quad (5.14)$$

pero en realidad nos preocupamos por el error de prueba, $\frac{1}{\text{metros(prueba)}} / \|X(\text{prueba})W - Y(\text{prueba})\|_2^2$.

¿Cómo podemos afectar el rendimiento en el conjunto de prueba cuando podemos observar solo el

¿conjunto de entrenamiento? El campo **deteoría del aprendizaje estadístico** proporciona algunas respuestas. Si el entrenamiento y el conjunto de prueba se recopilan arbitrariamente, es poco lo que podemos hacer. Si se nos permite hacer algunas suposiciones sobre cómo se recopilan los conjuntos de entrenamiento y prueba, entonces podemos hacer algún progreso.

Los datos del tren y de la prueba son generados por una distribución de probabilidad sobre conjuntos de datos llamados **proceso de generacion de datos**. Por lo general, hacemos un conjunto de suposiciones conocidas colectivamente como **elsupuestos iid**. Estas suposiciones son que los ejemplos en cada conjunto de datos son **independiente** uno del otro, y que el conjunto de tren y el conjunto de prueba son **distribuidas idénticamente**, extraídas de la misma distribución de probabilidad entre sí. Esta suposición nos permite describir el proceso de generación de datos con una distribución de probabilidad sobre un solo ejemplo. A continuación, se utiliza la misma distribución para generar cada ejemplo de tren y cada ejemplo de prueba. A esa distribución subyacente compartida la llamamos **distribución de generación de datos**, denotada p_{datos} . Este marco probabilístico y los supuestos iid nos permiten estudiar matemáticamente la relación entre el error de entrenamiento y el error de prueba.

Una conexión inmediata que podemos observar entre el entrenamiento y el error de prueba es que el error de entrenamiento esperado de un modelo seleccionado aleatoriamente es igual al error de prueba esperado de ese modelo. Supongamos que tenemos una distribución de probabilidad $p_{\text{datos}}(X, Y)$ y tomamos muestras de él repetidamente para generar el conjunto de trenes y el conjunto de prueba. Por algún valor fijo w , el error del conjunto de entrenamiento esperado es exactamente el mismo que el error del conjunto de prueba esperado, porque ambas expectativas se forman utilizando el mismo proceso de muestreo del conjunto de datos. La única diferencia entre las dos condiciones es el nombre que asignamos al conjunto de datos que muestreamos.

Por supuesto, cuando usamos un algoritmo de aprendizaje automático, no fijamos los parámetros con anticipación y luego muestreamos ambos conjuntos de datos. Probamos el conjunto de entrenamiento, luego lo usamos para elegir los parámetros para reducir el error del conjunto de entrenamiento, luego probamos el conjunto de prueba. Bajo este proceso, el error de prueba esperado es mayor o igual al valor esperado del error de entrenamiento. Los factores que determinan qué tan bien funcionará un algoritmo de aprendizaje automático son su capacidad para:

1. Haz que el error de entrenamiento sea pequeño.
2. Haga que la brecha entre el entrenamiento y el error de prueba sea pequeña.

Estos dos factores corresponden a los dos desafíos centrales en el aprendizaje automático: **desajustary sobreajuste**. El ajuste insuficiente ocurre cuando el modelo no puede obtener un valor de error suficientemente bajo en el conjunto de entrenamiento. El sobreajuste ocurre cuando la brecha entre el error de entrenamiento y el error de prueba es demasiado grande.

Podemos controlar si es más probable que un modelo se sobreajuste o no se ajuste alterando su **capacidad**. Informalmente, la capacidad de un modelo es su capacidad para adaptarse a una amplia variedad de

funciones Los modelos con poca capacidad pueden tener dificultades para adaptarse al conjunto de entrenamiento. Los modelos con alta capacidad pueden sobreajustarse al memorizar propiedades del conjunto de entrenamiento que no les sirven bien en el conjunto de prueba.

Una forma de controlar la capacidad de un algoritmo de aprendizaje es eligiendo su **espacio de hipótesis**, el conjunto de funciones que el algoritmo de aprendizaje puede seleccionar como solución. Por ejemplo, el algoritmo de regresión lineal tiene el conjunto de todas las funciones lineales de su entrada como su espacio de hipótesis. Podemos generalizar la regresión lineal para incluir polinomios, en lugar de solo funciones lineales, en su espacio de hipótesis. Si lo hace, aumenta la capacidad del modelo.

Un polinomio de grado uno nos da el modelo de regresión lineal con el que ya estamos familiarizados, con predicción

$$\hat{y} = b + wx. \quad (5.15)$$

Introduciendo x_2 como otra característica proporcionada al modelo de regresión lineal, podemos aprender un modelo que es cuadrático en función de x :

$$\hat{y} = b + w_1 x_1 + w_2 x_2. \quad (5.16)$$

Aunque este modelo implementa una función cuadrática de *su apoyo*, la salida sigue siendo una función lineal de *los parámetros*, por lo que aún podemos usar las ecuaciones normales para entrenar el modelo en forma cerrada. Podemos seguir agregando más poderes de x como características adicionales, por ejemplo para obtener un polinomio de grado 9:

$$\hat{y} = b + \sum_{i=1}^{9} w_i x_i. \quad (5.17)$$

Los algoritmos de aprendizaje automático generalmente funcionarán mejor cuando su capacidad sea adecuada para la verdadera complejidad de la tarea que deben realizar y la cantidad de datos de entrenamiento que se les proporcionan. Los modelos con capacidad insuficiente no pueden resolver tareas complejas. Los modelos con alta capacidad pueden resolver tareas complejas, pero cuando su capacidad es superior a la necesaria para resolver la tarea presente, pueden sobreajustarse.

Cifra 5.2 muestra este principio en acción. Comparamos un predictor lineal, cuadrático y de grado 9 intentando ajustar un problema donde la verdadera función subyacente es cuadrática. La función lineal no puede capturar la curvatura en el verdadero problema subyacente, por lo que no se ajusta. El predictor de grado 9 es capaz de representar la función correcta, pero también es capaz de representar infinitas funciones que pasan exactamente por los puntos de entrenamiento, porque

tienen más parámetros que ejemplos de entrenamiento. Tenemos pocas posibilidades de elegir una solución que generalice bien cuando existen tantas soluciones tremadamente diferentes. En este ejemplo, el modelo cuadrático se adapta perfectamente a la verdadera estructura de la tarea, por lo que se generaliza bien a los nuevos datos.

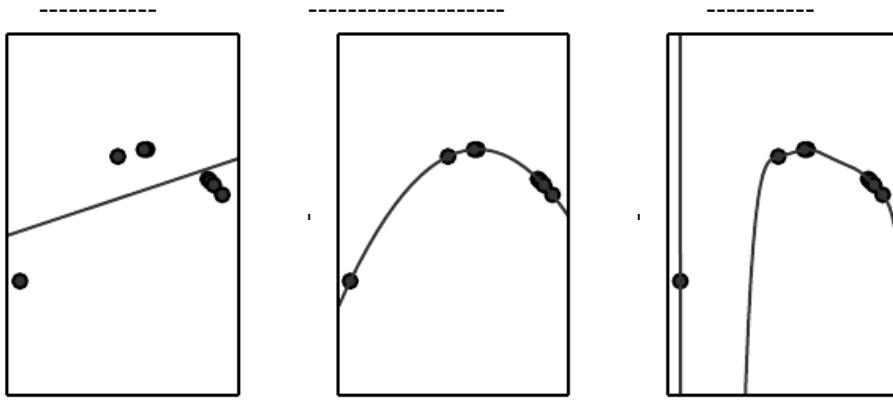


Figura 5.2: Ajustamos tres modelos a este conjunto de entrenamiento de ejemplo. Los datos de entrenamiento se generaron sintéticamente, mediante muestreo aleatorio. Xvalores y elección y determinísticamente mediante la evaluación de una función cuadrática. (Izquierda) Una función lineal ajustada a los datos sufre de ajuste insuficiente: no puede capturar la curvatura que está presente en los datos. (Centro) Una función cuadrática ajustada a los datos se generaliza bien a puntos no vistos. No sufre de una cantidad significativa de sobreajuste o desajuste. (Derecha) Un polinomio de grado 9 ajustado a los datos sufre de sobreajuste. Aquí usamos la pseudoinversa de Moore-Penrose para resolver las ecuaciones normales indeterminadas. La solución pasa exactamente por todos los puntos de entrenamiento, pero no hemos tenido la suerte de que extraiga la estructura correcta. Ahora tiene un valle profundo entre dos puntos de entrenamiento que no aparece en la verdadera función subyacente. También aumenta bruscamente en el lado izquierdo de los datos, mientras que la función verdadera disminuye en esta área.

Hasta ahora, hemos descrito solo una forma de cambiar la capacidad de un modelo: cambiando la cantidad de características de entrada que tiene y, al mismo tiempo, agregando nuevos parámetros asociados con esas características. De hecho, hay muchas formas de cambiar la capacidad de un modelo. La capacidad no está determinada únicamente por la elección del modelo. El modelo especifica qué familia de funciones puede elegir el algoritmo de aprendizaje al variar los parámetros para reducir un objetivo de entrenamiento. Esto se llama **el capacidad representativa** del modelo. En muchos casos, encontrar la mejor función dentro de esta familia es un problema de optimización muy difícil. En la práctica, el algoritmo de aprendizaje en realidad no encuentra la mejor función, sino simplemente una que reduce significativamente el error de entrenamiento. Estas limitaciones adicionales, tales como

la imperfección del algoritmo de optimización, significa que el algoritmo de aprendizaje **capacidad efectiva** puede ser inferior a la capacidad representativa de la familia modelo.

Nuestras ideas modernas sobre cómo mejorar la generalización de los modelos de aprendizaje automático son refinamientos del pensamiento que se remontan a los filósofos al menos desde Ptolomeo. Muchos de los primeros eruditos invocan un principio de parsimonia que ahora se conoce más ampliamente como **La navaja de Occam** (C. 1287-1347). Este principio establece que entre las hipótesis en competencia que explican igualmente bien las observaciones conocidas, se debe elegir la “más simple”. Esta idea fue formalizada y precisada en el siglo XX por los fundadores de la teoría del aprendizaje estadístico (**Vapnik y Chervonenkis, 1971; Vapnik, mil novecientos ochenta y dos; Blumer et al., 1989; Vapnik, 1995**).

La teoría del aprendizaje estadístico proporciona varios medios para cuantificar la capacidad del modelo. Entre estos, el más conocido es el **Dimensión Vapnik-Chervonenkis**, o dimensión VC. La dimensión VC mide la capacidad de un clasificador binario. La dimensión VC se define como el mayor valor posible de m para el cual existe un conjunto de entrenamiento de m diferentes X puntos que el clasificador puede etiquetar arbitrariamente.

Cuantificar la capacidad del modelo permite que la teoría del aprendizaje estadístico haga predicciones cuantitativas. Los resultados más importantes en la teoría del aprendizaje estadístico muestran que la discrepancia entre el error de entrenamiento y el error de generalización está limitada desde arriba por una cantidad que crece a medida que crece la capacidad del modelo, pero se reduce a medida que aumenta el número de ejemplos de entrenamiento (**Vapnik y Chervonenkis, 1971; Vapnik, mil novecientos ochenta y dos; Blumer et al., 1989; Vapnik, 1995**). Estos límites brindan una justificación intelectual de que los algoritmos de aprendizaje automático pueden funcionar, pero rara vez se usan en la práctica cuando se trabaja con algoritmos de aprendizaje profundo. Esto se debe en parte a que los límites suelen ser bastante flexibles y en parte a que puede ser bastante difícil determinar la capacidad de los algoritmos de aprendizaje profundo. El problema de determinar la capacidad de un modelo de aprendizaje profundo es especialmente difícil porque la capacidad efectiva está limitada por las capacidades del algoritmo de optimización, y tenemos poca comprensión teórica de los problemas generales de optimización no convexa involucrados en el aprendizaje profundo.

Debemos recordar que si bien es más probable que las funciones más simples se generalicen (tengan una pequeña brecha entre el entrenamiento y el error de prueba), aún debemos elegir una hipótesis lo suficientemente compleja para lograr un bajo error de entrenamiento. Por lo general, el error de entrenamiento disminuye hasta que se convierte en el valor de error mínimo posible a medida que aumenta la capacidad del modelo (suponiendo que la medida de error tiene un valor mínimo). Normalmente, el error de generalización tiene una curva en forma de U en función de la capacidad del modelo. Esto se ilustra en la figura 5.3.

Para llegar al caso más extremo de capacidad arbitrariamente alta, introducimos

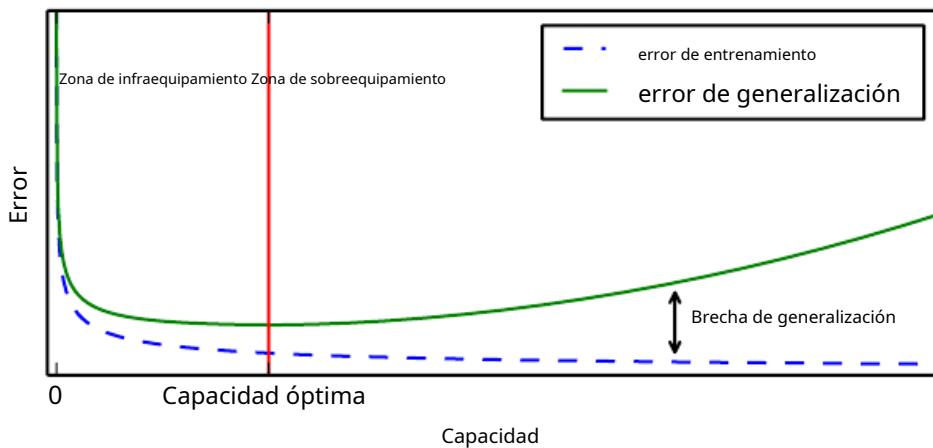


Figura 5.3: Relación típica entre capacidad y error. El entrenamiento y el error de prueba se comportan de manera diferente. En el extremo izquierdo del gráfico, el error de entrenamiento y el error de generalización son altos. Este es el **régimen de inhabilitación**. A medida que aumentamos la capacidad, el error de entrenamiento disminuye, pero aumenta la brecha entre el error de entrenamiento y el de generalización. Eventualmente, el tamaño de esta brecha supera la disminución en el error de entrenamiento, y entramos en el **régimen de sobrecondicionamiento**, donde la capacidad es demasiado grande, por encima de la **capacidad óptima**.

el concepto de **no paramétricos** modelos Hasta ahora, solo hemos visto modelos paramétricos, como la regresión lineal. Los modelos paramétricos aprenden una función descrita por un vector de parámetros cuyo tamaño es finito y fijo antes de que se observe ningún dato. Los modelos no paramétricos no tienen tal limitación.

A veces, los modelos no paramétricos son solo abstracciones teóricas (como un algoritmo que busca todas las distribuciones de probabilidad posibles) que no se pueden implementar en la práctica. Sin embargo, también podemos diseñar modelos no paramétricos prácticos haciendo que su complejidad sea una función del tamaño del conjunto de entrenamiento. Un ejemplo de tal algoritmo es **regresión del vecino más cercano**. A diferencia de la regresión lineal, que tiene un vector de pesos de longitud fija, el modelo de regresión del vecino más cercano simplemente almacena el X_i y y_i del conjunto de entrenamiento. Cuando se le pide que clasifique un punto de prueba X , el modelo busca la entrada más cercana en el conjunto de entrenamiento y devuelve el objetivo de regresión asociado. En otras palabras, $\hat{y} = y_i$ donde $i = \text{mínimo de argumento } \|X_i - X\|_2^2$.
El algoritmo también se puede generalizar a métricas de distancia distintas de la L_2 norma, como métricas de distancia aprendidas (Goldberger et al., 2005). Si se permite que el algoritmo rompa los empates promediando los valores para todos X_i que están vinculados al más cercano, entonces este algoritmo puede lograr el mínimo error de entrenamiento posible (que podría ser mayor que cero, si dos entradas idénticas están asociadas con salidas diferentes) en cualquier conjunto de datos de regresión.

Finalmente, también podemos crear un algoritmo de aprendizaje no paramétrico envolviendo un

algoritmo de aprendizaje paramétrico dentro de otro algoritmo que aumenta el número de parámetros según sea necesario. Por ejemplo, podríamos imaginar un ciclo externo de aprendizaje que cambia el grado del polinomio aprendido por regresión lineal sobre una expansión polinomial de la entrada.

El modelo ideal es un oráculo que simplemente conoce la verdadera distribución de probabilidad que genera los datos. Incluso un modelo así incurrirá en algún error en muchos problemas, porque todavía puede haber algo de ruido en la distribución. En el caso del aprendizaje supervisado, el mapeo de X a y puede ser inherentemente estocástico, o y puede ser una función determinista que involucre otras variables además de las incluidas en X . El error en que incurre un oráculo al hacer predicciones a partir de la distribución verdadera $p_{\text{true}}(X, y)$ se llama el **error bayesiano**.

El error de entrenamiento y generalización varía a medida que varía el tamaño del conjunto de entrenamiento. El error de generalización esperado nunca puede aumentar a medida que aumenta el número de ejemplos de entrenamiento. Para modelos no paramétricos, más datos producen una mejor generalización hasta que se logra el mejor error posible. Cualquier modelo paramétrico fijo con una capacidad inferior a la óptima generará una asíntota con un valor de error que excede el error de Bayes. Ver figura 5.4 para una ilustración. Tenga en cuenta que es posible que el modelo tenga una capacidad óptima y aún así tenga una gran brecha entre el entrenamiento y el error de generalización. En esta situación, es posible que podamos reducir esta brecha recopilando más ejemplos de capacitación.

5.2.1 El teorema de que no hay almuerzo gratis

La teoría del aprendizaje afirma que un algoritmo de aprendizaje automático puede generalizarse bien a partir de un conjunto finito de ejemplos de entrenamiento. Esto parece contradecir algunos principios básicos de la lógica. El razonamiento inductivo, o inferir reglas generales a partir de un conjunto limitado de ejemplos, no es lógicamente válido. Para inferir lógicamente una regla que describa cada miembro de un conjunto, uno debe tener información sobre cada miembro de ese conjunto.

En parte, el aprendizaje automático evita este problema al ofrecer solo reglas probabilísticas, en lugar de las reglas completamente seguras que se usan en el razonamiento puramente lógico. El aprendizaje automático promete encontrar reglas que sean *probablemente correcto* sobre *mayoría* miembros del conjunto al que se refieren.

Desafortunadamente, incluso esto no resuelve todo el problema. El **no hay teorema de almuerzo gratis** para el aprendizaje automático (Wolpert, 1996) establece que, promediando sobre todas las posibles distribuciones de generación de datos, cada algoritmo de clasificación tiene la misma tasa de error al clasificar puntos previamente no observados. En otras palabras, en cierto sentido, ningún algoritmo de aprendizaje automático es universalmente mejor que otro. El algoritmo más sofisticado que podemos concebir tiene el mismo promedio

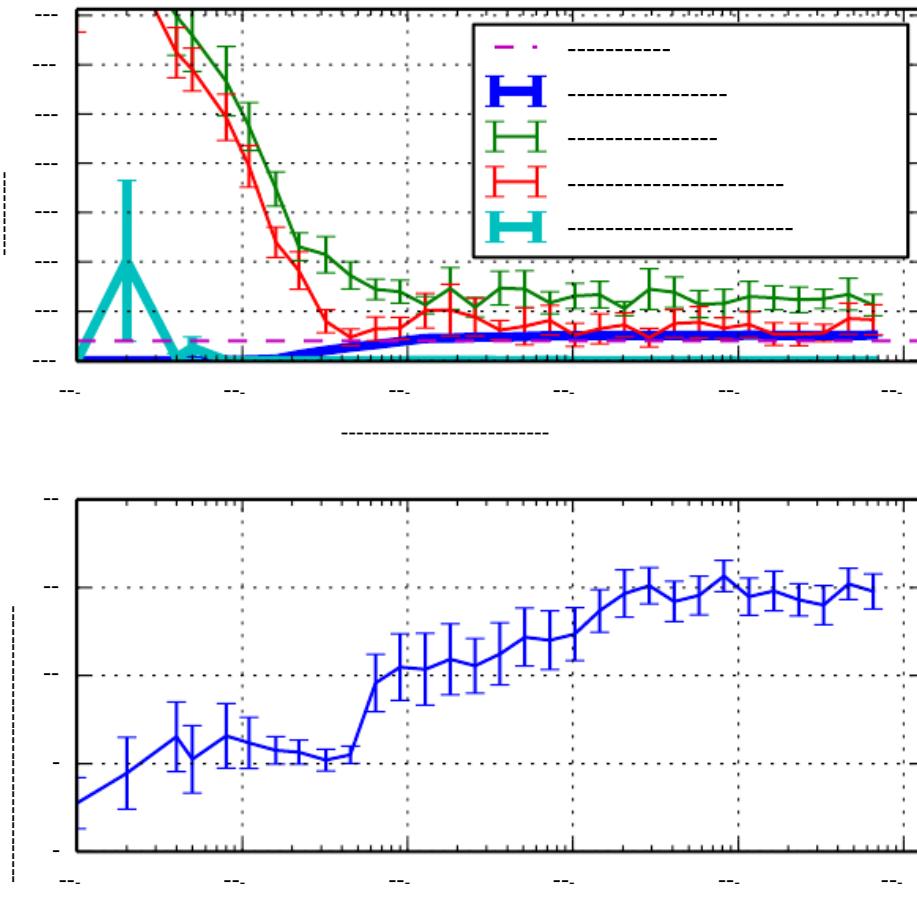


Figura 5.4: El efecto del tamaño del conjunto de datos de entrenamiento en el error de entrenamiento y prueba, así como en la capacidad óptima del modelo. Construimos un problema de regresión sintética basado en agregar una cantidad moderada de ruido a un polinomio de grado 5, generamos un solo conjunto de prueba y luego generamos varios tamaños diferentes de conjunto de entrenamiento. Para cada tamaño, generamos 40 conjuntos de entrenamiento diferentes para trazar barras de error que muestran intervalos de confianza del 95 por ciento. (Arriba) El MSE en el conjunto de entrenamiento y prueba para dos modelos diferentes: un modelo cuadrático y un modelo con grado elegido para minimizar el error de prueba. Ambos se ajustan en forma cerrada. Para el modelo cuadrático, el error de entrenamiento aumenta a medida que aumenta el tamaño del conjunto de entrenamiento. Esto se debe a que los conjuntos de datos más grandes son más difíciles de ajustar. Simultáneamente, el error de prueba disminuye, porque menos hipótesis incorrectas son consistentes con los datos de entrenamiento. El modelo cuadrático no tiene suficiente capacidad para resolver la tarea, por lo que su error de prueba arroja asíntotas a un valor alto. El error de prueba en las asíntotas de capacidad óptima para el error de Bayes. El error de entrenamiento puede caer por debajo del error de Bayes, debido a la capacidad del algoritmo de entrenamiento para memorizar instancias específicas del conjunto de entrenamiento. A medida que el tamaño de entrenamiento aumenta hasta el infinito, el error de entrenamiento de cualquier modelo de capacidad fija (aquí, / Abajo) A medida que aumenta el tamaño del conjunto de entrenamiento, aumenta la capacidad óptima (mostrada aquí como el grado del regresor polinomial óptimo). La capacidad óptima se estanca después de alcanzar la complejidad suficiente para resolver la tarea.

rendimiento (sobre todas las tareas posibles) como simplemente predecir que cada punto pertenece a la misma clase.

Afortunadamente, estos resultados solo se mantienen cuando promediamos *todos* los posibles distribuciones generadoras de datos. Si hacemos suposiciones sobre los tipos de distribuciones de probabilidad que encontramos en las aplicaciones del mundo real, podemos diseñar algoritmos de aprendizaje que funcionen bien en estas distribuciones.

Esto significa que el objetivo de la investigación del aprendizaje automático no es buscar un algoritmo de aprendizaje universal o el mejor algoritmo de aprendizaje absoluto. En cambio, nuestro objetivo es comprender qué tipos de distribuciones son relevantes para el "mundo real" que experimenta un agente de IA y qué tipos de algoritmos de aprendizaje automático funcionan bien en los datos extraídos de los tipos de distribuciones de generación de datos que nos interesan.

5.2.2 Regularización

El teorema de que no hay almuerzo gratis implica que debemos diseñar nuestros algoritmos de aprendizaje automático para que funcionen bien en una tarea específica. Lo hacemos mediante la construcción de un conjunto de preferencias en el algoritmo de aprendizaje. Cuando estas preferencias están alineadas con los problemas de aprendizaje que le pedimos al algoritmo que resuelva, funciona mejor.

Hasta ahora, el único método de modificar un algoritmo de aprendizaje que hemos discutido concretamente es aumentar o disminuir la capacidad de representación del modelo agregando o eliminando funciones del espacio de hipótesis de soluciones que el algoritmo de aprendizaje es capaz de elegir. Dimos el ejemplo específico de aumentar o disminuir el grado de un polinomio para un problema de regresión. La vista que hemos descrito hasta ahora está demasiado simplificada.

El comportamiento de nuestro algoritmo se ve fuertemente afectado no solo por cuán grande hacemos el conjunto de funciones permitidas en su espacio de hipótesis, sino también por la identidad específica de esas funciones. El algoritmo de aprendizaje que hemos estudiado hasta ahora, la regresión lineal, tiene un espacio de hipótesis que consiste en el conjunto de funciones lineales de su entrada. Estas funciones lineales pueden ser muy útiles para problemas donde la relación entre entradas y salidas realmente es casi lineal. Son menos útiles para problemas que se comportan de una manera muy no lineal. Por ejemplo, la regresión lineal no funcionaría muy bien si tratáramos de usarla para predecir $precio(X)$ de X . Por lo tanto, podemos controlar el rendimiento de nuestros algoritmos eligiendo de qué tipo de funciones les permitimos extraer soluciones, así como controlando la cantidad de estas funciones.

También podemos dar a un algoritmo de aprendizaje preferencia por una solución en su espacio de hipótesis a otra. Esto significa que ambas funciones son elegibles, pero se prefiere una. La solución no preferida se elegirá solo si se ajusta a la formación.

significativamente mejor que la solución preferida.

Por ejemplo, podemos modificar el criterio de entrenamiento para la regresión lineal para incluir **decaimiento de peso**. Para realizar una regresión lineal con caída de peso, minimizamos una suma que comprende tanto el error cuadrático medio en el entrenamiento como un criterio que expresa una preferencia por que los pesos tengan cuadrados más pequeños L_2 -norma. Específicamente,

$$J(w) = \text{MSE}_{\text{tren}} + \lambda w \cdot w, \quad (5.18)$$

dónde λ es un valor elegido con anticipación que controla la fuerza de nuestra preferencia por pesos más pequeños. Cuando $\lambda=0$, no imponemos preferencia, y más grande λ obliga a los pesos a ser más pequeños, minimizando $J(w)$ da como resultado una elección de pesos que hacen un compromiso entre ajustarse a los datos de entrenamiento y ser pequeños. Esto nos brinda soluciones que tienen una pendiente más pequeña o ponen peso en menos características. Como ejemplo de cómo podemos controlar la tendencia de un modelo a sobreajustarse o no ajustarse a través de la caída del peso, podemos entrenar un modelo de regresión polinomial de alto grado con diferentes valores de λ . Ver figura 5.5 por los resultados

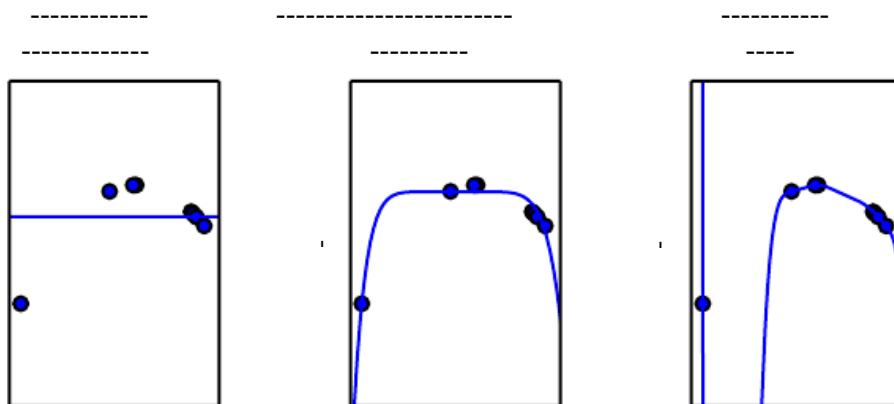


Figura 5.5: Ajustamos un modelo de regresión polinomial de alto grado a nuestro conjunto de entrenamiento de ejemplo de la figura 5.2. La verdadera función es cuadrática, pero aquí usamos solo modelos con grado 9. Variamos la cantidad de caída de peso para evitar que estos modelos de alto grado se sobreajusten. (Izquierda) Con muy grande λ , podemos obligar al modelo a aprender una función sin pendiente. Esto no se ajusta porque solo puede representar una función constante. (Centro) Con un valor medio de λ , el algoritmo de aprendizaje recupera una curva con la forma general correcta. Aunque el modelo es capaz de representar funciones con formas mucho más complicadas, la disminución del peso lo ha animado a usar una función más simple descrita por coeficientes más pequeños. (Derecha) Con el decaimiento del peso acercándose a cero (es decir, usando la pseudoinversa de Moore-Penrose para resolver el problema indeterminado con una regularización mínima), el polinomio de grado 9 se sobreajusta significativamente, como vimos en la figura 5.2.

Más generalmente, podemos regularizar un modelo que aprende una función $J(X; \theta)$ añadiendo una penalización llamada **regularizadora** la función de costo. En el caso de caída de peso, el regularizador es $\Omega(w) = w \cdot w$. en el capítulo 7, veremos que muchos otros regularizadores son posibles.

Expresar preferencias por una función sobre otra es una forma más general de controlar la capacidad de un modelo que incluir o excluir miembros del espacio de hipótesis. Podemos pensar en excluir una función de un espacio de hipótesis como expresando una preferencia infinitamente fuerte contra esa función.

En nuestro ejemplo de disminución de peso, expresamos nuestra preferencia por las funciones lineales definidas con pesos más pequeños explícitamente, a través de un término adicional en el criterio que minimizamos. Hay muchas otras formas de expresar preferencias por diferentes soluciones, tanto implícita como explícitamente. Juntos, estos diferentes enfoques se conocen como **regularización**. La regularización es cualquier modificación que hacemos a un algoritmo de aprendizaje que pretende reducir su error de generalización pero no su error de entrenamiento. La regularización es una de las preocupaciones centrales del campo del aprendizaje automático, rivalizada en su importancia solo por la optimización.

El teorema de que no hay almuerzo gratis ha dejado en claro que no existe el mejor algoritmo de aprendizaje automático y, en particular, no existe la mejor forma de regularización. En su lugar, debemos elegir una forma de regularización que se adapte bien a la tarea particular que queremos resolver. La filosofía del aprendizaje profundo en general y de este libro en particular es que una amplia gama de tareas (como todas las tareas intelectuales que las personas pueden hacer) pueden resolverse de manera efectiva utilizando formas de regularización de propósito muy general.

5.3 Hiperparámetros y Conjuntos de Validación

La mayoría de los algoritmos de aprendizaje automático tienen varias configuraciones que podemos usar para controlar el comportamiento del algoritmo de aprendizaje. Estos ajustes se llaman **hiperparámetros**. Los valores de los hiperparámetros no son adaptados por el propio algoritmo de aprendizaje (aunque podemos diseñar un procedimiento de aprendizaje anidado donde un algoritmo de aprendizaje aprende los mejores hiperparámetros para otro algoritmo de aprendizaje).

En el ejemplo de regresión polinomial que vimos en la figura 5.2, hay un solo hiperparámetro: el grado del polinomio, que actúa como un **capacidad** hiperparámetro. El valor utilizado para controlar la fuerza de la caída del peso es otro ejemplo de un hiperparámetro.

A veces, se elige una configuración para que sea un hiperparámetro que el algoritmo de aprendizaje no aprende porque es difícil de optimizar. Con mayor frecuencia, el

la configuración debe ser un hiperparámetro porque no es apropiado aprender ese hiperparámetro en el conjunto de entrenamiento. Esto se aplica a todos los hiperparámetros que controlan la capacidad del modelo. Si se aprendieran en el conjunto de entrenamiento, tales hiperparámetros siempre elegirían la máxima capacidad posible del modelo, lo que resultaría en un sobreajuste (consulte la figura 5.3). Por ejemplo, siempre podemos ajustar mejor el conjunto de entrenamiento con un polinomio de mayor grado y una configuración de caída de peso de $\lambda=0$ de lo que podríamos con un polinomio de menor grado y una configuración de caída de peso positiva.

Para resolver este problema, necesitamos un **conjunto de validación** de ejemplos que el algoritmo de entrenamiento no observa.

Anteriormente discutimos cómo un conjunto de prueba retenido, compuesto de ejemplos provenientes de la misma distribución que el conjunto de entrenamiento, puede usarse para estimar el error de generalización de un alumno, una vez que se ha completado el proceso de aprendizaje. Es importante que los ejemplos de prueba no se utilicen de ninguna manera para tomar decisiones sobre el modelo, incluidos sus hiperparámetros. Por este motivo, no se puede utilizar ningún ejemplo del conjunto de prueba en el conjunto de validación. Por lo tanto, siempre construimos el conjunto de validación a partir de la *capacitación* de los datos. Específicamente, dividimos los datos de entrenamiento en dos subconjuntos separados. Uno de estos subconjuntos se utiliza para aprender los parámetros. El otro subconjunto es nuestro conjunto de validación, utilizado para estimar el error de generalización durante o después del entrenamiento, lo que permite que los hiperparámetros se actualicen en consecuencia. El subconjunto de datos que se usa para aprender los parámetros todavía se suele denominar conjunto de entrenamiento, aunque esto puede confundirse con el grupo más grande de datos que se usa para todo el proceso de entrenamiento. El subconjunto de datos utilizado para guiar la selección de hiperparámetros se denomina conjunto de validación. Por lo general, se usa alrededor del 80 % de los datos de entrenamiento para el entrenamiento y el 20 % para la validación. Dado que el conjunto de validación se utiliza para "entrenar" los hiperparámetros, el error del conjunto de validación subestimará el error de generalización, aunque típicamente por una cantidad menor que el error de entrenamiento. Una vez completada toda la optimización de hiperparámetros, el error de generalización se puede estimar utilizando el conjunto de prueba.

En la práctica, cuando el mismo conjunto de prueba se ha utilizado repetidamente para evaluar el rendimiento de diferentes algoritmos durante muchos años, y especialmente si consideramos todos los intentos de la comunidad científica por superar el rendimiento de última generación informado en ese conjunto de prueba, terminar teniendo evaluaciones optimistas con el conjunto de prueba también. Por lo tanto, los puntos de referencia pueden volverse obsoletos y luego no reflejar el verdadero rendimiento de campo de un sistema entrenado. Afortunadamente, la comunidad tiende a pasar a conjuntos de datos de referencia nuevos (y generalmente más ambiciosos y más grandes).

5.3.1 Validación cruzada

Dividir el conjunto de datos en un conjunto de entrenamiento fijo y un conjunto de prueba fijo puede ser problemático si el conjunto de prueba es pequeño. Un conjunto de prueba pequeño implica incertidumbre estadística en torno al error de prueba promedio estimado, lo que dificulta afirmar que el algoritmo A funciona mejor que el algoritmo B sobre la tarea dada.

Cuando el conjunto de datos tiene cientos de miles de ejemplos o más, esto no es un problema grave. Cuando el conjunto de datos es demasiado pequeño, existen procedimientos alternativos que permiten utilizar todos los ejemplos en la estimación del error de prueba medio, al precio de un mayor costo computacional. Estos procedimientos se basan en la idea de repetir el cálculo de entrenamiento y prueba en diferentes subconjuntos o divisiones elegidos al azar del conjunto de datos original. El más común de estos es el k -procedimiento de validación cruzada, mostrado en el algoritmo 5.1, en el que se forma una partición del conjunto de datos al dividirlo en k subconjuntos que no se superponen. Entonces, el error de prueba se puede estimar tomando el error de prueba promedio en k juicios. En juicio i , el i -th subconjunto de los datos se usa como conjunto de prueba y el resto de los datos se usa como conjunto de entrenamiento. Un problema es que no existen estimadores insesgados de la varianza de tales estimadores de error promedio (Bengio y Grandvalet, 2004), pero normalmente se utilizan aproximaciones.

5.4 Estimadores, sesgo y varianza

El campo de la estadística nos brinda muchas herramientas que pueden usarse para lograr el objetivo de aprendizaje automático de resolver una tarea no solo en el conjunto de entrenamiento sino también para generalizar. Los conceptos fundamentales como la estimación de parámetros, el sesgo y la varianza son útiles para caracterizar formalmente las nociones de generalización, subajuste y sobreajuste.

5.4.1 Estimación puntual

La estimación puntual es el intento de proporcionar la única "mejor" predicción de alguna cantidad de interés. En general, la cantidad de interés puede ser un solo parámetro o un vector de parámetros en algún modelo paramétrico, como los pesos en nuestro ejemplo de regresión lineal en la sección 5.1.4, pero también puede ser una función completa.

Para distinguir las estimaciones de los parámetros de su valor real, nuestra convención será denotar una estimación puntual de un parámetro θ por $\hat{\theta}$.

Dejar $\{X(1), \dots, X(metro)\}$ ser un conjunto de $metro$ independientes e idénticamente distribuidas

Algoritmo 5.1 El k -fold algoritmo de validación cruzada. Se puede utilizar para estimar el error de generalización de un algoritmo de aprendizaje. Cuando el conjunto de datos dado D es demasiado pequeño para un simple tren/prueba o entrenamiento/división válida para producir una estimación precisa del error de generalización, porque la media de una pérdida L en un conjunto de prueba pequeño puede tener una variación demasiado alta. El conjunto de datos D contiene como elementos los ejemplos abstractos z_i (Para el i -ésimo ejemplo), que podría representar un par (entrada, objetivo) $z_i = (x_i, y_i)$ en el caso de aprendizaje supervisado, o solo para una entrada $z_i = x_i$ en el caso del aprendizaje no supervisado. El algoritmo devuelve el vector de errores m para cada ejemplo en D , cuya media es el error de generalización estimado. Los errores en ejemplos individuales se pueden usar para calcular un intervalo de confianza alrededor de la media (ecuación 5.47). Si bien estos intervalos de confianza no están bien justificados después del uso de la validación cruzada, todavía es una práctica común usarlos para declarar que el algoritmo A es mejor que el algoritmo B solo si el intervalo de confianza del error de algoritmo A se encuentra debajo y no interseca el intervalo de confianza del algoritmo B .

Definir K doblar XV(D, A, L, k):

Requerir: D , el conjunto de datos dado, con elementos z_i

Requerir: A , el algoritmo de aprendizaje, visto como una función que toma un conjunto de datos como entrada y salida de una función aprendida

Requerir: L , la función de pérdida, vista como una función de una función aprendida F y un ejemplo $z_i \in D$ a un escalar $\in \mathbb{R}$

Requerir: k , el número de pliegues

Dividir D en k subconjuntos mutuamente excluyentes D_i , cuya unión es D .

para i de 1 a k **hacer**

$F_i = A(D \setminus D_i)$ **para** z_i

enD **hacer**

$m_{ij} = L(F_i, z_j)$ **fin**

para

fin para

Devolver m

(iid) puntos de datos. A **estimador puntual estadística** es cualquier función de los datos:

$$\hat{\theta}_{\text{metro}} = \text{gramo}(X_{(1)}, \dots, X_{(\text{metro})}). \quad (5.19)$$

La definición no requiere que gramo devuelva un valor cercano al verdadero θ_0 incluso que el rango de gramo sea igual al conjunto de valores permisibles de θ . Esta definición de estimador puntual es muy general y permite al diseñador de un estimador una gran flexibilidad. Si bien casi cualquier función califica como un estimador, un buen estimador es una función cuya salida está cerca del verdadero valor subyacente θ que generó los datos de entrenamiento.

Por ahora, adoptamos la perspectiva frecuentista de las estadísticas. Es decir, suponemos que el verdadero valor del parámetro θ es fijo pero desconocido, mientras que la estimación puntual $\hat{\theta}$ es una función de los datos. Dado que los datos se extraen de un proceso aleatorio, cualquier función de los datos es aleatoria. Por lo tanto $\hat{\theta}$ es una variable aleatoria.

La estimación puntual también puede referirse a la estimación de la relación entre las variables de entrada y objetivo. Nos referimos a estos tipos de estimaciones puntuales como estimadores de funciones.

Estimación de funciones Como mencionamos anteriormente, a veces nos interesa realizar la estimación de funciones (o aproximación de funciones). Aquí estamos tratando de predecir una variable y dado un vector de entrada X . Suponemos que hay una función $F(X)$ que describe la relación aproximada entre y y X . Por ejemplo, podemos suponer que $y = F(X) + \epsilon$, donde ϵ representa la parte de y que no es predecible desde X . En la estimación de funciones, estamos interesados en aproximar F con un modelo o estimación \hat{F} . La estimación de funciones es realmente lo mismo que estimar un parámetro θ ; el estimador de funciones \hat{F} es simplemente un estimador puntual en el espacio de funciones. El ejemplo de regresión lineal (discutido anteriormente en la sección 5.1.4) y el ejemplo de regresión polinomial (discutido en la sección 5.2) son ejemplos de escenarios que pueden interpretarse como la estimación de un parámetro w o estimar una función \hat{F} mapeo desde X a y .

Ahora repasamos las propiedades más comúnmente estudiadas de los estimadores puntuales y discutimos lo que nos dicen acerca de estos estimadores.

5.4.2 Sesgo

El sesgo de un estimador se define como:

$$\text{inclinación}(\hat{\theta}_{\text{metro}}) = \text{MI}(\hat{\theta}_{\text{metro}}) - \theta \quad (5.20)$$

donde la expectativa está sobre los datos (vistos como muestras de una variable aleatoria) y θ es el verdadero valor subyacente de θ se utiliza para definir la distribución de generación de datos. un estimador $\hat{\theta}_{metro}$ se ha dicho **imparcial** si $inclinación(\hat{\theta}_{metro}) = 0$, lo que implica que $MI(\hat{\theta}_{metro}) = \theta$. un estimador $\hat{\theta}_{metro}$ se ha dicho **asintóticamente imparcial** si $límite_{metro \rightarrow \infty} inclinación(\hat{\theta}_{metro}) = 0$, lo que implica que $límite_{metro \rightarrow \infty} MI(\hat{\theta}_{metro}) = \theta$.

Ejemplo: distribución de Bernoulli Considere un conjunto de muestras $\{X_{(1)}, \dots, X_{(metro)}\}$ que se distribuyen de forma independiente e idéntica de acuerdo con una distribución de Bernoulli con media θ :

$$PAG(X_{(i)}; \theta) = \theta X_{(i)} (1 - \theta)(1 - X_{(i)}). \quad (5.21)$$

Un estimador común para el θ El parámetro de esta distribución es la media de las muestras de entrenamiento:

$$\hat{\theta}_{metro} = \frac{1}{metro} \sum_{i=1}^{metro} X_{(i)}. \quad (5.22)$$

Para determinar si este estimador está sesgado, podemos sustituir la ecuación 5.22 en ecuación 5.20:

$$inclinación(\hat{\theta}_{metro}) = MI[\hat{\theta}_{metro}] - \theta \quad (5.23)$$

$$= mi \left[\frac{1}{metro} \sum_{i=1}^{metro} X_{(i)} \right] - \theta \quad (5.24)$$

$$= \frac{1}{metro} \sum_{i=1}^{metro} mi X_{(i)} - \theta \quad (5.25)$$

$$= \frac{1}{metro} \sum_{i=1}^{metro} (1 - 1) - X_{(i)} \theta X_{(i)} (1 - \theta)(1 - X_{(i)}) - \theta \quad (5.26)$$

$$= \frac{1}{metro} \sum_{i=1}^{metro} (\theta) - \theta \quad (5.27)$$

$$= \theta - \theta = 0 \quad (5.28)$$

Desde $inclinación(\hat{\theta}) = 0$, decimos que nuestro estimador $\hat{\theta}$ es imparcial.

Ejemplo: Estimador de Distribución Gaussiana de la Media Ahora, considere un conjunto de muestras $\{X_{(1)}, \dots, X_{(metro)}\}$ que se distribuyen de forma independiente e idéntica de acuerdo con una distribución gaussiana $pag(X_{(i)}) = norte(X_{(i)}; \mu, \sigma^2)$, donde $i \in \{1, \dots, metro\}$.

Recuerde que la función de densidad de probabilidad gaussiana está dada por

$$p_{\text{gauss}}(x_{(i)}; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp - \frac{1}{2} \frac{(X_{(i)} - \mu)^2}{\sigma^2}. \quad (5.29)$$

Un estimador común del parámetro de la media gaussiana se conoce como el **muestra significar**:

$$\hat{\mu}_{\text{metro}} = \frac{1}{n} \sum_{i=1}^n X_{(i)} \quad (5.30)$$

Para determinar el sesgo de la media muestral, nuevamente nos interesa calcular su expectativa:

$$\text{inclinación}(\hat{\mu}_{\text{metro}}) = \text{MI}[\hat{\mu}_{\text{metro}}] - \mu \quad (5.31)$$

$$= \text{mi} \left[\frac{1}{n} \sum_{i=1}^n X_{(i)} \right] - \mu \quad (5.32)$$

$$= \frac{1}{n} \sum_{i=1}^n \text{mi}[X_{(i)}] - \mu \quad (5.33)$$

$$= \frac{1}{n} \sum_{i=1}^n \mu - \mu \quad (5.34)$$

$$= \mu - \mu = 0 \quad (5.35)$$

Por lo tanto, encontramos que la media de la muestra es un estimador no sesgado del parámetro de la media gaussiana.

Ejemplo: Estimadores de la Varianza de una Distribución Gaussiana Como ejemplo, comparamos dos estimadores diferentes del parámetro de varianza σ^2 de una distribución gaussiana. Nos interesa saber si alguno de los estimadores está sesgado.

El primer estimador de σ^2 consideramos que se conoce como el **varianza muestral**:

$$\hat{\sigma}_{\text{metro}}^2 = \frac{1}{n} \sum_{i=1}^n (X_{(i)} - \hat{\mu}_{\text{metro}})^2, \quad (5.36)$$

dónde $\hat{\mu}_{\text{metro}}$ es la media de la muestra, definida anteriormente. Más formalmente, estamos interesados en la computación

$$\text{inclinación}(\hat{\sigma}_{\text{metro}}^2) = \text{MI}[\hat{\sigma}_{\text{metro}}^2] - \sigma^2 \quad (5.37)$$

Comenzamos evaluando el término $\text{MI}[\hat{\sigma}_\text{metro}^2]$:

$$\text{MI}[\hat{\sigma}_\text{metro}^2] = \text{mi} \left(\frac{1}{m-1} \sum_{i=1}^{m-1} (X(i) - \hat{\mu}_\text{metro})^2 \right)^{-2} \quad (5.38)$$

$$= \frac{m-1}{m} \sigma^2 \quad (5.39)$$

Volviendo a la ecuación 5.37, concluimos que el sesgo de $\hat{\sigma}_\text{metro}$ es $-\sigma^2/m$. Por lo tanto, la varianza muestral es un estimador sesgado.

El varianza muestral imparcial estimador

$$\tilde{\sigma}_\text{metro}^2 = \frac{1}{m-1} \sum_{i=1}^{m-1} (X(i) - \hat{\mu}_\text{metro})^2 \quad (5.40)$$

proporciona un enfoque alternativo. Como su nombre indica, este estimador es imparcial.

Es decir, encontramos que $\text{MI}[\tilde{\sigma}_\text{metro}^2] = \sigma^2$:

$$\text{MI}[\tilde{\sigma}_\text{metro}^2] = \text{mi} \left(\frac{1}{m-1} \sum_{i=1}^{m-1} (X(i) - \hat{\mu}_\text{metro})^2 \right)^{-2} \quad (5.41)$$

$$= \frac{m}{m-1} \text{MI}[\hat{\sigma}_\text{metro}^2] \quad (5.42)$$

$$= \frac{m}{m-1} \frac{m-1}{m} \sigma^2 \quad (5.43)$$

$$= \sigma^2. \quad (5.44)$$

Tenemos dos estimadores: uno está sesgado y el otro no. Si bien los estimadores insesgados son claramente deseables, no siempre son los "mejores" estimadores. Como veremos, a menudo usamos estimadores sesgados que poseen otras propiedades importantes.

5.4.3 Varianza y error estándar

Otra propiedad del estimador que podríamos querer considerar es cuánto esperamos que varíe en función de la muestra de datos. Así como calculamos la expectativa del estimador para determinar su sesgo, podemos calcular su varianza. El diferenciade un estimador es simplemente la varianza

$$\text{Var}(\theta) \quad (5.45)$$

donde la variable aleatoria es el conjunto de entrenamiento. Alternativamente, la raíz cuadrada de la varianza se llama **Error estándar**, denotado $\text{SE}(\theta)$.

La varianza o el error estándar de un estimador proporciona una medida de cómo esperaríamos que varíe la estimación que calculamos a partir de los datos a medida que volvemos a muestrear de forma independiente el conjunto de datos del proceso de generación de datos subyacente. Así como nos gustaría que un estimador exhibiera un sesgo bajo, también nos gustaría que tuviera una varianza relativamente baja.

Cuando calculamos cualquier estadística utilizando un número finito de muestras, nuestra estimación del verdadero parámetro subyacente es incierta, en el sentido de que podríamos haber obtenido otras muestras de la misma distribución y sus estadísticas habrían sido diferentes. El grado de variación esperado en cualquier estimador es una fuente de error que queremos cuantificar.

El error estándar de la media viene dado por

$$SE(\hat{\mu}_{\text{metro}}) = \sqrt{\frac{1}{n} \sum_{i=1}^n \sigma_{x_i}^2}, \quad (5.46)$$

dónde σ_x^2 es la varianza verdadera de las muestras X_i . El error estándar a menudo se estima utilizando una estimación de σ . Desafortunadamente, ni la raíz cuadrada de la varianza de la muestra ni la raíz cuadrada del estimador insesgado de la varianza proporcionan una estimación insesgada de la desviación estándar. Ambos enfoques tienden a subestimar la verdadera desviación estándar, pero todavía se usan en la práctica. La raíz cuadrada del estimador insesgado de la varianza es una subestimación menor. Para grandes n , la aproximación es bastante razonable.

El error estándar de la media es muy útil en experimentos de aprendizaje automático. A menudo estimamos el error de generalización calculando la media muestral del error en el conjunto de prueba. El número de ejemplos en el conjunto de prueba determina la precisión de esta estimación. Aprovechando el teorema del límite central, que nos dice que la media se distribuirá aproximadamente con una distribución normal, podemos usar el error estándar para calcular la probabilidad de que la expectativa verdadera caiga en cualquier intervalo elegido. Por ejemplo, el intervalo de confianza del 95% centrado en la media $\hat{\mu}_{\text{metro}}$ es

$$(\hat{\mu}_{\text{metro}} - 1.96SE(\hat{\mu}_{\text{metro}}), \hat{\mu}_{\text{metro}} + 1.96SE(\hat{\mu}_{\text{metro}})), \quad (5.47)$$

bajo la distribución normal con media $\hat{\mu}_{\text{metro}}$ y varianza $SE(\hat{\mu}_{\text{metro}})^2$. En los experimentos de aprendizaje automático, es común decir que el algoritmo A es mejor que el algoritmo B si el límite superior del intervalo de confianza del 95% para el error de algoritmo A es menor que el límite inferior del intervalo de confianza del 95 % para el error del algoritmo B .

Ejemplo: distribución de Bernoulli Consideraremos una vez más un conjunto de muestras $\{X_1, \dots, X_{\text{metro}}\}$ extraído de forma independiente e idéntica de una distribución de Bernoulli (recordar $P(X_i; \theta) = \theta X_i (1 - \theta)^{1-X_i}$). Esta vez nos interesa la computación. la varianza del estimador $\hat{\theta}_{\text{metro}} = \frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} X_i$.

$$\text{Var} \hat{\theta}_{\text{metro}} = \text{Var} \left(\frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} X_i \right) \quad (5.48)$$

$$= \frac{1}{\text{metro}^2} \sum_{i=1}^{\text{metro}} \text{Var} (X_i) \quad (5.49)$$

$$= \frac{1}{\text{metro}^2} \sum_{i=1}^{\text{metro}} \theta(1 - \theta) \quad (5.50)$$

$$= \frac{1}{\text{metro}^2} m\theta(1 - \theta) \quad (5.51)$$

$$= \frac{1}{\text{metro}} \theta(1 - \theta) \quad (5.52)$$

La varianza del estimador disminuye en función de m , el número de ejemplos en el conjunto de datos. Esta es una propiedad común de los estimadores populares a la que volveremos cuando discutamos la consistencia (ver la sección 5.4.5).

5.4.4 Intercambio de sesgo y varianza para minimizar el error cuadrático medio

El sesgo y la varianza miden dos fuentes diferentes de error en un estimador. El sesgo mide la desviación esperada del valor verdadero de la función o parámetro. La varianza, por otro lado, proporciona una medida de la desviación del valor esperado del estimador que es probable que cause cualquier muestreo particular de los datos.

¿Qué sucede cuando se nos da a elegir entre dos estimadores, uno con más sesgo y otro con más varianza? ¿Cómo elegimos entre ellos? Por ejemplo, imagina que estamos interesados en aproximar la función que se muestra en la figura 5.2 y solo se nos ofrece la elección entre un modelo con un gran sesgo y uno que sufre de una gran varianza. ¿Cómo elegimos entre ellos?

La forma más común de negociar esta compensación es utilizar la validación cruzada. Empíricamente, la validación cruzada tiene mucho éxito en muchas tareas del mundo real. Alternativamente, también podemos comparar el **error medio cuadrado** (MSE) de las estimaciones:

$$\text{MSE} = \text{MI}[(\hat{\theta}_{\text{metro}} - \theta)^2] \quad (5.53)$$

$$= \text{Sesgo}(\hat{\theta}_{\text{metro}})^2 + \text{Var}(\hat{\theta}_{\text{metro}}) \quad (5.54)$$

El MSE mide la desviación general esperada, en un sentido de error cuadrático, entre el estimador y el valor verdadero del parámetro θ . Como se desprende de la ecuación 5.54, evaluar el MSE incorpora tanto el sesgo como la varianza. Los estimadores deseables son aquellos con MSE pequeño y estos son estimadores que logran mantener tanto su sesgo como su varianza bajo control.

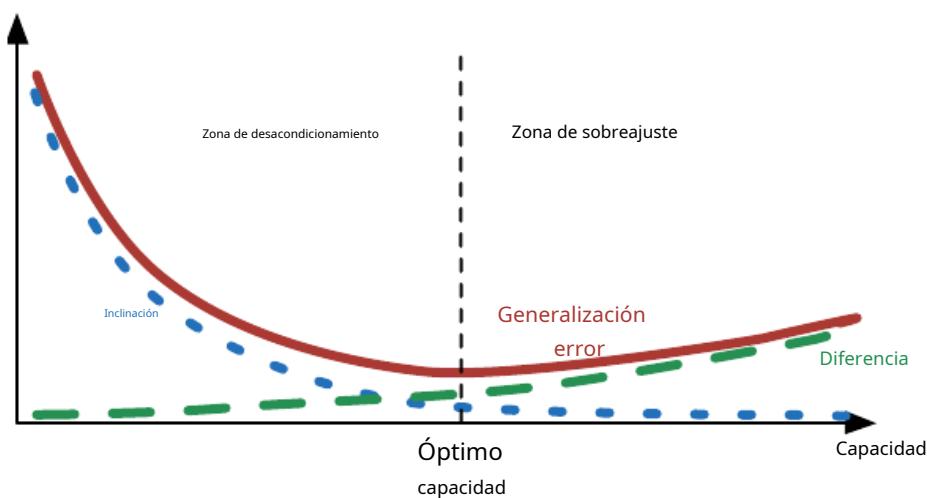


Figura 5.6: A medida que aumenta la capacidad (X -eje), el sesgo (punteado) tiende a disminuir y la varianza (discontinua) tiende a aumentar, produciendo otra curva en forma de U para el error de generalización (curva en negrita). Si variamos la capacidad a lo largo de un eje, existe una capacidad óptima, con subajuste cuando la capacidad está por debajo de este óptimo y sobreajuste cuando está por encima. Esta relación es similar a la relación entre capacidad, infraadaptación y sobreadaptación, discutida en la sección 5.2 y figura 5.3.

La relación entre el sesgo y la varianza está estrechamente vinculada a los conceptos de aprendizaje automático de capacidad, ajuste insuficiente y sobreajuste. En el caso de que el MSE mida el error de generalización (donde el sesgo y la varianza son componentes significativos del error de generalización), el aumento de la capacidad tiende a aumentar la varianza y a disminuir el sesgo. Esto se ilustra en la figura 5.6, donde vemos nuevamente la curva en forma de U del error de generalización en función de la capacidad.

5.4.5 Coherencia

Hasta ahora hemos discutido las propiedades de varios estimadores para un conjunto de entrenamiento de tamaño fijo. Por lo general, también nos preocupa el comportamiento de un estimador a medida que crece la cantidad de datos de entrenamiento. En particular, generalmente deseamos que, como el número de puntos de datos *metróen* nuestro conjunto de datos aumenta, nuestras estimaciones puntuales convergen al verdadero

valor de los parámetros correspondientes. Más formalmente, nos gustaría que

$$\text{plim}_{\text{metro} \rightarrow \infty} \hat{\theta}_{\text{metro}} = \theta. \quad (5.55)$$

El símbolo plim indica convergencia en probabilidad, lo que significa que para cualquier $\epsilon > 0$, $PAG(|\hat{\theta}_{\text{metro}} - \theta| > \epsilon) \rightarrow 0$ como $\text{metro} \rightarrow \infty$. La condición descrita por la ecuación 5.55 que se conoce como **consistencia**. A veces se le llama consistencia débil, con consistencia fuerte refiriéndose a la casi seguroconvergencia de $\hat{\theta}$ a θ . **Convergencia casi segura** de una secuencia de variables aleatorias $X_{(1)}, X_{(2)}, \dots$ a un valor X ocurre cuando $Pag(\lim_{\text{metro} \rightarrow \infty} X_{(\text{metro})} = X) = 1$.

La consistencia asegura que el sesgo inducido por el estimador disminuya a medida que crece el número de ejemplos de datos. Sin embargo, lo contrario no es cierto: la imparcialidad asintótica no implica consistencia. Por ejemplo, considere estimar el parámetro medio μ de una distribución normal $norte(X; \mu, \sigma^2)$, con un conjunto de datos compuesto por metro muestras: $\{X_{(1)}, \dots, X_{(\text{metro})}\}$. Podríamos usar la primera muestra $X_{(1)}$ del conjunto de datos como un estimador insesgado: $\hat{\theta} = X_{(1)}$. En ese caso, $MI(\hat{\theta}_{\text{metro}}) = \theta$ por lo que el estimador es imparcial sin importar cuántos puntos de datos se vean. Esto, por supuesto, implica que la estimación es asintóticamente insesgada. Sin embargo, este no es un estimador consistente ya que es *no* el caso de que $\hat{\theta}_{\text{metro}} \rightarrow \theta$ como $\text{metro} \rightarrow \infty$.

5.5 Estimación de Máxima Verosimilitud

Previamente, hemos visto algunas definiciones de estimadores comunes y analizado sus propiedades. Pero, ¿de dónde vienen estos estimadores? En lugar de suponer que alguna función podría ser un buen estimador y luego analizar su sesgo y varianza, nos gustaría tener algún principio del cual podamos derivar funciones específicas que sean buenos estimadores para diferentes modelos.

El principio más común de este tipo es el principio de máxima verosimilitud.

Considere un conjunto de metro ejemplos $X = \{X_{(1)}, \dots, X_{(\text{metro})}\}$ extraído independientemente de la distribución de generación de datos verdadera pero desconocida $Pag_{\text{datos}}(X)$.

Dejar $Pag_{\text{modelo}}(X; \theta)$ ser una familia paramétrica de distribuciones de probabilidad sobre el mismo espacio indexado por θ . En otras palabras, $Pag_{\text{modelo}}(X; \theta)$ mapea cualquier configuración X a un número real estimando la verdadera probabilidad $Pag_{\text{datos}}(X)$.

El estimador de máxima verosimilitud para θ entonces se define como

$$\theta_{ML} = \text{argumento máximo } pag_{\text{modelo}}(X; \theta) \quad (5.56)$$

$$\theta_{ML} = \arg \max_{\theta} \prod_{i=1}^{\text{metro}} pag_{\text{modelo}}(X_i; \theta) \quad (5.57)$$

Este producto con muchas probabilidades puede ser inconveniente por una variedad de razones. Por ejemplo, es propenso al subdesbordamiento numérico. Para obtener un problema de optimización más conveniente pero equivalente, observamos que tomar el logaritmo de la probabilidad no cambia su argumento máximo pero transforma convenientemente un producto en una suma:

$$\theta_{ML} = \underset{\theta}{\text{argmax}} \left[\sum_i \log p_{\text{registro}}(x_i; \theta) \right]. \quad (5.58)$$

Porque el argumento máximo cambia cuando cambiamos la escala de la función de costo, podemos dividir por metro obtener una versión del criterio que se exprese como una expectativa con respecto a la distribución empírica p_{datos} definido por los datos de entrenamiento:

$$\theta_{ML} = \underset{\theta}{\text{argmax}} \left[\text{mix}_{\sim p_{\text{datos}}} [\log p_{\text{registro}}(x; \theta)] \right]. \quad (5.59)$$

Una forma de interpretar la estimación de máxima verosimilitud es considerar que minimiza la diferencia entre la distribución empírica p_{datos} definido por el conjunto de entrenamiento y la distribución del modelo, con el grado de disimilitud entre los dos medido por la divergencia KL. La divergencia KL está dada por

$$D_{KL}(p_{\text{datos}} - p_{\text{modelo}}) = \text{mix}_{\sim p_{\text{datos}}} \left[\log p_{\text{registro}}(x; \theta) - \log p_{\text{modelo}}(x; \theta) \right]. \quad (5.60)$$

El término de la izquierda es una función únicamente del proceso de generación de datos, no del modelo. Esto significa que cuando entrenamos el modelo para minimizar la divergencia KL, solo necesitamos minimizar

$$- \text{mix}_{\sim p_{\text{datos}}} [\log p_{\text{registro}}(x; \theta)] \quad (5.61)$$

que por supuesto es lo mismo que la maximización en la ecuación 5.59.

Minimizar esta divergencia KL corresponde exactamente a minimizar la entropía cruzada entre las distribuciones. Muchos autores usan el término "entropía cruzada" para identificar específicamente la probabilidad logarítmica negativa de una distribución de Bernoulli o softmax, pero ese es un nombre inapropiado. Cualquier pérdida que consista en una verosimilitud logarítmica negativa es una entropía cruzada entre la distribución empírica definida por el conjunto de entrenamiento y la distribución de probabilidad definida por el modelo. Por ejemplo, el error cuadrático medio es la entropía cruzada entre la distribución empírica y un modelo gaussiano.

Por lo tanto, podemos ver la máxima verosimilitud como un intento de hacer que la distribución del modelo coincida con la distribución empírica p_{datos} . Idealmente, nos gustaría hacer coincidir la verdadera distribución de generación de datos p_{datos} , pero no tenemos acceso directo a esta distribución.

Mientras que lo óptimo θ es el mismo independientemente de si estamos maximizando la verosimilitud o minimizando la divergencia KL, los valores de las funciones objetivo

son diferentes. En software, a menudo expresamos que ambos minimizan una función de costo. La máxima verosimilitud se convierte así en la minimización del log-verosimilitud negativo (NLL), o de manera equivalente, la minimización de la entropía cruzada. La perspectiva de máxima verosimilitud como mínima divergencia KL se vuelve útil en este caso porque la divergencia KL tiene un valor mínimo conocido de cero. La log-verosimilitud negativa en realidad puede volverse negativa cuando λ es de valor real.

5.5.1 Log-verosimilitud condicional y error cuadrático medio

El estimador de máxima verosimilitud se puede generalizar fácilmente al caso en el que nuestro objetivo es estimar una probabilidad condicional $PAG(y/X; \theta)$ para predecir y dado X . Esta es en realidad la situación más común porque forma la base para la mayoría del aprendizaje supervisado. Si X representa todas nuestras entradas y y todos nuestros objetivos observados, entonces el estimador de máxima verosimilitud condicional es

$$\theta_{ML} = \underset{\theta}{\text{argumento máximo}} PAG(Y/X; \theta). \quad (5.62)$$

Si se supone que los ejemplos son iid, esto se puede descomponer en

$$\theta_{ML} = \underset{\theta}{\text{argumento máximo}} \prod_{i=1}^n PAG(y_i/X_i; \theta). \quad (5.63)$$

Ejemplo: regresión lineal como probabilidad máxima Regresión lineal, presentada anteriormente en la sección 5.1.4, puede justificarse como un procedimiento de máxima verosimilitud. Previamente, motivamos la regresión lineal como un algoritmo que aprende a tomar una entrada X y producir un valor de salida y . El mapeo de X a y se elige para minimizar el error cuadrático medio, un criterio que introdujimos de forma más o menos arbitraria. Ahora revisaremos la regresión lineal desde el punto de vista de la estimación de máxima verosimilitud. En lugar de producir una sola predicción y , ahora pensamos en el modelo como si produjera una distribución condicional $pag(y/X)$. Podemos imaginar que con un conjunto de entrenamiento infinitamente grande, podríamos ver varios ejemplos de entrenamiento con el mismo valor de entrada X pero diferentes valores de y . El objetivo del algoritmo de aprendizaje ahora es ajustar la distribución $pag(y/X)$ a todos esos diferentes y y valores que son todos compatibles con X . Para derivar el mismo algoritmo de regresión lineal que obtuvimos antes, definimos $pag(y/X) = norte(y; \mu(X; w), \sigma^2)$. La función $\mu(X; w)$ da la predicción de la media de la Gaussiana. En este ejemplo, suponemos que la varianza está fijada a alguna constante σ^2 elegida por el usuario. Veremos que esta elección de la forma funcional de $pag(y/X)$ hace que el procedimiento de estimación de máxima verosimilitud produzca el mismo algoritmo de aprendizaje que hemos desarrollado antes. Desde el

se supone que los ejemplos son iid, la log-verosimilitud condicional (ecuación 5.63) es dado por

$$\text{registro}_{\text{metro}} \hat{y}_i / X_i; \theta \quad (5.64)$$

$$\begin{aligned} &= -\ln \text{registro}_{\text{metro}} \hat{y}_i / X_i; \theta \\ &\quad + \frac{1}{2} \sum_{i=1}^{\text{metro}} \left(\hat{y}_i - y_i \right)^2 / 2\sigma^2, \end{aligned} \quad (5.65)$$

dónde \hat{y}_i es el resultado de la regresión lineal en el i -ésima entrada X_i y metro es el número de ejemplos de entrenamiento. Comparando el log-verosimilitud con el error cuadrático medio,

$$\text{MSE}_{\text{tren}} = \frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} \left(\hat{y}_i - y_i \right)^2 / 2, \quad (5.66)$$

inmediatamente vemos que maximizar la probabilidad logarítmica con respecto a θ produce la misma estimación de los parámetros θ igual que minimizar el error cuadrático medio. Los dos criterios tienen valores diferentes pero la misma ubicación del óptimo. Esto justifica el uso del MSE como procedimiento de estimación de máxima verosimilitud. Como veremos, el estimador de máxima verosimilitud tiene varias propiedades deseables.

5.5.2 Propiedades de Máxima Verosimilitud

El principal atractivo del estimador de máxima verosimilitud es que se puede demostrar que es el mejor estimador asintóticamente, ya que el número de ejemplos $\text{metro} \rightarrow \infty$, en términos de su tasa de convergencia como metro aumenta

Bajo condiciones apropiadas, el estimador de máxima verosimilitud tiene la propiedad de consistencia (ver sección 5.4.5 anterior), lo que significa que a medida que el número de ejemplos de entrenamiento se acerca al infinito, la estimación de máxima verosimilitud de un parámetro converge al valor verdadero del parámetro. Estas condiciones son:

- La verdadera distribución p_{datos} debe estar dentro de la familia modelo $p_{\text{modelo}}(\cdot; \theta)$. De lo contrario, ningún estimador puede recuperar p_{datos} .
- La verdadera distribución p_{datos} debe corresponder exactamente a un valor de θ . De lo contrario, la máxima verosimilitud puede recuperar la correcta p_{datos} , pero no podrá determinar qué valor de θ fue utilizado por el procesamiento de generación de datos.

Existen otros principios inductivos además del estimador de máxima verosimilitud, muchos de los cuales comparten la propiedad de ser estimadores consistentes. Sin embargo,

estimadores consistentes pueden diferir en sus **eficiencia estadística**, lo que significa que un estimador consistente puede obtener un error de generalización más bajo para un número fijo de muestras *metro*, o de manera equivalente, puede requerir menos ejemplos para obtener un nivel fijo de error de generalización.

La eficiencia estadística se estudia típicamente en el **caso paramétrico** (como en la regresión lineal) donde nuestro objetivo es estimar el valor de un parámetro (y suponiendo que sea posible identificar el verdadero parámetro), no el valor de una función. Una forma de medir qué tan cerca estamos del parámetro verdadero es mediante el error cuadrático medio esperado, calculando la diferencia cuadrática entre los valores del parámetro estimado y verdadero, donde la expectativa ha terminado. *metromuestras* de entrenamiento de la distribución de generación de datos. Ese error cuadrático medio paramétrico disminuye a medida que *metro* aumenta, y para *metro* grande, el límite inferior de Cramér-Rao ([Rao, 1945](#); [Cramér, 1946](#)) muestra que ningún estimador consistente tiene un error cuadrático medio más bajo que el estimador de máxima verosimilitud.

Por estas razones (consistencia y eficiencia), la máxima verosimilitud a menudo se considera el estimador preferido para el aprendizaje automático. Cuando el número de ejemplos es lo suficientemente pequeño como para generar un comportamiento de sobreajuste, se pueden usar estrategias de regularización como la disminución del peso para obtener una versión sesgada de máxima verosimilitud que tenga menos variación cuando los datos de entrenamiento son limitados.

5.6 Estadísticas bayesianas

Hasta ahora hemos discutido **estadísticas frecuentistas** y enfoques basados en la estimación de un único valor de θ , y luego hacer todas las predicciones basadas en esa única estimación. Otro enfoque consiste en considerar todos los valores posibles de θ al hacer una predicción. Este último es el dominio de **estadísticas bayesianas**.

Como se discutió en la sección [5.4.1](#), la perspectiva frecuentista es que el verdadero valor del parámetro θ es fijo pero desconocido, mientras que la estimación puntual $\hat{\theta}$ es una variable aleatoria debido a que es una función del conjunto de datos (que se ve como aleatorio).

La perspectiva bayesiana de las estadísticas es bastante diferente. El bayesiano utiliza la probabilidad para reflejar grados de certeza de estados de conocimiento. El conjunto de datos se observa directamente y, por lo tanto, no es aleatorio. Por otro lado, el verdadero parámetro θ es desconocida o incierta y, por lo tanto, se representa como una variable aleatoria.

Antes de observar los datos, representamos nuestro conocimiento de θ utilizando el **distribución de probabilidad previa**, $p_{\text{ag}}(\theta)$ (a veces denominado simplemente “el anterior”). Generalmente, el practicante de aprendizaje automático selecciona una distribución previa que es bastante amplia (es decir, con alta entropía) para reflejar un alto grado de incertidumbre en el

valor de θ antes de observar cualquier dato. Por ejemplo, se podría suponer *a priori* que θ se encuentra en un rango o volumen finito, con una distribución uniforme. En cambio, muchos priores reflejan una preferencia por soluciones "más simples" (como coeficientes de menor magnitud o una función que está más cerca de ser constante).

Ahora considere que tenemos un conjunto de muestras de datos $\{X(1), \dots, X(metro)\}$. Podemos recuperar el efecto de los datos sobre nuestra creencia acerca de θ combinando la probabilidad de los datos $pag(X(1), \dots, X(metro) / \theta)$ con el previo vía la regla de Bayes:

$$pag(\theta / X(1), \dots, X(metro)) = \frac{pag(X(1), \dots, X(metro) / \theta) pag(\theta)}{pag(X(1), \dots, X(metro))} \quad (5.67)$$

En los escenarios en los que se suele utilizar la estimación bayesiana, la distribución anterior comienza como una distribución relativamente uniforme o gaussiana con alta entropía, y la observación de los datos suele provocar que la posterior pierda entropía y se concentre en unos pocos valores muy probables de los parámetros.

En relación con la estimación de máxima verosimilitud, la estimación bayesiana ofrece dos diferencias importantes. Primero, a diferencia del enfoque de máxima verosimilitud que hace predicciones utilizando una estimación puntual de θ , el enfoque bayesiano es hacer predicciones utilizando una distribución completa sobre θ . Por ejemplo, después de observar $metro$ ejemplos, el distribución predicha sobre la siguiente muestra de datos, $X(metro+1)$, es dado por

$$pag(X(metro+1) / X(1), \dots, X(metro)) = \int pag(X(metro+1) / \theta) pag(\theta / X(1), \dots, X(metro)) d\theta. \quad (5.68)$$

Aquí cada valor de θ con densidad de probabilidad positiva contribuye a la predicción del siguiente ejemplo, con la contribución ponderada por la propia densidad posterior. Después de haber observado $\{X(1), \dots, X(metro)\}$, si todavía estamos bastante inseguros sobre el valor de θ , entonces esta incertidumbre se incorpora directamente a cualquier predicción que podamos hacer.

En la sección 5.4, discutimos cómo el enfoque frecuentista aborda la incertidumbre en una estimación puntual dada de θ evaluando su varianza. La varianza del estimador es una evaluación de cómo podría cambiar la estimación con muestreos alternativos de los datos observados. La respuesta bayesiana a la pregunta de cómo lidiar con la incertidumbre en el estimador es simplemente integrar sobre ella, lo que tiende a proteger bien contra el sobreajuste. Esta integral es, por supuesto, solo una aplicación de las leyes de probabilidad, lo que hace que el enfoque bayesiano sea fácil de justificar, mientras que la maquinaria frecuentista para construir un estimador se basa en la decisión más bien ad hoc de resumir todo el conocimiento contenido en el conjunto de datos con un solo punto. estimar.

La segunda diferencia importante entre el enfoque bayesiano de estimación y el enfoque de máxima verosimilitud se debe a la contribución del enfoque bayesiano

distribución previa. El anterior tiene una influencia al cambiar la densidad de masa de probabilidad hacia las regiones del espacio de parámetros que se prefieren *a priori*. En la práctica, el anterior a menudo expresa una preferencia por modelos que son más simples o más fluidos. Los críticos del enfoque bayesiano identifican lo anterior como una fuente de juicio humano subjetivo que afecta las predicciones.

Los métodos bayesianos generalmente se generalizan mucho mejor cuando hay datos de entrenamiento limitados disponibles, pero normalmente tienen un alto costo computacional cuando la cantidad de ejemplos de entrenamiento es grande.

Ejemplo: regresión lineal bayesiana Aquí consideramos el enfoque de estimación bayesiano para aprender los parámetros de regresión lineal. En la regresión lineal, aprendemos un mapeo lineal de un vector de entrada $X \in \mathbb{R}^{n_{\text{tren}}}$ predecir el valor de un escalar $y \in \mathbb{R}$. La predicción está parametrizada por el vector $w \in \mathbb{R}^{n_{\text{tren}}}$:

$$\hat{y} = w \cdot X. \quad (5.69)$$

Dado un conjunto de *metromuestras* de entrenamiento $(X_{\text{tren}}, y_{\text{tren}})$, podemos expresar la predicción de y sobre todo el conjunto de entrenamiento como:

$$\hat{y}_{\text{tren}} = X_{\text{tren}} w. \quad (5.70)$$

Expresado como una distribución condicional gaussiana en y_{tren} , tenemos

$$p_{\text{ag}}(y_{\text{tren}} | X_{\text{tren}}, w) = \text{norte}(y_{\text{tren}}; X_{\text{tren}} w, \text{Wisconsin}) \quad (5.71)$$

$$\propto \text{Exp} \left(-\frac{1}{2} (y_{\text{tren}} - X_{\text{tren}} w)^T (y_{\text{tren}} - X_{\text{tren}} w) \right), \quad (5.72)$$

donde seguimos la formulación MSE estándar al suponer que la varianza gaussiana en y es uno. En lo que sigue, para reducir la carga notacional, nos referimos a $(X_{\text{tren}}, y_{\text{tren}})$ como simplemente (X, y) .

Para determinar la distribución posterior sobre el vector de parámetros del modelo w , primero necesitamos especificar una distribución previa. Lo anterior debería reflejar nuestra creencia ingenua sobre el valor de estos parámetros. Si bien a veces es difícil o antinatural expresar nuestras creencias previas en términos de los parámetros del modelo, en la práctica generalmente asumimos una distribución bastante amplia que expresa un alto grado de incertidumbre acerca de θ . Para parámetros de valor real, es común usar una Gaussiana como distribución previa:

$$p_{\text{ag}}(w) = \text{norte}(w; \mu_0, \Lambda_0) \propto \text{Exp} \left(-\frac{1}{2} (w - \mu_0)^T \Lambda_0^{-1} (w - \mu_0) \right), \quad (5.73)$$

dónde μ_0 y Λ_0 son el vector medio de distribución previa y la matriz de covarianza respectivamente.¹

Con el previo así especificado, ahora podemos proceder a determinar el **posterior** distribución sobre los parámetros del modelo.

$$pag(w | X, y) \propto pag(y | X) pag(w) \quad (5.74)$$

$$\propto \text{Exp} \left(-\frac{1}{2}(y - Xw)^T (y - Xw) \right) \text{Exp} \left(-\frac{1}{2}(w - \mu)^T \Lambda^{-1} (\mu_0 - \mu) \right) \quad (5.75)$$

$$\propto \text{Exp} \left(-\frac{1}{2} (-2y^T X w + w^T X^T X w + w^T \Lambda^{-1}) \right) \quad (5.76)$$

Ahora definimos $\Lambda_{metro} = X^T X + \Lambda^{-1}$, $y \mu_{metro} = \Lambda_{metro}^{-1} X^T y + \Lambda^{-1} \mu_0$. Usando estas nuevas variables, encontramos que la posterior se puede reescribir como una distribución gaussiana:

$$pag(w | X, y) \propto \text{Exp} \left(-\frac{1}{2}(w - \mu_{metro})^T \Lambda_{metro} (w - \mu_{metro}) + \frac{1}{2} \mu_{metro}^T \Lambda_{metro}^{-1} \mu_{metro} \right) \quad (5.77)$$

$$\propto \text{Exp} \left(-\frac{1}{2}(w - \mu_{metro})^T \Lambda_{metro} (w - \mu_{metro}) \right). \quad (5.78)$$

Todos los términos que no incluyen el vector de parámetros w han sido omitidos; están implícitos por el hecho de que la distribución debe normalizarse para integrar a 1.

Ecuación 3.23 muestra cómo normalizar una distribución gaussiana multivariante.

Examinar esta distribución posterior nos permite ganar algo de intuición para la efecto de la inferencia bayesiana. En la mayoría de las situaciones, establecemos $\mu_0 = 0$. Si establecemos $\Lambda_0 = \alpha I_n$, entonces μ_{metro} da la misma estimación de w igual que la regresión lineal frecuentista con una penalización por caída de peso de αw . Una diferencia es que la estimación bayesiana no está definida si α se establece en cero: no se nos permite comenzar el proceso de aprendizaje bayesiano con un antecedente infinitamente amplio en w . La diferencia más importante es que la estimación bayesiana proporciona una matriz de covarianza, que muestra la probabilidad de que todos los diferentes valores de w son, en lugar de proporcionar sólo la estimación μ_{metro} .

5.6.1 Máximo Posteriormente(MAPA) Estimación

Si bien el enfoque más basado en principios es hacer predicciones utilizando la distribución posterior bayesiana completa sobre el parámetro θ , todavía es a menudo deseable tener un

¹A menos que haya una razón para suponer una estructura de covarianza particular, generalmente asumimos una matriz de covarianza diagonal $\Lambda_0 = \text{diag}(\lambda_0)$.

estimación de un solo punto. Una razón común para desear una estimación puntual es que la mayoría de las operaciones que involucran la parte posterior bayesiana para la mayoría de los modelos interesantes son intratables, y una estimación puntual ofrece una aproximación manejable. En lugar de simplemente volver a la estimación de máxima verosimilitud, aún podemos obtener algunos de los beneficios del enfoque bayesiano al permitir que el anterior influya en la elección de la estimación puntual. Una forma racional de hacer esto es elegir el **máximo a posteriori**(MAP) estimación puntual. La estimación MAP elige el punto de máxima probabilidad posterior (o máxima densidad de probabilidad en el caso más común de continuo θ):

$$\theta_{\text{MAP}} = \underset{\theta}{\text{argmax}} \, p(\theta / X) = \underset{\theta}{\text{argmax}} \, p(x / \theta) + \underset{\theta}{\text{argmax}} \, p(\theta). \quad (5.79)$$

Reconocemos, arriba en el lado derecho, $p(x / \theta)$, es decir, el término estándar de logverosimilitud, y $p(\theta)$, correspondiente a la distribución anterior.

Como ejemplo, considere un modelo de regresión lineal con un gaussiano previo en los pesos w . Si este previo está dado por $n(\mu, \sigma^2)$, entonces el término logarítmico anterior en ecuación 5.79 es proporcional a lo familiar λw -penalización por pérdida de peso, más un término que no depende de w y no afecta el proceso de aprendizaje. La inferencia bayesiana de MAP con un previo gaussiano en los pesos corresponde, por lo tanto, a la caída del peso.

Al igual que con la inferencia bayesiana completa, la inferencia bayesiana de MAP tiene la ventaja de aprovechar la información que trae el anterior y que no se puede encontrar en los datos de entrenamiento. Esta información adicional ayuda a reducir la varianza en la estimación puntual MAP (en comparación con la estimación ML). Sin embargo, lo hace al precio de un mayor sesgo.

Muchas estrategias de estimación regularizadas, como el aprendizaje de máxima verosimilitud regularizado con decaimiento de peso, pueden interpretarse como una aproximación MAP a la inferencia bayesiana. Esta visión se aplica cuando la regularización consiste en agregar un término extra a la función objetivo que corresponde a $p(\theta)$. No todas las penalizaciones de regularización corresponden a la inferencia MAP Bayesiana. Por ejemplo, algunos términos del regularizador pueden no ser el logaritmo de una distribución de probabilidad. Otros términos de regularización dependen de los datos, lo que por supuesto no se permite hacer con una distribución de probabilidad previa.

La inferencia bayesiana de MAP proporciona una forma sencilla de diseñar términos de regularización complicados pero interpretables. Por ejemplo, se puede derivar un término de penalización más complicado usando una mezcla de gaussianas, en lugar de una única distribución gaussiana, como anterior ([Nowlan y Hinton, 1992](#)).

5.7 Algoritmos de aprendizaje supervisado

Recuperar de la sección 5.1.3 que los algoritmos de aprendizaje supervisado son, en términos generales, algoritmos de aprendizaje que aprenden a asociar alguna entrada con alguna salida, dado un conjunto de entrenamiento de ejemplos de entradas X y salidas y . En muchos casos las salidas y puede ser difícil de recopilar automáticamente y debe ser proporcionado por un "supervisor" humano, pero el término aún se aplica incluso cuando los objetivos del conjunto de entrenamiento se recopilaron automáticamente.

5.7.1 Aprendizaje supervisado probabilístico

La mayoría de los algoritmos de aprendizaje supervisado de este libro se basan en la estimación de una distribución de probabilidad $p(y | X)$. Podemos hacer esto simplemente usando la estimación de máxima verosimilitud para encontrar el mejor vector de parámetros θ para una familia paramétrica de distribuciones $p(y | X; \theta)$.

Ya hemos visto que la regresión lineal corresponde a la familia

$$p(y | X; \theta) = \text{norte}(y; \theta \cdot x + \theta_0). \quad (5.80)$$

Podemos generalizar la regresión lineal al escenario de clasificación definiendo una familia diferente de distribuciones de probabilidad. Si tenemos dos clases, clase 0 y clase 1, entonces solo necesitamos especificar la probabilidad de una de estas clases. La probabilidad de la clase 1 determina la probabilidad de la clase 0, porque estos dos valores deben sumar 1.

La distribución normal sobre números con valores reales que usamos para la regresión lineal está parametrizada en términos de una media. Cualquier valor que proporcionemos para esta media es válido. Una distribución sobre una variable binaria es un poco más complicada, porque su media siempre debe estar entre 0 y 1. Una forma de resolver este problema es usar la función logística sigmoida para aplastar la salida de la función lineal en el intervalo (0, 1) e interpretar ese valor como una probabilidad:

$$p(y=1 | X; \theta) = \sigma(\theta \cdot X). \quad (5.81)$$

Este enfoque se conoce como **Regresión logística** (un nombre algo extraño ya que usamos el modelo para clasificación en lugar de regresión).

En el caso de la regresión lineal, pudimos encontrar los pesos óptimos resolviendo las ecuaciones normales. La regresión logística es algo más difícil. No existe una solución de forma cerrada para sus pesos óptimos. En su lugar, debemos buscarlos maximizando la probabilidad logarítmica. Podemos hacer esto minimizando la probabilidad logarítmica negativa (NLL) mediante el descenso de gradiente.

Esta misma estrategia se puede aplicar esencialmente a cualquier problema de aprendizaje supervisado, escribiendo una familia paramétrica de distribuciones de probabilidad condicional sobre el tipo correcto de variables de entrada y salida.

5.7.2 Máquinas de vectores de soporte

Uno de los enfoques más influyentes para el aprendizaje supervisado es la máquina de vectores de soporte ([Boser et al., 1992](#); [Cortés y Vapnik, 1995](#)). Este modelo es similar a la regresión logística en que está impulsado por una función lineal $w \cdot X + b$. A diferencia de la regresión logística, la máquina de vectores de soporte no proporciona probabilidades, sino que solo genera una identidad de clase. La SVM predice que la clase positiva está presente cuando $w \cdot X + b$ es positivo. Asimismo, predice que la clase negativa está presente cuando $w \cdot X + b$ es negativo

Una innovación clave asociada con las máquinas de vectores de soporte es la **truco del núcleo**. El truco del núcleo consiste en observar que muchos algoritmos de aprendizaje automático se pueden escribir exclusivamente en términos de productos punto entre ejemplos. Por ejemplo, se puede demostrar que la función lineal utilizada por la máquina de vectores de soporte puede ser reescrito como

$$w \cdot X + b = b + \sum_{i=1}^m a_i X^{(i)} \quad (5.82)$$

dónde $X^{(i)}$ es un ejemplo de entrenamiento y a_i es un vector de coeficientes. Reescribir el algoritmo de aprendizaje de esta manera nos permite reemplazar X por la salida de una función característica dada $\varphi(X)$ y el producto escalar con una función $k(x, x^{(i)}) = \varphi(x) \cdot \varphi(x^{(i)})$ llamado **núcleo**. El operador \cdot representa un producto interno análogo a $\varphi(x) \cdot \varphi(x^{(i)})$. Para algunos espacios de características, es posible que no usemos literalmente el producto interno del vector. En algunos espacios de dimensión infinita, necesitamos usar otros tipos de productos internos, por ejemplo, productos internos basados en la integración en lugar de la suma. Un desarrollo completo de este tipo de productos internos está más allá del alcance de este libro.

Después de reemplazar los productos punto con evaluaciones del núcleo, podemos hacer predicciones usando la función

$$f(X) = b + \sum_i a_i k(x, x^{(i)}). \quad (5.83)$$

Esta función es no lineal con respecto a X , pero la relación entre $\varphi(X)$ y $f(X)$ es lineal. Asimismo, la relación entre a_i y $f(X)$ es lineal. La función basada en el kernel es exactamente equivalente a preprocesar los datos aplicando $\varphi(x)$ a todas las entradas, luego aprendiendo un modelo lineal en el nuevo espacio transformado.

El truco del kernel es poderoso por dos razones. Primero, nos permite aprender modelos que no son lineales en función de X usando técnicas de optimización convexa que son

garantizado para converger de manera eficiente. Esto es posible porque consideramos φ parreglado y optimizado solamente a, es decir, el algoritmo de optimización puede ver la función de decisión como lineal en un espacio diferente. En segundo lugar, la función del kernel k a menudo admite una implementación que es significativamente más eficiente computacionalmente que construir ingenuamente dos $\varphi(X)$ vectores y tomando explícitamente su producto escalar.

En algunos casos, $\varphi(X)$ puede incluso tener una dimensión infinita, lo que daría como resultado un costo computacional infinito para el enfoque ingenuo y explícito. En muchos casos, $k(x, x_i)$ es una función no lineal y manejable de X incluso cuando $\varphi(X)$ es intratable. Como ejemplo de un espacio de características de dimensión infinita con un kernel manejable, construimos un mapeo de características $\varphi(X)$ sobre los enteros no negativos X . Suponga que este mapeo devuelve un vector que contiene X unos seguidos de infinitos ceros. Podemos escribir una función kernel $k(x, x_i) = \min(x, x_i)$ eso es exactamente equivalente al correspondiente producto punto de dimensión infinita.

El núcleo más utilizado es el **Núcleo gaussiano**

$$k(tu, v) = norte(tu - v; 0, \sigma^2 I) \quad (5.84)$$

dónde $norte(X; \mu, \Sigma)$ es la densidad normal estándar. Este núcleo también se conoce como **elfuncion de base radial** (RBF), porque su valor disminuye a lo largo de las líneas en v espacio que irradian hacia afuera desde tu . El núcleo gaussiano corresponde a un producto escalar en un espacio de dimensión infinita, pero la derivación de este espacio es menos directa que en nuestro ejemplo del minnúcleo sobre los enteros.

Podemos pensar en el núcleo gaussiano como si realizara una especie de **comparación de plantillas**. Un ejemplo de entrenamiento X asociado con la etiqueta de entrenamiento y se convierte en una plantilla para la clase y . Cuando un punto de prueba x está cerca de acuerdo con la distancia euclíadiana, el núcleo gaussiano tiene una gran respuesta, lo que indica que x es muy similar a la X plantilla. Luego, el modelo pone un gran peso en la etiqueta de entrenamiento asociada y . En general, la predicción combinará muchas de estas etiquetas de entrenamiento ponderadas por la similitud de los ejemplos de entrenamiento correspondientes.

Las máquinas de vectores de soporte no son el único algoritmo que se puede mejorar usando el truco del kernel. Muchos otros modelos lineales se pueden mejorar de esta manera. La categoría de algoritmos que emplean el truco del kernel se conoce como **máquinas del núcleo** o **métodos del núcleo** (Williams y Rasmussen, 1996; Scholkopf et al., 1999).

Un gran inconveniente de las máquinas kernel es que el costo de evaluar la función de decisión es lineal en el número de ejemplos de entrenamiento, porque el i -ésimo ejemplo aporta un término $a_i k(x, x_i)$ a la función de decisión. Las máquinas de vectores de soporte pueden mitigar esto aprendiendo un vector que contiene principalmente ceros. Clasificar un nuevo ejemplo requiere evaluar la función del kernel solo para los ejemplos de entrenamiento que tienen un valor distinto de cero. a_i . Estos ejemplos de entrenamiento son conocidos

como **Vectores de apoyo**.

Las máquinas kernel también sufren un alto costo computacional de entrenamiento cuando el conjunto de datos es grande. Revisaremos esta idea en la sección 5.9. Las máquinas kernel con kernels genéricos luchan por generalizar bien. Explicaremos por qué en la sección 5.11. La encarnación moderna del aprendizaje profundo se diseñó para superar estas limitaciones de las máquinas kernel. El renacimiento actual del aprendizaje profundo comenzó cuando [Hinton et al.](#) (2006) demostró que una red neuronal podría superar a la SVM del kernel RBF en el punto de referencia MNIST.

5.7.3 Otros algoritmos simples de aprendizaje supervisado

Ya hemos encontrado brevemente otro algoritmo de aprendizaje supervisado no probabilístico, la regresión del vecino más cercano. Más generalmente, k -vecinos más cercanos es una familia de técnicas que se pueden utilizar para la clasificación o la regresión. Como un algoritmo de aprendizaje no paramétrico, k -Los vecinos más cercanos no están restringidos a un número fijo de parámetros. Solemos pensar en el k -Algoritmo de vecinos más cercanos que no tiene ningún parámetro, sino que implementa una función simple de los datos de entrenamiento. De hecho, ni siquiera existe realmente una etapa de formación o proceso de aprendizaje. En cambio, en el momento de la prueba, cuando queremos producir una salida para una nueva entrada de prueba X , encontramos los k -vecinos más cercanos a X en los datos de entrenamiento X . Luego devolvemos el promedio de los correspondientes y valores en el conjunto de entrenamiento. Esto funciona esencialmente para cualquier tipo de aprendizaje supervisado donde podemos definir un promedio sobre y valores. En el caso de la clasificación, podemos promediar sobre vectores de código one-hot C con $C_j=1$ y $C_i=0$ para todos los demás valores de i . Entonces podemos interpretar el promedio de estos códigos de una sola caliente como una distribución de probabilidad sobre las clases. Como un algoritmo de aprendizaje no paramétrico, k -El vecino más cercano puede lograr una capacidad muy alta. Por ejemplo, supongamos que tenemos una tarea de clasificación multiclas y medimos el rendimiento con una pérdida de 0-1. En esta configuración, 1-el vecino más cercano converge para duplicar el error de Bayes a medida que el número de ejemplos de entrenamiento se acerca al infinito. El error superior al error de Bayes resulta de elegir un solo vecino rompiendo aleatoriamente los lazos entre vecinos igualmente distantes. Cuando hay datos de entrenamiento infinitos, todos los puntos de prueba X tendrán infinitos vecinos del conjunto de entrenamiento en la distancia cero. Si permitimos que el algoritmo use todos estos vecinos para votar, en lugar de elegir uno de ellos al azar, el procedimiento converge a la tasa de error de Bayes. La alta capacidad de k -los vecinos más cercanos le permiten obtener una alta precisión dado un gran conjunto de entrenamiento. Sin embargo, lo hace a un alto costo computacional y puede generalizarse muy mal dado un conjunto de entrenamiento pequeño y finito. Una debilidad de k -vecinos más cercanos es que no puede aprender que una característica es más discriminatoria que otra. Por ejemplo, imagina que tenemos una tarea de regresión con $X \in \mathbb{R}^{100}$ extraído de una Gaussiana isotrópica

distribución, pero sólo una variable X_1 es relevante para la salida. Supongamos además que esta función simplemente codifica la salida directamente, es decir, que $y = X_1$ en todos los casos. La regresión del vecino más cercano no podrá detectar este patrón simple. El vecino más cercano de la mayoría de los puntos X vendrá determinado por la gran cantidad de características X_2 a través de X_{100} , no por la característica solitaria X_1 . Por lo tanto, la salida en pequeños conjuntos de entrenamiento será esencialmente aleatoria.

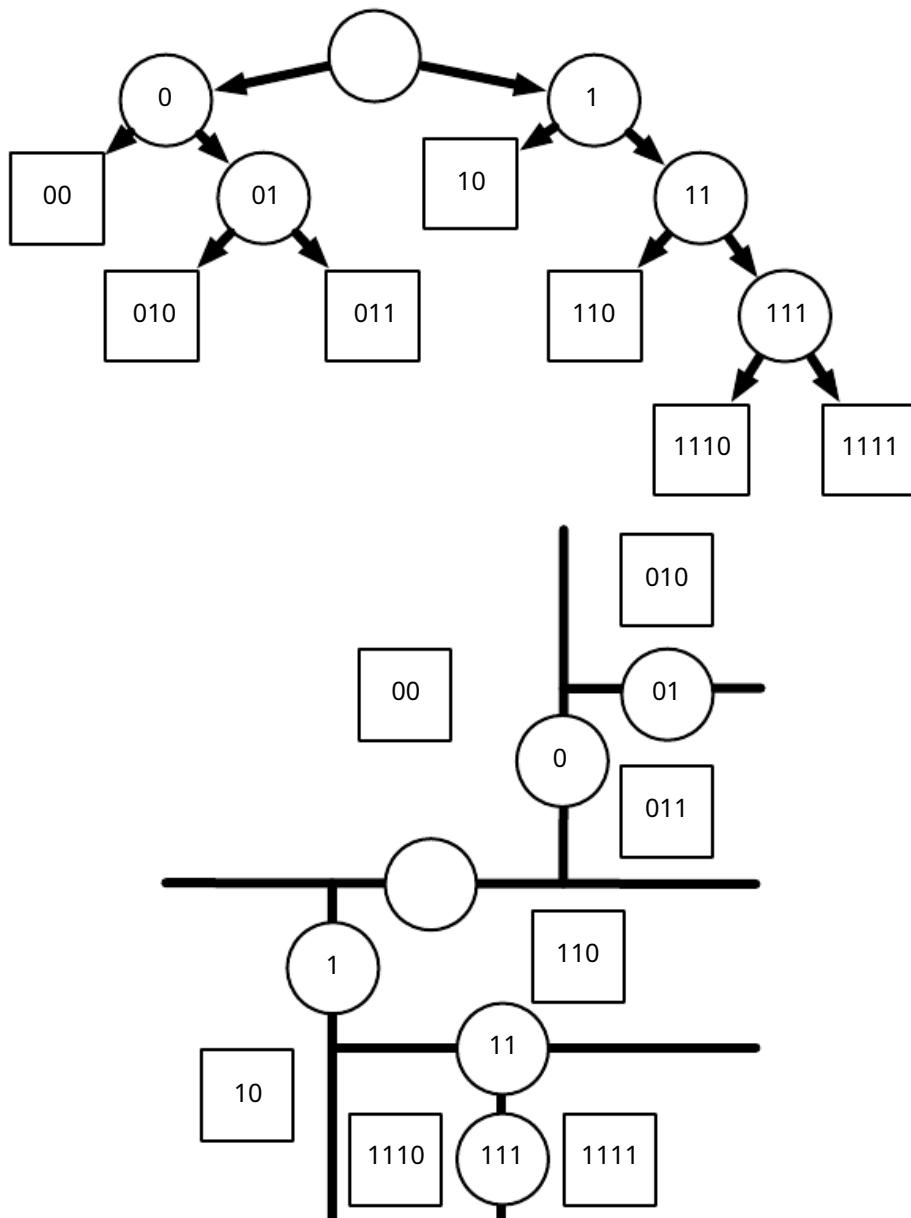


Figura 5.7: Diagramas que describen cómo funciona un árbol de decisión.(Arriba)Cada nodo del árbol elige enviar el ejemplo de entrada al nodo secundario de la izquierda (0) o al nodo secundario de la derecha (1). Los nodos internos se dibujan como círculos y los nodos de hoja como cuadrados. Cada nodo se muestra con un identificador de cadena binaria correspondiente a su posición en el árbol, obtenido agregando un bit a su identificador principal (0 = elegir izquierda o arriba, 1 = elegir derecha o abajo). (Abajo)El árbol divide el espacio en regiones. El plano 2D muestra cómo se podría dividir un árbol de decisiónR₂. Los nodos del árbol se trazan en este plano, con cada nodo interno dibujado a lo largo de la línea divisoria que usa para categorizar los ejemplos, y los nodos de hoja dibujados en el centro de la región de ejemplos que reciben. El resultado es una función constante por partes, con una pieza por hoja. Cada hoja requiere al menos un ejemplo de entrenamiento para definir, por lo que no es posible que el árbol de decisión aprenda una función que tenga más máximos locales que el número de ejemplos de entrenamiento.

Otro tipo de algoritmo de aprendizaje que también divide el espacio de entrada en regiones y tiene parámetros separados para cada región es el **árbol de decisión**([Breiman et al., 1984](#)) y sus múltiples variantes. Como se muestra en la figura5.7, cada nodo del árbol de decisión está asociado con una región en el espacio de entrada y los nodos internos dividen esa región en una subregión para cada elemento secundario del nodo (normalmente mediante un corte alineado con el eje). Por lo tanto, el espacio se subdivide en regiones que no se superponen, con una correspondencia uno a uno entre los nodos hoja y las regiones de entrada. Cada nodo de hoja generalmente asigna cada punto en su región de entrada a la misma salida. Los árboles de decisión generalmente se entrenan con algoritmos especializados que están más allá del alcance de este libro. El algoritmo de aprendizaje puede considerarse no paramétrico si se le permite aprender un árbol de tamaño arbitrario, aunque los árboles de decisión suelen regularizarse con restricciones de tamaño que los convierten en modelos paramétricos en la práctica. Árboles de decisión como se usan normalmente, con divisiones alineadas con el eje y salidas constantes dentro de cada nodo, luchan por resolver algunos problemas que son fáciles incluso para la regresión logística. Por ejemplo, si tenemos un problema de dos clases y la clase positiva ocurre dondequiera que $X_2 > X_1$, el límite de decisión no está alineado con el eje. Por lo tanto, el árbol de decisión deberá aproximarse al límite de decisión con muchos nodos, implementando una función de paso que constantemente avanza y retrocede a través de la verdadera función de decisión con pasos alineados con el eje.

Como hemos visto, los predictores de vecinos más cercanos y los árboles de decisión tienen muchas limitaciones. No obstante, son algoritmos de aprendizaje útiles cuando los recursos computacionales son limitados. También podemos construir la intuición para algoritmos de aprendizaje más sofisticados al pensar en las similitudes y diferencias entre algoritmos sofisticados y k -NN o líneas base del árbol de decisión.

[Ver Murphy\(2012\)](#),[obispo\(2006\)](#),[prisa et al.\(2001\)](#) u otros libros de texto de aprendizaje automático para obtener más material sobre los algoritmos de aprendizaje supervisado tradicionales.

5.8 Algoritmos de aprendizaje no supervisados

Recuperar de la sección5.1.3que los algoritmos no supervisados son aquellos que experimentan solo "características" pero no una señal de supervisión. La distinción entre algoritmos supervisados y no supervisados no está definida formal y rígidamente porque no existe una prueba objetiva para distinguir si un valor es una característica o un objetivo proporcionado por un supervisor. Informalmente, el aprendizaje no supervisado se refiere a la mayoría de los intentos de extraer información de una distribución que no requiere trabajo humano para anotar ejemplos. El término generalmente se asocia con la estimación de densidad, aprender a extraer muestras de una distribución, aprender a eliminar el ruido de los datos de alguna distribución, encontrar una variedad cercana a los datos o agrupar los datos en grupos de

ejemplos relacionados.

Una tarea clásica de aprendizaje no supervisado es encontrar la "mejor" representación de los datos. Por 'mejor' podemos referirnos a diferentes cosas, pero en general estamos buscando una representación que conserve la mayor cantidad de información sobre X como sea posible, obedeciendo a alguna sanción o restricción dirigida a mantener la representación *más simple* más accesible que *sí mismo*.

Existen múltiples formas de definir un *más simple* representación. Tres de los más comunes incluyen representaciones de menor dimensión, representaciones dispersas y representaciones independientes. Las representaciones de baja dimensión intentan comprimir tanta información sobre X como sea posible en una representación más pequeña. Representaciones dispersas ([Barlow, 1989](#); [Olshausen y campo, 1996](#); [Hinton y Ghahramani, 1997](#)) incrustar el conjunto de datos en una representación cuyas entradas son en su mayoría ceros para la mayoría de las entradas. El uso de representaciones dispersas generalmente requiere aumentar la dimensionalidad de la representación, de modo que la representación que se convierte en su mayoría en ceros no descarte demasiada información. Esto da como resultado una estructura general de la representación que tiende a distribuir los datos a lo largo de los ejes del espacio de representación. Las representaciones independientes intentan *desenredar* las fuentes de variación que subyacen a la distribución de datos, de modo que las dimensiones de la representación sean estadísticamente independientes.

Por supuesto, estos tres criterios ciertamente no son mutuamente excluyentes. Las representaciones de baja dimensión a menudo producen elementos que tienen menos o más débiles dependencias que los datos originales de alta dimensión. Esto se debe a que una forma de reducir el tamaño de una representación es encontrar y eliminar las redundancias. Identificar y eliminar más redundancia permite que el algoritmo de reducción de dimensionalidad logre una mayor compresión y descarte menos información.

La noción de representación es uno de los temas centrales del aprendizaje profundo y, por lo tanto, uno de los temas centrales de este libro. En esta sección, desarrollamos algunos ejemplos simples de algoritmos de aprendizaje de representación. Juntos, estos algoritmos de ejemplo muestran cómo hacer operativos los tres criterios anteriores. La mayoría de los capítulos restantes introducen algoritmos de aprendizaje de representación adicionales que desarrollan estos criterios de diferentes maneras o introducen otros criterios.

5.8.1 Análisis de componentes principales

En la sección [2.12](#), vimos que el algoritmo de análisis de componentes principales proporciona un medio para comprimir datos. También podemos ver PCA como un algoritmo de aprendizaje no supervisado que aprende una representación de datos. Esta representación se basa en dos de los criterios para una representación simple descritos anteriormente. PCA aprende un

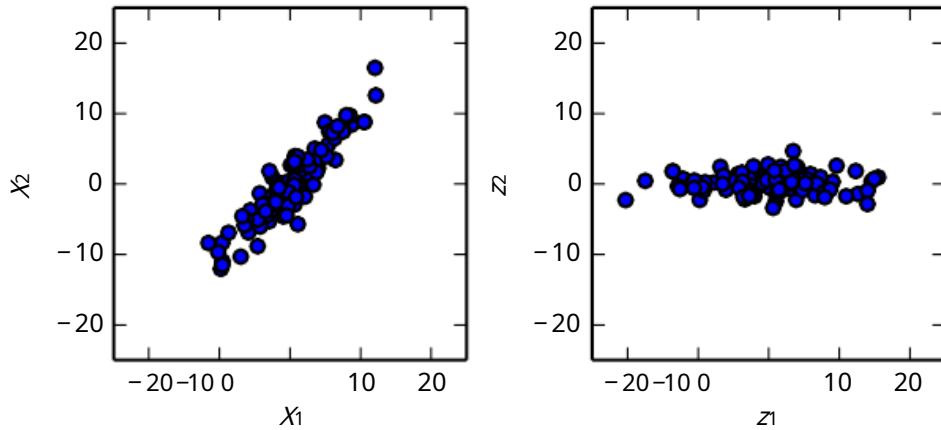


Figura 5.8: PCA aprende una proyección lineal que alinea la dirección de mayor variación con los ejes del nuevo espacio.(Izquierda) Los datos originales consisten en muestras de X . En este espacio, la variación puede ocurrir a lo largo de direcciones que no están alineadas con el eje.(Derecha) Los datos transformados $Z = X \cdot W$ ahora varía más a lo largo del eje Z_1 . La dirección de la segunda mayor variación es ahora a lo largo de Z_2 .

representación que tiene menor dimensionalidad que la entrada original. También aprende una representación cuyos elementos no tienen correlación lineal entre sí. Este es un primer paso hacia el criterio de aprender representaciones cuyos elementos son estadísticamente independientes. Para lograr una independencia total, un algoritmo de aprendizaje de representación también debe eliminar las relaciones no lineales entre variables.

PCA aprende una transformación lineal ortogonal de los datos que proyecta una entrada X a una representación Z como se muestra en la figura 5.8. En la sección 2.12, vimos que podíamos aprender una representación unidimensional que mejor reconstruye los datos originales (en el sentido de error cuadrático medio) y que esta representación en realidad corresponde al primer componente principal de los datos. Por lo tanto, podemos usar PCA como un método de reducción de dimensionalidad simple y efectivo que conserva la mayor cantidad posible de información en los datos (nuevamente, medida por el error de reconstrucción de mínimos cuadrados). A continuación, estudiaremos cómo la representación PCA decorrelaciona la representación de datos original X .

Consideremos el *metroxnortheast*-matriz de diseño dimensional X . Supondremos que los datos tienen una media de cero, $M[X] = 0$. Si este no es el caso, los datos se pueden centrar fácilmente restando la media de todos los ejemplos en un paso de preprocessamiento.

La matriz de covarianza muestral no sesgada asociada con X es dado por:

$$\text{Var}[X] = \frac{1}{m-1} X \cdot X^T. \quad (5.85)$$

PCA encuentra una representación (a través de la transformación lineal) $z = X \cdot W$ donde $\text{Var}[z]$ es diagonal.

En la sección 2.12, vimos que los componentes principales de una matriz de diseño X están dados por los vectores propios de $X \cdot X$. Desde esta vista,

$$X \cdot X = W \Lambda W^T. \quad (5.86)$$

En esta sección, explotamos una derivación alternativa de los componentes principales. Los componentes principales también pueden obtenerse mediante la descomposición en valores singulares. Específicamente, son los vectores rectos singulares de X . Para ver esto, dejemos W ser los vectores singulares correctos en la descomposición $X = tU\Sigma W$. Luego recuperamos la ecuación del vector propio original con W como base del vector propio:

$$X \cdot X = tU\Sigma W^T \quad tU\Sigma W = W\Sigma_2 W^T. \quad (5.87)$$

El SVD es útil para mostrar que PCA da como resultado una diagonal $\text{Var}[z]$. Usando el SVD de X , podemos expresar la varianza de X como:

$$\text{Var}[X] = \frac{1}{m-1} X \cdot X \quad (5.88)$$

$$= \frac{1}{m-1} (tU\Sigma W^T) \cdot tU\Sigma W \quad (5.89)$$

$$= \frac{1}{m-1} W\Sigma_2 W^T \cdot tU \cdot tU\Sigma W \quad (5.90)$$

$$= \frac{1}{m-1} W\Sigma_2 W, \quad (5.91)$$

donde usamos el hecho de que $tU \cdot tU = I$ porque tU la matriz de la descomposición en valores singulares se define como ortogonal. Esto demuestra que si tomamos $z = X \cdot W$, podemos asegurar que la covarianza de z es diagonal como se requiere:

$$\text{Var}[z] = \frac{1}{m-1} Z \cdot Z \quad (5.92)$$

$$= \frac{1}{m-1} W \cdot X \cdot X \cdot W^T \quad (5.93)$$

$$= \frac{1}{m-1} W \cdot W \Sigma_2 W^T \cdot W \quad (5.94)$$

$$= \frac{1}{m-1} \Sigma_2, \quad (5.95)$$

donde esta vez usamos el hecho de que $W \cdot W^T = I$, de nuevo de la definición de la SVD.

El análisis anterior muestra que cuando proyectamos los datos X a través de la transformación lineal W , la representación resultante tiene una matriz de covarianza diagonal (tal como está dada por Σ_2) lo que implica inmediatamente que los elementos individuales de z están mutuamente correlacionados.

Esta capacidad de PCA para transformar datos en una representación en la que los elementos no están correlacionados entre sí es una propiedad muy importante de PCA. Es un ejemplo simple de una representación que intenta *desentrañar los factores desconocidos de variación* subyacente a los datos. En el caso de PCA, este desenredo toma la forma de encontrar una rotación del espacio de entrada (descrito por W) que alinea los ejes principales de varianza con la base del nuevo espacio de representación asociado con z .

Si bien la correlación es una categoría importante de dependencia entre elementos de los datos, también estamos interesados en aprender representaciones que desentrañen formas más complicadas de dependencias de características. Para esto, necesitaremos más de lo que se puede hacer con una simple transformación lineal.

5.8.2 k -significa agrupamiento

Otro ejemplo de un algoritmo de aprendizaje de representación simple es k -significa agrupamiento. El k - significa que el algoritmo de agrupamiento divide el conjunto de entrenamiento en k diferentes grupos de ejemplos que están cerca unos de otros. Por lo tanto, podemos pensar que el algoritmo proporciona un k vector de código unidimensional $-dimensional/h$ representando una entrada X . Si X pertenece al grupo i , entonces $h=1$ y todas las demás entradas de la representación h son cero.

El código one-hot proporcionado por k El agrupamiento de medios es un ejemplo de una representación dispersa, porque la mayoría de sus entradas son cero para cada entrada. Más adelante, desarrollaremos otros algoritmos que aprendan representaciones dispersas más flexibles, donde más de una entrada puede ser distinta de cero para cada entrada X . Los códigos one-hot son un ejemplo extremo de representaciones dispersas que pierden muchos de los beneficios de una representación distribuida. El código one-hot todavía confiere algunas ventajas estadísticas (naturalmente transmite la idea de que todos los ejemplos en el mismo grupo son similares entre sí) y confiere la ventaja computacional de que la representación completa puede ser capturada por un solo número entero.

El k -significa que el algoritmo funciona inicializando k centroides diferentes $\{\mu(1), \dots, \mu(k)\}$ a diferentes valores, luego alternando entre dos pasos diferentes hasta la convergencia. En un solo paso, cada ejemplo de entrenamiento se asigna a un grupo i , donde i es el índice del centroide más cercano $\mu(i)$. En el otro paso, cada centroide $\mu(i)$ se actualiza a la media de todos los ejemplos de entrenamiento X asignado al clúster i .

Una dificultad relacionada con el agrupamiento es que el problema del agrupamiento está inherentemente mal planteado, en el sentido de que no existe un criterio único que mida qué tan bien se corresponde un agrupamiento de datos con el mundo real. Podemos medir las propiedades de la agrupación, como la distancia euclídea promedio desde el centroide de un grupo hasta los miembros del grupo. Esto nos permite decir qué tan bien podemos reconstruir los datos de entrenamiento a partir de las asignaciones de grupos. No sabemos qué tan bien corresponden las asignaciones de grupos a las propiedades del mundo real. Además, puede haber muchos agrupamientos diferentes que se correspondan bien con alguna propiedad del mundo real. Podemos esperar encontrar un agrupamiento que se relacione con una característica pero obtener un agrupamiento diferente e igualmente válido que no sea relevante para nuestra tarea. Por ejemplo, supongamos que ejecutamos dos algoritmos de agrupamiento en un conjunto de datos que consta de imágenes de camiones rojos, imágenes de automóviles rojos, imágenes de camiones grises e imágenes de automóviles grises. Si le pedimos a cada algoritmo de agrupamiento que encuentre dos grupos, un algoritmo puede encontrar un grupo de automóviles y un grupo de camiones, mientras que otro puede encontrar un grupo de vehículos rojos y un grupo de vehículos grises. Supongamos que también ejecutamos un tercer algoritmo de agrupamiento, que puede determinar el número de agrupamientos. Esto puede asignar los ejemplos a cuatro grupos, automóviles rojos, camiones rojos, automóviles grises y camiones grises. Este nuevo agrupamiento ahora al menos captura información sobre ambos atributos, pero ha perdido información sobre la similitud. Los autos rojos están en un grupo diferente al de los autos grises, al igual que están en un grupo diferente al de los camiones grises. El resultado del algoritmo de agrupamiento no nos dice que los autos rojos sean más similares a los autos grises que a los camiones grises. Son diferentes de ambas cosas, y eso es todo lo que sabemos.

Estos problemas ilustran algunas de las razones por las que podemos preferir una representación distribuida a una representación única. Una representación distribuida podría tener dos atributos para cada vehículo: uno que represente su color y otro que represente si es un automóvil o un camión. Todavía no está del todo claro cuál es la representación distribuida óptima (¿cómo puede saber el algoritmo de aprendizaje si los dos atributos que nos interesan son el color y la comparación entre el automóvil y el camión en lugar del fabricante y la edad?) pero tener muchos atributos reduce la carga sobre el algoritmo para adivinar qué atributo único nos importa, y nos permite medir la similitud entre los objetos de manera detallada al comparar muchos atributos en lugar de solo probar si un atributo coincide.

5.9 Descenso de gradiente estocástico

Casi todo el aprendizaje profundo está impulsado por un algoritmo muy importante:**descenso de gradiente estocástico** o SGD. El descenso de gradiente estocástico es una extensión del

algoritmo de descenso de gradiente presentado en la sección 4.3.

Un problema recurrente en el aprendizaje automático es que se necesitan grandes conjuntos de entrenamiento para una buena generalización, pero los grandes conjuntos de entrenamiento también son más costosos desde el punto de vista computacional.

La función de costo utilizada por un algoritmo de aprendizaje automático a menudo se descompone como una suma de ejemplos de entrenamiento de alguna función de pérdida por ejemplo. Por ejemplo, la probabilidad logarítmica condicional negativa de los datos de entrenamiento se puede escribir como

$$J(\theta) = \text{mix}, y \sim p_{\text{datos}} \quad L(X, y, \theta) = \frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} L(X(i), y(i), \theta) \quad (5.96)$$

dónde L es la pérdida por ejemplo $L(X, y, \theta) = -\text{registro} \log(y / X; \theta)$.

Para estas funciones de costo aditivo, el descenso de gradiente requiere computación

$$\nabla_{\theta} J(\theta) = \frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} \nabla L(X(i), y(i), \theta) \quad (5.97)$$

El costo computacional de esta operación es $\mathcal{O}(\text{metro})$. A medida que el tamaño del conjunto de entrenamiento crece a miles de millones de ejemplos, el tiempo para dar un solo paso de gradiente se vuelve prohibitivamente largo.

La idea del descenso de gradiente estocástico es que el gradiente es una expectativa. La expectativa se puede estimar aproximadamente utilizando un pequeño conjunto de muestras. Específicamente, en cada paso del algoritmo, podemos muestrear un **minilote** de ejemplos $B = \{X(1), \dots, X(\text{metro})\}$ extraído uniformemente del conjunto de entrenamiento. El tamaño del mini lote metro -normalmente se elige para ser un número relativamente pequeño de ejemplos, que van desde 1 a unos pocos cientos. Crucialmente, metro -generalmente se mantiene fijo como el tamaño del conjunto de entrenamiento metro crece. Podemos ajustar un conjunto de entrenamiento con miles de millones de ejemplos utilizando actualizaciones calculadas en solo cien ejemplos.

La estimación del gradiente se forma como

$$\nabla_{\theta} J(\theta) = \frac{1}{\text{metro}} \sum_{i=1}^{\text{metro}} \nabla L(X(i), y(i), \theta). \quad (5.98)$$

usando ejemplos del minibatch B . El algoritmo de descenso de gradiente estocástico sigue el gradiente estimado cuesta abajo:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta), \quad (5.99)$$

dónde η es la tasa de aprendizaje.

El descenso de gradiente en general se ha considerado a menudo como lento o poco fiable. En el pasado, la aplicación del gradiente descendente a problemas de optimización no convexos se consideraba temeraria o carente de principios. Hoy sabemos que los modelos de aprendizaje automático descritos en parte Y funcionan muy bien cuando se entran con descenso de gradiente. Es posible que no se garantice que el algoritmo de optimización llegue incluso a un mínimo local en un período de tiempo razonable, pero a menudo encuentra un valor muy bajo de la función de costo lo suficientemente rápido como para ser útil.

El descenso de gradiente estocástico tiene muchos usos importantes fuera del contexto del aprendizaje profundo. Es la forma principal de entrenar modelos lineales grandes en conjuntos de datos muy grandes. Para un tamaño de modelo fijo, el costo por actualización de SGD no depende del tamaño del conjunto de entrenamiento. En la práctica, a menudo usamos un modelo más grande a medida que aumenta el tamaño del conjunto de entrenamiento, pero no estamos obligados a hacerlo. La cantidad de actualizaciones requeridas para alcanzar la convergencia generalmente aumenta con el tamaño del conjunto de entrenamiento. Sin embargo, como *metrōse* acerca al infinito, el modelo finalmente convergerá a su mejor error de prueba posible antes de que SGD haya muestreado cada ejemplo en el conjunto de entrenamiento. Creciente *metromás* no extenderá la cantidad de tiempo de entrenamiento necesario para alcanzar el mejor error de prueba posible del modelo. Desde este punto de vista, se puede argumentar que el costo asintótico de entrenar un modelo con SGD es $O(1)$ como una función de *metro*.

Antes de la llegada del aprendizaje profundo, la forma principal de aprender modelos no lineales era usar el truco del núcleo en combinación con un modelo lineal. Muchos algoritmos de aprendizaje del núcleo requieren la construcción de un $metro \times metro$ matriz $GRAM O_{yo,j} = k(X(i), X(j))$. La construcción de esta matriz tiene un costo computacional $O(metro^2)$, lo cual es claramente indeseable para conjuntos de datos con miles de millones de ejemplos. En el mundo académico, a partir de 2006, el aprendizaje profundo fue inicialmente interesante porque podía generalizar a nuevos ejemplos mejor que los algoritmos de la competencia cuando se entrenaba en conjuntos de datos de tamaño mediano con decenas de miles de ejemplos. Poco después, el aprendizaje profundo atrajo un interés adicional en la industria porque proporcionaba una forma escalable de entrenar modelos no lineales en grandes conjuntos de datos.

El descenso de gradiente estocástico y muchas mejoras se describen más adelante en el capítulo 8.

5.10 Construcción de un algoritmo de aprendizaje automático

Casi todos los algoritmos de aprendizaje profundo se pueden describir como instancias particulares de una receta bastante simple: combinar una especificación de un conjunto de datos, una función de costo, un procedimiento de optimización y un modelo.

Por ejemplo, el algoritmo de regresión lineal combina un conjunto de datos que consta de

Xy , la función de costo

$$J(w, b) = -\text{mix}_{y \sim p_{\text{datos}}} \underset{\text{registro}}{\text{pag}} \underset{\text{modelo}}{\text{model}}(y / X), \quad (5.100)$$

la especificación del modelo $p_{\text{model}}(y / X) = \text{norte}(y; X \cdot w + b, 1)$, y, en la mayoría de los casos, el algoritmo de optimización definido al resolver donde el gradiente del costo es cero usando las ecuaciones normales.

Al darnos cuenta de que podemos reemplazar cualquiera de estos componentes en su mayoría independientemente de los demás, podemos obtener una variedad muy amplia de algoritmos.

La función de costo generalmente incluye al menos un término que hace que el proceso de aprendizaje realice una estimación estadística. La función de costo más común es la verosimilitud logarítmica negativa, por lo que minimizar la función de costo provoca una estimación de máxima verosimilitud.

La función de costo también puede incluir términos adicionales, como los términos de regularización. Por ejemplo, podemos agregar la disminución del peso a la función de costo de regresión lineal para obtener

$$J(w, b) = \lambda / \|w\|_2^2 - \text{mix}_{y \sim p_{\text{datos}}} \underset{\text{registro}}{\text{pag}} \underset{\text{modelo}}{\text{model}}(y / X). \quad (5.101)$$

Esto todavía permite la optimización de forma cerrada.

Si cambiamos el modelo para que sea no lineal, entonces la mayoría de las funciones de costo ya no se pueden optimizar en forma cerrada. Esto requiere que elijamos un procedimiento de optimización numérica iterativa, como el descenso de gradiente.

La receta para construir un algoritmo de aprendizaje combinando modelos, costos y algoritmos de optimización admite tanto el aprendizaje supervisado como el no supervisado. El ejemplo de regresión lineal muestra cómo apoyar el aprendizaje supervisado. El aprendizaje no supervisado se puede respaldar definiendo un conjunto de datos que contenga solo Xy proporcionando un costo y modelo no supervisado apropiado. Por ejemplo, podemos obtener el primer vector PCA especificando que nuestra función de pérdida es

$$J(w) = \text{mix}_{y \sim p_{\text{datos}}} \|Xw - r(X; w)\|_2^2 \quad (5.102)$$

mientras que nuestro modelo está definido para tener w con norma uno y función de reconstrucción $r(X) = w \cdot Xw$.

En algunos casos, la función de costo puede ser una función que en realidad no podemos evaluar, por razones de cálculo. En estos casos, aún podemos minimizarlo aproximadamente usando optimización numérica iterativa siempre que tengamos alguna forma de aproximar sus gradientes.

La mayoría de los algoritmos de aprendizaje automático hacen uso de esta receta, aunque puede que no sea inmediatamente obvio. Si un algoritmo de aprendizaje automático parece especialmente único o

diseñado a mano, por lo general se puede entender como el uso de un optimizador de casos especiales. Algunos modelos como los árboles de decisión ok- significa que requieren optimizadores de casos especiales porque sus funciones de costo tienen regiones planas que las hacen inapropiadas para la minimización mediante optimizadores basados en gradientes. Reconocer que la mayoría de los algoritmos de aprendizaje automático se pueden describir usando esta receta ayuda a ver los diferentes algoritmos como parte de una taxonomía de métodos para realizar tareas relacionadas que funcionan por razones similares, en lugar de como una larga lista de algoritmos que tienen justificaciones separadas.

5.11 Desafíos que motivan el aprendizaje profundo

Los algoritmos simples de aprendizaje automático descritos en este capítulo funcionan muy bien en una amplia variedad de problemas importantes. Sin embargo, no han logrado resolver los problemas centrales de la IA, como el reconocimiento del habla o el reconocimiento de objetos.

El desarrollo del aprendizaje profundo estuvo motivado en parte por el fracaso de los algoritmos tradicionales para generalizar bien tales tareas de IA.

Esta sección trata sobre cómo el desafío de generalizar a nuevos ejemplos se vuelve exponencialmente más difícil cuando se trabaja con datos de alta dimensión, y cómo los mecanismos utilizados para lograr la generalización en el aprendizaje automático tradicional son insuficientes para aprender funciones complicadas en espacios de alta dimensión. Dichos espacios también suelen imponer altos costos computacionales. El aprendizaje profundo se diseñó para superar estos y otros obstáculos.

5.11.1 La maldición de la dimensionalidad

Muchos problemas de aprendizaje automático se vuelven extremadamente difíciles cuando la cantidad de dimensiones en los datos es alta. Este fenómeno se conoce como el**maldición de dimensionalidad**. De particular preocupación es que el número de posibles configuraciones distintas de un conjunto de variables aumenta exponencialmente a medida que aumenta el número de variables.

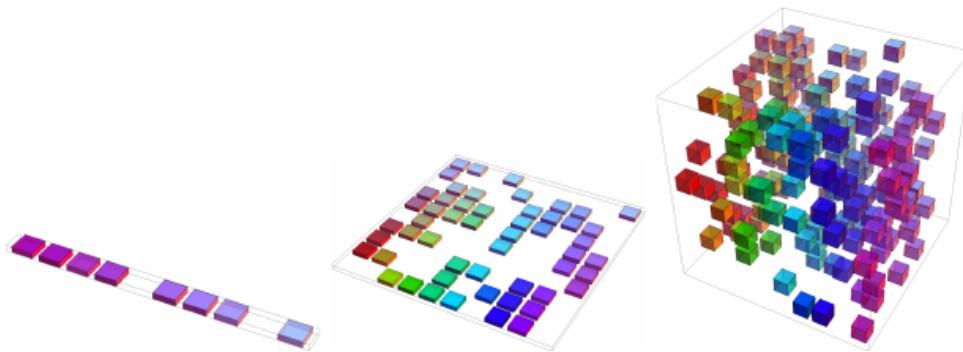


Figura 5.9: A medida que aumenta el número de dimensiones relevantes de los datos (de izquierda a derecha), el número de configuraciones de interés puede crecer exponencialmente. (Izquierda) En este ejemplo unidimensional, tenemos una variable para la que solo nos importa distinguir 10 regiones de interés. Con suficientes ejemplos dentro de cada una de estas regiones (cada región corresponde a una celda en la ilustración), los algoritmos de aprendizaje pueden generalizarse correctamente fácilmente. Una forma sencilla de generalizar es estimar el valor de la función objetivo dentro de cada región (y posiblemente interpolar entre regiones vecinas). (Centro) Con 2 dimensiones es más difícil distinguir 10 valores diferentes de cada variable. Necesitamos hacer un seguimiento de hasta $10 \times 10 = 100$ regiones, y necesitamos al menos esa cantidad de ejemplos para cubrir todas esas regiones. (Bien) Con 3 dimensiones esto crece a $10^3 = 1000$ regiones y al menos esa cantidad de ejemplos. Para d dimensiones y v valores a ser distinguidos a lo largo de cada eje, parece que necesitamos $O(v^d)$ regiones y ejemplos. Este es un ejemplo de la maldición de la dimensionalidad. Figura cedida gentilmente por Nicolás Chapados.

La maldición de la dimensionalidad surge en muchos lugares de la informática, y especialmente en el aprendizaje automático.

Un desafío planteado por la maldición de la dimensionalidad es un desafío estadístico. Como se ilustra en la figura 5.9, surge un desafío estadístico porque el número de configuraciones posibles de X es mucho mayor que el número de ejemplos de entrenamiento. Para entender el problema, consideremos que el espacio de entrada está organizado en una cuadrícula, como en la figura. Podemos describir un espacio de baja dimensión con un número reducido de celdas de cuadrícula que en su mayoría están ocupadas por los datos. Al generalizar a un nuevo punto de datos, generalmente podemos decir qué hacer simplemente inspeccionando los ejemplos de entrenamiento que se encuentran en la misma celda que la nueva entrada. Por ejemplo, si se estima la densidad de probabilidad en algún punto X , podemos devolver el número de ejemplos de entrenamiento en la misma celda de volumen unitario como X , dividido por el número total de ejemplos de entrenamiento. Si deseamos clasificar un ejemplo, podemos devolver la clase más común de ejemplos de entrenamiento en la misma celda. Si estamos haciendo una regresión, podemos promediar los valores objetivo observados sobre los ejemplos en esa celda. Pero, ¿qué pasa con las células de las que no hemos visto ningún ejemplo? Debido a que en espacios de alta dimensión la cantidad de configuraciones es enorme, mucho mayor que nuestra cantidad de ejemplos, una celda de cuadrícula típica no tiene ningún ejemplo de entrenamiento asociado. ¿Cómo podríamos decir algo

significativo acerca de estas nuevas configuraciones? Muchos algoritmos tradicionales de aprendizaje automático simplemente asumen que la salida en un nuevo punto debe ser aproximadamente la misma que la salida en el punto de entrenamiento más cercano.

5.11.2 Regularización de Constancia y Suavidad Local

Para generalizar bien, los algoritmos de aprendizaje automático deben guiarse por creencias previas sobre qué tipo de función deben aprender. Anteriormente, hemos visto estos antecedentes incorporados como creencias explícitas en forma de distribuciones de probabilidad sobre los parámetros del modelo. Más informalmente, también podemos discutir las creencias previas como influencias directas en la *función* mismo y solo actuando indirectamente sobre los parámetros a través de su efecto sobre la función. Además, discutimos de manera informal que las creencias previas se expresan implícitamente, eligiendo algoritmos que están sesgados hacia la elección de una clase de funciones sobre otra, aunque estos sesgos no se expresen (o incluso no se puedan expresar) en términos de una distribución de probabilidad que represente nuestro grado de creencia en varias funciones.

Entre los más utilizados de estos "priores" implícitos está el **suavidad previa o constancia local previa**. Este prior establece que la función que aprendemos no debería cambiar mucho dentro de una pequeña región.

Muchos algoritmos más simples se basan exclusivamente en esto antes de generalizar bien y, como resultado, no logran adaptarse a los desafíos estadísticos que implica resolver tareas de nivel de IA. A lo largo de este libro, describiremos cómo el aprendizaje profundo introduce antecedentes adicionales (explícitos e implícitos) para reducir el error de generalización en tareas sofisticadas. Aquí, explicamos por qué la suavidad previa por sí sola es insuficiente para estas tareas.

Hay muchas formas diferentes de expresar implícita o explícitamente una creencia previa de que la función aprendida debe ser uniforme o localmente constante. Todos estos métodos diferentes están diseñados para fomentar el proceso de aprendizaje para aprender una función. F^* que satisface la condición

$$F^*(X) \approx F^*(X + \cdot) \quad (5.103)$$

para la mayoría de las configuraciones X y pequeño cambio-. En otras palabras, si conocemos una buena respuesta para una entrada X (por ejemplo, si X es un ejemplo de entrenamiento etiquetado), entonces esa respuesta es probablemente buena en el vecindario de X . Si tenemos varias respuestas buenas en algún vecindario, las combinariamos (mediante algún tipo de promedio o interpolación) para producir una respuesta que concuerde con tantas de ellas como sea posible.

Un ejemplo extremo del enfoque de constancia local es el k -Familia de vecinos más cercanos de algoritmos de aprendizaje. Estos predictores son literalmente constantes sobre cada

región que contiene todos los puntos x que tienen el mismo conjunto de k vecinos más cercanos en el conjunto de entrenamiento. Para $k=1$, el número de regiones distinguibles no puede ser mayor que el número de ejemplos de entrenamiento.

Mientras que el algoritmo de los vecinos más cercanos copia la salida de los ejemplos de entrenamiento cercanos, la mayoría de las máquinas del kernel interpolan entre las salidas del conjunto de entrenamiento asociadas con los ejemplos de entrenamiento cercanos. Una clase importante de granos es la familia de **núcleos locales** donde $k(t_u, v)$ es grande cuando $t_u = v$ y disminuye a medida que $t_u \neq v$ crecer más lejos unos de otros. Se puede pensar en un kernel local como una función de similitud que realiza la coincidencia de plantillas, midiendo qué tan cerca se encuentra un ejemplo de prueba. X se parece a cada ejemplo de entrenamiento X_i . Gran parte de la motivación moderna para el aprendizaje profundo se deriva del estudio de las limitaciones de la coincidencia de plantillas locales y cómo los modelos profundos pueden tener éxito en los casos en que falla la coincidencia de plantillas locales ([Bengio et al., 2006b](#)).

Los árboles de decisión también sufren las limitaciones del aprendizaje basado exclusivamente en la suavidad porque dividen el espacio de entrada en tantas regiones como hojas y usan un parámetro separado (o, a veces, muchos parámetros para extensiones de árboles de decisión) en cada región. Si la función objetivo requiere un árbol con al menos n hojas para ser representadas con precisión, entonces al menos n ejemplos de entrenamiento para ajustarse al árbol. Un múltiplo de n es necesario para lograr cierto nivel de confianza estadística en el resultado previsto.

En general, para distinguir $O(k)$ regiones en el espacio de entrada, todos estos métodos requieren $O(k)$ ejemplos. Típicamente hay $O(k)$ parámetros, con $O(1)$ parámetros asociados a cada uno de los $O(k)$ regiones. El caso de un escenario de vecino más cercano, donde cada ejemplo de entrenamiento se puede usar para definir como máximo una región, se ilustra en la figura 5.10.

¿Hay alguna manera de representar una función compleja que tenga muchas más regiones para distinguir que la cantidad de ejemplos de entrenamiento? Claramente, asumir solo la suavidad de la función subyacente no permitirá que un alumno haga eso. Por ejemplo, imagina que la función objetivo es una especie de tablero de ajedrez. Un tablero de ajedrez contiene muchas variaciones, pero tienen una estructura simple. Imagine lo que sucede cuando la cantidad de ejemplos de capacitación es sustancialmente menor que la cantidad de cuadrados blancos y negros en el tablero de ajedrez. Basándonos solo en la generalización local y la suavidad o la constancia local previa, estaríamos garantizados para adivinar correctamente el color de un nuevo punto si se encuentra dentro del mismo cuadro de tablero de ajedrez como ejemplo de entrenamiento. No hay garantía de que el alumno pueda extender correctamente el patrón de tablero de ajedrez a puntos que se encuentran en cuadrados que no contienen ejemplos de entrenamiento. Solo con este previo, la única información que nos dice un ejemplo es el color de su cuadrado, y la única forma de obtener los colores de los

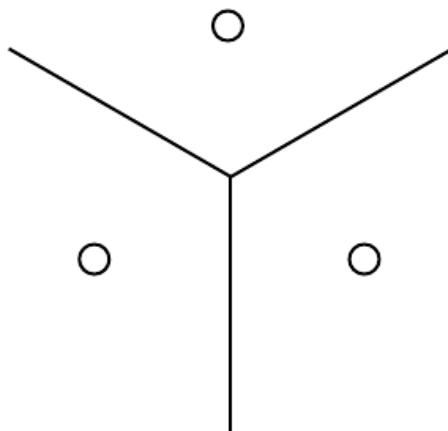


Figura 5.10: Ilustración de cómo el algoritmo del vecino más cercano divide el espacio de entrada en regiones. Un ejemplo (representado aquí por un círculo) dentro de cada región define el límite de la región (representado aquí por las líneas). El valor asociado con cada ejemplo define cuál debe ser la salida para todos los puntos dentro de la región correspondiente. Las regiones definidas por la coincidencia del vecino más cercano forman un patrón geométrico llamado diagrama de Voronoi. El número de estas regiones contiguas no puede crecer más rápido que el número de ejemplos de entrenamiento. Si bien esta figura ilustra específicamente el comportamiento del algoritmo del vecino más cercano, otros algoritmos de aprendizaje automático que se basan exclusivamente en la suavidad local antes de la generalización exhiben comportamientos similares: cada ejemplo de capacitación solo informa al alumno sobre cómo generalizar en algún vecindario que rodea inmediatamente ese ejemplo.

todo el derecho del tablero de ajedrez es cubrir cada una de sus celdas con al menos un ejemplo.

La suposición de suavidad y los algoritmos de aprendizaje no paramétrico asociados funcionan extremadamente bien siempre que haya suficientes ejemplos para que el algoritmo de aprendizaje observe puntos altos en la mayoría de los picos y puntos bajos en la mayoría de los valles de la verdadera función subyacente que se va a aprender. Esto es generalmente cierto cuando la función que se va a aprender es lo suficientemente suave y varía en pocas dimensiones. En dimensiones altas, incluso una función muy suave puede cambiar suavemente pero de forma diferente a lo largo de cada dimensión. Si la función además se comporta de manera diferente en diferentes regiones, puede volverse extremadamente complicado de describir con un conjunto de ejemplos de entrenamiento. Si la función es complicada (queremos distinguir una gran cantidad de regiones en comparación con la cantidad de ejemplos), ¿hay alguna esperanza de generalizar bien?

La respuesta a ambas preguntas, si es posible representar una función complicada de manera eficiente y si es posible que la función estimada se generalice bien a nuevas entradas, es sí. La idea clave es que un gran número de regiones, por ejemplo, $\mathcal{O}(2^k)$, se puede definir con $\mathcal{O}(k)$ ejemplos, siempre que introduzcamos algunas dependencias entre las regiones a través de suposiciones adicionales sobre la distribución de generación de datos subyacente. De esta manera, podemos generalizar de forma no local ([Bengio y Monperrus, 2005](#); [bengio et al., 2006c](#)). Muchos algoritmos de aprendizaje profundo diferentes proporcionan suposiciones implícitas o explícitas que son razonables para una amplia gama de tareas de IA con el fin de capturar estas ventajas.

Otros enfoques del aprendizaje automático a menudo hacen suposiciones más sólidas y específicas de la tarea. Por ejemplo, podríamos resolver fácilmente la tarea del tablero de ajedrez suponiendo que la función objetivo es periódica. Por lo general, no incluimos suposiciones tan fuertes y específicas de la tarea en las redes neuronales para que puedan generalizarse a una variedad mucho más amplia de estructuras. Las tareas de IA tienen una estructura que es demasiado compleja para limitarse a propiedades simples especificadas manualmente, como la periodicidad, por lo que queremos algoritmos de aprendizaje que incorporen suposiciones de propósito más general. La idea central en el aprendizaje profundo es que asumimos que los datos fueron generados por el *composición de factores* o características, potencialmente en múltiples niveles en una jerarquía. Muchas otras suposiciones genéricas similares pueden mejorar aún más los algoritmos de aprendizaje profundo. Estas suposiciones aparentemente moderadas permiten una ganancia exponencial en la relación entre el número de ejemplos y el número de regiones que se pueden distinguir. Estas ganancias exponenciales se describen con mayor precisión en las secciones [6.4.1, 15.4](#) y [15.5](#). Las ventajas exponenciales conferidas por el uso de representaciones profundas y distribuidas contrarrestan los desafíos exponenciales que plantea la maldición de la dimensionalidad.

5.11.3 Aprendizaje múltiple

Un concepto importante que subyace a muchas ideas en el aprendizaje automático es el de una variedad.

Acoletores una región conectada. Matemáticamente, es un conjunto de puntos, asociados a una vecindad alrededor de cada punto. Desde cualquier punto dado, la variedad localmente parece ser un espacio euclíadiano. En la vida cotidiana, experimentamos la superficie del mundo como un plano bidimensional, pero en realidad es una variedad esférica en el espacio tridimensional.

La definición de un entorno que rodea a cada punto implica la existencia de transformaciones que se pueden aplicar para moverse en la variedad de una posición a una vecina. En el ejemplo de la superficie del mundo como una variedad, uno puede caminar hacia el norte, el sur, el este o el oeste.

Aunque existe un significado matemático formal para el término "variedad", en el aprendizaje automático tiende a usarse de manera más flexible para designar un conjunto conectado de puntos que se pueden aproximar bien considerando solo una pequeña cantidad de grados de libertad o dimensiones, incrustado en un espacio de dimensiones superiores. Cada dimensión corresponde a una dirección local de variación. Ver figura 5.11 para ver un ejemplo de datos de entrenamiento que se encuentran cerca de una variedad unidimensional incrustada en un espacio bidimensional. En el contexto del aprendizaje automático, permitimos que la dimensionalidad de la variedad varíe de un punto a otro. Esto sucede a menudo cuando una variedad se corta a sí misma. Por ejemplo, una figura ocho es una variedad que tiene una sola dimensión en la mayoría de los lugares pero dos dimensiones en la intersección en el centro.

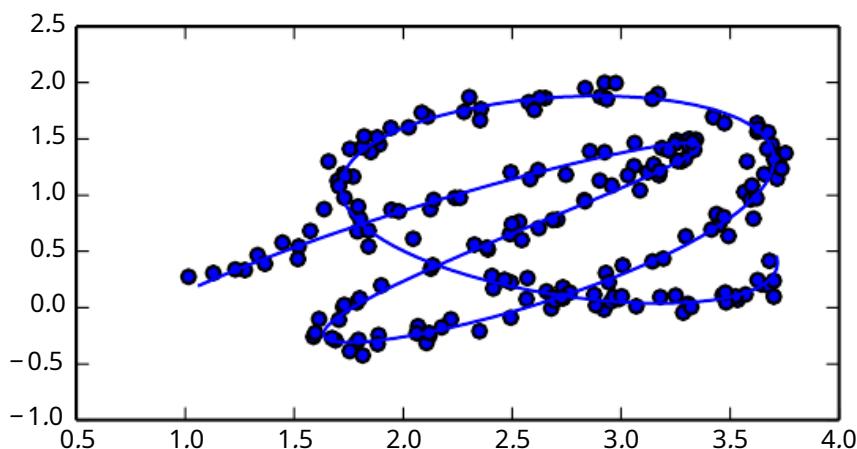


Figura 5.11: Datos muestreados de una distribución en un espacio bidimensional que en realidad está concentrado cerca de una variedad unidimensional, como una cuerda retorcida. La línea continua indica la variedad subyacente que el alumno debe inferir.

Muchos problemas de aprendizaje automático parecen inútiles si esperamos que el algoritmo de aprendizaje automático aprenda funciones con variaciones interesantes en todos los aspectos. *Rnorte*.

Aprendizaje múltiple Los algoritmos superan este obstáculo asumiendo que la mayoría de *Rnorte* consiste en entradas inválidas, y que las entradas interesantes ocurren solo a lo largo de una colección de variedades que contienen un pequeño subconjunto de puntos, con variaciones interesantes en la salida de la función aprendida que ocurren solo a lo largo de las direcciones que se encuentran en la variedad, o con variaciones interesantes que ocurren solo cuando nos movemos de una variedad a otra. El aprendizaje múltiple se introdujo en el caso de datos de valores continuos y el entorno de aprendizaje no supervisado, aunque esta idea de concentración de probabilidad se puede generalizar tanto para datos discretos como para el entorno de aprendizaje supervisado: la suposición clave sigue siendo que la masa de probabilidad está altamente concentrada.

La suposición de que los datos se encuentran a lo largo de una variedad de baja dimensión puede no ser siempre correcta o útil. Argumentamos que en el contexto de las tareas de IA, como aquellas que involucran el procesamiento de imágenes, sonidos o texto, la suposición múltiple es al menos aproximadamente correcta. La evidencia a favor de esta suposición consiste en dos categorías de observaciones.

La primera observación a favor de la **hipótesis múltiple** es que la distribución de probabilidad sobre imágenes, cadenas de texto y sonidos que ocurren en la vida real está altamente concentrada. El ruido uniforme esencialmente nunca se parece a las entradas estructuradas de estos dominios. Cifra 5.12 muestra cómo, en cambio, los puntos muestreados uniformemente se parecen a los patrones de estática que aparecen en los televisores analógicos cuando no hay señal disponible. De manera similar, si genera un documento eligiendo letras uniformemente al azar, ¿cuál es la probabilidad de que obtenga un texto significativo en inglés? Casi cero, de nuevo, porque la mayoría de las largas secuencias de letras no corresponden a una secuencia de lenguaje natural: la distribución de secuencias de lenguaje natural ocupa un volumen muy pequeño en el espacio total de secuencias de letras.

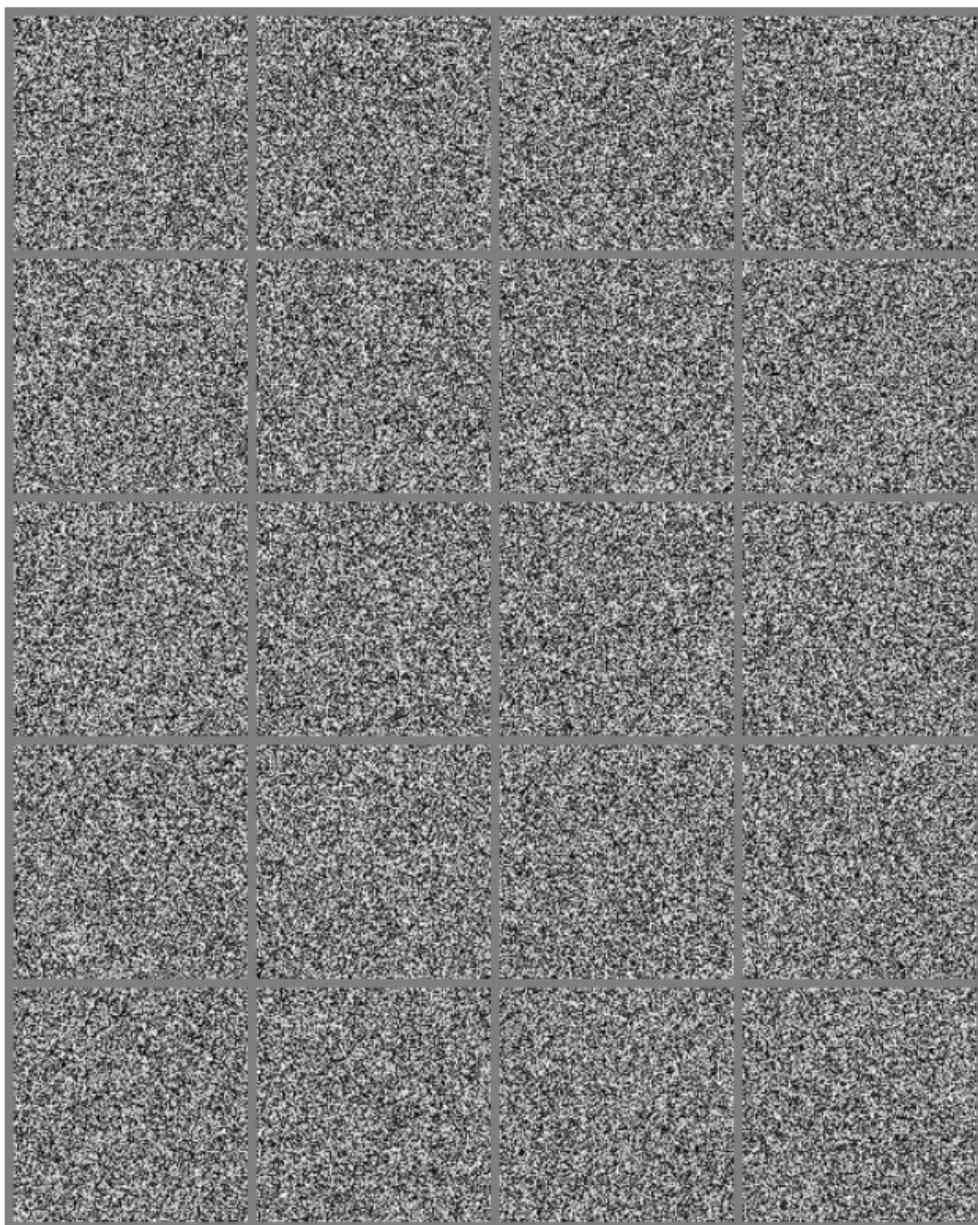


Figura 5.12: Muestrear imágenes uniformemente al azar (elegir aleatoriamente cada píxel de acuerdo con una distribución uniforme) da lugar a imágenes ruidosas. Aunque existe una probabilidad distinta de cero de generar una imagen de una cara o cualquier otro objeto que se encuentre con frecuencia en las aplicaciones de IA, en realidad nunca observamos que esto suceda en la práctica. Esto sugiere que las imágenes que se encuentran en las aplicaciones de IA ocupan una proporción insignificante del volumen del espacio de la imagen.

Por supuesto, las distribuciones de probabilidad concentrada no son suficientes para mostrar que los datos se encuentran en un número razonablemente pequeño de variedades. También debemos establecer que los ejemplos que encontramos están conectados entre sí por otros

ejemplos, con cada ejemplo rodeado de otros ejemplos muy similares que se pueden alcanzar mediante la aplicación de transformaciones para atravesar la variedad. El segundo argumento a favor de la hipótesis múltiple es que también podemos imaginar tales vecindarios y transformaciones, al menos de manera informal. En el caso de las imágenes, ciertamente podemos pensar en muchas transformaciones posibles que nos permiten trazar una variedad en el espacio de la imagen: podemos atenuar o aumentar gradualmente las luces, mover o rotar gradualmente los objetos en la imagen, alterar gradualmente los colores en las superficies de objetos, etc. Es probable que haya múltiples variedades involucradas en la mayoría de las aplicaciones. Por ejemplo, la variedad de imágenes de rostros humanos puede no estar conectada a la variedad de imágenes de caras de gatos.

Estos experimentos mentales que respaldan las múltiples hipótesis transmiten algunas razones intuitivas que las respaldan. Experimentos más rigurosos (Caytón, 2005; Narayanan y Mitter, 2010; Scholkopf et al., 1998; Roweis y Saúl, 2000; Tenenbaum et al., 2000; Marca, 2003; Belkin y Niyogi, 2003; Donoho y Grimes, 2003; Weinberger y Saúl, 2004) apoyan claramente la hipótesis de una gran clase de conjuntos de datos de interés en IA.

Cuando los datos se encuentran en una variedad de baja dimensión, puede ser más natural que los algoritmos de aprendizaje automático representen los datos en términos de coordenadas en la variedad, en lugar de en términos de coordenadas en \mathbb{R}^n . En la vida cotidiana, podemos pensar en las carreteras como variedades 1-D incrustadas en un espacio 3-D. Damos direcciones a direcciones específicas en términos de números de dirección a lo largo de estos caminos 1-D, no en términos de coordenadas en el espacio 3-D. Extraer estas múltiples coordenadas es un desafío, pero promete mejorar muchos algoritmos de aprendizaje automático. Este principio general se aplica en muchos contextos. Cifra 5.13 muestra la estructura múltiple de un conjunto de datos que consta de caras. Al final de este libro, habremos desarrollado los métodos necesarios para aprender tal estructura múltiple. En figura 20.6, veremos cómo un algoritmo de aprendizaje automático puede lograr con éxito este objetivo.

Esto concluye parte I, que ha proporcionado los conceptos básicos de matemáticas y aprendizaje automático que se emplean en las partes restantes del libro. Ahora está preparado para embarcarse en su estudio de aprendizaje profundo.

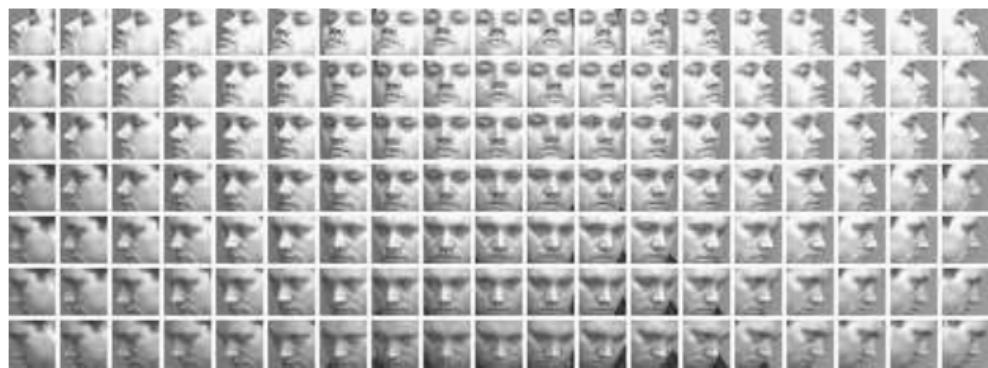


Figura 5.13: Ejemplos de entrenamiento del QMUL Multiview Face Dataset ([Gonget al., 2000](#)) para lo cual se pidió a los sujetos que se movieran de tal manera que cubrieran la variedad bidimensional correspondiente a dos ángulos de rotación. Nos gustaría aprender algoritmos para poder descubrir y desentrañar tales múltiples coordenadas. Cifra [20.6](#) ilustra tal hazaña.

Parte II

Redes profundas: modernas Prácticas

Esta parte del libro resume el estado del aprendizaje profundo moderno tal como se utiliza para resolver aplicaciones prácticas.

El aprendizaje profundo tiene una larga historia y muchas aspiraciones. Se han propuesto varios enfoques que aún no han dado sus frutos por completo. Varios objetivos ambiciosos aún no se han realizado. Estas ramas menos desarrolladas del aprendizaje profundo aparecen en la parte final del libro.

Esta parte se enfoca solo en aquellos enfoques que son esencialmente tecnologías de trabajo que ya se usan mucho en la industria.

El aprendizaje profundo moderno proporciona un marco muy poderoso para el aprendizaje supervisado. Al agregar más capas y más unidades dentro de una capa, una red profunda puede representar funciones de complejidad creciente. La mayoría de las tareas que consisten en mapear un vector de entrada a un vector de salida, y que son fáciles de hacer rápidamente para una persona, se pueden lograr a través del aprendizaje profundo, dados modelos suficientemente grandes y conjuntos de datos suficientemente grandes de ejemplos de entrenamiento etiquetados. Otras tareas, que no pueden describirse como la asociación de un vector con otro, o que son lo suficientemente difíciles como para que una persona requiera tiempo para pensar y reflexionar para realizar la tarea, quedan fuera del alcance del aprendizaje profundo por ahora.

Esta parte del libro describe la tecnología central de aproximación de funciones paramétricas que está detrás de casi todas las aplicaciones prácticas modernas de aprendizaje profundo. Comenzamos describiendo el modelo de red profunda feedforward que se utiliza para representar estas funciones. A continuación, presentamos técnicas avanzadas para la regularización y optimización de dichos modelos. Escalar estos modelos a grandes entradas, como imágenes de alta resolución o secuencias temporales largas, requiere especialización. Presentamos la red convolucional para escalar a imágenes grandes y la red neuronal recurrente para procesar secuencias temporales. Finalmente, presentamos pautas generales para la metodología práctica involucrada en el diseño, construcción y configuración de una aplicación que involucra aprendizaje profundo, y revisamos algunas de las aplicaciones del aprendizaje profundo.

Estos capítulos son los más importantes para un profesional, alguien que quiere comenzar a implementar y usar algoritmos de aprendizaje profundo para resolver problemas del mundo real hoy.

Capítulo 6

Redes de retroalimentación profunda

Redes feedforward profundas, también llamado a menudo **redes neuronales feedforward**, **operceptrones multicapa**(MLP), son los modelos de aprendizaje profundo por excelencia. El objetivo de una red feedforward es aproximar alguna función F^* . Por ejemplo, para un clasificador, $y=F^*(X)$ asigna una entrada X a una categoría y . Una red feedforward define un mapeo $y=F(X; \theta)$ y aprende el valor de los parámetros θ que dan como resultado la mejor aproximación de la función.

Estos modelos se llaman **retroalimentación** porque la información fluye a través de la función que se evalúa desde X a través de los cálculos intermedios utilizados para definir F , y finalmente a la salida y . No existen **comentario** conexiones en las que las salidas del modelo se retroalimentan a sí mismo. Cuando las redes neuronales feedforward se amplían para incluir conexiones de retroalimentación, se denominan **redes neuronales recurrentes**, presentado en el capítulo [10](#).

Las redes feedforward son de extrema importancia para los profesionales del aprendizaje automático. Forman la base de muchas aplicaciones comerciales importantes. Por ejemplo, las redes convolucionales utilizadas para el reconocimiento de objetos a partir de fotografías son un tipo especializado de red de avance. Las redes feedforward son un trampolín conceptual en el camino hacia las redes recurrentes, que impulsan muchas aplicaciones de lenguaje natural.

Las redes neuronales feedforward se denominan **redes** porque normalmente se representan componiendo muchas funciones diferentes. El modelo está asociado con un gráfico acíclico dirigido que describe cómo se componen las funciones juntas. Por ejemplo, podríamos tener tres funciones F_1 , F_2 , y F_3 conectados en cadena, para formar $F(X) = F_3(F_2(F_1(X)))$. Estas estructuras de cadena son las estructuras más utilizadas de las redes neuronales. En este caso, F_1 se llama la **primera capa** de la red, F_2 se llama la **segunda capa**, etcétera. El general

longitud de la cadena da la**profundidad** del modelo. Es a partir de esta terminología que surge el nombre de “aprendizaje profundo”. La capa final de una red feedforward se llama **capa de salida**. Durante el entrenamiento de redes neuronales, manejamos $F(X)$ para hacer coincidir $F(X)$. Los datos de entrenamiento nos proporcionan ejemplos ruidosos y aproximados de $F(X)$ evaluados en diferentes puntos de entrenamiento, cada ejemplo X va acompañado de una etiqueta $y \approx F(X)$. Los ejemplos de entrenamiento especifican directamente lo que debe hacer la capa de salida en cada punto X ; debe producir un valor cercano a y . Los datos de entrenamiento no especifican directamente el comportamiento de las otras capas. El algoritmo de aprendizaje debe decidir cómo usar esas capas para producir el resultado deseado, pero los datos de entrenamiento no dicen qué debe hacer cada capa individual. En cambio, el algoritmo de aprendizaje debe decidir cómo usar estas capas para implementar mejor una aproximación de F . Debido a que los datos de entrenamiento no muestran el resultado deseado para cada una de estas capas, estas capas se denominan **capas ocultas**.

Finalmente, estas redes se denominan *neural* porque están vagamente inspirados en la neurociencia. Cada capa oculta de la red suele tener un valor vectorial. La dimensionalidad de estas capas ocultas determina la **anchura** del modelo. Puede interpretarse que cada elemento del vector desempeña un papel análogo al de una neurona. En lugar de pensar que la capa representa una sola función de vector a vector, también podemos pensar que la capa consta de muchas **unidades** que actúan en paralelo, cada uno representando una función de vector a escalar. Cada unidad se asemeja a una neurona en el sentido de que recibe información de muchas otras unidades y calcula su propio valor de activación. La idea de usar muchas capas de representación con valores vectoriales se extrae de la neurociencia. La elección de las funciones $F_l(X)$ utilizado para computar estas representaciones también está vagamente guiado por observaciones neurocientíficas sobre las funciones que computan las neuronas biológicas. Sin embargo, la investigación moderna de redes neuronales está guiada por muchas disciplinas matemáticas y de ingeniería, y el objetivo de las redes neuronales no es modelar perfectamente el cerebro. Es mejor pensar en las redes feedforward como máquinas de aproximación de funciones que están diseñadas para lograr la generalización estadística, extrayendo ocasionalmente algunas ideas de lo que sabemos sobre el cerebro, en lugar de modelos de la función cerebral.

Una forma de comprender las redes feedforward es comenzar con modelos lineales y considerar cómo superar sus limitaciones. Los modelos lineales, como la regresión logística y la regresión lineal, son atractivos porque pueden ajustarse de manera eficiente y confiable, ya sea en forma cerrada o con optimización convexa. Los modelos lineales también tienen el defecto obvio de que la capacidad del modelo está limitada a funciones lineales, por lo que el modelo no puede comprender la interacción entre dos variables de entrada.

Extender modelos lineales para representar funciones no lineales de X , podemos aplicar el modelo lineal a no X sí mismo sino a una entrada transformada $\varphi(X)$, donde φ es un

transformación no lineal. De manera equivalente, podemos aplicar el truco del núcleo descrito en la sección 5.7.2, para obtener un algoritmo de aprendizaje no lineal basado en aplicar implícitamente el φ -cartografía. podemos pensar en φ como proporcionando un conjunto de características que describen X , o como proporcionar una nueva representación para X .

La pregunta es entonces cómo elegir el mapeo φ .

1. Una opción es usar una muy genérica φ , como el infinito-dimensional φ que es utilizado implícitamente por máquinas kernel basadas en el kernel RBF. Si $\varphi(X)$ tiene una dimensión lo suficientemente alta, siempre podemos tener suficiente capacidad para adaptarse al conjunto de entrenamiento, pero la generalización al conjunto de prueba a menudo sigue siendo deficiente. Los mapeos de características muy genéricos generalmente se basan solo en el principio de la suavidad local y no codifican suficiente información previa para resolver problemas avanzados.
2. Otra opción es diseñar manualmente φ . Hasta la llegada del aprendizaje profundo, este era el enfoque dominante. Este enfoque requiere décadas de esfuerzo humano para cada tarea por separado, con profesionales que se especializan en diferentes dominios, como el reconocimiento de voz o la visión por computadora, y con poca transferencia entre dominios.
3. La estrategia del aprendizaje profundo es aprender φ . En este enfoque, tenemos un modelo $y = f(X; \theta, w) = \varphi(X; \theta) \cdot w$. Ahora tenemos parámetros θ que usamos para aprender φ de una amplia clase de funciones y parámetros w es mapa de $\varphi(X)$ a la salida deseada. Este es un ejemplo de una red feedforward profunda, con φ definiendo una capa oculta. Este enfoque es el único de los tres que renuncia a la convexidad del problema del entrenamiento, pero los beneficios superan los daños. En este enfoque, parametrizamos la representación como $\varphi(X; \theta)$ y use el algoritmo de optimización para encontrar el θ que corresponde a una buena representación. Si lo deseamos, este enfoque puede capturar el beneficio del primer enfoque al ser muy genérico; lo hacemos utilizando una familia muy amplia. $\varphi(X; \theta)$. Este enfoque también puede capturar el beneficio del segundo enfoque. Los profesionales humanos pueden codificar su conocimiento para ayudar a la generalización mediante el diseño de familias $\varphi(X; \theta)$ que esperan que funcione bien. La ventaja es que el diseñador humano solo necesita encontrar la familia de funciones general correcta en lugar de encontrar precisamente la función correcta.

Este principio general de mejorar los modelos mediante el aprendizaje de características se extiende más allá de las redes feedforward descritas en este capítulo. Es un tema recurrente del aprendizaje profundo que se aplica a todos los tipos de modelos descritos a lo largo de este libro. Las redes feedforward son la aplicación de este principio al aprendizaje determinista.

asignaciones de X y que carecen de conexiones de retroalimentación. Otros modelos presentados más adelante aplicarán estos principios para aprender mapeos estocásticos, funciones de aprendizaje con retroalimentación y distribuciones de probabilidad de aprendizaje sobre un solo vector.

Comenzamos este capítulo con un ejemplo simple de una red feedforward. A continuación, abordamos cada una de las decisiones de diseño necesarias para implementar una red feedforward. Primero, entrenar una red feedforward requiere tomar muchas de las mismas decisiones de diseño que son necesarias para un modelo lineal: elegir el optimizador, la función de costo y la forma de las unidades de salida. Revisamos estos conceptos básicos del aprendizaje basado en gradientes, luego procedemos a confrontar algunas de las decisiones de diseño que son exclusivas de las redes feedforward. Las redes feedforward han introducido el concepto de una capa oculta, y esto requiere que elijamos la **funciones de activación** que se utilizará para calcular los valores de la capa oculta. También debemos diseñar la arquitectura de la red, incluidas cuántas capas debe contener la red, cómo deben conectarse estas capas entre sí y cuántas unidades debe haber en cada capa. El aprendizaje en redes neuronales profundas requiere calcular los gradientes de funciones complicadas. Presentamos el **retropropagación** algoritmo y sus generalizaciones modernas, que se pueden utilizar para calcular de manera eficiente estos gradientes. Finalmente, cerramos con alguna perspectiva histórica.

6.1 Ejemplo: Aprender XOR

Para hacer más concreta la idea de una red feedforward, comenzamos con un ejemplo de una red feedforward completamente funcional en una tarea muy simple: aprender la función XOR.

La función XOR (“o exclusivo”) es una operación sobre dos valores binarios, X_1 y X_2 . Cuando exactamente uno de estos valores binarios es igual a 1, la función XOR devuelve 1. De lo contrario, devuelve 0. La función XOR proporciona la función de destino $y = F(X)$ que queremos aprender. Nuestro modelo proporciona una función $y = f(X; \theta)$ y nuestro algoritmo de aprendizaje adaptará los parámetros θ para hacer f lo más parecido posible a F .

En este ejemplo simple, no nos ocuparemos de la generalización estadística. Queremos que nuestra red funcione correctamente en los cuatro puntos $X = \{[0,0], [0,1], [1,0], [1,1]\}$. Capacitaremos a la red en estos cuatro puntos. El único desafío es adaptarse al conjunto de entrenamiento.

Podemos tratar este problema como un problema de regresión y usar una función de pérdida de error cuadrático medio. Elegimos esta función de pérdida para simplificar las matemáticas de este ejemplo tanto como sea posible. En aplicaciones prácticas, MSE no suele ser un

función de costo apropiada para modelar datos binarios. Los enfoques más apropiados se describen en la sección 6.2.2.2.

Evaluada en todo nuestro conjunto de entrenamiento, la función de pérdida MSE es

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (F(x) - F(x; \theta))^2. \quad (6.1)$$

Ahora debemos elegir la forma de nuestro modelo, $F(x; \theta)$. Supongamos que elegimos un modelo lineal, con θ que consiste en w y b . Nuestro modelo se define como

$$F(x; w, b) = x \cdot w + b. \quad (6.2)$$

Podemos minimizar $J(\theta)$ en forma cerrada con respecto a w y b utilizando las ecuaciones normales.

Después de resolver las ecuaciones normales, obtenemos $w=0$ y $b=1$. El modelo lineal simplemente salidas 0.5 en todos los lados. ¿Por qué pasó esto? Cifra 6.1 muestra cómo un modelo lineal no puede representar la función XOR. Una forma de resolver este problema es usar un modelo que aprenda un espacio de características diferente en el que un modelo lineal pueda representar la solución.

Específicamente, presentaremos una red feedforward muy simple con una capa oculta que contiene dos unidades ocultas. Ver figura 6.2 para una ilustración de este modelo. Esta red feedforward tiene un vector de unidades ocultas h que son calculados por una función $F_1(X; W_C)$. Los valores de estas unidades ocultas se utilizan luego como entrada para una segunda capa. La segunda capa es la capa de salida de la red. La capa de salida sigue siendo solo un modelo de regresión lineal, pero ahora se aplica a h en lugar de a X . La red ahora contiene dos funciones encadenadas juntas: $h = F_1(X; W_C)$ y $y = F_2(h; w, b)$, siendo el modelo completo $F(X; w, c, w, b) = F_2(F_1(X))$.

que función debe F_1 calcular? Los modelos lineales nos han servido bien hasta ahora, y puede ser tentador hacer F_1 ser lineal también. Desafortunadamente, si F_1 fueran lineales, entonces la red feedforward como un todo seguiría siendo una función lineal de su entrada. Ignorando los términos de intersección por el momento, supongamos $F_1(X) = W \cdot X$ y $F_2(h) = h \cdot w$. Entonces $F(X) = w \cdot W \cdot X$. Podríamos representar esta función como $F(X) = X \cdot w$ donde $w = w \cdot w$.

Claramente, debemos usar una función no lineal para describir las características. La mayoría de las redes neuronales lo hacen mediante una transformación afín controlada por parámetros aprendidos, seguida de una función no lineal fija denominada función de activación. Usamos esa estrategia aquí, definiendo $h = g(w \cdot W \cdot X + b)$, donde W proporciona los pesos de una transformación lineal y b los sesgos. Anteriormente, para describir una regresión lineal

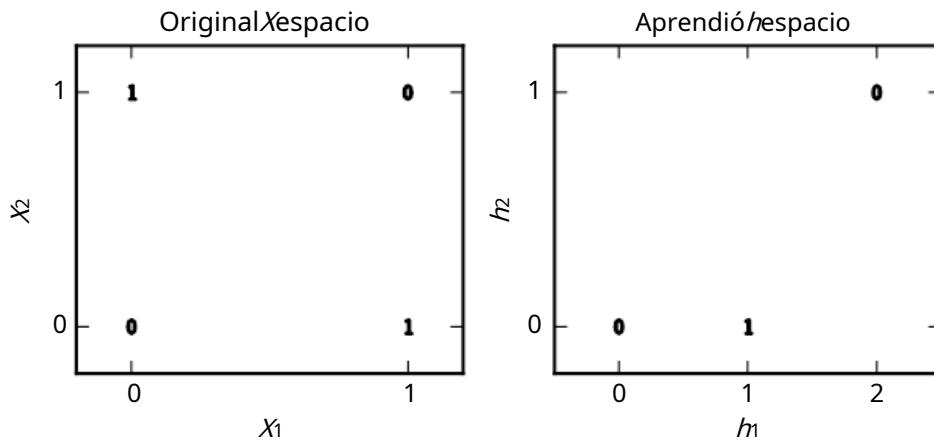


Figura 6.1: Resolución del problema XOR aprendiendo una representación. Los números en negrita impresos en el gráfico indican el valor que la función aprendida debe generar en cada punto.

(Izquierda) Un modelo lineal aplicado directamente a la entrada original no puede implementar la función XOR. Cuando $X_1=0$, la salida del modelo debe aumentar como X_2 aumenta. Cuando $X_1=1$, la salida del modelo debe disminuir como X_2 aumenta. Un modelo lineal debe aplicar un coeficiente fijo a X_2 . Por lo tanto, el modelo lineal no puede utilizar el valor de X_1 para cambiar el coeficiente de X_2 y no puede resolver este problema. (Bien) En el espacio transformado representado por las características extraídas por una red neuronal, un modelo lineal ahora puede resolver el problema. En nuestra solución de ejemplo, los dos puntos que deben tener salida 1 se han colapsado en un solo punto en el espacio de características. En otras palabras, las características no lineales han mapeado tanto $X=[1,0]$ -y $X=[0,1]$ a un solo punto en el espacio de características, $h=[1,0]$. El modelo lineal ahora puede describir la función como creciente en h_1 y decreciendo en h_2 . En este ejemplo, la motivación para aprender el espacio de características es solo aumentar la capacidad del modelo para que pueda adaptarse al conjunto de entrenamiento. En aplicaciones más realistas, las representaciones aprendidas también pueden ayudar a generalizar el modelo.

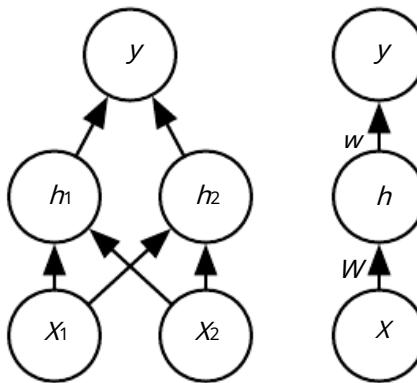


Figura 6.2: Un ejemplo de una red feedforward, dibujada en dos estilos diferentes.

Especificamente, esta es la red feedforward que usamos para resolver el ejemplo XOR. Tiene una sola capa oculta que contiene dos unidades. (*Izquierda*) En este estilo, dibujamos cada unidad como un nodo en el gráfico. Este estilo es muy explícito e inequívoco pero para redes más grandes que este ejemplo puede consumir demasiado espacio. (*Bien*) En este estilo, dibujamos un nodo en el gráfico para cada vector completo que representa las activaciones de una capa. Este estilo es mucho más compacto. A veces anotamos los bordes de este gráfico con el nombre de los parámetros que describen la relación entre dos capas. Aquí, indicamos que una matriz W describe el mapeo de X a h y un vector w describe el mapeo de h a y . Por lo general, omitimos los parámetros de intercepción asociados con cada capa al etiquetar este tipo de dibujo.

modelo, usamos un vector de pesos y un parámetro de sesgo escalar para describir una transformación afín de un vector de entrada a un escalar de salida. Ahora, describimos una transformación afín de un vector X a un vector h , por lo que se necesita un vector completo de parámetros de sesgo. La función de activación *gramonormalmente* se elige como una función que se aplica por elementos, con $h = \text{gramo}(X \cdot W + C)$. En las redes neuronales modernas, la recomendación predeterminada es utilizar el **unidad lineal rectificada** o ReLU ([Jarrett et al., 2009; Nair y Hinton, 2010; gloria et al., 2011a](#)) definida por la función de activación $\text{gramo}(z) = \max\{0, z\}$ representado en la figura 6.3.

Ahora podemos especificar nuestra red completa como

$$f(X; w, c, w, b) = w \cdot \max\{0, W \cdot X + C\} + b. \quad (6.3)$$

Ahora podemos especificar una solución al problema XOR. Dejar

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad (6.4)$$

$$C = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad (6.5)$$

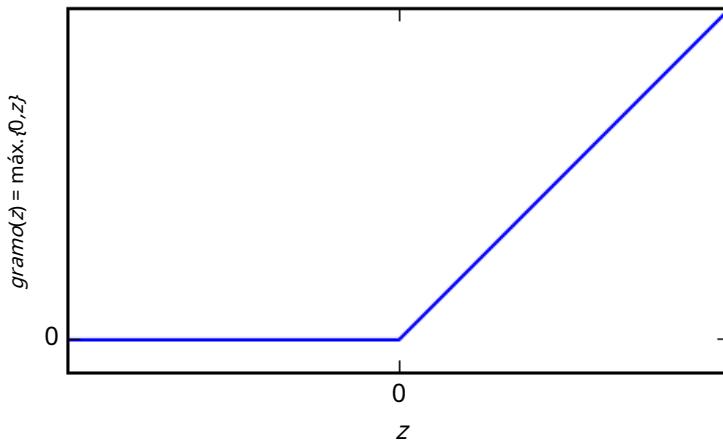


Figura 6.3: La función de activación lineal rectificada. Esta función de activación es la función de activación predeterminada recomendada para su uso con la mayoría de las redes neuronales feedforward. La aplicación de esta función a la salida de una transformación lineal produce una transformación no lineal. Sin embargo, la función se mantiene muy cercana a la lineal, en el sentido de que es una función lineal por partes con dos partes lineales. Debido a que las unidades lineales rectificadas son casi lineales, conservan muchas de las propiedades que facilitan la optimización de los modelos lineales con métodos basados en gradientes. También conservan muchas de las propiedades que hacen que los modelos lineales se generalicen bien. Un principio común en toda la informática es que podemos construir sistemas complicados a partir de componentes mínimos. Así como la memoria de una máquina de Turing solo necesita poder almacenar 0 o 1 estados,

$$w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad (6.6)$$

$y b=0$.

Ahora podemos recorrer la forma en que el modelo procesa un lote de entradas. Dejar X sea la matriz de diseño que contiene los cuatro puntos en el espacio de entrada binaria, con un ejemplo por fila:

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ -1 & 0 \\ 1 & 1 \end{bmatrix}. \quad (6.7)$$

El primer paso en la red neuronal es multiplicar la matriz de entrada por la matriz de peso de la primera capa:

$$XW = \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}. \quad (6.8)$$

A continuación, agregamos el vector de sesgo C , para obtener

$$\begin{bmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \quad (6.9)$$

En este espacio, todos los ejemplos se encuentran a lo largo de una línea con pendiente 1. A medida que nos movemos a lo largo de esta línea, la salida debe comenzar en 0, luego subir a 1, luego vuelve a bajar a 0. Un modelo lineal no puede implementar tal función. Para terminar de calcular el valor de h para cada ejemplo, aplicamos la transformación lineal rectificada:

$$\begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \quad (6.10)$$

Esta transformación ha cambiado la relación entre los ejemplos. Ya no mienten en una sola línea. Como se muestra en la figura 6.1, ahora se encuentran en un espacio donde un modelo lineal puede resolver el problema.

Terminamos multiplicando por el vector de pesos w :

$$\begin{bmatrix} 0 \\ -1 \\ 1 \\ -1 \\ 0 \end{bmatrix}. \quad (6.11)$$

La red neuronal ha obtenido la respuesta correcta para cada ejemplo del lote.

En este ejemplo, simplemente especificamos la solución y luego mostramos que obtuvo cero errores. En una situación real, puede haber miles de millones de parámetros de modelo y miles de millones de ejemplos de entrenamiento, por lo que uno no puede simplemente adivinar la solución como lo hicimos aquí. En cambio, un algoritmo de optimización basado en gradientes puede encontrar parámetros que producen muy pocos errores. La solución que describimos al problema XOR está en un mínimo global de la función de pérdida, por lo que el descenso de gradiente podría converger en este punto. Hay otras soluciones equivalentes al problema XOR que también podría encontrar el descenso de gradiente. El punto de convergencia del descenso del gradiente depende de los valores iniciales de los parámetros. En la práctica, el descenso de gradiente generalmente no encontraría soluciones limpias, fáciles de entender y con valores enteros como la que presentamos aquí.

6.2 Aprendizaje basado en gradientes

Diseñar y entrenar una red neuronal no es muy diferente de entrenar cualquier otro modelo de aprendizaje automático con descenso de gradiente. En la sección 5.10, describimos cómo construir un algoritmo de aprendizaje automático especificando un procedimiento de optimización, una función de costo y una familia de modelos.

La mayor diferencia entre los modelos lineales que hemos visto hasta ahora y las redes neuronales es que la no linealidad de una red neuronal hace que las funciones de pérdida más interesantes se vuelvan no convexas. Esto significa que las redes neuronales generalmente se entrena mediante el uso de optimizadores iterativos basados en gradientes que simplemente llevan la función de costo a un valor muy bajo, en lugar de los solucionadores de ecuaciones lineales que se usan para entrenar modelos de regresión lineal o los algoritmos de optimización convexos con garantías de convergencia global que se usan para entrenar regresión logística o SVM. La optimización convexa converge a partir de cualquier parámetro inicial (en teoría, en la práctica es muy robusta pero puede encontrar problemas numéricos). El descenso de gradiente estocástico aplicado a funciones de pérdida no convexas no tiene tal garantía de convergencia y es sensible a los valores de los parámetros iniciales. Para redes neuronales feedforward, es importante inicializar todos los pesos a valores aleatorios pequeños. Los sesgos pueden inicializarse a cero o a pequeños valores positivos. Los algoritmos iterativos de optimización basados en gradientes utilizados para entrenar redes feedforward y casi todos los demás modelos profundos se describirán en detalle en el capítulo 8, con la inicialización de parámetros en particular discutida en la sección 8.4. Por el momento, basta entender que el algoritmo de entrenamiento casi siempre se basa en usar el gradiente para descender la función de costo de una forma u otra. Los algoritmos específicos son mejoras y refinamientos de las ideas de descenso de gradiente, presentadas en la sección 4.3, y,

más específicamente, son más a menudo mejoras del algoritmo de descenso de gradiente estocástico, presentado en la sección 5.9.

Por supuesto, también podemos entrenar modelos como regresión lineal y máquinas de vectores de soporte con descenso de gradiente y, de hecho, esto es común cuando el conjunto de entrenamiento es extremadamente grande. Desde este punto de vista, entrenar una red neuronal no es muy diferente de entrenar cualquier otro modelo. Calcular el gradiente es un poco más complicado para una red neuronal, pero aún se puede hacer de manera eficiente y exacta. Sección 6.5 describirá cómo obtener el gradiente utilizando el algoritmo de retropropagación y las generalizaciones modernas del algoritmo de retropropagación.

Al igual que con otros modelos de aprendizaje automático, para aplicar el aprendizaje basado en gradientes debemos elegir una función de costo y debemos elegir cómo representar la salida del modelo. Ahora revisamos estas consideraciones de diseño con especial énfasis en el escenario de las redes neuronales.

6.2.1 Funciones de costo

Un aspecto importante del diseño de una red neuronal profunda es la elección de la función de coste. Afortunadamente, las funciones de costo de las redes neuronales son más o menos las mismas que las de otros modelos paramétricos, como los modelos lineales.

En la mayoría de los casos, nuestro modelo paramétrico define una distribución $p(y | X; \theta)$ y simplemente usamos el principio de máxima verosimilitud. Esto significa que usamos la entropía cruzada entre los datos de entrenamiento y las predicciones del modelo como función de costo.

A veces, tomamos un enfoque más simple, donde en lugar de predecir una distribución de probabilidad completa sobre y , simplemente predecimos alguna estadística de y condicionado a X . Las funciones de pérdida especializadas nos permiten entrenar un predictor de estas estimaciones.

La función de costo total utilizada para entrenar una red neuronal a menudo combinará una de las funciones de costo principales descritas aquí con un término de regularización. Ya hemos visto algunos ejemplos sencillos de regularización aplicada a modelos lineales en la sección 5.2.2. El enfoque de disminución de peso utilizado para los modelos lineales también se aplica directamente a las redes neuronales profundas y se encuentra entre las estrategias de regularización más populares. Las estrategias de regularización más avanzadas para redes neuronales se describirán en el capítulo 7.

6.2.1.1 Aprendizaje de Distribuciones Condicionales con Máxima Verosimilitud

La mayoría de las redes neuronales modernas se entranan utilizando la máxima verosimilitud. Esto significa que la función de costo es simplemente la probabilidad logarítmica negativa, descrita de manera equivalente

como la entropía cruzada entre los datos de entrenamiento y la distribución del modelo. Esta función de costo está dada por

$$J(\theta) = -\text{mix}, y \sim p_{\text{datos}} \text{ registropag}_{\text{modelo}}(y / X). \quad (6.12)$$

La forma específica de la función de costo cambia de modelo a modelo, dependiendo de la forma específica de $\text{registropag}_{\text{modelo}}$. La expansión de la ecuación anterior generalmente produce algunos términos que no dependen de los parámetros del modelo y pueden descartarse. Por ejemplo, como vimos en la sección 5.5.1, si $p_{\text{modelo}}(y / X) = \text{norte}(y; F(X; \theta), I)$, luego recuperamos el costo del error cuadrático medio,

$$J(\theta) = \text{mi} \frac{1}{2} \sum_{X, y \sim p_{\text{datos}}} \|y - F(X; \theta)\|^2 + \text{constante}, \quad (6.13)$$

hasta un factor de escala de 1/2 y un término que no depende de θ . Los desechados constantes se basan en la varianza de la distribución gaussiana, que en este caso decidimos no parametrizar. Anteriormente, vimos que la equivalencia entre la estimación de máxima verosimilitud con una distribución de salida y la minimización del error cuadrático medio se cumple para un modelo lineal, pero de hecho, la equivalencia se cumple independientemente de la $F(X; \theta)$ utilizada para predecir la media de la Gaussiana.

Una ventaja de este enfoque de derivar la función de costo a partir de la máxima verosimilitud es que elimina la carga de diseñar funciones de costo para cada modelo. Especificación de un modelo $p_{\text{ag}}(y / X)$ determina automáticamente una función de costo $\text{registropag}(y / X)$.

Un tema recurrente en el diseño de redes neuronales es que el gradiente de la función de costo debe ser lo suficientemente grande y predecible para servir como una buena guía para el algoritmo de aprendizaje. Las funciones que saturan (se vuelven muy planas) socavan este objetivo porque hacen que el gradiente se vuelva muy pequeño. En muchos casos esto sucede porque las funciones de activación utilizadas para producir la salida de las unidades ocultas o las unidades de salida se saturan. El log-verosimilitud negativo ayuda a evitar este problema para muchos modelos. Muchas unidades de salida implican una función que puede saturarse cuando su argumento es muy negativo. El registro en la función de costo de verosimilitud logarítmica negativa deshace el efecto de algunas unidades de producción. Discutiremos la interacción entre la función de costo y la elección de la unidad de producción en la sección 6.2.2.

Una propiedad inusual de la función de costo de la entropía cruzada que se utiliza para realizar la estimación de máxima verosimilitud es que, por lo general, no tiene un valor mínimo cuando se aplica a los modelos comúnmente utilizados en la práctica. Para las variables de salida discretas, la mayoría de los modelos están parametrizados de tal manera que no pueden representar una probabilidad de cero o uno, pero pueden acercarse arbitrariamente a hacerlo. La regresión logística es un ejemplo de este modelo. Para variables de salida de valor real, si el modelo

puede controlar la densidad de la distribución de salida (por ejemplo, aprendiendo el parámetro de varianza de una distribución de salida gaussiana), entonces es posible asignar una densidad extremadamente alta a las salidas del conjunto de entrenamiento correcto, lo que da como resultado una entropía cruzada que se acerca al infinito negativo. Técnicas de regularización descritas en el capítulo 7 proporcionan varias formas diferentes de modificar el problema de aprendizaje para que el modelo no pueda obtener una recompensa ilimitada de esta manera.

6.2.1.2 Aprendizaje de estadísticas condicionales

En lugar de aprender una distribución de probabilidad completa $p(y | X; \theta)$ a menudo queremos aprender solo una estadística condicional de y dado X .

Por ejemplo, podemos tener un predictor $f(X; \theta)$ que deseamos predecir la media de y .

Si usamos una red neuronal lo suficientemente poderosa, podemos pensar en la red neuronal como capaz de representar cualquier función f de una amplia clase de funciones, con esta clase limitada solo por características tales como continuidad y acotación en lugar de tener una forma paramétrica específica. Desde este punto de vista, podemos ver la función de costo como una **funcional** en lugar de simplemente una función. Un funcional es un mapeo de funciones a números reales. Por lo tanto, podemos pensar en el aprendizaje como elegir una función en lugar de simplemente elegir un conjunto de parámetros. Podemos diseñar nuestro costo funcional para que su mínimo ocurra en alguna función específica que deseemos. Por ejemplo, podemos diseñar el funcional de costos para que tenga su mínimo en la función que mapea X al valor esperado de y dado X . Resolver un problema de optimización con respecto a una función requiere una herramienta matemática llamada **cálculo de variaciones**, descrito en la sección 19.4.2. No es necesario saber cálculo de variaciones para entender el contenido de este capítulo. Por el momento, solo es necesario entender que el cálculo de variaciones puede usarse para derivar los siguientes dos resultados.

Nuestro primer resultado derivado usando cálculo de variaciones es que resolviendo el problema de optimización

$$F = \text{mínimo de argumentos } m_{x,y} \sim p_{\text{datos}}(y|x) / f(x) \quad (6.14)$$

rendimientos

$$f^*(X) = m_{y \sim p_{\text{datos}}(y|x)}[y], \quad (6.15)$$

siempre que esta función se encuentre dentro de la clase sobre la que optimizamos. En otras palabras, si pudiéramos entrenar en un número infinito de muestras de la verdadera distribución de generación de datos, minimizar la función de costo del error cuadrático medio da una función que predice la media de y por cada valor de x .

Diferentes funciones de costo dan diferentes estadísticas. Un segundo resultado derivado del cálculo de variaciones es que

$$F = \underset{F}{\text{mínimo de argumento}} \max_{y \sim p_{\text{datos}}} \|y - F(X)\|_1 \quad (6.16)$$

produce una función que predice la *mediana* valor de y para cada X , siempre que dicha función pueda ser descrita por la familia de funciones sobre las que optimizamos. Esta función de costo es comúnmente llamada **error absoluto medio**.

Desafortunadamente, el error cuadrático medio y el error absoluto medio a menudo conducen a resultados deficientes cuando se utilizan con optimización basada en gradientes. Algunas unidades de salida que saturan producen gradientes muy pequeños cuando se combinan con estas funciones de costo. Esta es una de las razones por las que la función de costo de entropía cruzada es más popular que el error cuadrático medio o el error absoluto medio, incluso cuando no es necesario estimar una distribución completa, $p(y | X)$.

6.2.2 Unidades de salida

La elección de la función de costo está estrechamente relacionada con la elección de la unidad de producción. La mayoría de las veces, simplemente usamos la entropía cruzada entre la distribución de datos y la distribución del modelo. La elección de cómo representar la salida determina la forma de la función de entropía cruzada.

Cualquier tipo de unidad de red neuronal que pueda usarse como salida también puede usarse como unidad oculta. Aquí, nos enfocamos en el uso de estas unidades como salidas del modelo, pero en principio también se pueden usar internamente. Revisamos estas unidades con detalles adicionales sobre su uso como unidades ocultas en la sección 6.3.

A lo largo de esta sección, suponemos que la red feedforward proporciona un conjunto de características ocultas definidas por $h = f(X; \theta)$. La función de la capa de salida es entonces proporcionar alguna transformación adicional de las características para completar la tarea que debe realizar la red.

6.2.2.1 Unidades lineales para distribuciones de salida gaussianas

Un tipo simple de unidad de salida es una unidad de salida basada en una transformación afín sin no linealidad. Estos a menudo se llaman simplemente unidades lineales.

características dadas h , una capa de unidades de salida lineales produce un vector $\hat{y} = W \cdot h + b$.

Las capas de salida lineal se utilizan a menudo para producir la media de una distribución gaussiana condicional:

$$p_{\text{ag}}(y | X) = \text{norte}(y; \hat{y}, I). \quad (6.17)$$

Entonces, maximizar la probabilidad logarítmica es equivalente a minimizar el error cuadrático medio.

El marco de máxima verosimilitud también hace que sea sencillo aprender la covarianza de la Gaussiana, o hacer que la covarianza de la Gaussiana sea una función de la entrada. Sin embargo, la covarianza debe limitarse a ser una matriz definida positiva para todas las entradas. Es difícil satisfacer tales restricciones con una capa de salida lineal, por lo que normalmente se utilizan otras unidades de salida para parametrizar la covarianza. Los enfoques para modelar la covarianza se describen brevemente, en la sección 6.2.2.4.

Debido a que las unidades lineales no se saturan, presentan poca dificultad para los algoritmos de optimización basados en gradientes y pueden usarse con una amplia variedad de algoritmos de optimización.

6.2.2.2 Unidades sigmoideas para distribuciones de salida de Bernoulli

Muchas tareas requieren predecir el valor de una variable binaria y . Los problemas de clasificación con dos clases pueden formularse de esta forma.

El enfoque de máxima verosimilitud consiste en definir una distribución de Bernoulli sobre y condicionado a X .

Una distribución de Bernoulli se define por un solo número. La red neuronal solo necesita predecir $PAG(y=1 | X)$. Para que este número sea una probabilidad válida, debe estar en el intervalo $[0, 1]$.

Satisfacer esta restricción requiere un cuidadoso esfuerzo de diseño. Supongamos que tuviéramos que usar una unidad lineal y umbralizar su valor para obtener una probabilidad válida:

$$PAG(y=1 | X) = \text{máximo } 0,1 \text{ minuto}, w \cdot h + b \quad . \quad (6.18)$$

De hecho, esto definiría una distribución condicional válida, pero no seríamos capaces de entrenarla de manera muy efectiva con descenso de gradiente. En cualquier momento que $w \cdot h + b$ desviado fuera del intervalo unitario, el gradiente de la salida del modelo con respecto a sus parámetros sería 0. Un gradiente de 0 suele ser problemático porque el algoritmo de aprendizaje ya no tiene una guía sobre cómo mejorar los parámetros correspondientes.

En cambio, es mejor usar un enfoque diferente que asegure que siempre haya un fuerte gradiente cada vez que el modelo tenga una respuesta incorrecta. Este enfoque se basa en el uso de unidades de salida sigmoideas combinadas con la máxima verosimilitud.

Una unidad de salida sigmoidea se define por

$$\hat{y} = \sigma(w \cdot h + b) \quad (6.19)$$

dónde σ es la función sigmoidea logística descrita en la sección 3.10.

Podemos pensar que la unidad de salida sigmoidea tiene dos componentes. Primero, usa una capa lineal para calcular $z = w \cdot h + b$. A continuación, utiliza la función de activación sigmoidea para convertir z en una probabilidad.

Omitimos la dependencia de X por el momento para discutir cómo definir una distribución de probabilidad sobre y usando el valor z . El sigmoide puede motivarse construyendo una distribución de probabilidad no normalizada $PAG(y)$, que no suma 1. Luego podemos dividir por una constante adecuada para obtener una distribución de probabilidad válida. Si comenzamos con la suposición de que las probabilidades logarítmicas no normalizadas son lineales en y y z , podemos exponenciar para obtener las probabilidades no normalizadas. Luego normalizamos para ver que esto produce una distribución de Bernoulli controlada por una transformación sigmoidal de z .

$$\text{registro } PAG(y) = yz \quad (6.20)$$

$$PAG(y) = \exp(yz) \quad (6.21)$$

$$PAG(y) = \frac{\exp(yz)}{1 - \exp(yz)} \quad (6.22)$$

$$PAG(y) = \sigma(2y - 1)z. \quad (6.23)$$

Las distribuciones de probabilidad basadas en exponenciación y normalización son comunes en toda la literatura de modelado estadístico. La variable que define tal distribución sobre variables binarias se llama **logit**.

Este enfoque para predecir las probabilidades en el espacio logarítmico es natural para usar con el aprendizaje de máxima probabilidad. Debido a que la función de costo utilizada con máxima verosimilitud es $-\text{registro } PAG(y / X)$, el registro en la función de costo deshace el \exp del sigmoide. Sin este efecto, la saturación del sigmoide podría impedir que el aprendizaje basado en gradientes progrese bien. La función de pérdida para el aprendizaje de máxima verosimilitud de un Bernoulli parametrizado por un sigmoide es

$$J(\theta) = -\text{registro } PAG(y / X) \quad (6.24)$$

$$= -\text{registro } \sigma(2y - 1)z \quad (6.25)$$

$$= \zeta(1 - 2y)z. \quad (6.26)$$

Esta derivación hace uso de algunas propiedades de la sección 3.10. Al reescribir la pérdida en términos de la función softplus, podemos ver que se satura solo cuando $(1 - 2y)z$ es muy negativo. Por lo tanto, la saturación ocurre solo cuando el modelo ya tiene la respuesta correcta, cuando $y = 1$ y z es muy positivo, o $y = 0$ y z es muy negativo. Cuando z tiene el signo incorrecto, el argumento de la función softplus,

$(1 - 2y)z$, puede simplificarse a $/z/$. Como $/z/$ se vuelve grande mientras z tiene el signo incorrecto, las asíntotas de la función softplus simplemente devuelven su argumento $/z/$. La derivada con respecto a z esas asíntotas definen $\sigma(z)$, por lo que, en el límite de lo extremadamente incorrecto z , la función softplus no reduce el gradiente en absoluto. Esta propiedad es muy útil porque significa que el aprendizaje basado en gradientes puede actuar para corregir rápidamente un error z .

Cuando usamos otras funciones de pérdida, como el error cuadrático medio, la pérdida puede saturarse en cualquier momento. $\sigma(z)$ se satura a 0 cuando z se vuelve muy negativo y se satura a 1 cuando z se vuelve muy positivo. El gradiente puede reducirse demasiado para ser útil para el aprendizaje siempre que esto suceda, ya sea que el modelo tenga la respuesta correcta o incorrecta. Por esta razón, la máxima verosimilitud es casi siempre el enfoque preferido para entrenar unidades de salida sigmoideas.

Analíticamente, el logaritmo del sigmoide siempre está definido y es finito, porque el sigmoide devuelve valores restringidos al intervalo abierto $(0, 1)$, en lugar de usar todo el intervalo cerrado de probabilidades válidas $[0, 1]$. En implementaciones de software, para evitar problemas numéricos, es mejor escribir el log-verosimilitud negativo como una función $\text{de}z$, más que como una función de $y = \sigma(z)$. Si la función sigmoidea se desborda hasta cero, entonces tomando el logaritmo de y da infinito negativo.

6.2.2.3 Unidades Softmax para distribuciones de salida Multinoulli

Cada vez que deseemos representar una distribución de probabilidad sobre una variable discreta con n valores posibles, podemos usar la función softmax. Esto puede verse como una generalización de la función sigmoidea que se utilizó para representar una distribución de probabilidad sobre una variable binaria.

Las funciones Softmax se utilizan con mayor frecuencia como salida de un clasificador, para representar la distribución de probabilidad sobre n diferentes clases. Más raramente, las funciones softmax se pueden usar dentro del propio modelo, si deseamos que el modelo elija entre uno de n diferentes opciones para alguna variable interna.

En el caso de variables binarias, deseamos producir un solo número

$$\hat{y} = PAG(y=1 | X). \quad (6.27)$$

Debido a que este número necesitaba estar entre 0 y 1, y como queríamos que el logaritmo del número tuviera un buen comportamiento para la optimización basada en gradientes de la verosimilitud logarítmica, optamos por predecir un número $z = \text{registro} PAG(y=1 | X)$. Exponenciar y normalizar nos dio una distribución de Bernoulli controlada por la función sigmoidea.

Para generalizar al caso de una variable discreta con n valores, ahora necesitamos producir un vector \hat{y} , con $\hat{y} = PAG(y=y_0 | X)$. Requerimos no sólo que cada elemento de \hat{y} esté entre 0 y 1, pero también que todo el vector sume a 1 de modo que represente una distribución de probabilidad válida. El mismo enfoque que funcionó para la distribución de Bernoulli se generaliza para la distribución multinomial. Primero, una capa lineal predice probabilidades logarítmicas no normalizadas:

$$z = W \cdot h + b, \quad (6.28)$$

dónde z = registro $PAG(y=y_0 | X)$. La función softmax puede entonces exponenciar y normalizar z para obtener lo deseado \hat{y} . Formalmente, la función softmax viene dada por

$$\text{softmax}(z) = -\frac{\text{Exp}(z)}{\sum_j \text{Exp}(z_j)}. \quad (6.29)$$

Al igual que con el sigmoide logístico, el uso del ExpLa función funciona muy bien cuando se entrena el softmax para generar un valor objetivo y usando la máxima verosimilitud logarítmica. En este caso, deseamos maximizar registro $PAG(y=i; z) =$ registro $\text{softmax}(z_i)$. Definiendo el softmax en términos de Exp es natural porque el registro en el log-verosimilitud puede deshacer el Exp del softmax:

$$\text{registro softmax}(z) = z_i - \text{registro} \sum_j \text{Exp}(z_j). \quad (6.30)$$

El primer término de la ecuación 6.30 muestra que la entrada z_i siempre tiene una contribución directa a la función de costo. Debido a que este término no puede saturar, sabemos que el aprendizaje puede proceder, incluso si la contribución de z_i al segundo término de la ecuación 6.30 se vuelve muy pequeño. Al maximizar la verosimilitud logarítmica, el primer término alienta a z_i ser empujado hacia arriba, mientras que el segundo término encorta a todos z_j ser empujado hacia abajo. Para ganar algo de intuición para el segundo término, observe que este término se puede aproximar aproximadamente por $\text{máximo}_j z_j$. Esta aproximación se basa en la idea de que $\text{Exp}(z_k)$ es insignificante para cualquier z_k que sea notablemente menor que $\text{máximo}_j z_j$. La intuición que podemos obtener de esta aproximación es que la función de costo de verosimilitud negativa siempre penaliza fuertemente la predicción incorrecta más activa. Si el answer ya tiene la entrada más grande para softmax, entonces el $-z_i$ término y el registro $\sum_j \text{Exp}(z_j) \approx \text{máximo}_j z_j = z_i$ los términos se cancelarán aproximadamente. Este ejemplo contribuirá poco al costo general de capacitación, que estará dominado por otros ejemplos que aún no están clasificados correctamente.

Hasta ahora hemos discutido un solo ejemplo. En general, la máxima verosimilitud no regularizada impulsará al modelo a aprender parámetros que impulsan al softmax a predecir