Machine Learning [Mitchell-1997]

Aprendizaje Basado en Instancias [Chapter 8]

Data Scientist
Jhoan Esteban Ruiz Borja Msc

A diferencia de los métodos de aprendizaje que construyen una descripción general y explícita de la función objetivo cuando se proporcionan ejemplos de formación, los métodos de aprendizaje basados en la instancia simplemente almacenan los ejemplos de formación. Generalizar más allá de estos ejemplos se pospone hasta que una nueva instancia debe ser clasificada. Cada vez que se encuentra una nueva instancia de consulta, se examina su relación con los ejemplos previamente almacenados para asignar un valor de función de destino para la nueva instancia. El aprendizaje basado en instancias incluye el vecino más cercano y los métodos de regresión localmente ponderados que asumen que las instancias pueden ser representadas como puntos en un espacio euclidiano. También incluye métodos de razonamiento basados en casos que utilizan representaciones simbólicas más complejas para instancias. Los métodos basados en la instancia a veces se denominan métodos de aprendizaje "perezosos" porque retrasan el procesamiento hasta que una nueva instancia deba ser clasificada. Una ventaja clave de este tipo de aprendizaje retrasado o perezoso es que en lugar de estimar la función de destino una vez para todo el espacio de instancia, estos métodos pueden estimarlo localmente y de manera diferente para cada nueva instancia que se va a clasificar.

Introducción [8.1]

Los métodos de aprendizaje basados en la instancia, como el vecino más cercano y la regresión localmente ponderada, son enfoques conceptualmente sencillos para aproximar funciones de valor real o discreto. El aprendizaje en estos algoritmos consiste simplemente en almacenar los datos de entrenamiento presentados. Cuando se encuentra una nueva instancia de consulta, unos conjuntos de instancias relacionadas de forma similares se recuperan de la memoria y se utilizan para clasificar la nueva instancia de consulta. Una diferencia clave entre estos enfoques y los métodos discutidos en otros capítulos es que los enfoques basados en instancias pueden construir una aproximación diferente a la función de destino para cada instancia de consulta distinta que debe clasificarse. De hecho, muchas técnicas construyen sólo una aproximación local a la función de destino que se aplica en la vecindad de la nueva instancia de consulta, y nunca construyen una aproximación diseñada para funcionar bien en todo el espacio de instancia. Esto tiene ventajas significativas cuando la función objetivo es muy compleja, pero todavía puede ser descrita por una colección de aproximaciones locales menos complejas.

Los métodos basados en instancias también pueden usar representaciones simbólicas más complejas para instancias. En el aprendizaje basado en casos, las instancias se representan de esta manera y el proceso para identificar instancias "vecinas" se elabora en consecuencia. El

razonamiento basado en casos se ha aplicado a tareas tales como almacenar y reutilizar la experiencia pasada en una mesa de ayuda, razonar sobre casos legales haciendo referencia a casos anteriores y resolver problemas complejos de programación reutilizando porciones relevantes de problemas previamente resueltos.

Una desventaja de los enfoques basados en instancias es que el costo de clasificar nuevas instancias puede ser alto. Esto se debe al hecho de que casi todos los cálculos tienen lugar en el momento de la clasificación en lugar de cuando los ejemplos de entrenamiento se encuentran por primera vez. Por lo tanto, las técnicas para indexar eficientemente los ejemplos de entrenamiento son un problema práctico importante en la reducción de la computación requerida en el momento de la consulta. Una segunda desventaja de muchos enfoques basados en instancias, especialmente enfoques de vecindad más cercana, es que normalmente consideran todos los atributos de las instancias al intentar recuperar ejemplos de entrenamiento similares de la memoria. Si el concepto de destino depende sólo de algunos de los muchos atributos disponibles, entonces las instancias que son realmente más "similares" pueden ser una gran distancia.

En la siguiente sección presentamos el algoritmo de aprendizaje k-NEAREST NEIGHBOR, incluyendo varias variantes de este enfoque ampliamente utilizado. En la sección subsiguiente se analiza la regresión localmente ponderada, un método de aprendizaje que construye aproximaciones locales a la función objetivo y que puede ser visto como una generalización de los algoritmos k-NEAREST NEIGHBOR. Describimos las redes de función de base radial, que proporcionan un puente interesante entre los algoritmos de aprendizaje basados en la instancia y la red neuronal. La siguiente sección discute el razonamiento basado en casos, un enfoque basado en la instancia que emplea representaciones simbólicas e inferencia basada en el conocimiento. Esta sección incluye un ejemplo de aplicación del razonamiento basado en casos a un problema en el diseño de ingeniería. Finalmente, discutimos las diferencias fundamentales en capacidades que distinguen los métodos de aprendizaje perezosos discutidos en este capítulo de los métodos de aprendizaje ansiosos discutidos en los otros capítulos de este libro.

Aprendizaje de k-vecinos más cercanos (k-NEAREST NEIGHBOR) [8.2]

El método más básico basado en la instancia es el algoritmo k-NEAREST NEIGHBOR. Este algoritmo asume que todas las instancias corresponden a puntos en el espacio **n-dimensional** $\mathfrak{R}^n$. Los vecinos más próximos de una instancia se definen en términos de la distancia euclidiana estándar. Más precisamente, deje que una instancia arbitraria $\boldsymbol{x}$ sea descrita por el vector de entidad

$$\left(\boldsymbol{a_1}(\boldsymbol{x}), \boldsymbol{a_2}(\boldsymbol{x}), \ldots, \boldsymbol{a_n}(\boldsymbol{x})\right)$$

Donde $\boldsymbol{a_r}(\boldsymbol{x})$ denota el valor del $\boldsymbol{r}$**-ésimo** atributo de la instancia $\boldsymbol{x}$. Entonces la distancia entre dos instancias $\boldsymbol{x_i}$ y $\boldsymbol{x_j}$ se define como $\boldsymbol{d}(\boldsymbol{x_i}, \boldsymbol{x_j})$, donde

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} \left(a_r(x_i) - a_r(x_j)\right)^2}$$

En el aprendizaje de vecinos más cercanos, la función objetivo puede ser un valor discreto o un valor real. Consideremos primero el aprendizaje de funciones de valor discreto de la forma $f: \Re^n \rightarrow V$, donde V es el conjunto finito $\{v_1, \ldots v_s\}$. El algoritmo k-NEAREST NEIGHBOR para aproximar una función objetivo discreta se da en la Tabla 8.1. Como se muestra aquí, el valor $\hat{f}(x_q)$ devuelto por este algoritmo como su estimación de $f(x_q)$ es sólo el valor más común de f entre los k ejemplos de entrenamiento más cercano a x_q. Si elegimos $k = 1$, entonces el algoritmo **1-NEAREST NEIGHBOR** asigna a $\hat{f}(x_q)$ el valor $f(x_i)$ donde x_i es la instancia de entrenamiento más cercana a x_q. Para valores mayores de k, el algoritmo asigna el valor más común entre los k ejemplos de entrenamiento más cercanos.

La figura 8.1 ilustra el funcionamiento del algoritmo k-NEAREST NEIGHBOR para el caso en el que las instancias son puntos en un espacio bidimensional y donde la función objetivo es booleana. Los ejemplos de entrenamiento positivos y negativos se muestran con "+" y "-", respectivamente. También se muestra un punto de consulta x_q. Obsérvese que el algoritmo 1-NEAREST NEIGHBOR clasifica x_q como un ejemplo positivo en esta figura, mientras que el algoritmo 5-NEAREST NEIGHBOR lo clasifica como un ejemplo negativo.

¿Cuál es la naturaleza del espacio H de hipótesis implícitamente considerado por el algoritmo k-NEAREST NEIGHBOR? Obsérvese que el algoritmo k-NEAREST NEIGHBOR nunca forma una hipótesis general explícita $\hat{f}$ con respecto a la función objetivo f. Simplemente calcula la clasificación de cada nueva instancia de consulta según sea necesario. Sin embargo, todavía podemos preguntar cuál es la función general implícita, o qué clasificaciones se asignarían si tuviéramos que mantener los ejemplos de entrenamiento constantes y consultar el algoritmo con cada posible instancia en X. El diagrama en el lado derecho de la figura 8.1 muestra la forma de esta superficie de decisión inducida por 1-NEARBOR NEIGHBOR en todo el espacio de la instancia. La superficie de decisión es una combinación de poliedros convexos que rodean cada uno de los ejemplos de entrenamiento. Para cada ejemplo de entrenamiento, el poliedro indica el conjunto de puntos de consulta cuya clasificación estará completamente determinada por ese ejemplo de entrenamiento. Los puntos de consulta fuera del poliedro están más cerca de algún otro ejemplo de entrenamiento. Este tipo de diagrama se llama a menudo el diagrama de Voronoi del conjunto de ejemplos de entrenamiento.

El algoritmo k-NEAREST NEIGHBOR se adapta fácilmente a la aproximación de funciones objetivo de valor continuo. Para lograr esto, tenemos que el algoritmo calcule el valor medio de los k ejemplos de entrenamiento más cercanos en lugar de calcular su valor más común. Más precisamente, para aproximar una función de destino real $f: \Re^n \rightarrow \Re$ reemplazamos la línea final del algoritmo anterior por la línea

$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} f(x_i)}{k} \qquad (8.1)$$

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

TABLE 8.1
The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

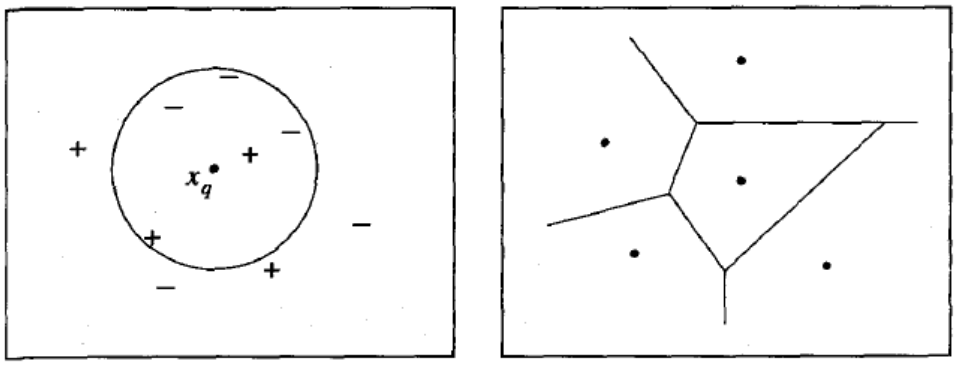

FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

Distance-Weighted KNN [8.2.1]

Un refinamiento obvio al algoritmo ***k***-NEAREST NEIGHBOR es ponderar la contribución de cada uno de los ***k*** vecinos de acuerdo con su distancia al punto de consulta x_q dando mayor peso a los vecinos más cercanos. Por ejemplo, en el algoritmo de la tabla 8.1, que se aproxima a las funciones objetivo de valores discretos, podríamos ponderar el voto de cada vecino de acuerdo con el cuadrado inverso de su distancia desde x_q.

Esto puede lograrse reemplazando la línea final del algoritmo por

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \qquad (8.2)$$

Donde

$$w_i = \frac{1}{d(x_q, x_i)^2} \qquad (8.3)$$

Para acomodar el caso en que el punto de consulta x_q coincide exactamente con una de las instancias de entrenamiento x_i y el denominador $d(x_q, x_i)^2$ es, por tanto, cero, asignamos $\hat{f}(x_q)$ a $f(x_i)$ en este caso. Son varios ejemplos de entrenamiento, asignamos la clasificación mayoritaria entre ellos.

Podemos distanciar las instancias de las funciones objetivo real de una manera similar, reemplazando la línea final del algoritmo en este caso por

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \qquad (8.4)$$

Donde w_i es definido en la ecuación (8.3). Obsérvese que el denominador en la ecuación (8.4) es una constante que normaliza las contribuciones de los distintos pesos (por ejemplo, asegura que si $f(x_i) = c$ para todos los ejemplos de entrenamiento, entonces $\hat{f}(x_q) \leftarrow c$ también).

Observe que todas las variantes del algoritmo k-NEAREST NEIGHBOR consideran sólo los k vecinos más próximos para clasificar el punto de consulta. Una vez que añadimos la ponderación de distancia, no hay realmente ningún daño al permitir que todos los ejemplos de entrenamiento tengan una influencia en la clasificación de los x_q porque los ejemplos muy lejanos tendrán muy poco efecto sobre $\hat{f}(x_q)$. La única desventaja de considerar todos los ejemplos es que nuestro clasificador funcionará más lentamente. Si todos los ejemplos de entrenamiento se consideran al clasificar una nueva instancia de consulta, llamamos al algoritmo un método global. Si sólo se consideran los ejemplos de entrenamiento más cercanos, lo llamamos un método local. Cuando la regla de la ecuación (8.4) se aplica como un método global, utilizando todos los ejemplos de entrenamiento, se conoce como el método de Shepard (Shepard 1968).

Observaciones sobre el algoritmo k-NEAREST NEIGHBOR [8.2.2]

El algoritmo **k-NEAREST NEIGHBOR** ponderado a distancia es un método de inferencia inductiva altamente efectivo para muchos problemas prácticos. Es robusto para los datos de entrenamientos ruidosos y bastante eficaces cuando se proporciona un conjunto suficientemente grande de datos de entrenamiento. Tenga en cuenta que al tomar el promedio

ponderado de los k vecinos más cercanos al punto de consulta, puede suavizar el impacto de ejemplos de entrenamiento ruidosos aislados.

¿Cuál es el sesgo inductivo de k-**NEAREST NEIGHBOR**? La base para clasificar nuevos puntos de consulta se entiende fácilmente según los diagramas de la Figura 8.1. El sesgo inductivo corresponde a la suposición de que la clasificación de una instancia x_q será más similar a la clasificación de otras instancias que están cerca en la distancia euclidiana.

Un problema práctico al aplicar los algoritmos k-**NEAREST NEIGHBOR** es que la distancia entre las instancias se calcula en función de todos los atributos de la instancia (es decir, en todos los ejes del espacio euclidiano que contiene las instancias). Esto se encuentra en contraste con los métodos tales como los sistemas de aprendizaje de árboles de reglas y decisiones que seleccionan solo un subconjunto de los atributos de instancia al formar la hipótesis. Para ver el efecto de esta política, considere aplicar k-**NEAREST NEIGHBOR** a un problema en el que cada instancia se describe mediante 20 atributos, pero donde solo 2 de estos atributos son relevantes para determinar la clasificación para la función de destino particular. En este caso, las instancias que tienen valores idénticos para los 2 atributos relevantes, pueden, no obstante, estar distantes entre sí en el espacio de instancia de 20 dimensiones. Como resultado, la métrica de similitud utilizada por k-**NEAREST NEIGHBOUR** (dependiendo de los 20 atributos) será engañosa. La distancia entre vecinos estará dominada por la gran cantidad de atributos irrelevantes. Esta dificultad, que surge cuando muchos atributos irrelevantes están presentes, a veces se conoce como la maldición de la dimensionalidad. Los enfoques del vecino más cercano son especialmente sensibles a este problema.

Un enfoque interesante para superar este problema es ponderar cada atributo de forma diferente al calcular la distancia entre dos instancias. Esto corresponde a estirar los ejes en el espacio euclidiano, acortar los ejes que corresponden a atributos menos relevantes y alargar los ejes que corresponden a atributos más relevantes. La cantidad por la cual se debe estirar cada eje se puede determinar automáticamente usando un enfoque de validación cruzada. Para ver cómo, primero tenga en cuenta que deseamos estirar (multiplicar) el eje j-ésimo por algún factor z_j, donde los valores $z_j, \ldots, z_n$ se eligen para minimizar el verdadero error de clasificación del algoritmo de aprendizaje. En segundo lugar, tenga en cuenta que este verdadero error puede estimarse utilizando la validación cruzada. Por lo tanto, un algoritmo es seleccionar un subconjunto aleatorio de los datos disponibles para usar como ejemplos de entrenamiento, luego determinar los valores de $z_j, \ldots, z_n$ que conducen al error mínimo en la clasificación de los ejemplos restantes. Al repetir este proceso varias veces, la estimación de estos factores de ponderación puede ser más precisa. Este proceso de estiramiento de los ejes para optimizar el rendimiento de k-**NEARESNT EIGHBOR** proporciona un mecanismo para suprimir el impacto de los atributos irrelevantes.

Una alternativa aún más drástica es eliminar por completo los atributos menos relevantes del espacio de instancia. Esto es equivalente a establecer algunos de los factores de escala z_i a cero. Moore y Lee (1994) discuten métodos eficientes de validación cruzada para seleccionar subconjuntos relevantes de los atributos para los algoritmos k-**NEARESNT EIGHBOR**. En

particular, exploran métodos basados en la validación cruzada de dejar uno fuera, en el cual el conjunto de $\boldsymbol{m}$ instancias de capacitación se divide repetidamente en un conjunto de entrenamiento de tamaño $\boldsymbol{m-1}$ y un conjunto de prueba de tamaño 1, de todas las formas posibles. Este enfoque leave-one-out se implementa fácilmente en los algoritmos $\boldsymbol{k}$-**NEARESNT EIGHBOR** porque no se requiere esfuerzo adicional de entrenamiento cada vez que se redefine el conjunto de entrenamiento. Tenga en cuenta que ambos enfoques anteriores se pueden ver como un estiramiento de cada eje por algún factor constante. Alternativamente, podríamos estirar cada eje por un valor que varía en el espacio de la instancia. Sin embargo, a medida que aumentamos el número de grados de libertad disponibles para el algoritmo para redefinir su métrica de distancia de esa manera, también aumentamos el riesgo de sobreajuste. Por lo tanto, el enfoque de estirar localmente los ejes es mucho menos común.

Un problema práctico adicional en la aplicación de $\boldsymbol{k}$-**NEARESNT EIGHBOR** es la indexación de memoria eficiente. Debido a que este algoritmo demora todo el procesamiento hasta que se recibe una nueva consulta, es posible que se requiera un cálculo significativo para procesar cada consulta nueva. Se han desarrollado varios métodos para indexar los ejemplos de entrenamiento almacenados para que los vecinos más cercanos puedan identificarse de manera más eficiente a algún costo adicional en la memoria. Uno de estos métodos de indexación es el **kd-tree** (Bentley 1975; Friedman et al., 1977), en el cual las instancias se almacenan en las hojas de un árbol, con instancias cercanas almacenadas en el mismo nodo o en los nodos cercanos. Los nodos internos del árbol ordenan la nueva consulta $\boldsymbol{x_q}$, a la hoja correspondiente al probar los atributos seleccionados de $\boldsymbol{x_q}$.

Una nota sobre terminología [8.2.3]

Gran parte de la literatura sobre métodos de vecinos más cercanos y regresión local ponderada utiliza una terminología que ha surgido del campo del reconocimiento estadístico de patrones. Al leer esa literatura, es útil conocer los siguientes términos:

- La regresión significa aproximar una función objetivo de valor real.
- Residual es el error $\hat{f}(x) - f(x)$ al aproximar la función objetivo.
- La función Kernel es la función de distancia que se usa para determinar el peso de cada ejemplo de entrenamiento. En otras palabras, la función kernel es la función $\boldsymbol{K}$ tal que $\boldsymbol{w_i = K\left(d(x_i, x_q)\right)}$.

REGRESIÓN LOCALMENTE PONDERADA [8.3]

Se puede pensar que los enfoques del vecino más cercano descritos en la sección anterior se aproximan a la función objetivo $\boldsymbol{f(x)}$ en el único punto de consulta $\boldsymbol{x = x_q}$. La regresión ponderada localmente es una generalización de este enfoque. Construye una aproximación explícita a $\boldsymbol{f}$ sobre una región local que rodea a $\boldsymbol{x_q}$. La regresión ponderada localmente usa ejemplos de entrenamiento ponderados a distancia o cercanos para formar esta aproximación

local a $\boldsymbol{f}$. Por ejemplo, podríamos aproximar la función objetivo en el entorno que rodea a $\boldsymbol{x_q}$ utilizando una función lineal, una función cuadrática, una red neuronal de múltiples capas o alguna otra forma funcional. La frase "regresión ponderada localmente" se denomina local porque la función se aproxima basándose únicamente en los datos cercanos al punto de consulta, ponderada porque la contribución de cada ejemplo de capacitación se pondera por su distancia desde el punto de consulta y la regresión porque este es el término ampliamente utilizado en la comunidad estadística de aprendizaje para el problema de aproximar las funciones de valores reales.

Dada una nueva instancia de consulta $\boldsymbol{x_q}$, el enfoque general en la regresión ponderada localmente es construir una aproximación $\hat{\boldsymbol{f}}$ que se ajuste a los ejemplos de capacitación en el vecindario que rodea a $\boldsymbol{x_q}$. Esta aproximación se utiliza para calcular el valor $\hat{\boldsymbol{f}}(\boldsymbol{x_q})$, que se genera como el valor objetivo estimado para la instancia de consulta. La descripción de $\hat{\boldsymbol{f}}$ se puede eliminar, porque se calculará una aproximación local diferente para cada instancia instancia de consulta.

Regresión lineal ponderada localmente [8.3.1]

Consideremos el caso de la regresión ponderada localmente en la cual la función objetivo $\boldsymbol{f}$ se aproxima a $\boldsymbol{x_q}$ usando una función lineal de la forma

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{w_0} + \boldsymbol{w_1 a_1}(\boldsymbol{x}) + \cdots + \boldsymbol{w_n a_n}(\boldsymbol{x})$$

Como antes, $\boldsymbol{a_1}(\boldsymbol{x})$ denota el valor del $\boldsymbol{i}$-ésimo atributo de la instancia $\boldsymbol{x}$.

Recordemos que en el Capítulo 4 discutimos métodos como el descenso de gradiente para encontrar los coeficientes $\boldsymbol{w_0} \ldots \boldsymbol{w_n}$ para minimizar el error al ajustar tales funciones lineales a un conjunto dado de ejemplos de entrenamiento. En ese capítulo, estábamos interesados en una aproximación global a la función objetivo. Por lo tanto, derivamos métodos para elegir pesos que minimicen el error cuadrado sumado sobre el conjunto $\boldsymbol{D}$ de ejemplos de entrenamiento

$$\boldsymbol{E} = \frac{\boldsymbol{1}}{\boldsymbol{2}} \sum_{\boldsymbol{x \in D}} \left(\boldsymbol{f}(\boldsymbol{x}) - \hat{\boldsymbol{f}}(\boldsymbol{x}) \right)^{\boldsymbol{2}} \qquad (\boldsymbol{8.5})$$

lo que nos llevó a la regla de entrenamiento de descenso de gradiente

$$\Delta \boldsymbol{w_j} = \boldsymbol{\eta} \sum_{\boldsymbol{x \in D}} \left(\boldsymbol{f}(\boldsymbol{x}) - \hat{\boldsymbol{f}}(\boldsymbol{x}) \right) \boldsymbol{a_j}(\boldsymbol{x}) \qquad (\boldsymbol{8.6})$$

donde $\boldsymbol{\eta}$ es una velocidad de aprendizaje constante, y donde la regla de entrenamiento se ha vuelto a expresar a partir de la notación del capítulo 4 para ajustarla a nuestra notación actual (es decir, $t \rightarrow \hat{f}(x)$, y $x_j \rightarrow a_j(x)$).

¿Cómo modificaremos este procedimiento para derivar una aproximación local en lugar de una aproximación global? La manera simple es redefinir el criterio de error $\boldsymbol{E}$ para enfatizar

el ajuste de los ejemplos de capacitación local. Tres posibles criterios se dan a continuación. Tenga en cuenta que escribimos el error $\boldsymbol{E(x_q)}$ para enfatizar el hecho de que ahora el error se está definiendo como una función del punto de consulta $\boldsymbol{x_q}$.

1. Minimice el error cuadrado sobre los $\boldsymbol{k}$ vecinos más cercanos:
$$E_1(x_q) = \frac{1}{2} \sum_{x \in k \ vecinos \ nbrs \ de \ x_q} \left(f(x) - \hat{f}(x)\right)^2$$
2. Minimice el error al cuadrado en todo el conjunto $\boldsymbol{D}$ de ejemplos de entrenamiento, mientras pondera el error de cada ejemplo de entrenamiento por alguna función decreciente $\boldsymbol{K}$ de su distancia desde $\boldsymbol{x_q}$:
$$E_2(x_q) = \frac{1}{2} \sum_{x \in D} \left(f(x) - \hat{f}(x)\right)^2 K\left(d(x_q, x)\right)$$
3. Combine 1 y 2:
$$E_3(x_q) = \frac{1}{2} \sum_{x \in k \ \ vecinos \ nbrs \ de \ x_q} \left(f(x) - \hat{f}(x)\right)^2 K\left(d(x_q, x)\right)$$

El segundo criterio es quizás el más estéticamente agradable porque permite que cada ejemplo de capacitación tenga un impacto en la clasificación de $\boldsymbol{x_q}$. Sin embargo, este enfoque requiere un cálculo que crece linealmente con el número de ejemplos de entrenamiento. El criterio tres es una buena aproximación al criterio dos y tiene la ventaja de que el costo computacional es independiente del número total de ejemplos de capacitación; su costo depende solo del número $\boldsymbol{k}$ de vecinos considerados.

Si elegimos el criterio tres anterior y revocamos la regla de descenso del gradiente utilizando el mismo estilo de argumento que en el Capítulo 4, obtenemos la siguiente regla de entrenamiento (ver Ejercicio 8.1):

$$\Delta w_j = \eta \sum_{x \in k \ vecinoes \ nbrs \ de \ x_q} K\left(d(x_q, x)\right)\left(f(x) - \hat{f}(x)\right) a_j(x) \qquad (8.7)$$

Observe que las únicas diferencias entre esta nueva regla y la regla dada por la ecuación (8.6) son que la contribución de la instancia $\boldsymbol{x}$ a la actualización de ponderación ahora se multiplica por la penalización de distancia $\boldsymbol{K\left(d(x_q, x)\right)}$, y que el error se resume solo en los $\boldsymbol{k}$ ejemplos de entrenamiento más cercanos. De hecho, si estamos ajustando una función lineal a un conjunto fijo de ejemplos de entrenamiento, entonces hay métodos mucho más eficientes que el descenso de gradiente para resolver directamente los coeficientes deseados $\boldsymbol{w_0 \ldots w_n}$ urna. Atkeson et al. (1997a) y Bishop (1995) estudian varios de estos métodos.

Comentarios sobre la regresión ponderada localmente [8.3.2]

Más arriba consideramos el uso de una función lineal para aproximar $\boldsymbol{f}$ en el entorno de la instancia de consulta $\boldsymbol{x_q}$. La literatura sobre la regresión ponderada localmente contiene una amplia gama de métodos alternativos para ponderar a distancia los ejemplos de entrenamiento, y una variedad de métodos para aproximar localmente la función objetivo. En la mayoría de los casos, la función objetivo se aproxima mediante una función constante, lineal o cuadrática. No se encuentran a menudo formas funcionales más complejas porque (1) el costo de ajustar funciones más complejas para cada instancia de consulta es prohibitivamente alto, y (2) estas aproximaciones simples modelan la función objetivo bastante bien en una subregión suficientemente pequeña del espacio de instancia.

FUNCIONES BASE RADIAL [8.4]

Un enfoque para la aproximación de funciones que está estrechamente relacionado con la regresión ponderada por distancia y también con las redes neuronales artificiales es el aprendizaje con funciones de base radial (Powell 1987, Broomhead y Lowe 1988, Moody y Darken 1989). En este enfoque, la hipótesis aprendida es una función de la forma

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u\big(d(x_u, x)\big) \qquad (8.8)$$

donde cada x_u es una instancia de X y donde la función del núcleo $K_u\big(d(x_u, x)\big)$ se define de manera que disminuye a medida que aumenta la distancia $d(x_u, x)$. Aquí k es una constante proporcionada por el usuario que especifica el número de funciones del núcleo que se incluirán. Aunque $\hat{f}(x)$ es una aproximación global a $f(x)$, la contribución de cada uno de los términos $K_u\big(d(x_u, x)\big)$ se localiza en una región cercana al punto x_u. Es común elegir cada función $K_u\big(d(x_u, x)\big)$ como una función gaussiana (ver Tabla 5.4) centrada en el punto x_u con alguna varianza a σ_u^2.

$$K_u\big(d(x_u, x)\big) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Restringiremos nuestra discusión aquí a esta función común del kernel de Gauss. Como lo muestran Hartman et al. (1990), la forma funcional de la ecuación (8.8) puede aproximar cualquier función con un error arbitrariamente pequeño, siempre que haya un número suficientemente grande k de dichos núcleos gaussianos y siempre que el ancho σ^2 de cada núcleo pueda especificarse por separado.

La función dada por la ecuación (8.8) puede verse como una red de dos capas en la que la primera capa de unidades calcula los valores de los diversos $K_u\big(d(x_u, x)\big)$ y donde la segunda capa calcula una combinación lineal de estos valores de unidad de primera capa. En la figura 8.2 se ilustra una red de función de base radial (RBF) de ejemplo.

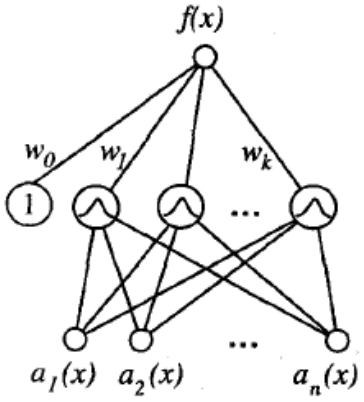

FIGURE 8.2
A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u. Therefore, its activation will be close to zero unless the input x is near x_u. The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

Dado un conjunto de ejemplos de entrenamiento de la función de destino, las redes de RBF típicamente se entrenan en un proceso de dos etapas. Primero, se determina el número k de unidades ocultas y cada unidad oculta u se define eligiendo los valores de x_u y σ_u^2 que definen su función kernel $K_u\big(d(x_u, x)\big)$. Segundo, los pesos w_u están entrenados para maximizar el ajuste de la red a los datos de entrenamiento, utilizando el criterio de error global dado por la ecuación (8.5). Debido a que las funciones del kernel se mantienen fijas durante esta segunda etapa, los valores de peso lineal w_u se pueden entrenar de manera muy eficiente.

Se han propuesto varios métodos alternativos para elegir una cantidad adecuada de unidades ocultas o, de forma equivalente, funciones de kernel. Un enfoque es asignar una función de núcleo Gaussiana para cada ejemplo de entrenamiento $\big(x_i, f(x_i)\big)$, centrando este gaussiano en el punto x_i. Cada uno de estos núcleos puede tener asignado el mismo ancho σ^2. Dado este enfoque, la red RBF aprende una aproximación global a la función objetivo en la que cada ejemplo de entrenamiento $\big(x_i, f(x_i)\big)$ puede influir en el valor de $\hat{f}$ solo en el entorno de x_i. Una de las ventajas de esta elección de funciones del núcleo es que permite que la red RBF se ajuste exactamente a los datos de entrenamiento. Es decir, para cualquier conjunto de m ejemplos de entrenamiento, los pesos $w_0 \ldots w_m$ para combinar las m funciones gaussianas del kernel se puede configurar de manera que $\hat{f}(x_i) = f(x_i)$ para cada ejemplo de entrenamiento $\big(x_i, f(x_i)\big)$.

Un segundo enfoque es elegir un conjunto de funciones de núcleo que sea más pequeño que el número de ejemplos de entrenamiento. Este enfoque puede ser mucho más eficiente que el primer enfoque, especialmente cuando el número de ejemplos de capacitación es grande. El conjunto de funciones del kernel puede distribuirse con centros espaciados uniformemente a lo largo del espacio de instancia X. Alternativamente, podemos desear distribuir los centros de forma no uniforme, especialmente si se encuentra que las instancias mismas se distribuyen de forma no uniforme sobre X. En este último caso, podemos escoja los centros de función del kernel seleccionando aleatoriamente un subconjunto de las instancias de capacitación, con lo que se muestrea la distribución subyacente de las instancias. Alternativamente, podemos identificar clusters prototípicos de instancias, luego agregar una función kernel centrada en cada clúster. La colocación de las funciones del kernel de esta manera se puede lograr usando algoritmos de agrupación no supervisados que se ajusten a las instancias de

capacitación (pero no a sus valores objetivo) a una mezcla de gaussianos. El algoritmo EM discutido en la Sección 6.12.1 proporciona un algoritmo para elegir los medios de una mezcla de k Gaussianos para que se ajuste mejor a las instancias observadas. En el caso del algoritmo EM, los medios se eligen para maximizar la probabilidad de observar las instancias x_i, dados los k medios estimados. Tenga en cuenta que el valor de la función objetivo $f(x_i)$ de la instancia no entra en el cálculo de los centros del kernel mediante métodos de agrupamiento no supervisados. La única función de los valores objetivo $f(x_i)$ en este caso es determinar los pesos de la capa de salida w_u.

En resumen, las redes de funciones de base radial proporcionan una aproximación global a la función objetivo, representada por una combinación lineal de muchas funciones locales del núcleo. El valor para cualquier función dada del kernel no es insignificante solo cuando la entrada x cae en la región definida por su centro y ancho particular. Por lo tanto, la red se puede ver como una combinación lineal sin problemas de muchas aproximaciones locales a la función objetivo. Una ventaja clave de las redes RBF es que se pueden entrenar de manera mucho más eficiente que las redes feedforward entrenadas con BACKPROPAGATTIOhNis. se deduce del hecho de que la capa de entrada y la capa de salida de un RBF se entrenan por separado.

(Data Mining concepts and Techniques [Han-3ed-2012]) (Chapter 9.5)

Estudiantes perezosos (o aprendiendo de sus vecinos) [9.5]

Los métodos de clasificación discutidos hasta ahora en este libro: inducción del árbol de decisión, clasificación bayesiana, clasificación basada en reglas, clasificación por retro-propagación, máquinas de vectores de soporte y clasificación basada en la minería de reglas de asociación, son todos ejemplos de alumnos ansiosos. Los estudiantes ansiosos, cuando se les da un conjunto de tuplas de entrenamiento, construirán un modelo de generalización (es decir, clasificación) antes de recibir nuevas tuplas para clasificar. Podemos pensar que el modelo aprendido está listo y ansioso por clasificar tuplas que no se habían visto anteriormente.

Imagínese un enfoque perezoso contrastante, en el que el alumno en cambio espera hasta el último minuto antes de hacer cualquier construcción de modelo para clasificar una tupla de prueba determinada. Es decir, cuando se le da una tupla de entrenamiento, un aprendiz perezoso simplemente la almacena (o solo hace un pequeño procesamiento) y espera hasta que se le dé una tupla de prueba. Solo cuando ve la tupla de prueba realiza una generalización para clasificar la tupla en función de su similitud con las tuplas de entrenamiento almacenadas. A diferencia de los métodos de aprendizaje ansiosos, los estudiantes perezosos hacen menos trabajo cuando se presenta una tupla de capacitación y más trabajo cuando hacen una clasificación o predicción numérica. Debido a que los estudiantes perezosos almacenan las tuplas de capacitación o "instancias", también se les conoce como aprendices basados en instancias, aunque todo el aprendizaje se basa esencialmente en instancias.

Al hacer una clasificación o predicción numérica, los estudiantes perezosos pueden ser computacionalmente caros. Requieren técnicas de almacenamiento eficientes y se adaptan bien a la implementación en hardware paralelo. Ofrecen poca explicación o información sobre la estructura de los datos. Los estudiantes perezosos, sin embargo, naturalmente apoyan el aprendizaje incremental. Son capaces de modelar espacios de decisión complejos que tienen formas hiperpoligonales que pueden no ser tan fácilmente descriptas por otros algoritmos de aprendizaje (como formas hiperrectangulares modeladas por árboles de decisión). En esta sección, observamos dos ejemplos de estudiantes perezosos: $\boldsymbol{k}$ clasificadores de vecinos más cercanos (sección 9.5.1) y clasificadores de razonamiento basados en casos (sección 9.5.2).

k-Clasificadores de vecinos más cercanos [9.5.1]

El método del $\boldsymbol{k}$-vecino más cercano se describió por primera vez a principios de la década de 1950. El método requiere mucha mano de obra cuando se le administran grandes conjuntos de entrenamiento y no ganó popularidad hasta la década de 1960, cuando se incrementó el poder de computación. Desde entonces se ha utilizado ampliamente en el área de reconocimiento de patrones.

Los clasificadores de vecinos más cercanos se basan en el aprendizaje por analogía, es decir, al comparar una tupla de prueba dada con tuplas de entrenamiento que son similares a ella. Las tuplas de entrenamiento están descritas por $\boldsymbol{n}$ atributos. Cada tupla representa un punto en un espacio $\boldsymbol{n}$-dimensional. De esta forma, todas las tuplas de entrenamiento se almacenan en un espacio de patrón $\boldsymbol{n}$-dimensional. Cuando se le da una tupla desconocida, un clasificador de $\boldsymbol{k}$-vecino más cercano busca en el espacio de patrón las $\boldsymbol{k}$ tuplas de entrenamiento que están más cerca de la tupla desconocida. Estas tuplas de entrenamiento $\boldsymbol{k}$ son los $\boldsymbol{k}$ "vecinos más cercanos" de la tupla desconocida.

La "cercanía" se define en términos de una métrica de distancia, como la distancia euclidiana. La distancia euclidiana entre dos puntos o tuplas, por ejemplo, $\boldsymbol{X_1 = (x_{11},\ x_{12}, \dots, x_{1n})}$, y $\boldsymbol{X_2 = (x_{21},\ x_{22}, \dots, x_{2n})}$ es

$$\boldsymbol{dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}} \qquad \boldsymbol{(9.22)}$$

En otras palabras, para cada atributo numérico, tomamos la diferencia entre los valores correspondientes de ese atributo en la tupla $\boldsymbol{X_1}$ y en la tupla $\boldsymbol{X_2}$, cuadramos esta diferencia y la acumulamos. La raíz cuadrada se toma del conteo total de distancia acumulada. Por lo general, normalizamos los valores de cada atributo antes de usar Eq. (9.22). Esto ayuda a evitar que los atributos con rangos inicialmente grandes (p. Ej., Ingresos (incomes)) superen los atributos con rangos inicialmente más pequeños (p. Ej., Atributos binarios). La

normalización Min-max, por ejemplo, puede usarse para transformar un valor $\boldsymbol{v}$ de un atributo numérico $\boldsymbol{A}$ a un valor $\boldsymbol{v'}$ en el rango $[\mathbf{0}, \mathbf{1}]$ mediante computación

$$\boldsymbol{v' = \frac{v - min_A}{max_A - min_A}}, \qquad (\mathbf{9.23})$$

donde $\boldsymbol{min_A}$ y $\boldsymbol{max_A}$ son los valores mínimo y máximo del atributo $\boldsymbol{A}$. El Capítulo 3 describe otros métodos para la normalización de datos como una forma de transformación de datos.

Para la clasificación de $\boldsymbol{k}$-vecino más cercano, a la tupla desconocida se le asigna la clase más común entre sus $\boldsymbol{k}$ vecinos más cercanos. Cuando $\boldsymbol{k = 1}$, a la tupla desconocida se le asigna la clase de la tupla de entrenamiento que está más cerca de ella en el espacio del patrón. Los clasificadores de vecinos más cercanos también se pueden usar para la predicción numérica, es decir, para devolver una predicción de valor real para una tupla desconocida dada. En este caso, el clasificador devuelve el valor promedio de las etiquetas de valor real asociadas con los $\boldsymbol{k}$ vecinos más cercanos de la tupla desconocida.

"Pero, ¿cómo se puede calcular la distancia para atributos que no son numéricos, sino nominales (o categóricos) como el color?" La discusión anterior supone que los atributos utilizados para describir las tuplas son todos numéricos. Para los atributos nominales, un método simple es comparar el valor correspondiente del atributo en la tupla $\boldsymbol{X_1}$ con el de la tupla $\boldsymbol{X_2}$. Si los dos son idénticos (p. Ej., Las tuplas $\boldsymbol{X_1}$ y $\boldsymbol{X_2}$ tienen el color azul), la diferencia entre ambas se toma como $\mathbf{0}$. Si las dos son diferentes (p. Ej., La tupla $\boldsymbol{X_1}$ es azul pero la tupla $\boldsymbol{X_2}$ es roja), entonces se considera que la diferencia es $\mathbf{1}$. Otros métodos pueden incorporar esquemas más sofisticados para la clasificación diferencial (por ej., cuando se asigna un puntaje de diferencia más grande, por ejemplo, para azul y blanco que para azul y negro).

"¿Qué pasa con los valores perdidos?" En general, si falta el valor de un atributo $\boldsymbol{A}$ dado en la tupla $\boldsymbol{X_1}$ y/o en la tupla $\boldsymbol{X_2}$, asumimos la diferencia máxima posible. Supongamos que cada uno de los atributos ha sido asignado al rango $[\mathbf{0}, \mathbf{1}]$. Para los atributos nominales, consideramos que el valor de la diferencia es $\mathbf{1}$ si faltan uno o ambos valores correspondientes de $\boldsymbol{A}$. Si $\boldsymbol{A}$ es numérico y falta tanto de las dos tuplas $\boldsymbol{X_1}$ y $\boldsymbol{X_2}$, entonces la diferencia también se toma como $\mathbf{1}$. Si solo falta un valor y el otro (que llamaremos $\boldsymbol{v'}$) está presente y normalizado, entonces podemos tomar la diferencia ser $|\mathbf{1} - \boldsymbol{v'}|$ o $|\mathbf{0} - \boldsymbol{v'}|$ (es decir, $\mathbf{1} - \boldsymbol{v'}$ o $\boldsymbol{v'}$), el que sea mayor.

"¿Cómo puedo determinar un buen valor para $\boldsymbol{k}$, el número de vecinos?" Esto puede determinarse experimentalmente. Comenzando con $\boldsymbol{k = 1}$, usamos un conjunto de prueba para estimar la tasa de error del clasificador. Este proceso se puede repetir cada vez incrementando $\boldsymbol{k}$ para permitir un vecino más. Se puede seleccionar el valor de $\boldsymbol{k}$ que da la tasa de error mínima. En general, cuanto mayor sea el número de tuplas de entrenamiento, mayor será el valor de $\boldsymbol{k}$ (de modo que la clasificación y las decisiones de predicción numérica pueden basarse en una porción mayor de las tuplas almacenadas). A medida que el número de tuplas de entrenamiento se aproxima al infinito y $\boldsymbol{k = 1}$, la tasa de error no puede

ser peor que el doble de la tasa de error de Bayes (siendo este último el mínimo teórico). Si $\boldsymbol{k}$ también se aproxima al infinito, la tasa de error se aproxima a la tasa de error de Bayes.

Los clasificadores de vecinos más cercanos usan comparaciones basadas en la distancia que intrínsecamente asignan el mismo peso a cada atributo. Por lo tanto, pueden sufrir una mala precisión cuando se les dan atributos ruidosos o irrelevantes. El método, sin embargo, se ha modificado para incorporar la ponderación de atributos y la poda de las tuplas de datos ruidosas. La elección de una métrica de distancia puede ser crítica. La distancia de Manhattan (bloque de ciudad) (Sección 2.4.4) u otras mediciones de distancia también pueden ser utilizadas.

Los clasificadores de vecinos cercanos pueden ser extremadamente lentos al clasificar las tuplas de prueba. Si $\boldsymbol{D}$ es una base de datos de entrenamiento de tuplas $|\boldsymbol{D}|$ y $\boldsymbol{k} = \mathbf{1}$, entonces se requieren comparaciones $\boldsymbol{O}(|\boldsymbol{D}|)$ para clasificar una tupla de prueba determinada. Al preclasificar y ordenar las tuplas almacenadas en árboles de búsqueda, el número de comparaciones se puede reducir a $\boldsymbol{O}(\mathbf{log}(|\boldsymbol{D}|))$. La implementación paralela puede reducir el tiempo de ejecución a una constante, es decir, $\boldsymbol{O}(\mathbf{1})$ que es independiente de $|\boldsymbol{D}|$.

Otras técnicas para acelerar el tiempo de clasificación incluyen el uso de cálculos de distancia parcial y la edición de las tuplas almacenadas. En el método de **distancia parcial**, calculamos la distancia en función de un subconjunto de los $\boldsymbol{n}$ atributos. Si esta distancia excede un umbral, entonces el cómputo adicional para la tupla almacenada dada se detiene, y el proceso pasa a la siguiente tupla almacenada. El método de **edición** elimina las tuplas de entrenamiento que resultan inútiles. Este método también se conoce como **poda** o **condensación** porque reduce la cantidad total de tuplas almacenadas.

Razonamiento basado en casos [9.5.2]

Los clasificadores de razonamiento basado en casos (CBR) utilizan una base de datos de soluciones de problemas para resolver problemas nuevos. A diferencia de los clasificadores de vecinos más cercanos, que almacenan tuplas de entrenamiento como puntos en el espacio euclidiano, CBR almacena las tuplas o "casos" para la resolución de problemas como descripciones simbólicas complejas. Las aplicaciones comerciales de CBR incluyen la resolución de problemas para las mesas de ayuda de servicio al cliente, donde los casos describen problemas de diagnóstico relacionados con el producto. CBR también se ha aplicado a áreas como ingeniería y derecho, donde los casos son diseños técnicos o sentencias legales, respectivamente. La educación médica es otra área para CBR, donde los historiales de casos y tratamientos de pacientes se utilizan para ayudar a diagnosticar y tratar a pacientes nuevos.

Cuando se le presente un nuevo caso para clasificar, un razonador basado en un caso primero verificará si existe un caso de capacitación idéntico. Si se encuentra uno, se devuelve la solución correspondiente a ese caso. Si no se encuentra un caso idéntico, el razonador basado en casos buscará casos de capacitación que tengan componentes que sean similares a los del caso nuevo. Conceptualmente, estos casos de capacitación se pueden considerar como vecinos del nuevo caso. Si los casos se representan como gráficos, esto implica buscar

subgrafos que son similares a los subgrafos dentro del nuevo caso. El razonador basado en casos trata de combinar las soluciones de los casos de capacitación vecinos para proponer una solución para el nuevo caso. Si surgen incompatibilidades con las soluciones individuales, puede ser necesario retroceder para buscar otras soluciones. El razonador basado en casos puede emplear conocimiento de fondo y estrategias de resolución de problemas para proponer una solución combinada factible.

Los desafíos en el razonamiento basado en casos incluyen encontrar una buena métrica de similitud (por ejemplo, para subgrafos coincidentes) y métodos adecuados para combinar soluciones. Otros desafíos incluyen la selección de características destacadas para indexar casos de capacitación y el desarrollo de técnicas de indexación eficientes. Una compensación entre precisión y eficiencia evoluciona a medida que el número de casos almacenados se vuelve muy grande. A medida que este número aumenta, el razonador basado en casos se vuelve más inteligente. Sin embargo, después de cierto punto, la eficacia del sistema se verá afectada a medida que aumente el tiempo requerido para buscar y procesar casos relevantes. Al igual que con los clasificadores de vecinos más cercanos, una solución es editar la base de datos de entrenamiento. Los casos que son redundantes o que no han demostrado ser útiles pueden descartarse por el bien de un mejor rendimiento. Estas decisiones, sin embargo, no son claras y su automatización sigue siendo un área activa de investigación.