

# INSTANCE BASED LEARNING

Danica Kragic

Danica Kragic, 2004

## Instance-Based Learning

- no explicit description of the target function
- generalization postponed until new instance has to be classified
- “lazy learning”
- local approximation of the target function
- High cost: training vs. classification!!!
- Case-base reasoning: more complex, symbolic representations for instances

Danica Kragic, 2004

## Instance Based Learning

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

Danica Kragic, 2004

## Instance-Based Learning

Key idea: just store all training examples  $\langle x_i, f(x_i) \rangle$

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:

- Given  $x_q$ , take vote among its  $k$  nearest nbrs (if discrete-valued target function)
- take mean of  $f$  values of  $k$  nearest nbrs (if real-valued)

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

Danica Kragic, 2004

## When To Consider Nearest Neighbor

### Nearest neighbor

- Instances map to points in  $n$ -dimensional space  $\mathfrak{R}^n$
- Nearest neighbors defined by standard Euclidean distance given their feature vectors

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

- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Training is very fast
- Learn complex target functions
- Don't lose information

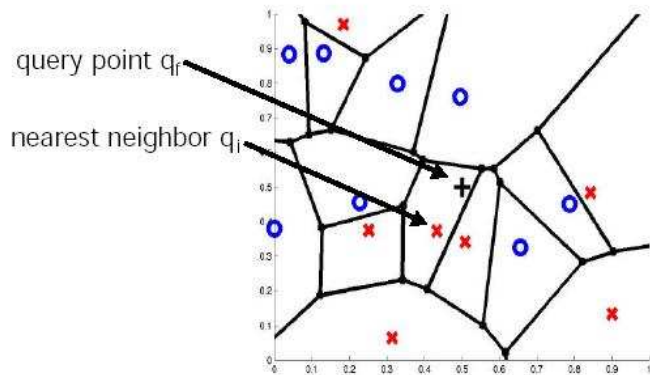
Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

Danica Kragic, 2004

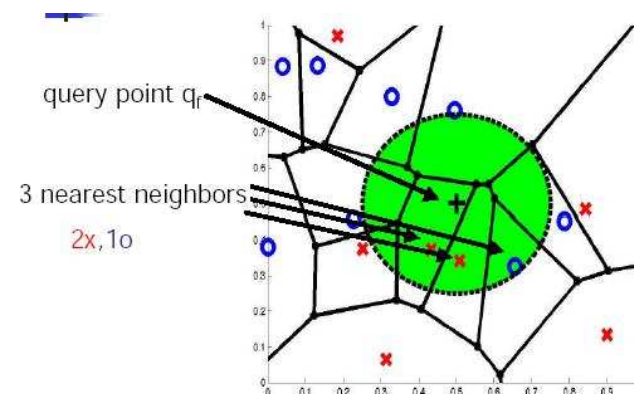
Danica Kragic, 2004

### Voronoi Diagram



Danica Kragic, 2004

### 3-Nearest Neighbors



Danica Kragic, 2004

### Behavior in the Limit

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

Nearest neighbor:

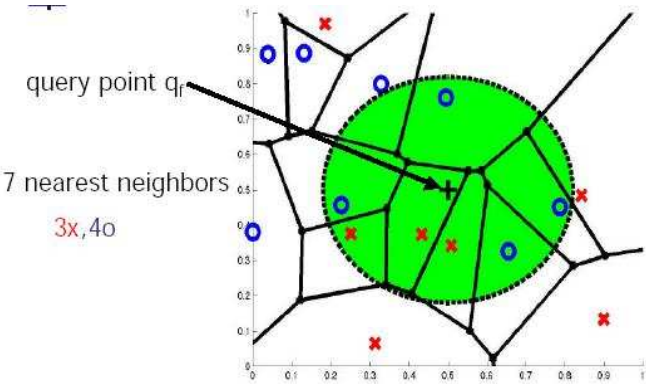
- As number of training examples  $\rightarrow \infty$ , approaches Gibbs Algorithm
- Gibbs: with probability  $p(x)$  predict 1, else 0

$k$ -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

### 7-Nearest neighbors

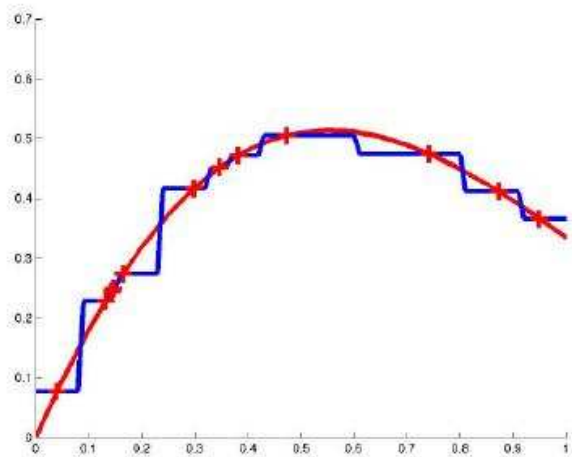


Danica Kragic, 2004

Danica Kragic, 2004

### Nearest Neighbors (continuous)

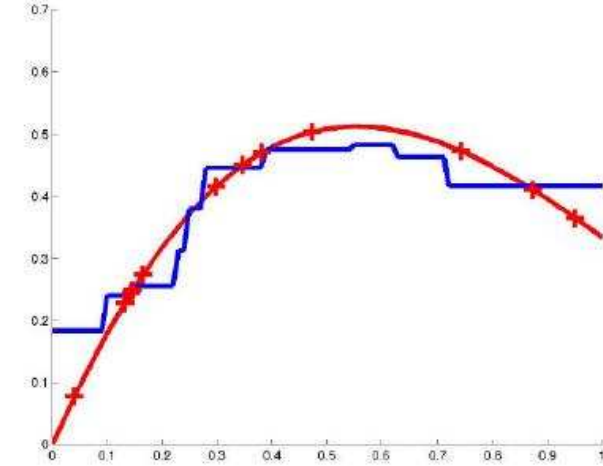
#### 1-nearest neighbor



Danica Kragic, 2004

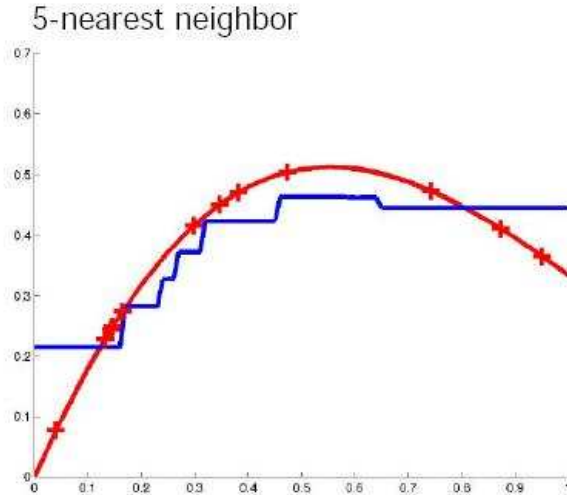
### Nearest Neighbors (continuous)

#### 3-nearest neighbor



Danica Kragic, 2004

## Nearest Neighbors (continuous)



Danica Kragic, 2004

## Locally Weighted Regression (terminology)

- Regression means approximating a real-valued target function
- Residual is the error  $\hat{f}(x) - f(x)$  in approximating the target function
- Kernel function is the function of distance that is used to determine the weight of each training example
- In other words: the kernel function is the function  $K$  such that  $w_i = K(d(x_i, x_q))$

Danica Kragic, 2004

## 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  $x_q$

- Fit linear function to  $k$  nearest neighbors
- Fit quadratic, ...
- Produces “piecewise approximation” to  $f$

Several choices of error to minimize:

- Squared error over  $k$  nearest neighbors

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

- Distance-weighted squared error over all nbrs

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

Danica Kragic, 2004

## Distance-Weighted $k$ NN

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

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

(or  $w_i = K(d(x_i, x_q))$ ) in general) 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  $k$

→ Shepard’s method

Danica Kragic, 2004

## Distance Weighted Average

- Weighting the data:

$$f^*(x_q) = \frac{\sum_i f(x_i)K(d(x_i, x_q))}{\sum_i K(d(x_i, x_q))}$$

- Relevance of a data point  $(x_i, f(x_i))$  is measured by calculating the distance  $d(x_i, x_q)$  between the query  $x_q$  and the input vector  $x_i$
- Weighting the error criterion:

$$E(x_q) = \sum_i (f^*(x_q) - f(x_i))^2 K(d(x_i, x_q))$$

the best estimate  $f^*(x_q)$  will minimize the cost  $E(x_q)$ , therefore

$$\frac{\partial E(x_q)}{\partial f^*(x_q)} = 0$$

Danica Kragic, 2004

Danica Kragic, 2004

## Distance Weighted k-NN

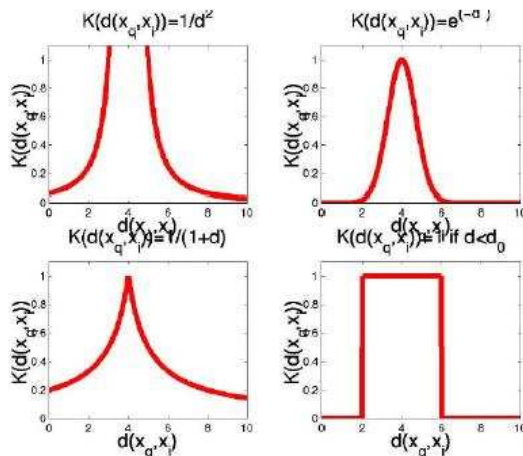
Give more weight to neighbors closer to the query point

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

where  $w_i = K(d(x_q, x_i))$  and  $d(x_q, x_i)$  is the distance between  $x_q$  and  $x_i$

Instead of only k-nearest neighbors, use all training examples (Shepard's method)

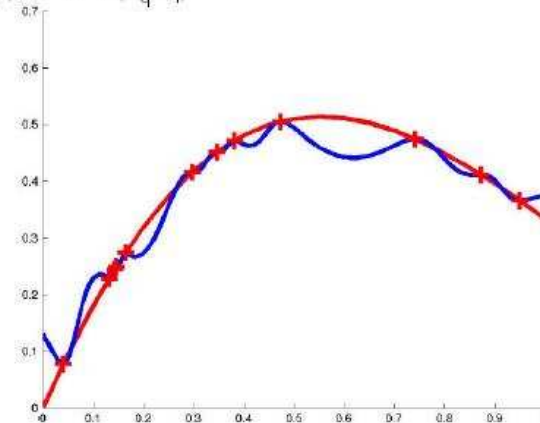
## Kernel Functions



Danica Kragic, 2004

## Distance Weighted NN

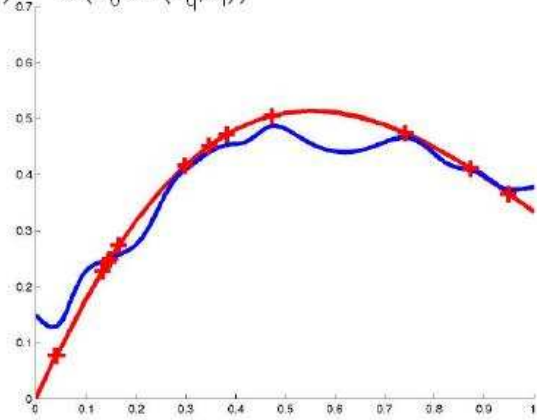
$$K(d(x_q, x_i)) = 1/d(x_q, x_i)^2$$



Danica Kragic, 2004

## Distance Weighted NN

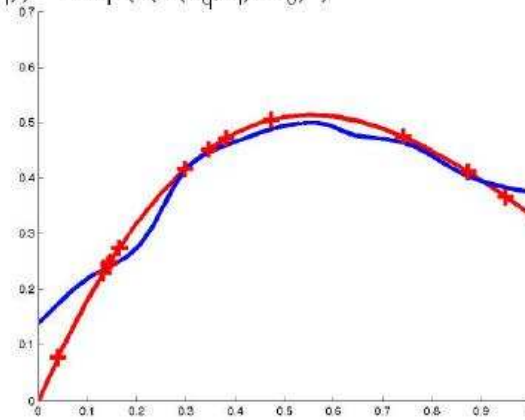
$$K(d(x_q, x_i)) = 1/(d_0 + d(x_q, x_i))^2$$



Danica Kragic, 2004

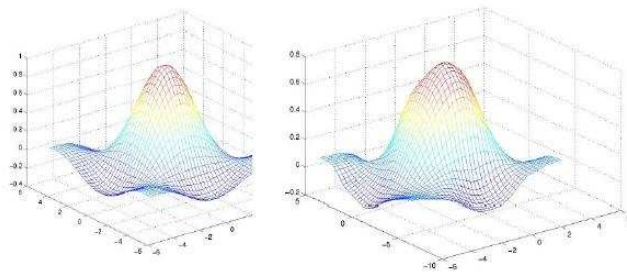
## Distance Weighted NN

$$K(d(x_q, x_i)) = \exp(-(d(x_q, x_i)/\sigma_0)^2)$$



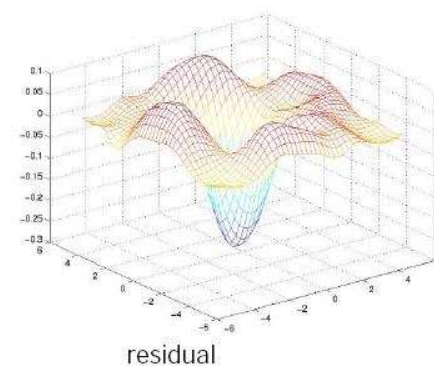
Danica Kragic, 2004

## Example: Mexican Hat



$$f(x_1, x_2) = \sin(x_1)\sin(x_2)/x_1x_2$$

approximation



residual

Danica Kragic, 2004

Danica Kragic, 2004

## Locally Weighted Linear Regression

- Local linear function

$$f^*(x) = w_0 + \sum_n w_n x_n$$

- Error criterion

$$E = \sum_i (w_0 + \sum_n w_n x_{qn} - f(x_i))^2 K(d(x_i, x_q))$$

- Gradient descent

$$\Delta w_n = \sum_i (f^*(x_q) - f(x_i)) x_n K(d(x_i, x_q))$$

- Least square solution

$$w = ((KX)^T KX)^{-1} (KX)^T f(X)$$

with  $KX$   $N \times M$  matrix of row vectors  $K(d(x_i, x_q))x_i$  and  $f(X)$  is a vector whose  $i$ -th element is  $f(x_i)$

Danica Kragic, 2004

## Radial Basis Function Networks

- Global approximation to target function, in terms of linear combination of local approximations
- Used, e.g., for image classification
- A different kind of neural network - similar to back-propagation neural network but activation function is Gaussian rather than sigmoid
- Closely related to distance-weighted regression, but “eager” instead of “lazy”

Danica Kragic, 2004

## Curse of Dimensionality

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

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

One approach:

- Stretch  $j$ th axis by weight  $z_j$ , where  $z_1, \dots, z_n$  chosen to minimize prediction error
- Use cross-validation to automatically choose weights  $z_1, \dots, z_n$
- Note setting  $z_j$  to zero eliminates this dimension altogether (feature subset selection)

Danica Kragic, 2004

## Radial Basis Function Networks

where  $a_i(x)$  are the attributes describing instance  $x$ , and

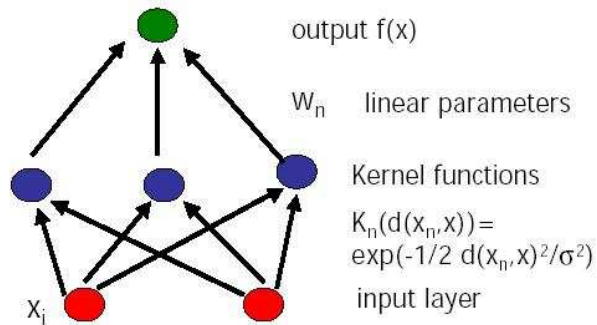
$$f(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

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

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

Danica Kragic, 2004

## Radial Basis Function Networks



$$f(x) = w_0 + \sum_{n=1}^k w_n K_n(d(x_n, x))$$

Danica Kragic, 2004

## Training Radial Basis Function Networks

Q1: What center  $x_u$  to use for each kernel function  $K_u(d(x_u, x))$

- Scatter uniformly throughout instance space
- use distribution of training instances (clustering)

Q2: How to train weights (assume here Gaussian  $K_u$ )

- First choose mean and variance for each  $K_u$ 
  - e.g., use EM
- Then hold  $K_u$  fixed, and train linear output layer
  - efficient methods to fit linear function

Danica Kragic, 2004

## Lazy and Eager Learning

Lazy: wait for query before generalizing

- $k$ -NEAREST NEIGHBOR, Case based reasoning

Eager: generalize before seeing query

- Radial basis function networks, ID3, Backpropagation, Naive-Bayes, ...

Does it matter?

- Eager learner must create global approximation
- Lazy learner can create many local approximations
- if they use same  $H$ , lazy can represent more complex fns (e.g., consider  $H =$  linear functions)

Danica Kragic, 2004

## Contents

Danica Kragic, 2004