# Aprendizaje basado en instancias

- Los métodos de aprendizaje basados en instancias como el algoritmo de los vecinos más cercanos y regresión local se utilizan para aproximar funciones objetivo con valores discretos o continuos.

- El aprendizaje de esos algoritmos consiste simplemente en almacenar los datos del conjunto de entrenamiento.

- Cuando se presenta una nueva instancia, se recupera un conjunto de instancias similares de la memoria y se usan para clasificar esa nueva instancia.

- Una diferencia entre esta aproximación y otros métodos que hemos visto, es que las aproximaciones basadas en instancias pueden construir una aproximación diferente a la función objetivo para cada instancia nueva que debe clasificarse.

- Desventajas:

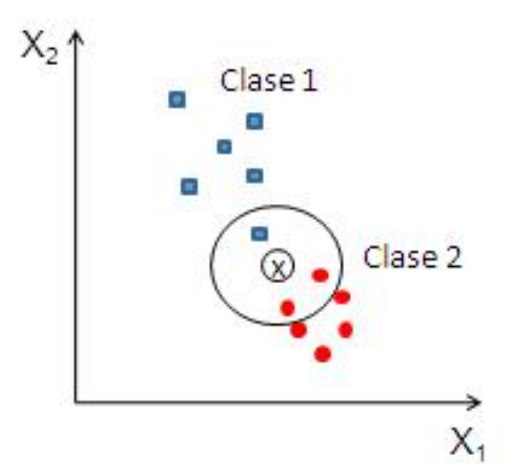
  - el costo de clasificar nuevas instancias puede ser alto.

    Esto se debe al hecho de que casi todos los cálculos se realizan en el momento de clasificar en vez de cuando se realiza el entrenamiento.

- generalmente consideran ***todos*** los atributos de las instancias cuando tratan de encontrar ejemplos de entrenamiento similares en la memoria.

  Si el concepto objetivo sólo depende de pocos atributos, entonces las instancias que realmente son las "más similares" pueden resultar a mayor distancia de la que realmente están.

# Aprendizaje de los $k$ –vecinos más cercanos




- Este algoritmo asume que todas las instancias corresponden a puntos en el espacio $\mathbb{R}^n$.

- Los vecinos más cercanos de una instancia se definen en términos de la distancia euclídea.

- Sea $x$ una instancia arbitraria descripta por el vector $\langle a_1(x), \cdots, a_n(x) \rangle$, donde $a_r(x)$ denota el valor del r-ésimo atributo de la instancia $x$.

- Entonces la distancia entre dos instancias $x_i$ y $x_j$ está definida por $d(x_i, x_j)$ donde

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

- En el aprendizaje del vecino más cercano la función objetivo puede tener tanto valores discretos como continuos.

## Caso discreto

- La función es de la forma $f: \mathbb{R}^n \to V$, donde $V$ es el conjunto finito $\{v_1, \cdots, v_n\}$.

# *Algoritmo de los $k$ – vecinos más cercanos*

- *Algoritmo de entrenamiento*

1. Para cada ejemplo de entrenamiento $\langle x, f(x) \rangle$:
   agregar el ejemplo a la lista *ejemplos_de_entrenamiento*

- *Algoritmo de clasificación*

2. Dada una instancia $x_q$ que debe clasificarse:

a. Sean $x_1, \cdots, x_k$ las $k$ instancias más cercanas a $x_q$ de *ejemplos_de_entrenamiento*

b. Devolver

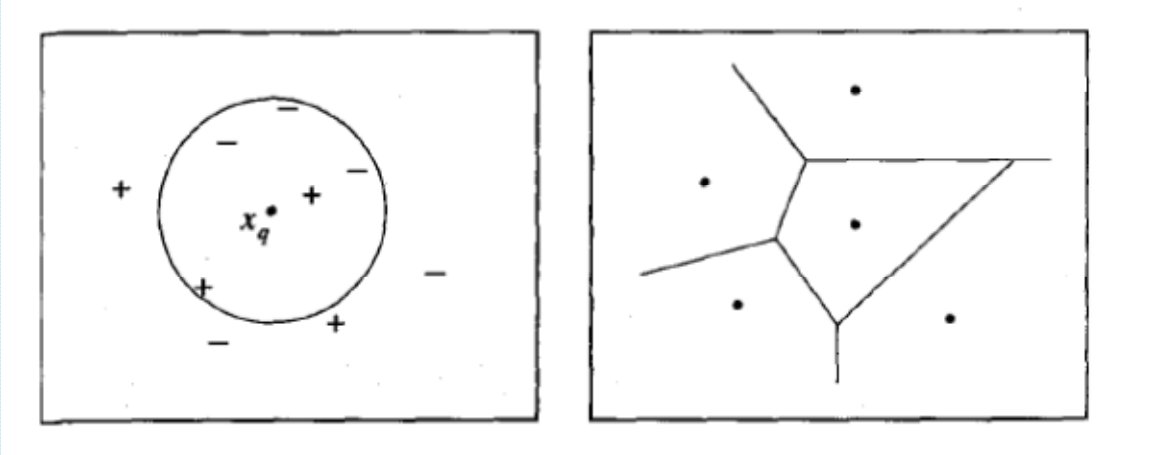
$$\hat{f}(x_q) \leftarrow \text{arg} \max_{v \in V} \delta(v, f(x_i))$$

donde $\delta(a, b) = \begin{cases} 1 & \text{si } a = b \\ 0 & \text{en otro caso} \end{cases}$

- El valor $\hat{f}(x_q)$ que es devuelto por el algoritmo es el estimador de $f(x_q)$, que es el valor más frecuente de $f$ entre los $k$ ejemplos de entrenamiento más cercanos a $x_q$.

- Si elegimos $k = 1$, entonces el algoritmo de los $k$ −vecinos más cercanos asigna el valor de $f(x_i)$ donde $x_i$ es la instancia de entrenamiento más cercana a $x_q$.

- La figura ilustra la operación del algoritmo de los $k$ –vecinos más cercanos para el caso donde las instancias son puntos en el espacio bidimensional y donde la función objetivo tiene valores booleanos.




- Los ejemplos positivos y negativos se denotan con "+" y "–" respectivamente.

- $x_q$ es un punto de consulta.

- El algoritmo del *1–vecino más cercano* clasifica a $x_q$ como un ejemplo positivo

- Pero el algoritmo de *los 5–vecinos más cercanos* lo clasifica como un ejemplo negativo.

- ¿Cuál es la naturaleza del espacio de hipótesis $H$ implícitamente considerada en el algoritmo de los $k$ –vecinos más cercanos?

    - El algoritmo de los $k$ –vecinos más cercanos nunca forma una hipótesis general explícita $\hat{f}$ para la función objetivo $f$.

- Simplemente calcula la clasificación de cada nueva instancia cuando se necesita.

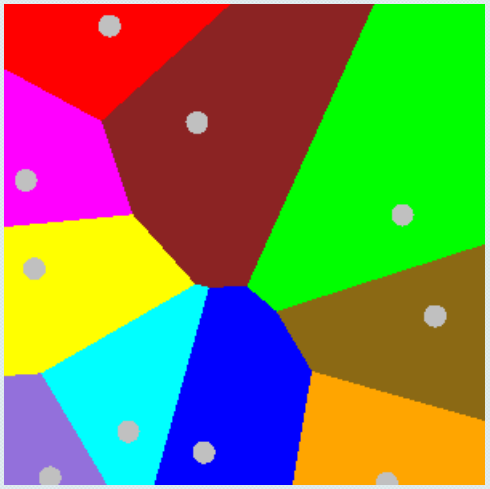
- El sistema de la derecha de la figura muestra la forma de la superficie de decisión inducida por el 1-vecino más cercano sobre el espacio completo de instancias.

  - La superficie de decisión es una combinación de poliedros convexos rodeando cada ejemplo de entrenamiento.

  - Para cada ejemplo de entrenamiento, el poliedro indica un conjunto de puntos de consulta cuya clasificación queda completamente determinada por ese ejemplo de entrenamiento.

- Los puntos consultados fuera del poliedro están cercanos a algún otro ejemplo de entrenamiento.

- Esta clase de diagrama se denomina diagrama de *Voronoi* del conjunto de ejemplos de entrenamiento.



## Caso continuo

- Para adaptar el algoritmo para aproximar funciones objetivo con valores continuos, hacemos que calcule la media de los *k* ejemplos de entrenamiento más cercanos en vez del valor más frecuente.

- De esta manera, para aproximar una función objetivo con valores reales $f: \mathbb{R}^n \rightarrow \mathbb{R}$ reemplazamos 2b. por:

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

# Algoritmo de la distancia pesada al vecino más cercano

- Una variante del algoritmo de los $k$ –vecinos más cercanos es pesar la contribución de cada uno de los $k$ vecinos de acuerdo a sus distancias al punto consultado $x_q$, dándole mayor peso a los más cercanos.

- Por ejemplo, en el algoritmo anterior, que aproxima funciones objetivo con valores discretos, podríamos pesar la contribución de cada vecino de acuerdo al inverso del cuadrado de la distancia a $x_q$.

- Para esto reemplazamos 2b. por:

$$\hat{f}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^{k} w_i \, \delta\big(v, f(x_i)\big)$$

donde

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \quad (*)$$

- Para considerar el caso donde el punto $x_q$ coincide exactamente con una de las instancias de entrenamiento $x_i$ y el denominador $d(x_q, x_i)$ es por lo tanto nulo, asignamos a $\hat{f}(x_q)$ el valor de $f(x_i)$ en este caso.

- Si hay varios de estos ejemplos de entrenamiento, se asigna la clasificación mayoritaria entre ellos.

- Se pueden pesar las instancias basándose en la distancia para funciones objetivos con valores reales de un modo similar, reemplazando 2b por:

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

donde $w_i$ está definida como en la ecuación (*).

- El denominador en la ecuación anterior es una constante que normaliza la contribución de los pesos
  (asegura que si $f(x_i) = c$ para todos los ejemplos de entrenamiento, entonces $\hat{f}(x_q) \leftarrow c$ también).

- Todas las variantes del algoritmo de $k$ –vecinos más cercanos consideran sólo los $k$ vecinos más cercanos para clasificar el punto que se consulta.

- Pero una vez que se agrega un peso basado en distancias, no hay problemas en permitir que todos los ejemplos de entrenamiento tengan influencia en la clasificación de $x_q$ porque ejemplos muy distantes tendrán poco efecto en $\hat{f}(x_q)$.

- La única desventaja de considerar todos los ejemplos es que el clasificador será más lento.

- Si se consideran todos los ejemplos de entrenamiento al clasificar una nueva instancia consultada, diremos que el algoritmo es un método **global**.

- Si sólo se consideran los ejemplos de entrenamiento más cercanos, diremos que es un método **local.**

- El algoritmo de distancias pesadas para los $k$ –vecinos más cercanos es un método de inferencia muy efectivo en muchos problemas prácticos.

    - Es robusto frente a datos de entrenamiento con ruido y muy efectivo cuando se dispone de un gran conjunto de datos de entrenamiento.

- Al tomar la media pesada sobre los $k$ vecinos más cercanos al punto consultado, se suaviza el impacto de un solo punto aislado con ruido en los ejemplos de entrenamiento.

- En este algoritmo la distancia entre instancias se calcula basándose en *todos* los atributos de la instancia.

  - Esto difiere de los métodos como las reglas y los árboles de decisión que seleccionan sólo un subconjunto de los atributos de las instancias para formar la hipótesis.

- Para ver el efecto de esta política, apliquemos el algoritmo de los $k$ −vecinos más cercanos al problema en el cual cada instancia se describe con 20 atributos, pero sólo 2 de estos atributos son relevantes para determinar la clasificación de la función objetivo.

  - En este caso, las instancias que tienen valores idénticos para los 2 atributos relevantes pueden estar a mucha distancia una de otra en el espacio 20-dimensional.

  - Como resultado, la métrica de similitud usada por el algoritmo de $k$ −vecinos más cercanos podría ser engañosa.

  - La distancia entre vecinos estaría dominada por un número irrelevante de atributos.

- Esta dificultad, que aparece cuando hay muchos atributos presentes, se denomina la ***maldición de la multidimensionalidad.***

- Este método es especialmente sensible a este problema.

- Una aproximación interesante para superar este problema es pesar a cada atributo de modo distinto cuando se calculan las distancias entre dos instancias.

    - Esto corresponde a comprimir/estirar los ejes en el espacio euclídeo, acortando los ejes que corresponden a atributos menos relevantes, y alargando los que corresponden a los más relevantes.

- La cantidad en que cada eje debe ser dilatado o comprimido puede determinarse automáticamente usando convalidación cruzada.

- Otra alternativa más drástica es eliminar completamente los atributos menos relevantes del espacio de instancias.

- Debido a que este algoritmo demora todo el proceso hasta que se recibe una nueva consulta, se requiere de un cómputo significativo para procesar cada consulta.

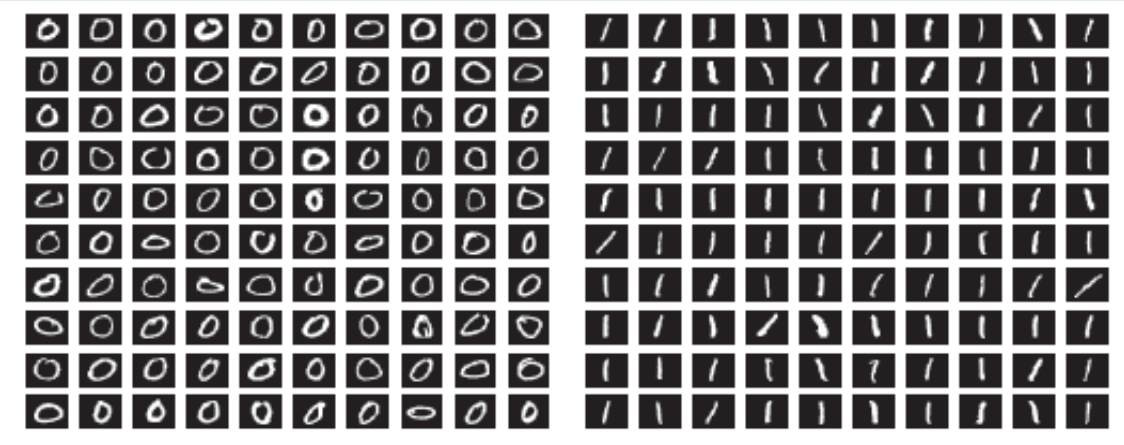
- Se han desarrollado varios métodos para indexar los ejemplos de entrenamiento de modo que los vecinos más cercanos puedan identificarse más eficientemente con algún costo adicional de memoria.

- Uno de tales métodos es el $kd$-árbol en el cual las instancias son almacenadas en las hojas del árbol, donde las instancias cercanas son almacenadas en el mismo nodo o en otro cercano.

  - Los nodos internos del árbol ordenan la nueva consulta $x_q$ en la hoja relevante testeando los atributos de $x_q$

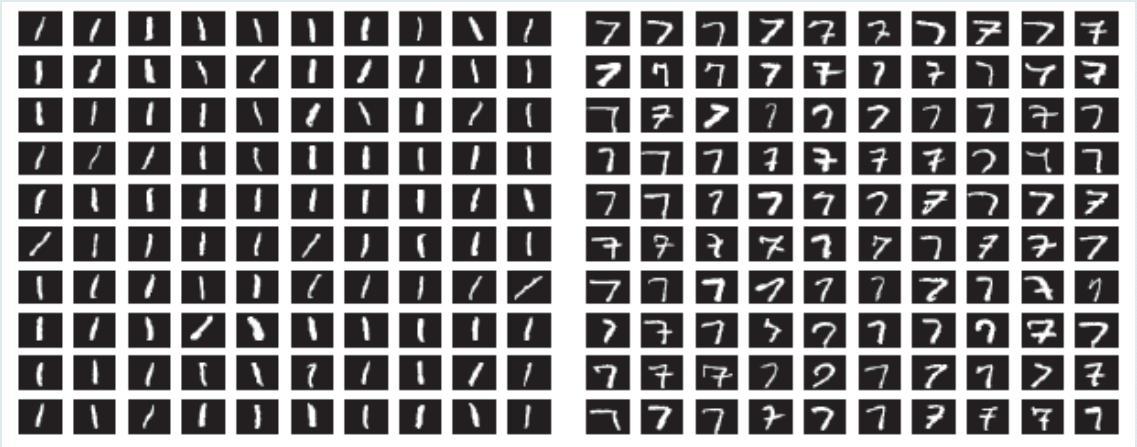
# Ejemplo

- Consideremos dos clases de dígitos manuscritos, ceros y unos.
- Cada dígito contiene 28x28=784 pixeles.
- Los datos de entrenamiento consisten en 300 ceros y 300 unos.



- Para testear la performance del algoritmo de 1–vecino más cercano, se utiliza un conjunto de testeo independiente consistente de otros 600 dígitos (300 ceros y 300 unos).
- El algoritmo aplicado a estos datos predice correctamente de qué dígito se trata en los 600 casos de testeo.

- La razón del éxito de la clasificación está en que los dígitos son lo suficientemente diferentes de modo que son fácilmente distinguibles.



- Una tarea más difícil es distinguir entre unos y sietes.

- Se repitió el experimento anterior ahora considerando un conjunto de 300 sietes y 300 unos.

- Nuevamente se volvió a tomar un conjunto de testeo de otros 600 dígitos para estudiar la performance.

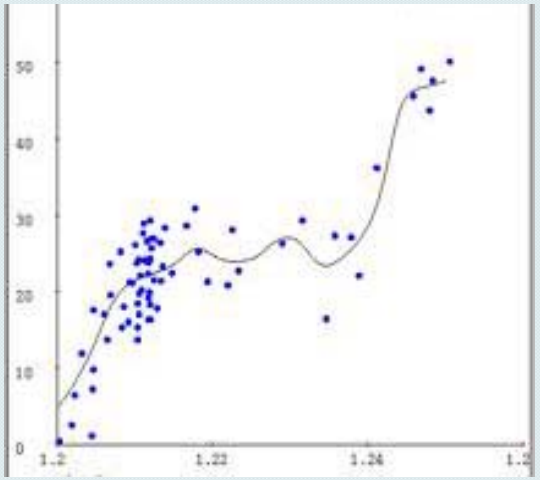
- Esta vez se encontraron 18 errores en la clasificación usando el algoritmo de 1−vecino más cercano (3% de error).

- Los 18 casos que fueron clasificados erróneamente se muestran en la figura.

- Si se utiliza el algoritmo de los 3−vecinos más cercanos, el error de clasificación se reduce a 14 dígitos mal clasificados (2.3% de error)

- Para comparar la magnitud del error, el mejor método de clasificación automático clasifica dígitos manuscritos con un error menor del 1%, un porcentaje mejor que la performance promedio de los humanos.

# Regresión local pesada




- El método del vecino más cercano se puede pensar como una aproximación de la función objetivo $f(x)$ en un único punto de consulta $x = x_q$.

- La regresión local pesada es una generalización de esta aproximación.

- Construye una aproximación explícita de $f$ sobre una región local que rodea a $x_q$.

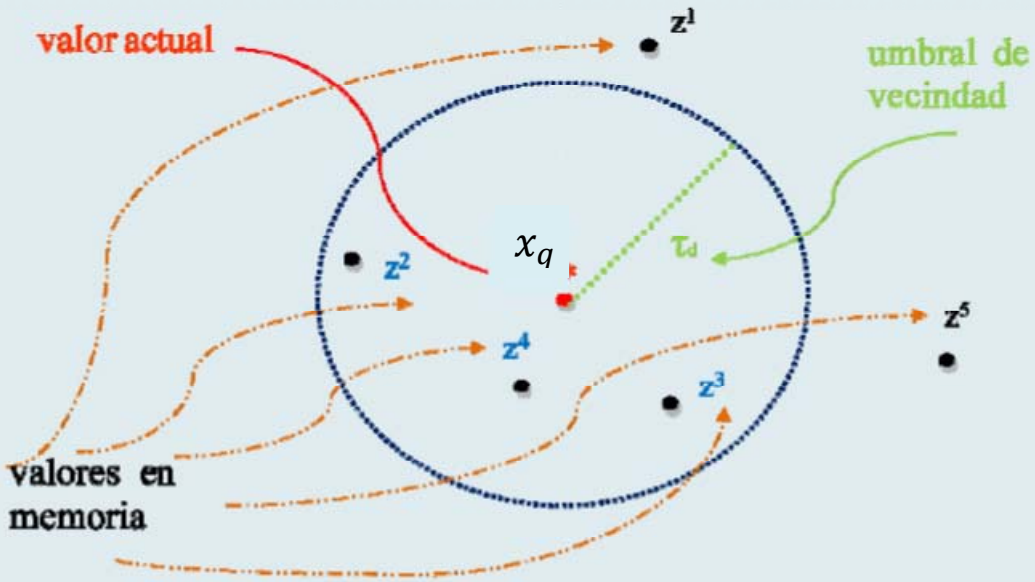
- Utiliza la cercanía o la distancia pesada a los ejemplos de entrenamiento para formar esta aproximación local de $f$.

- El nombre de este método se debe a que:

  - *local*: porque la función se aproxima basándose sólo en puntos cercanos al de consulta,

  - *pesada*: porque la contribución de cada ejemplo de entrenamiento se pesa por la distancia al punto de consulta

- *regresión:* porque este término se usa en el contexto estadístico para los problemas de aproximar funciones con valores reales.




- Dada una nueva instancia de consulta $x_q$, se construye una aproximación $\hat{f}$ que ajuste a los ejemplos de entrenamiento en un entorno que rodee a $x_q$.

- Luego la aproximación se usa para calcular el valor $\hat{f}(x_q)$, que es dado como output del valor estimado de la función en la nueva instancia.

- La descripción de $\hat{f}$ puede luego eliminarse, porque se calculará una aproximación local distinta para cada instancia distinta consultada.

# Regresión lineal local pesada

- Consideremos el caso de regresión local pesada en el cual la función objetivo $f$ se aproxima cerca de $x_q$ usando una función lineal de la forma

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

  donde $a_i(x)$ denota el valor del i-ésimo atributo de la instancia $x$.

- Para encontrar la aproximación vamos a elegir los pesos de modo que minimicen el error cuadrático medio sobre el conjunto $D$ de ejemplos de entrenamiento.

- Hay varias formas de hacerlo. Vamos a ver 3 posibilidades:

1. Minimizar el error cuadrático medio sólo sobre los $k$ vecinos más cercanos

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ vecinos más cercanos de } x_q} \left[f(x) - \hat{f}(x)\right]^2$$

2. Minimizar el error cuadrático medio sobre todo el conjunto $D$ de ejemplos de entrenamiento, pesando el error de cada ejemplo con alguna función decreciente $K$ de su distancia a $x_q$

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} \left[f(x) - \hat{f}(x)\right]^2 K\left(d(x_q, x)\right)$$

3. Combinar 1 y 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ vecinos más cercanos de } x_q} \left[f(x) - \hat{f}(x)\right]^2 K\left(d(x_q, x)\right)$$

- El criterio 2 permite que todos los ejemplos de entrenamiento tengan impacto en la clasificación de $x_q$.
  - Sin embargo esta aproximación requiere una capacidad de cálculo que crece linealmente con el número de ejemplos de entrenamiento.

- El criterio 3 es una buena aproximación al criterio 2 y tiene la ventaja de que el costo computacional es independiente del número total de ejemplos de entrenamiento; sólo depende del número $k$ de vecinos considerados.

- Si se considera el criterio 3, derivando se obtiene que

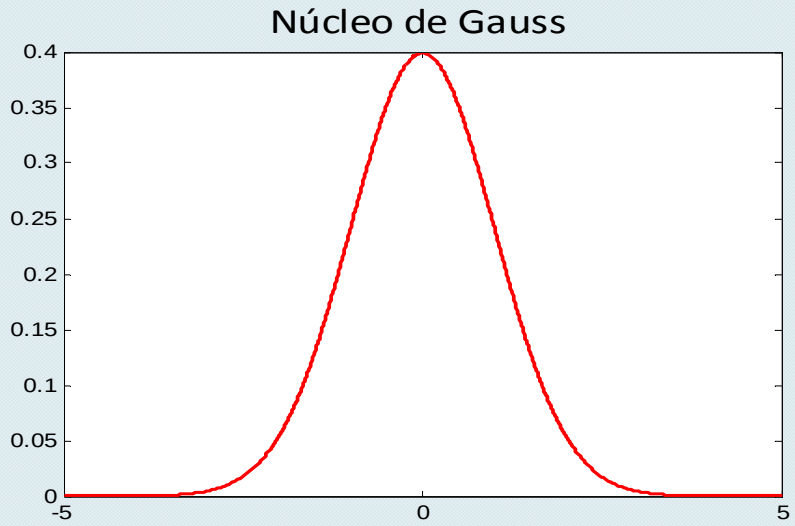
$$\Delta w_j = \eta \sum_{x \in k \text{ vecinos más cercanos de } x_q} K\left(d\left(x_q, x\right)\right)\left[f(x) - \hat{f}(x)\right] a_j(x)$$

donde $\eta$ es la tasa de aprendizaje, y la regla de entrenamiento está dada por:

$$w_j \leftarrow w_j + \Delta w_j$$

- La función de núcleo $K$ (kernel) es una función de la distancia que se usa para determinar el peso de cada ejemplo de entrenamiento.

- La función *K* se elige de modo que sea:
  - no negativa
  - simétrica
  - concentre la mayor parte de masa cerca del *0*
  - decrezca rápidamente o se anule fuera del intervalo [-1,1]



- Algunas funciones posibles son:

  - Gaussiana:

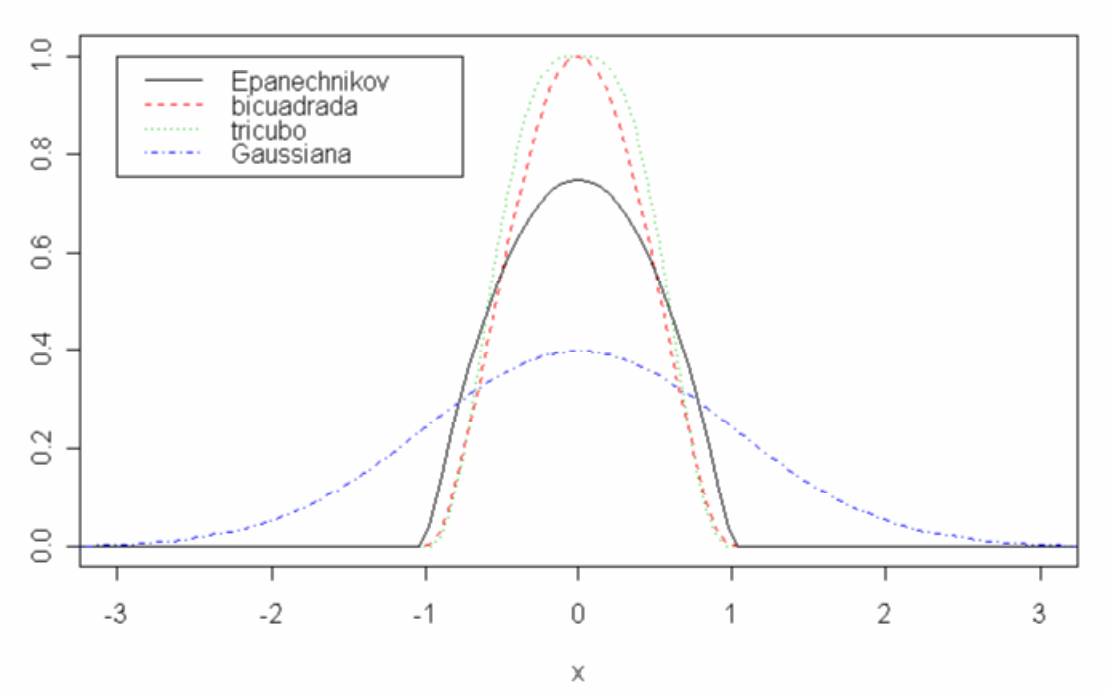
$$K(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}$$

  - Bicuadrada:

$$K(x) = (1 - x^2)^2 \, I^{(x)}_{[-1,1]}$$

  - Epanechnikov:

$$K(x) = \frac{3(1 - x^2)}{4} \, I^{(x)}_{[-1,1]}$$

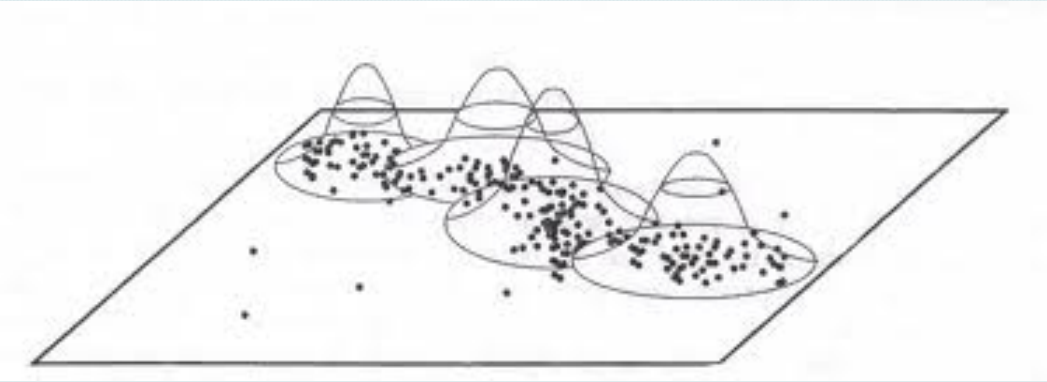



  - Tricubo:

$$K(x) = (1 - |x|^3)^3 \, I^{(x)}_{[-1,1]}$$

donde $I^{(x)}_{[-1,1]} = \begin{cases} 1 & \text{si } x \in [-1,1] \\ 0 & \text{si } x \notin [-1,1] \end{cases}$

# Funciones de base radial



- En este método la función aprendida es de la forma:

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

donde cada $x_u$ es una instancia de $X$ y donde el núcleo $K_u\big(d(x_u, x)\big)$ se define de modo que sea decreciente si la distancia $d(x_u, x)$ crece.

- Aquí $k$ es una constante provista por el usuario que especifica el número de núcleos que se incluirán.

- A pesar de que $\hat{f}(x)$ es una aproximación global a $f(x)$, la contribución de cada término $K_u\big(d(x_u, x)\big)$ está localizada en una región cercana al punto $x_u$.

- Es común elegir a cada función $K_u\big(d(x_u, x)\big)$ como una función gaussiana centrada en el punto $x_u$ con varianza $\sigma_u^2$

$$K_u\big(d(x_u, x)\big) = \exp\left[-\frac{d^2(x_u, x)}{2\sigma_u^2}\right]$$

- Una posible aproximación es elegir el número de núcleos menor que el número de ejemplos de entrenamiento.

- El conjunto de núcleos podría distribuirse con centros espaciados uniformemente a través del espacio de instancias $X$.

- Alternativamente, podríamos desear distribuir los centros no uniformemente, especialmente si las instancias no se distribuyen uniformemente sobre $X$.

    - En este último caso, elegimos los centros de los núcleos eligiendo aleatoriamente un conjunto de instancias

- Otra posibilidad es identificar clusters prototipos de instancias y agregar un núcleo centrado en cada cluster.
  - La ubicación de los núcleos puede lograrse usando algoritmos de clustering que ajusten las instancias de entrenamiento (pero no sus valores objetivo) como una mezcla de normales.
  - El algoritmo EM da un método para elegir las medias de una mezcla de *k* normales que mejor ajusten a las instancias observadas.

- Notemos que el valor de la función objetivo $f(x_i)$ de la instancia no interviene en el cálculo del centro de los núcleos.

- El único rol del valor de la función objetivo $f(x_i)$ es determinar el peso $w_u$ del output.

# Razonamiento basado en casos

- Los métodos basados en instancias como el algoritmo de los $k$ –vecinos más cercanos y la regresión local pesada comparten tres propiedades:

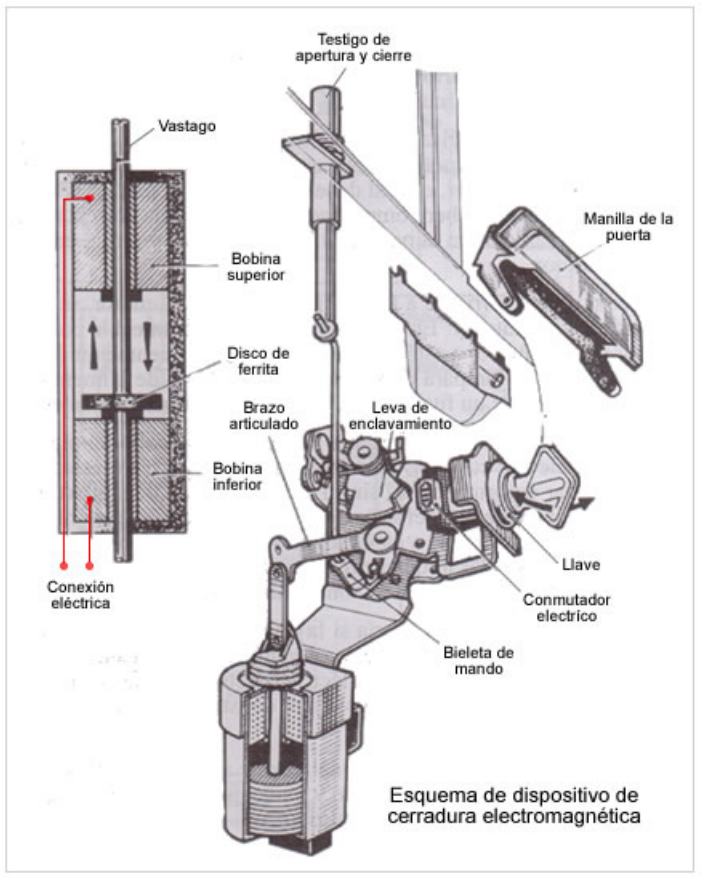
1. Son métodos ***perezosos*** en los que se difiere la decisión de cómo generalizar más allá de los datos de entrenamiento hasta que se consulta un nuevo dato

2. Clasifican una nueva instancia analizando instancias similares e ignorando instancias que son muy distintas de la consultada

3. Representan a las instancias como puntos con valores reales en el espacio euclídeo *n*– dimensional

- Los razonamientos basados en casos (CBR) es un paradigma de aprendizaje que se basa en los dos primeros principios anteriores, pero no en el tercero.

- Este método se puede aplicar a problemas reales como:




- el diseño conceptual de dispositivos mecánicos basados en una librería almacenada de diseños previos

- razonamiento acerca de casos legales basados en fallos previos


o resolver problemas de planificación de horarios reusando y combinando porciones de soluciones previas a problemas similares

- Consideramos un ejemplo típico: el sistema CADET emplea el razonamiento basado en casos para asistir al diseño conceptual de mecanismos mecánicos simples como canillas de agua.

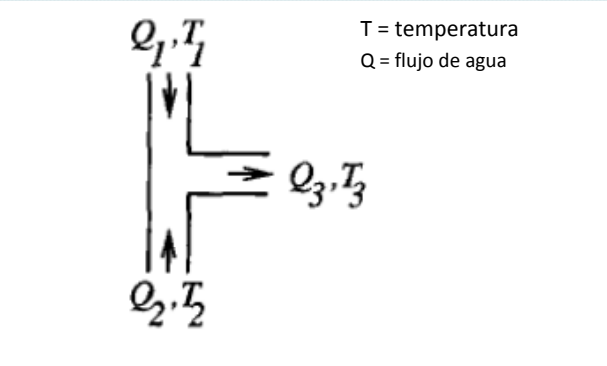
- Utiliza una librería que contiene aproximadamente 75 diseños previos y diseña fragmentos para sugerir diseños conceptuales que satisfagan las especificaciones de nuevos problemas de diseño.

- Cada instancia almacenada en memoria (por ej. una cañería) se representa describiendo tanto su estructura como su función cualitativa.

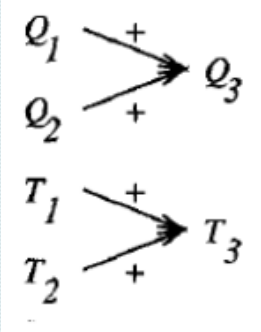
- Luego se presentan nuevos problemas de diseño especificando la función deseada y requiriendo la correspondiente estructura.

# Caso almacenado: empalme T

Estructura




Función




- Se muestra la descripción de un caso típico almacenado llamado **empalme T**.

- Su función se representa en términos de la relación cualitativa entre los niveles de flujo de agua y la temperatura como inputs y outputs.

o En la descripción funcional, una flecha con un signo "+" indica que la variable en la punta de la flecha se incrementa con la variable en la cola.

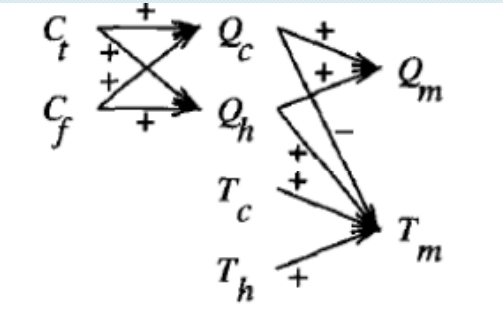
o Análogamente, un "–" indica que la variable en la punta decrece con la variable en la cola.

## Especificaciones del problema: canilla

Estructura

?

Función




o Se presenta un nuevo problema de diseño descripto por su función deseada.

- Esta función en particular describe el comportamiento requerido por un tipo de canilla.

- Aquí $Q_c$ se refiere al flujo de agua fría en la canilla, $Q_h$ al flujo entrante de agua caliente y $Q_m$ a un único flujo mixto que sale de la canilla.

- Análogamente, $T_c$, $T_h$ y $T_m$ se refieren a la temperatura del agua fría, de la caliente y de la mezcla respectivamente.

- La variable $C_t$ denota la señal de control para la temperatura que entra en la canilla, y $C_f$ denota la señal de control para el flujo de agua.

- Notemos que la descripción de la función deseada especifica que estos controles $C_t$ y $C_f$ están influenciados por los flujos de agua $Q_c$ y $Q_h$ respectivamente, mientras que influyen indirectamente la salida del flujo de la canilla $Q_m$ y la temperatura $T_m$.

- Dada una especificación funcional para el nuevo problema de diseño, CADET busca en su librería los casos almacenados cuyas descripciones funcionales coincidan con el diseño del problema.

  - Si se encuentra una coincidencia exacta, que indica que algún caso almacenado implementa exactamente la función deseada, entonces este caso se devuelve como una solución sugerida al problema de diseño.

- Si no se encuentra ninguna coincidencia, CADET puede encontrar casos que coincidan en varios subgrafos de las especificaciones deseadas.

- En la figura anterior, por ejemplo, el empalme T coincide en un subgrafo del grafo de la función de la canilla.

- Además, el sistema puede elaborar el grafo de la función de especificaciones original para crear grafos equivalentes que coincidan en muchos más casos.

- Utiliza conocimientos generales acerca de ciertas influencias físicas para cerrar estos grafos elaborados.

- Por ejemplo, reescribe la regla que permite reescribir la influencia

$$A \rightarrow^{+} B$$

como

$$A \rightarrow^{+} x \rightarrow^{+} B$$

- Esta regla reescrita puede interpretarse como si *B* debe incrementarse con *A*, entonces es suficiente encontrar alguna otra cantidad $x$ tal que *B* se incrementa con $x$ y $x$ se incremente con A.

- Aquí $x$ es una variable cuantificada cuyos valores están acotados cuando coincide la función grafo con la del caso de la librería.

- De hecho la función grafo para la cañería de la figura es una elaboración de las especificaciones funcionales originales producida al aplicar estas reglas reescritas.

- Al recuperar múltiples casos que coinciden con distintos subgrafos, el diseño entero puede algunas veces unirse.

- En general el proceso de producir una solución final de muchos casos recuperados puede ser muy complejo.

- CADET tiene capacitadas muy limitadas para combinar y adaptar casos recuperados para formar el diseño final y depende fuertemente del usuario para este proceso de adaptación

- En CADET cada ejemplo de entrenamiento almacenado describe un grafo de la función junto con la estructura que lo implementa.

- Las nuevas consultas corresponden a nuevos grafos de función.

- Luego se puede mapear el problema CADET a nuestra notación estándar definiendo el espacio de instancias $X$ como el espacio de todos los grafos de función.

- La función objetivo $f$ mapea grafos de función a estructuras que los implementan.

- Cada ejemplo de entrenamiento almacenado $\langle x, f(x) \rangle$ es un par que describe algún grafo de función $x$ y la estructura $f(x)$ que implementa $x$.

- El sistema debe aprender de los casos de ejemplos de entrenamiento cómo dar como output una estructura $f(x)$ que implemente exitosamente el input del grafo de función de consulta $x_q$.

- Algunas característica del CADET son:

  - las instancias o casos pueden representarse con descripciones simbólicas, tales como los grafos de función usados en CADET.

- Esto requiere una métrica de similitud distinta de la distancia euclídea, tal como el tamaño del subgrafo más grande compartido entre dos grafos de función.

- se pueden combinar múltiples casos recuperados para formar la solución para un nuevo problema.
  - Esto es similar a la aproximación de los *k* vecinos más cercanos, en los que múltiples casos similares se usan para construir una respuesta para una nueva consulta.
  - Sin embargo el proceso de combinar estos múltiples casos recuperados pude ser muy diferente, dependiendo del razonamiento en el conocimiento en vez de métodos estadísticos.

# Métodos perezosos y codiciosos de aprendizaje

- Hemos visto métodos perezosos de aprendizaje:
  - algoritmo de los *k*-vecinos más cercanos
  - regresión local pesada
  - razonamiento basado en casos

- Llamamos a estos métodos perezosos porque aplazan la decisión de cómo generalizar más allá de los datos de entrenamiento hasta que se encuentre una nueva instancia de consulta.

- También vimos un método codicioso de aprendizaje:

  - base de funciones radiales.

- Llamamos a este método codicioso porque generaliza más allá de los datos de entrenamiento antes de una nueva instancia de consulta.

- Hay dos clases de diferencias entre ambos métodos:

  - en el tiempo de cómputo

    - Los métodos perezosos generalmente requieren de menos tiempo de cómputo durante el entrenamiento, pero de más cálculo cuando deben predecir el valor de la función objetivo para una nueva consulta.

o en la clasificación producida por nuevas consultas

- Los métodos perezosos pueden considerar una nueva instancia $x_q$ cuando deciden cómo generalizar más allá de los datos de entrenamiento $D$

- Los métodos codiciosos, **no** pueden.
  En el momento de observar una nueva instancia $x_q$ ellos ya han elegido su aproximación (global) a la función objetivo.

- ¿Afecta esta distinción la precisión?
  Lo hace si se requiere que el aprendiz perezoso y el codicioso utilicen el mismo espacio de hipótesis $H$.

- Para ilustrar este hecho, consideramos el espacio de hipótesis que consiste en las funciones lineales.

- El algoritmo de regresión lineal local pesada es un método perezoso basado en este espacio de hipótesis.

  - Para cada nueva consulta $x_q$ generaliza a partir de los datos de entrenamiento eligiendo una nueva hipótesis basada en los ejemplos de entrenamiento cercanos a $x_q$.

- En contraste, un método codicioso que usa el mismo espacio de hipótesis de las funciones lineales, debe elegir su aproximación **antes** de que se observe la consulta.

- El aprendiz codicioso debe asignar una única función lineal que cubra el espacio completo de instancias y todas las futuras consultas.

- Los métodos perezosos efectivamente usan un espacio de hipótesis más rico debido a que usan muchas funciones lineales distintas locales para formar su aproximación global implícita a la función objetivo

- De este modo las diferencias entre aprendizaje perezoso y codicioso está relacionada con las distinción entre aproximaciones locales y globales a la función objetivo.

# FIN

# Aprendizaje automático