

Capítulo 9

Aprendizaje Basado en Instancias

En este tipo de aprendizaje, se almacenan los ejemplos de entrenamiento y cuando se quiere clasificar un nuevo objeto, se extraen los objetos más parecidos y se usa su clasificación para clasificar al nuevo objeto.

Contrario a los otros esquemas vistos, el proceso de aprendizaje es trivial y el de clasificación es el que consume el mayor tiempo.

Este tipo de aprendizaje también se conoce como *lazy learning* o *memory-based learning* donde los datos de entrenamiento se procesan solo hasta que se requiere (cuando se requiere constestar alguna pregunta), y la relevancia de los datos se mide en función de una medida de distancia.

9.1 Vecinos más cercanos

El algoritmo de k-NN (*k-nearest neighbours*) es el más simple.

El algoritmo es robusto con ejemplos que tienen ruido.

Los vecinos más cercanos a una instancia se obtienen, en caso de atributos continuos, utilizando la distancia Euclideana sobre los n posibles atributos

Tabla 9.1: El algoritmo de los k vecinos más cercanos.

Entrenamiento:

almacena todos los ejemplos de entrenamiento $(x, f(x))$

Clasificación:

Dada una instancia x_q :

Sean x_1, \dots, x_k los k vecinos más cercanos a x_q .

Entonces:

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

donde: $\delta(a, b) = 1$ si $a = b$ y 0 en caso contrario.

(luego veremos otro tipo de distancias):

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

El resultado de la clasificación de k -NN puede ser discreto o continuo.

En el caso discreto, el resultado de la clasificación es la clase más común de los k -vecinos (ver tabla ??).

La forma que se genera con $k = 1$ es un diagrama de Voronoi alrededor de las instancias almacenadas. A una nueva instancia se le asigna la clasificación del vecino más cercano.

Para clasificaciones continuas, se puede tomar la media de las clasificaciones.

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

Un extensión obvia al algoritmo es pesar las clasificaciones de los vecinos de acuerdo a su distancia con el objeto a clasificar (la clasificación de vecinos

más cercanos tienen más peso). Promedio ponderado (*weighed average*) promedia la salida de los puntos pesados inversamente por su distancia.

Para clases discretas:

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

donde: $w_i = \frac{1}{d(x_q, x_i)^2}$ (si la distancia es 0 entonces $w = 0$).

Para clase continuas:

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

Una suposición es que los vecinos más cercanos nos dan la mejor clasificación y esto se hace utilizando todos los atributos.

El problema es que es posible que se tengan muchos atributos irrelevantes que dominen sobre la clasificación (e.g., 2 atributos relevantes dentro de 20 irrelevantes no pintan).

Una posibilidad es pesar las distancias de cada atributo, dándole más peso a los atributos más relevantes.

Otra posibilidad es tratar de determinar estos pesos con ejemplos conocidos de entrenamiento. Alterando los pesos para minimizar el error.

Finalmente, también se pueden eliminar los atributos que se consideran irrelevantes.

Un elemento práctico adicional, tiene que ver con el almacenamiento de los ejemplos. En este caso se han sugerido representaciones basadas en árboles (*kd-trees*) donde las instancias están distribuidas en base a su cercanía.

9.2 Regresión pesada local

Locally weighed regression es una generalización que construye una función que ajusta los datos de entrenamiento que están en la vecindad de x_q .

Se pueden usar funciones lineales, cuadráticas, redes neuronales, etc. Si utilizamos una función lineal:

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

Podemos usar gradiente descendiente para ajustar los pesos que minimizan el error.

El error lo podemos expresar por diferencias de error al cuadrado de la siguiente forma:

$$E(W) = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Lo que queremos es determinar el vector de pesos que minimice el error E . Esto se logra alterando los pesos en la dirección que produce el máximo descenso en la superficie del error.

La dirección de cambio se obtiene mediante el gradiente. El gradiente nos especifica la dirección que produce el máximo incremento, por lo que el mayor descenso es el negativo de la dirección.

La regla de actualización de pesos es entonces:

$$\begin{aligned} W &\leftarrow W + \Delta W \\ \Delta W &= -\alpha \nabla E \end{aligned}$$

donde α es el factor de aprendizaje (qué tanto le creemos al error para ajustar nuestros pesos).

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \\ &= \sum_{x \in D} (f(x) - \hat{f}(x)) \frac{\partial}{\partial w_i} (f(x) - \vec{w} \cdot \vec{a}_x) \\ &= \sum_{x \in D} (f(x) - \hat{f}(x)) (-a_{i,x}) \end{aligned}$$

Por lo que:

$$\Delta w_i = \alpha \sum_{x \in D} (f(x) - \hat{f}(x))(-a_{i,x})$$

Para modificar los pesos se puede hacer:

1. Minimizar el error cuadrado usando los k vecinos más cercanos.

$$E(W) = \frac{1}{2} \sum_{x \in k \text{ vecinas más cercanos}} (f(x) - \hat{f}(x))^2$$

2. Minimizar el error cuadrado usando todos los ejemplos pesados por su distancia a x_q .

$$E(W) = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

3. Minimizar el error cuadrado usando los k vecinos más cercanos pesados por su distancia a x_q .

$$E(W) = \frac{1}{2} \sum_{x \in k \text{ vecinas más cercanos}} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

Para el último caso, la regla de actualización es entonces:

$$\Delta w_i = \alpha \sum_{x \in k \text{ vecinas más cercanos}} K(d(x_q, x))(f(x) - \hat{f}(x))(-a_{i,x})$$

9.3 Funciones de Distancia

Las funciones de distancia las podemos clasificar en:

- Funciones globales: se usa la misma función de distancia en todo el espacio.

- Funciones basadas en el *query*. Los parámetros de la función de distancia se ajustan con cada *query*, típicamente minimizando el error con validación cruzada.
- Funciones basadas en puntos. Cada dato tiene asociado su propia función de distancia

El cambiar/ajustar la función de distancia puede mejorar las predicciones.

Las funciones de distancia típicas para datos continuos son:

- Euclideana

$$d_E(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j (\mathbf{x}_j - \mathbf{q}_j)^2} = \sqrt{(\mathbf{x} - \mathbf{q})^T (\mathbf{x} - \mathbf{q})}$$

- Euclideana pesada diagonalmente

$$d_m(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j (m_j (\mathbf{x}_j - \mathbf{q}_j)^2)} = \sqrt{(\mathbf{x} - \mathbf{q})^T \mathbf{M}^T \mathbf{M} (\mathbf{x} - \mathbf{q})} = d_E(\mathbf{M}\mathbf{x}, \mathbf{M}\mathbf{q})$$

donde m_j es el factor de escala en la dimensión j y \mathbf{M} es una matriz diagonal con $\mathbf{M}_{jj} = m_j$.

- Euclideana completa o Mahalanobis

$$d_M(\mathbf{x}, \mathbf{q}) = \sqrt{(\mathbf{x} - \mathbf{q})^T \mathbf{M}^T \mathbf{M} (\mathbf{x} - \mathbf{q})} = d_E(\mathbf{M}\mathbf{x}, \mathbf{M}\mathbf{q})$$

donde \mathbf{M} puede ser arbitraria.

- Normal o Minkowski

$$d_p(\mathbf{x}, \mathbf{q}) = \left(\sum_i |\mathbf{x}_i - \mathbf{q}_i|^p \right)^{\frac{1}{p}}$$

- Normal pesada diagonal o completa. Igual que la Minkowski pero incluyendo pesos.

Matrices (\mathbf{M}) diagonales hacen escalas radiales simétricas. Se pueden crear elipses con orientaciones arbitrarias incluyendo otros elementos fuera de la diagonal.

También se puede incluir un rango o escala en donde aplicar la función de generalización. Algunas opciones son:

- Selección de ancho de banda fijo. h es un valor constante, por lo que se usan valores constantes de datos y forma.
- Selección de los vecinos más cercanos. h se pone como la distancia a los k vecinos más cercanos y el volumen de datos cambia de acuerdo a la densidad de los datos más cercanos.
- Selección de banda global. h se ajusta globalmente por un proceso de optimización.
- Basado en el *query*. h se selecciona de acuerdo al *query* siguiendo un proceso de optimización.
- Basada en puntos. Cada dato tiene asociado su propia h .

9.4 Funciones de pesos o Kernels

Las funciones de peso deben de ser máximas a distancia cero y decaer suavemente con la distancia.

No es necesario normalizar el kernel, tampoco tiene que ser unimodal, y tiene que ser positivo siempre.

Algunos ejemplos son:

- Elevar la distancia a una potencia negativa

$$K(d) = \frac{1}{d^p}$$

- Para evitar infinitos (*inverse distance*):

$$K(d) = \frac{1}{1 + d^p}$$

- Uno de los más populares, es el kernel Gaussiano:

$$K(d) = \exp(-d^2)$$

- Uno relacionado es el exponencial:

$$K(d) = \exp(-|d|)$$

Los dos últimos tienen una extensión infinita que se puede truncar después de un cierto umbral.

- Kernel cuadrático o Epanechnikov o Bartlett-Priestley:

$$K(d) = \begin{cases} (1 - d^2) & \text{si } |d| < 1 \\ 0 & \text{de otra forma} \end{cases}$$

el cual ignora datos más alejados que 1 unidad.

- El kernel *tricube*:

$$K(d) = \begin{cases} (1 - |d|^3)^3 & \text{si } |d| < 1 \\ 0 & \text{de otra forma} \end{cases}$$

- Kernel de *uniform weighting*:

$$K(d) = \begin{cases} 1 & \text{si } |d| < 1 \\ 0 & \text{de otra forma} \end{cases}$$

- Kernel triangular:

$$K(d) = \begin{cases} 1 - |d| & \text{si } |d| < 1 \\ 0 & \text{de otra forma} \end{cases}$$

- Variante del triangular:

$$K(d) = \begin{cases} \frac{1-|d|}{|d|} & \text{si } |d| < 1 \\ 0 & \text{de otra forma} \end{cases}$$

Se pueden crear nuevos kernels. Según los autores la definición del kernel no es tan crítica.

9.5 Pocos datos y otras consideraciones

Un posible problema que puede surgir es cuando se tienen pocos datos. Algunas de las posibles soluciones es o tratar de introducir nuevos datos artificialmente y/o reducir la dimensionalidad usando un proceso de selección de variables.

La eficiencia de LWR depende de cuantos datos se tengan. Se puede usar una representación de *kd-trees* para acceder datos cercanos más rápidamente.

En general, LWR es más caro que vecinos más cercanos y promedios pesados.

Por otro lado, cualquier representación se puede usar para construir el modelo local (e.g., árboles de decisión, reglas, redes neuronales, etc.).

Una forma sencilla de hacerlo, es tomar los vecinos más cercanos y entrenar un modelo/clasificador con ellos.

Lo que se requiere para implantar un LWR es:

- Una función de distancia. Aquí la suposición más grande de LWR es que datos más cercanos son los más relevantes. La función de distancia no tiene que cumplir con los requerimientos de una métrica de distancia.
- Criterio de separabilidad. Se calcula un peso para cada punto dado por el kernel aplicado a la función de distancia. Este criterio es aparte de la función de predicción ($C = \sum_i [L(\hat{y}_i, y_i)K(d(\mathbf{x}_i, \mathbf{q}))]$)
- Suficientes datos para construir los modelos
- Datos con salida y_i .
- Representación adecuada.

Algunas posibles direcciones futuras de investigación incluyen:

- Combinar datos continuos y discretos
- Mejores formas de sintonización de parámetros

- Sintonización local a múltiples escalas
- Usar gradientes para sintonizar parámetros
- Definir cuánta validación cruzada es suficiente
- Usar métodos probabilísticos
- Olvidar datos
- Mejorar aspectos computacionales con muchos datos
- No hacer el aprendizaje completamente *lazy*

9.6 Funciones de bases radiales

Radial basis functions (RBF) utilizan una combinación de funciones *Kernel* que decrecen con la distancia (correspondería a $K(d(x_u, x))$ en las expresiones de arriba).

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

Para cada instancia x_u existe una función Kernel que decrece con la distancia a x_u .

Lo más común es escoger funciones normales o Gaussianas para las K s.

$$K_u(d(x_u, x)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}d^2(x_u, x)}$$

La función $\hat{f}(x)$ consiste básicamente de dos elementos: uno que calcula las funciones Kernel y otro los pesos de estas.

Estas se pueden aprender dentro de una red neuronal de dos capas (ver figure ??).

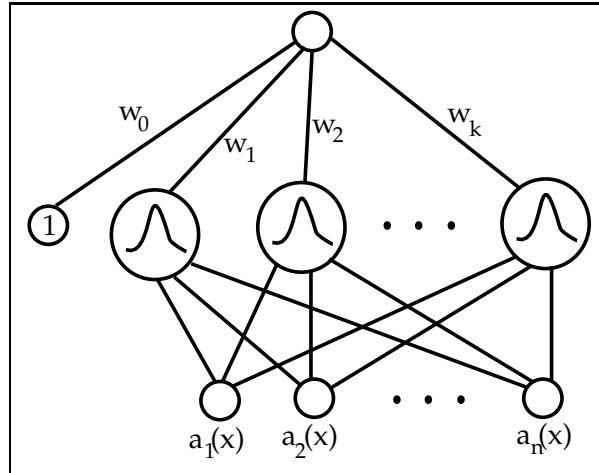


Figura 9.1: Una red de funciones bases radiales.

El entrenamiento se lleva en dos pasos. Se buscan las x_u y σ para cada función y después se buscan los pesos para las funciones minimizando el error global.

Posibilidades:

1. Centrar cada función en cada punto y a todas darles la misma desviación estandar.
2. Seleccionar un número limitado de funciones distribuidas uniformemente en el espacio de instancias.
3. Seleccionar funciones no distribuir las uniformemente (sobretudo si las instancias no estan distribuidas uniformemente).
 - Se puede hacer un muestreo sobre las instancias o tratar de identificar prototipos (posiblemente con un algoritmo de clustering).
 - Se puede utilizar EM para escoger k medias de las distribuciones Gaussianas que mejor se ajusten a los datos.

En el caso de RBF, se realiza un aprendizaje previo con las instancias de entrenamiento (como en los sistemas de aprendizaje que se han visto) y luego se trata de clasificar a las nuevas instancias.

9.7 Razonamiento Basado en Casos

Una alternativa para aprendizaje basado en instancias, es utilizar una representación simbólica mucho más rica para representar cada instancia.

Un Razonador Basado en Casos resuelve problemas nuevos mediante la adaptación de soluciones previas usadas para resolver problemas similares.

Las instancias o casos tienen normalmente representado el problema que solucionan, una descripción de cómo lo solucionaron, y el resultado obtenido.

Obviamente, las medidas de distancia se vuelven más complejas.

Las combinaciones de las instancias también se complica y generalmente involucra conocimiento del dominio y mecanismos de búsqueda y razonamiento sofisticados.