INSTANCE BASED LEARNING

[Read Ch. 8]

- k-Nearest Neighbor
- Locally weighted regression
- Radial basis functions
- Case-based reasoning
- Lazy and eager learning

k-Nearest Neighbor Classifier

- Instances are assumed to be n-dimensional feature vectors $x_p = (x_{p,1}, \ldots, x_{p,n})$

Learning Phase

- Store all training examples $\langle x_i, f(x_i) \rangle$ in memory

Classification/Approximation Phase

- Nearest Neighbor:

  Given query instance x_q, first locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

- k-Nearest Neighbor:

1. Given x_q, take vote among its k nearest nbrs (if discrete-valued target function — classification)

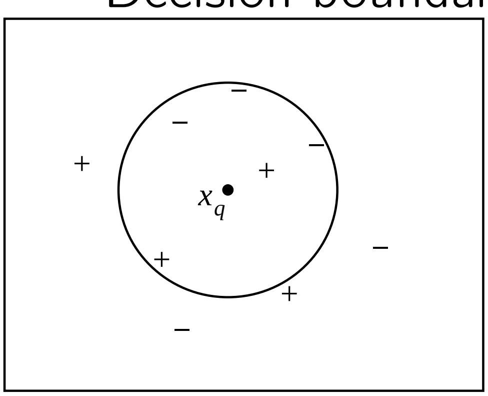
2. Take the mean of f values of k nearest nbrs (if real-valued — approximation)

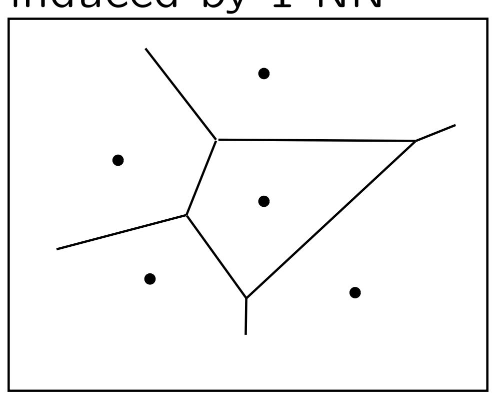
$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Nearest Neighbor Methods

- Conceptually simple
- Asymptotically have error rates that are no worse than twice that of the optimum Bayes classifier
- Learn by simply memorizing training examples
- Construct a different approximation on the fly for each input instance (query instance) unlike the other learning algorithms we have considered so far which construct a single approximation to the target function during the learning phase and use it thereafter for generating the output for each query instance
- The computational effort of learning is low
- The storage requirements of learning is high: need to memorize the examples in the training set
- Cost of classifying new instances is high
- A distance measure needs to be defined over input space: e.g. Euclidean distance, Hamming distance, etc as appropriate
- Performance degrades when there are many irrelevant attributes

Decision boundary induced by 1-NN





Voronoi Diagram

When To Consider Nearest Neighbor

- Instances map to points in $\Re^n$
- Less than 20 attributes per instance
- Lots of training data
- Advantages:
  1. Training is very fast
  2. Learn complex target functions
  3. Don't lose information
- Disadvantages:
  1. Slow at query time
  2. Easily fooled by irrelevant attributes

Behavior in the Limit

- Consider $p(x)$ defines probability that instance x will be labeled 1 (positive) versus 0 (negative).

- Nearest neighbor:

  As number of training examples $\rightarrow \infty$, approaches Gibbs Algorithm

  Gibbs: with probability $p(x)$ predict 1, else 0

- k-Nearest neighbor:

  As number of training examples $\rightarrow \infty$ and k gets large, approaches Bayes optimal

  Bayes optimal: if $p(x) > .5$ then predict 1, else 0

- Note: Gibbs has at most twice the expected error of Bayes optimal

Distance-Weighted k-Nearest Neighbor Classifier

- Might want to weight nearer neighbors more heavily ...

Learning Phase

- Store each training example $\langle x_i, f(x_i)\rangle$ in memory

Classification/Approximation Phase

- For discrete-valued target functions $f : \Re^n \rightarrow V$

  Given query instance x_q to be classified

  1. Let $x_1, \ldots, x_k$ denote the k nearest neighbors of x_q
  2. Return

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

  where $\delta(a,b) = 1$ if $a = b$ otherwise 0, and $w_i = \frac{1}{d(x_q, x_i)^2}$

- For real-valued target functions $f : \Re^n \rightarrow \Re$

  Given query instance x_q to be approximated

  2. Replace above by

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

  and $d(x_q, x_i)$ is distance between x_q and x_i

Note: now it makes sense to use *all* training examples instead of just k

$\rightarrow$ Shepard's method

Curse of Dimensionality

- Imagine instances described by 20 attributes, but only 2 are relevant to target function

- *Curse of dimensionality*: nearest nbr is easily mislead when high-dimensional X

- One approach:

  1. Stretch jth axis by weight z_j, where $z_1, \ldots, z_n$ chosen to minimize prediction error

  2. Use cross-validation to automatically choose weights $z_1, \ldots, z_n$

  3. Note: setting z_j to zero eliminates this dimension altogether

  See [Moore and Lee, 1994]

Locally Weighted Regression

- Note: k-NN forms local approximation to f for each query point x_q

- Why not form an explicit approximation $\hat{f}(x)$ for region surrounding x_q

  1. Fit linear function to k nearest neighbors
  2. Fit quadratic, ...
  3. Produces "piecewise approximation" to f

- Locally weighted regression involves calculating an approximation of the function value for a given input based on its nearest neighbors when needed during the approximation phase as opposed to during the learning phase.

- Example of linear approximation

  $$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x) = w_0 + \sum_{i=1}^{n} w_i a_i(x)$$

  where $a_i(x) = i$-th attribute of x.

Locally Weighted Regression Error Functions

1. Squared error over k-NN of x_q

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in\ k\text{NN}(x_q)} (f(x) - \hat{f}(x))^2$$

$$w_i \leftarrow w_i - \eta \frac{\partial E_1(x_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{x \in\ k\text{NN}(x_q)} (f(x) - \hat{f}(x)) a_i(x)$$

2. Distance-weighted squared error over all samples

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

$$w_i \leftarrow w_i - \eta \frac{\partial E_2(x_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{x \in D} K(d(x_q, x))(f(x) - \hat{f}(x)) a_i(x)$$

3. Distance-weighted squared error over k-NN of x_q

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in\ k\text{NN}(x_q)} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

$$w_i \leftarrow w_i - \eta \frac{\partial E_3(x_q)}{\partial w_i}$$

$$w_i \leftarrow w_i + \eta \sum_{x \in\ k\text{NN}(x_q)} K(d(x_q, x))(f(x) - \hat{f}(x)) a_i(x)$$

Radial Basis Function Networks

- Global approximation to target function, in terms of linear combination of local approximations

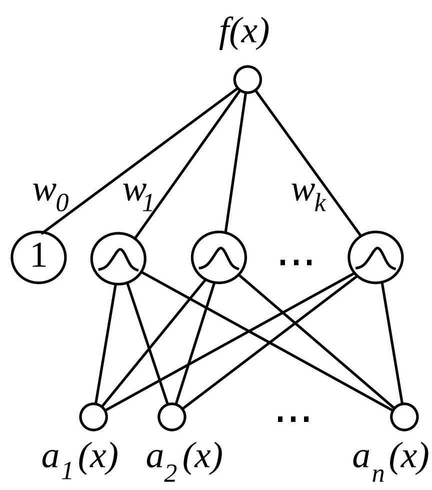
  A function is approximated as a linear combination of radial basis functions (RBF). RBFs capture local behaviors of functions

- A different kind of neural network where $a_i(x)$ are the attributes describing instance x, and

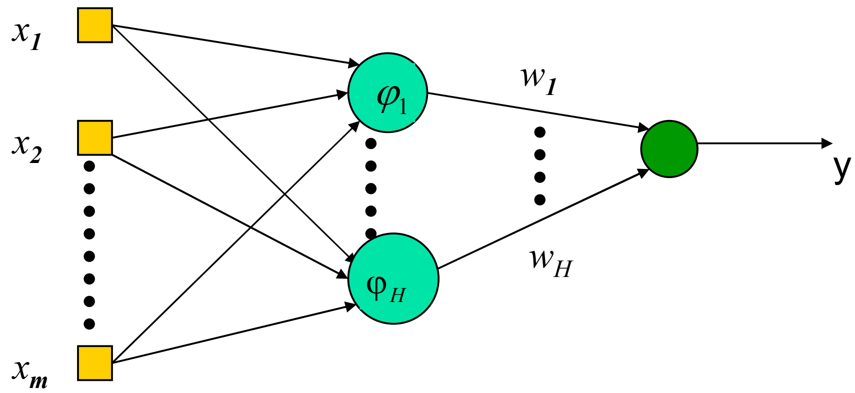
$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

  One common choice for $K_u(d(x_u, x))$ is

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



Radial Basis Function Networks



- Hidden layer representation

  1. Hidden layer applies a non-linear transformation from the input space to the hidden space

  2. Output layer applies a linear transformation from the hidden space to the output space

$$\varphi(\mathbf{x}) = \langle \varphi_1(\mathbf{x}), \ldots, \varphi_H(\mathbf{x}) \rangle = \{\varphi_i(\mathbf{x})\}_{i=1}^{H}$$

- φ-**Separability of patterns**: A (binary) partition, also called dichotomy, (C_1, C_2) of the training set C is φ-*separable* if there is a vector $\mathbf{w} = (w_1, \ldots, w_H)$ such that

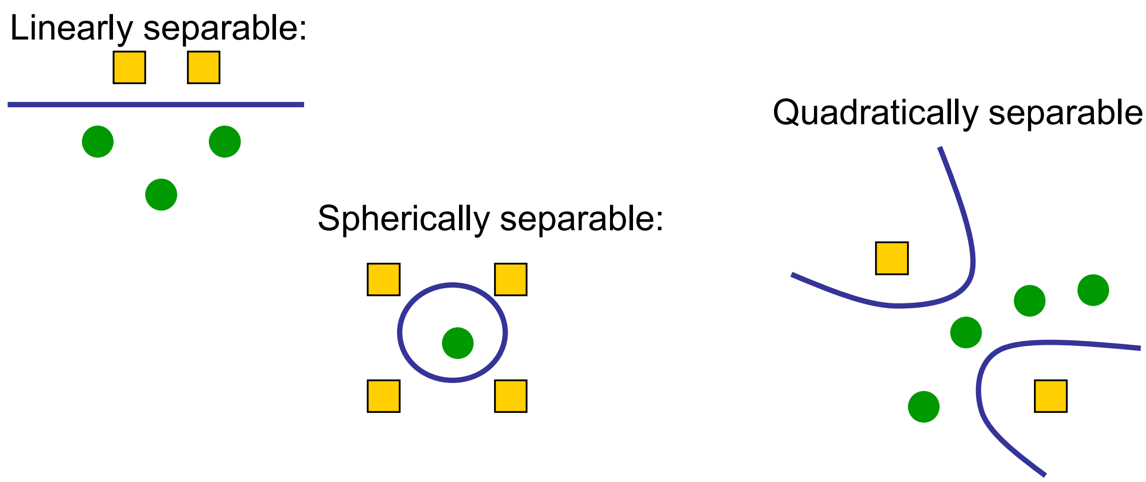
$$\mathbf{w} \cdot \varphi(\mathbf{x}) > 0 \text{ iff } \mathbf{x} \in C_1$$

$$\mathbf{w} \cdot \varphi(\mathbf{x}) < 0 \text{ iff } \mathbf{x} \in C_2$$

For all $\mathbf{x} = (x_1, \ldots, x_m)$

Examples of φ-Separability and RBF Functions

- Separating surface: $\mathbf{w} \cdot \varphi(\mathbf{x}) = 0$

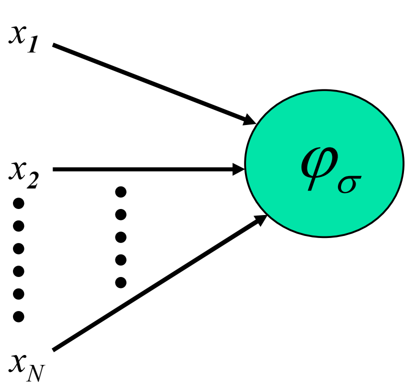


- Radial Basis Functions

  Hidden units use a radial basis function

$$\varphi_\sigma(||\mathbf{x} - \mathbf{w}||) = e^{-\frac{||\mathbf{x}-\mathbf{w}||^2}{2\sigma^2}}$$

The output depends on the distance of the input $\mathbf{x}$ from the center $t = \mathbf{w}$



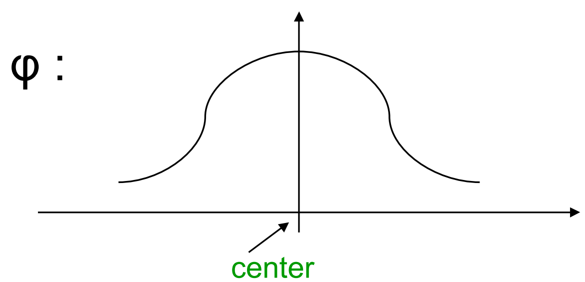
$\varphi_\sigma(|| \mathbf{X}\text{-} \mathbf{W}||^2)$

W is called center
σ is called spread
center and spread are parameters

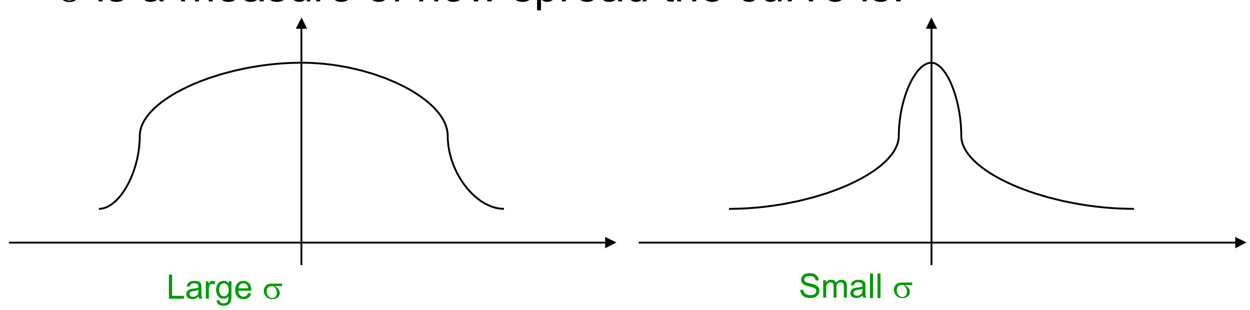
Radial Basis Functions

- A hidden neuron is more sensitive to data points near its center. This sensitivity may be tuned by adjusting the spread. Larger spread $\rightarrow$ less sensitivity

- Gaussian radial basis function: $\varphi_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}}$, $\sigma > 0$



σ is a measure of how spread the curve is:



- Other radial basis functions

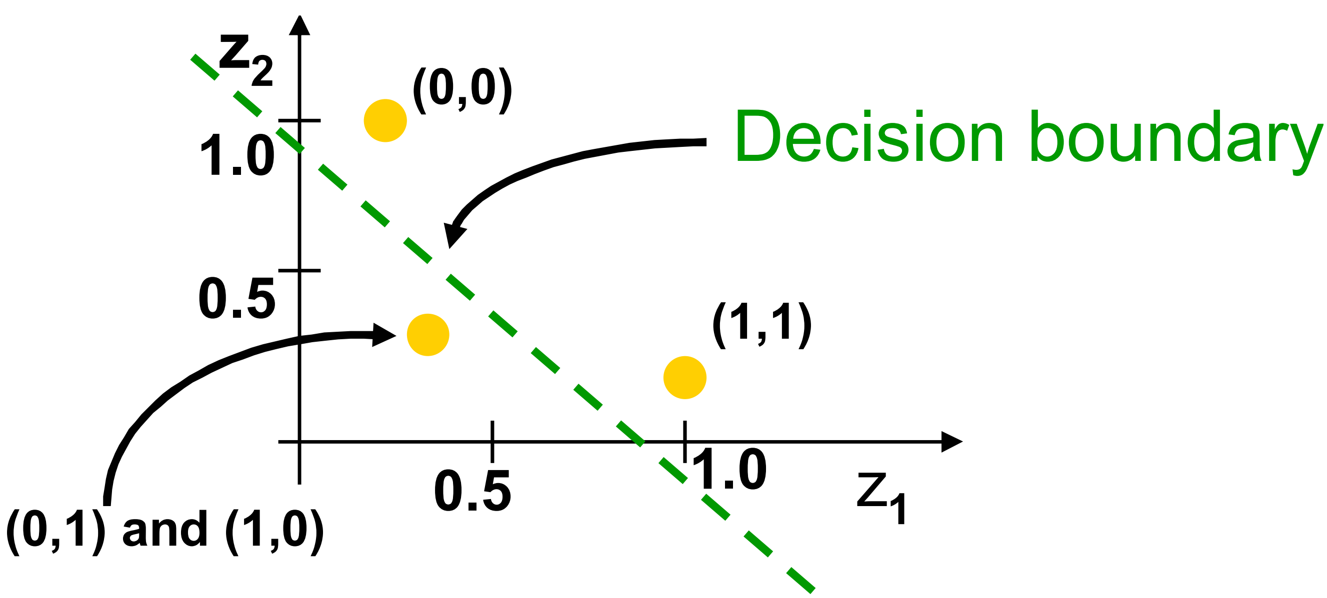
  Multiquadrics: $\varphi = (r^2 + c^2)^{\frac{1}{2}}$, $c > 0$, $r = ||\mathbf{x} - \mathbf{w}||$

  Inverse multiquadrics: $\varphi = \frac{1}{(r^2+c^2)^{\frac{1}{2}}}$

Implementing Exclusive-OR by RBF Network

- Construct an RBF pattern classifier such that
  1. $(0, 0)$ and $(1, 1)$ are mapped to 0, class C_1
  2. $(0, 1)$ and $(1, 0)$ are mapped to 1, class C_2

- In the feature (hidden) space:
  1. $\varphi_1(x_1, x_2) = e^{-||\mathbf{x}-\mathbf{w}_1||^2} = z_1$ where $\mathbf{w}_2 = [1, 1]^t$
  2. $\varphi_2(x_1, x_2) = e^{-||\mathbf{x}-\mathbf{w}_2||^2} = z_2$ where $\mathbf{w}_2 = [0, 0]^t$



- When mapped into the feature space $\langle z_1, z_2 \rangle$, C_1 and C_2 become *linearly separable*. So a linear classifier with $\varphi_1(\mathbf{x})$ and $\varphi_2(\mathbf{x})$ as inputs can be used to solve the XOR problem

Training Radial Basis Function Networks

- Q1: What x_u to use for each kernel function $K_u(d(x_u, x))$?
  1. Scatter uniformly throughout instance space
  2. Or use training instances (reflects instance distribution)
- Q2: How to train weights (assume here Gaussian K_u)?
  1. First choose variance (and perhaps mean) for each K_u

     e.g., use EM
  2. Then hold K_u fixed, and train linear output layer

     Efficient method to fit linear function
- Closely related to distance-weighted regression, but "eager" instead of "lazy"

RBF Learning Algorithms

$$\Delta\sigma_j = -\eta_{\sigma_j} \frac{\partial E_S}{\partial \sigma_j}$$

$$\Delta\alpha_j = -\eta_j \frac{\partial E_S}{\partial \alpha_j}$$

$$\Delta w_{ji} = -\eta_{ji} \frac{\partial E_S}{\partial w_{ji}}$$

Depending on the specific function can be computed using the chain rule of calculus

$$z_{jp} = e^{-\frac{-\|\mathbf{x}_p - \mathbf{w}_j\|^2}{2\sigma_j^2}}$$

$$y_p = \sum_{j=0}^{L} \alpha_j z_{jp}$$

$$E_p = \frac{1}{2}\left(t_p - y_p\right)^2$$

$$\mathbf{X}_p = \left[x_{1p} \ldots\ldots\ldots x_{Np}\right]^T$$

$$\mathbf{W}_j = \left[w_{j1} \ldots\ldots w_{jN}\right]^T$$

RBF Learning Algorithms

$$\Delta\alpha_j = -\eta_j \frac{\partial E_p}{\partial \alpha_j} = \eta_j (t_p - y_p) z_{jp}$$

$$\alpha_j \leftarrow \alpha_j + \eta_j (t_p - y_p) z_{jp}$$

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial w_{ji}}$$

$$= -(t_p - y_p)\alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

$$w_{ji} = w_{ji} + \eta_{ji} (t_p - y_p)\alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

$$\frac{\partial E_p}{\partial \sigma_j} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial \sigma_j}$$

$$= -(t_p - y_p)\alpha_j (-z_{jp}) \left(\left(\frac{2}{\sigma_j} \right) (\ln z_{jp}) \right)$$

$$\sigma_j \leftarrow \sigma_j - \eta_j (t_p - y_p)\alpha_j (z_{jp}) \left(\left(\frac{2}{\sigma_j} \right) (\ln z_{jp}) \right)$$

RBF Learning Algorithms

Initialize the parameters -- centers of the hidden neurons are typically initialized to coincide with a subset of the training set

Use gradient descent to adjust the parameters using the training data until the desired performance criterion is satisfied

Some Useful Facts

$$\|V\|^2 = V^T V \text{ (norm)}$$

$$\|V\|_C^2 = (CV)^T (CV) = V^T C^T C V \text{ (weighted norm)}$$

$$\|V\|_C^2 = \|V\|^2 \text{ if } C^T C = \text{identity matrix}$$

$$\frac{d}{d\mathbf{X}}(A\mathbf{X}) = A$$

$$\frac{d}{d\mathbf{X}}(\mathbf{X}^T A\mathbf{X}) = 2A\mathbf{X} \text{ (when A is a symmetric matrix)}$$

$$\frac{d}{dA}(\mathbf{X}^T A\mathbf{X}) = \mathbf{X}^T \mathbf{X}$$

Case-Based Reasoning

- Can apply instance-based learning even when $X \neq \Re^n$

  $\rightarrow$ Need different "distance" metric

- Case-Based Reasoning is instance-based learning applied to instances with symbolic logic descriptions

```
((user-complaint error53-on-shutdown)
 (cpu-model PowerPC)
 (operating-system Windows)
 (network-connection PCIA)
 (memory 48meg)
 (installed-applications Excel Netscape VirusScan)
 (disk 1gig)
 (likely-cause ???))
```

- Three properties of CBR

  1. *Lazy* learning methods
  2. Classification of intances by similarity
  3. Symbolic representation of instance

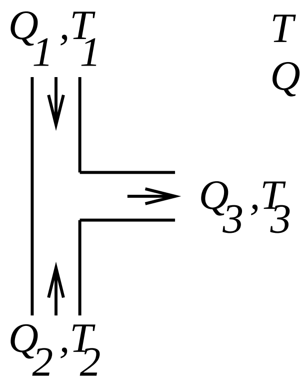
Case-Based Reasoning in CADET

- CADET: 75 stored examples of mechanical devices
  1. each training example: $\langle$qualitative function, mechanical structure$\rangle$
  2. new query: desired function,
  3. target value: mechanical structure for this function
- Distance metric: match qualitative function descriptions
- Instances represented by rich structural descriptions
- Multiple cases retrieved (and combined) to form solution to new problem
- Tight coupling between case retrieval and problem solving
- Bottom line:
  1. Simple matching of cases useful for tasks such as answering help-desk queries
  2. Area of ongoing research

Case-Based Reasoning in CADET

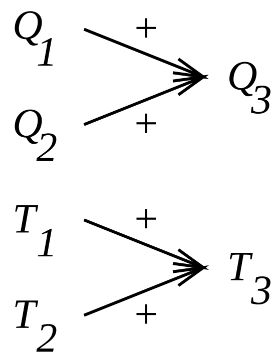
A stored case: T−junction pipe

Structure:



T = temperature
Q = waterflow

Function:

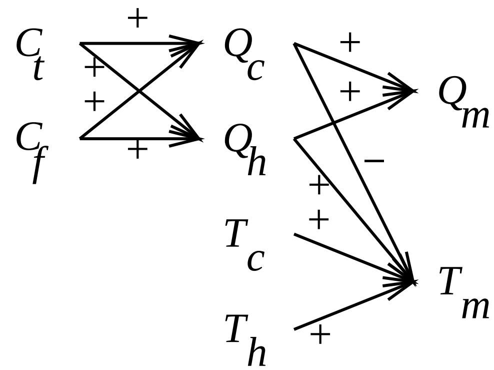


A problem specification: Water faucet

Structure:

?

Function:



Lazy versus Eager Learning

1. Lazy learner

   - Can produce many good local approximations based on *the training data and the query input*

   - Waits for query before generalizing

   - Requires a predefined distance measure over the input space, low computation effort but large memory for storing examples

   - Has to work hard during classification or approximation

   e.g. k-NN, LWR, CBR

2. Eager learner

   - Construct a global approximation based only on the training data *without regard to the query input*

   - Generalizes before seeing query

   e.g. CL, DT, ANN, BNN, RBF, . . .

- If they use the same H, lazy can represent more complex functions