INSTANCE-BASE LEARNING

- Instance-based learning methods simply store the training examples instead of learning explicit description of the target function.
  - Generalizing the examples is postponed until a new instance must be classified.
  - When a new instance is encountered, its relationship to the stored examples is examined in order to assign a target function value for the new instance.
- Instance-based learning includes ***nearest neighbor***, ***locally weighted regression*** and ***case-based reasoning*** methods.
- Instance-based methods are sometimes referred to as **lazy** learning methods because they delay processing until a new instance must be classified.
- A key advantage of lazy learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

k-Nearest Neighbor Learning

- k-Nearest Neighbor Learning algorithm assumes all instances correspond to points in the n-dimensional space $\Re^n$
- The nearest neighbors of an instance are defined in terms of Euclidean distance.
- Euclidean distance between the instances $x_i = <x_{i1},\ldots,x_{in}>$ and $x_j = <x_{j1},\ldots,x_{jn}>$ are:

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

- For a given query instance x_q, $f(x_q)$ is calculated the function values of k-nearest neighbor of x_q

k-Nearest Neighbor Learning

- Store all training examples $<x_i, f(x_i)>$
- Calculate $f(x_q)$ for a given query instance x_q using k-nearest neighbor
- **Nearest neighbor: (k=1)**
  - Locate the nearest traing example x_n, and estimate $f(x_q)$ as
  - $f(x_q) \leftarrow f(x_n)$
- **k-Nearest neighbor:**
  - Locate k nearest traing examples, and estimate $f(x_q)$ as
  - If the target function is real-valued, take mean of f-values of k nearest neighbors.

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

  - If the target function is discrete-valued, take a vote among f-values of k nearest neighbors.

When To Consider Nearest Neighbor

- Instances map to points in R^n
- Less than 20 attributes per instance
- Lots of training data
- **Advantages**
  - Training is very fast
  - Learn complex target functions
  - Can handle noisy data
  - Does not loose any information
- **Disadvantages**
  - Slow at query time
  - Easily fooled by irrelevant attributes

Distance-Weighted kNN

Might want weight nearer neighbors more heavily...

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

where

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

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

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:

- Stretch jth axis by weight z_j, where $z_1, \ldots, z_n$ chosen to minimize prediction error
- Use cross-validation to automatically choose weights $z_1, \ldots, z_n$
- Note setting z_j to zero eliminates this dimension altogether

Locally Weighted Regression

- KNN forms local approximation to f for each query point xq
- Why not form an explicit approximation f(x) for region surrounding xq

  ➔ Locally Weighted Regression
- **Locally weighted regression** uses nearby or distance-weighted training examples to form this local approximation to f.
- We might approximate the target function in the neighborhood surrounding x, using a linear function, a quadratic function, a multilayer neural network.
- The phrase "locally weighted regression" is called
  - ***local*** because the function is approximated based only on data near the query point,
  - ***weighted*** because the contribution of each training example is weighted by its distance from the query point, and
  - ***regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation f that fits the training examples in the neighborhood surrounding xq.
- This approximation is then used to calculate the value f(xq), which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

f is approximated near x_q using a linear function of the form

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

$a_i(x)$ denotes the value of the ith attribute of the instance x.

Minimize the squared error

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

Kernel function K is the function of distance that is used to determine the weight of each training example.

$$\Delta w_j = \eta \sum_{x \in \text{ k nearest nbrs of } x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x)$$

Radial Basis Functions

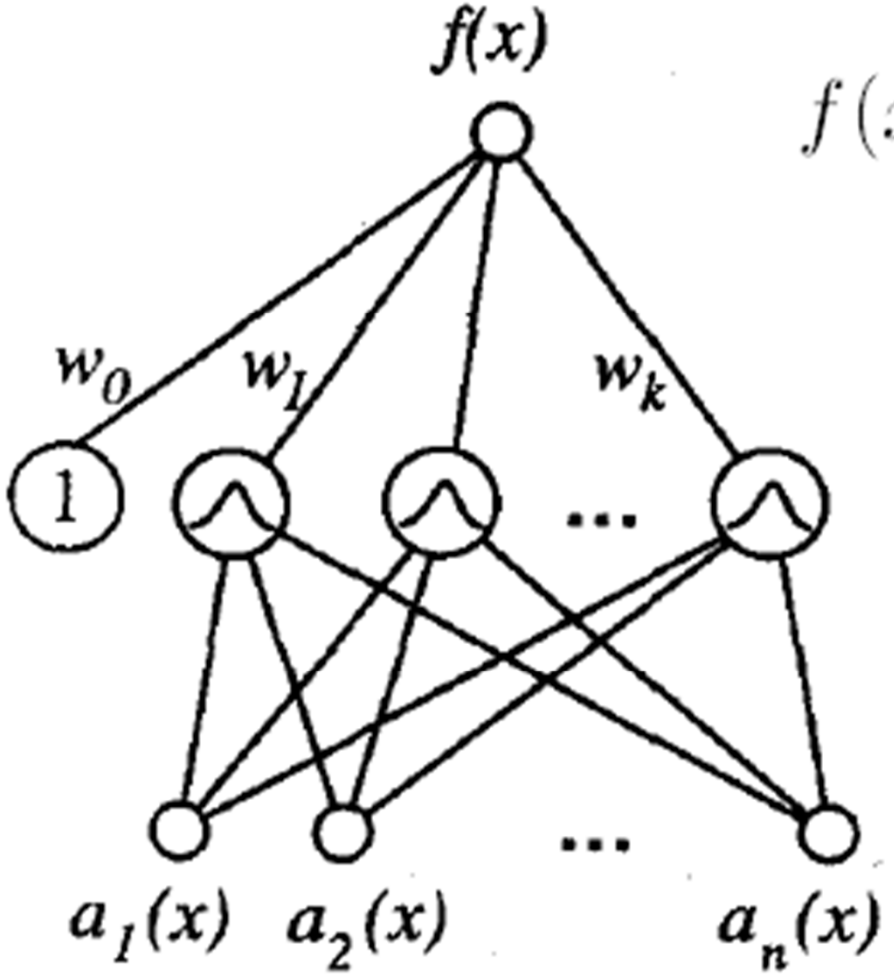
- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions.
- The learned hypothesis is a function of the form

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

where each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases. Here k is a user-provided constant that specifies the number of kernel functions to be included. Even though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u. It is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at the point x_u with some variance σ_u^2.

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

Radial Basis Function Networks



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

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

Each hidden unit produces an activation determined by a Gaussian function centered at some instance *xu.*

Therefore, its activation will be close to zero unless the input x is near *xu.*

The output unit produces a linear combination of the hidden unit activations.

Case-based reasoning

- Instance-based methods
  - lazy
  - classification based on classifications of near (similar) instances
  - data: points in n-dim. space
- Case-based reasoning
  - as above, but data represented in symbolic form
- New distance metrics required

Lazy & eager learning

- Lazy: generalize at query time
  - kNN, CBR
- Eager: generalize before seeing query
  - Radial basis, ID3, …
- Difference
  - eager *must* create global approximation
  - lazy *can* create many local approximation
  - lazy can represent more complex functions using same H (H = linear functions)