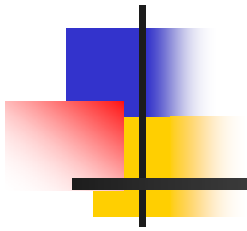
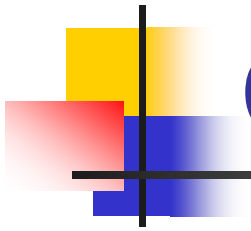


Machine Learning

Lecture 12

Instance Based Learning.

Radial basis functions



Outline

- K-Nearest Neighbor
- Locally weighted regression
- Radial basis functions

When to Consider Nearest Neighbors

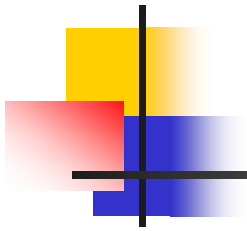
- Instances map to points in R^N
- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Training is very fast
- Learn complex target functions
- Do not loose information

Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes



Instance Based Learning

Key idea: just store all training examples $<x_i, f(x_i)>$

Nearest neighbor:

- Given query instance x_q, first locate nearest training example x_n, then estimate $f(x_q)=f(x_n)$

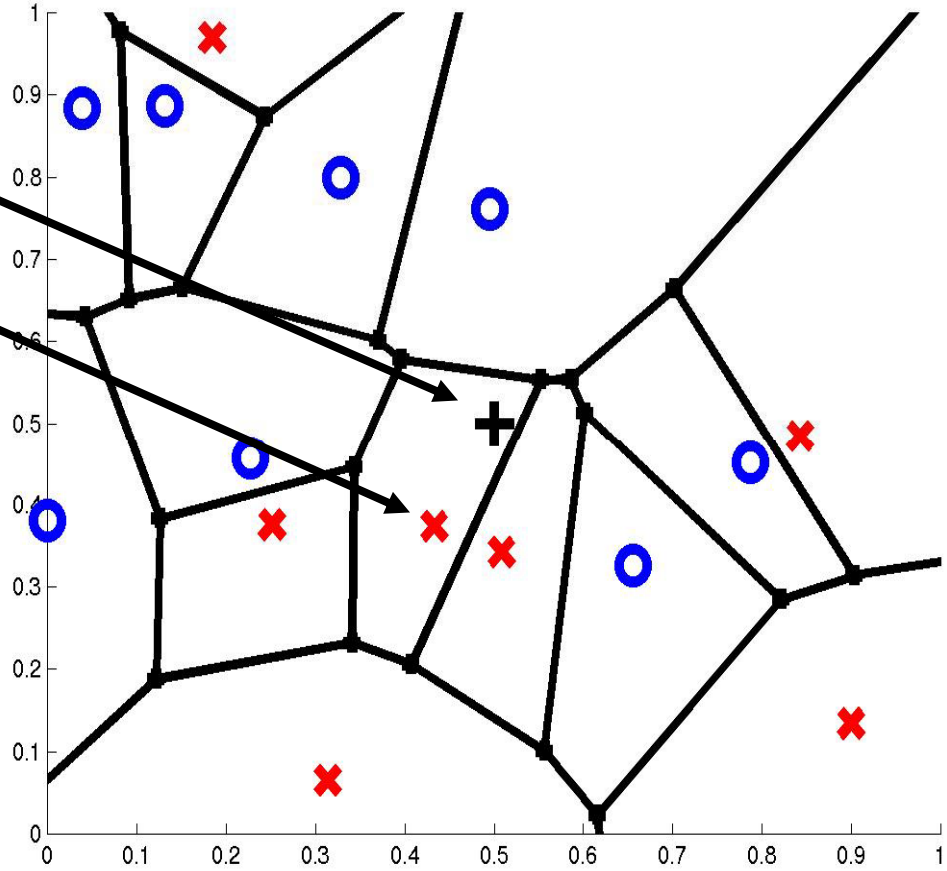
K-nearest neighbor:

- Given x_q, take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued) $f(x_q)=\sum_{i=1}^{k} f(x_i)/k$

Voronoi Diagram

query point q_f

nearest neighbor q_i

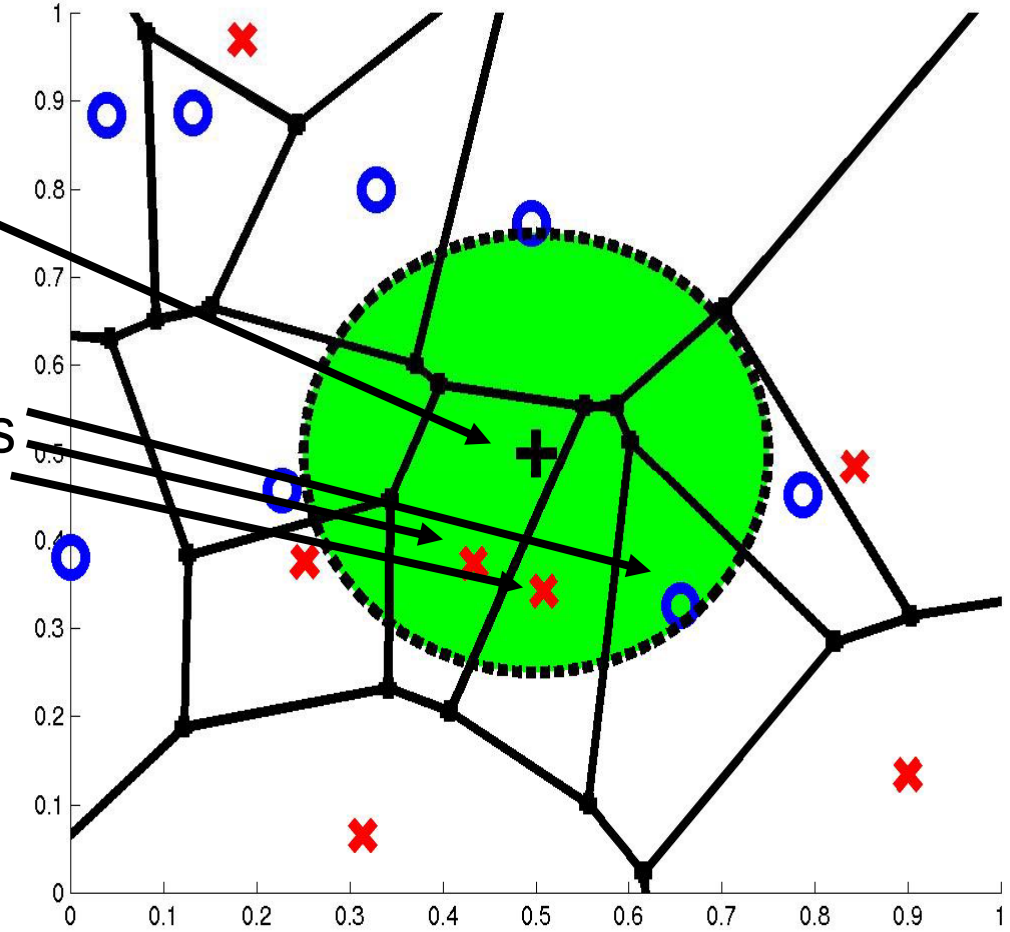


3-Nearest Neighbors

query point q_f

3 nearest neighbors

2x,1o

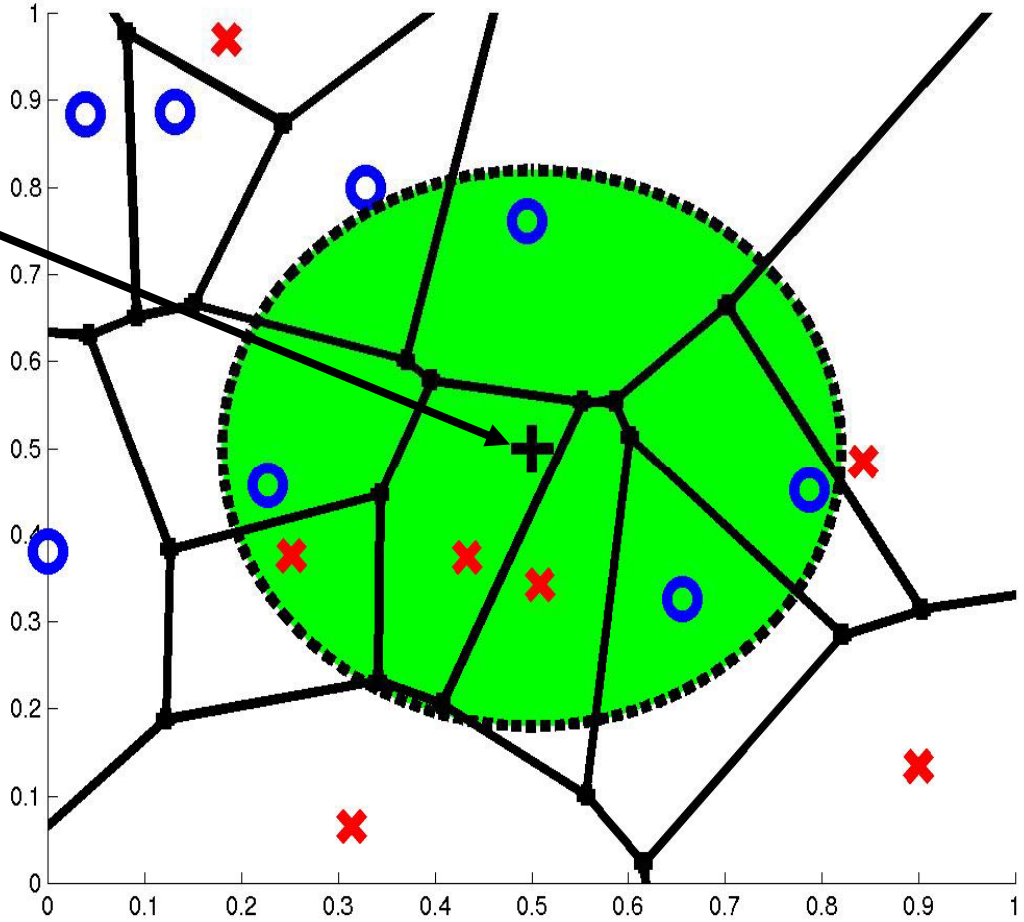


7-Nearest Neighbors

query point q_f

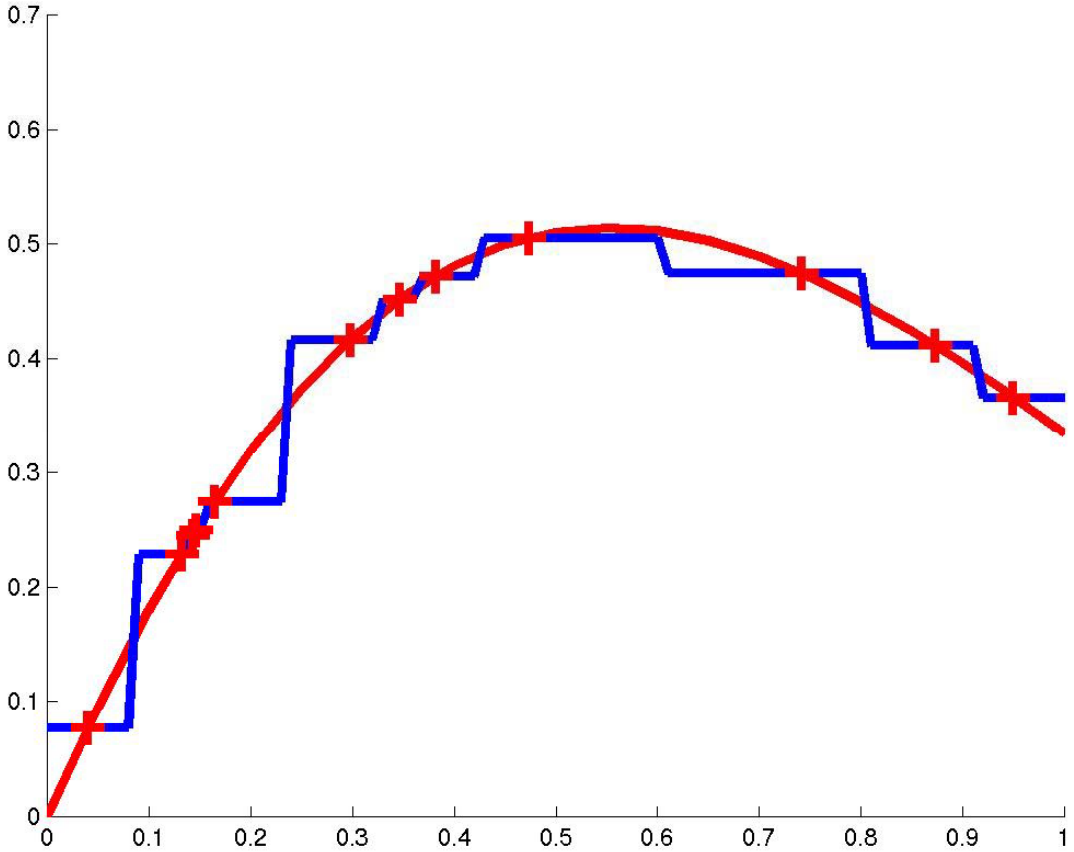
7 nearest neighbors

3x,4o



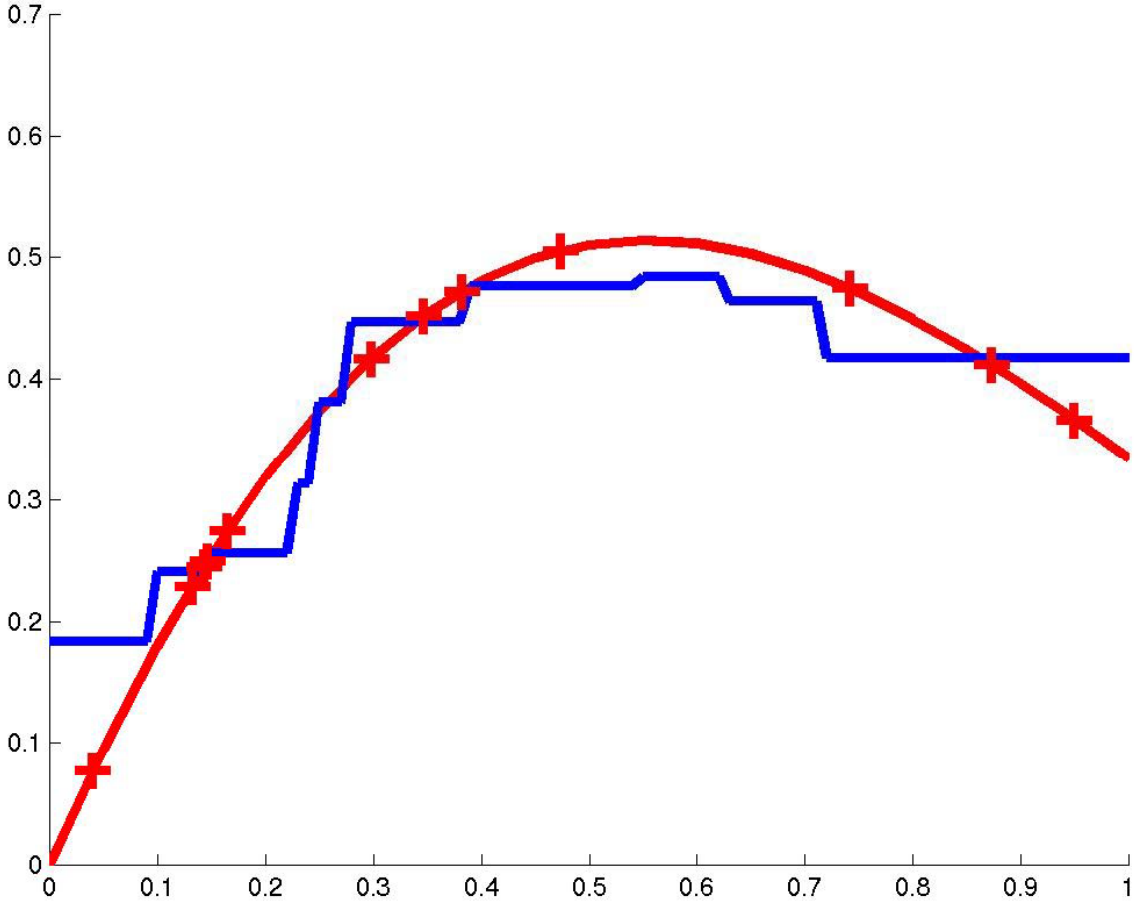
Nearest Neighbor (continuous)

1-nearest neighbor



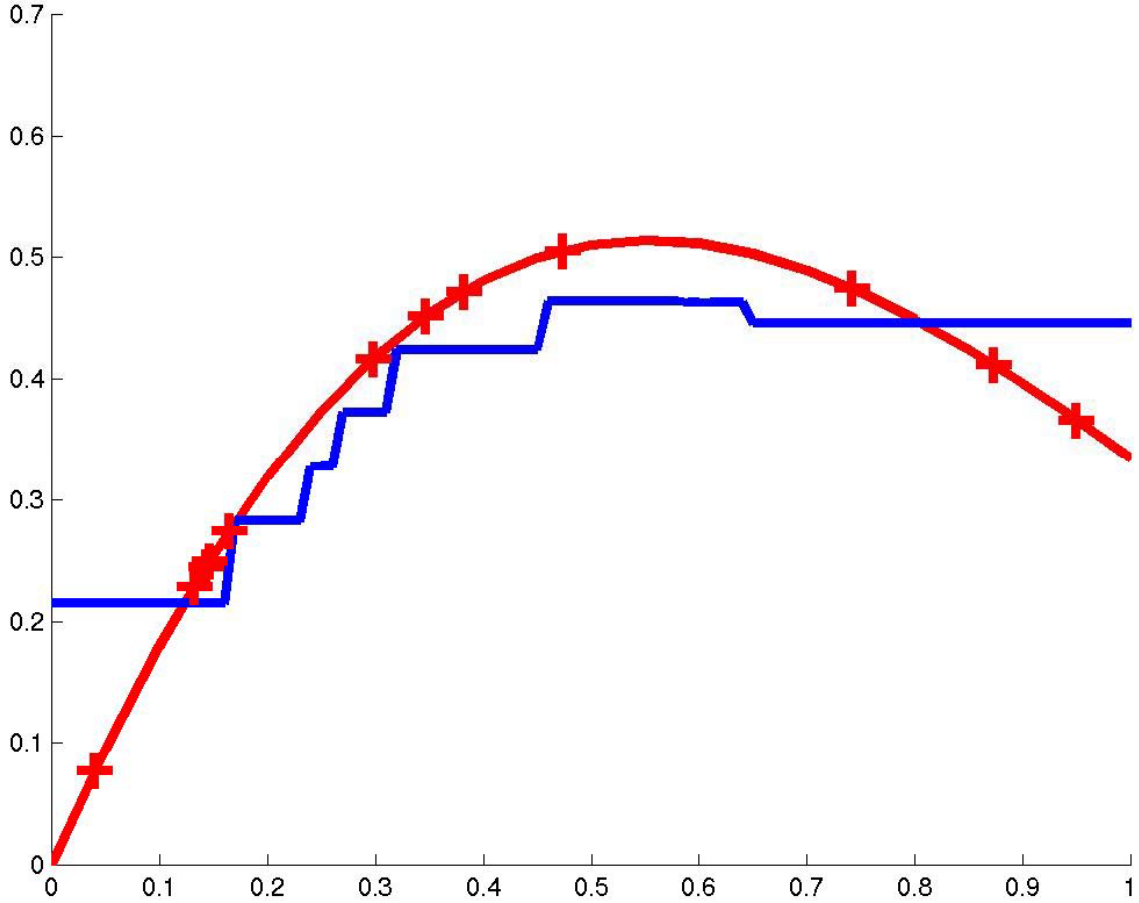
Nearest Neighbor (continuous)

3-nearest neighbor



Nearest Neighbor (continuous)

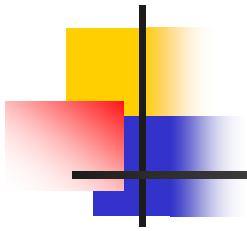
5-nearest neighbor



Locally Weighted Regression

- 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 $w_i = K(d(x_i, x_q))$



Distance Weighted k-NN

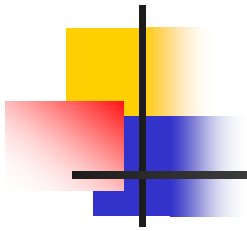
Give more weight to neighbors closer to the query point

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

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)



Distance Weighted Average

- Weighting the data:

  $\hat{f}(x_q) = \sum_i f(x_i)\, K(d(x_i,xq)) / \sum_i K(d(x_i,x_q))$

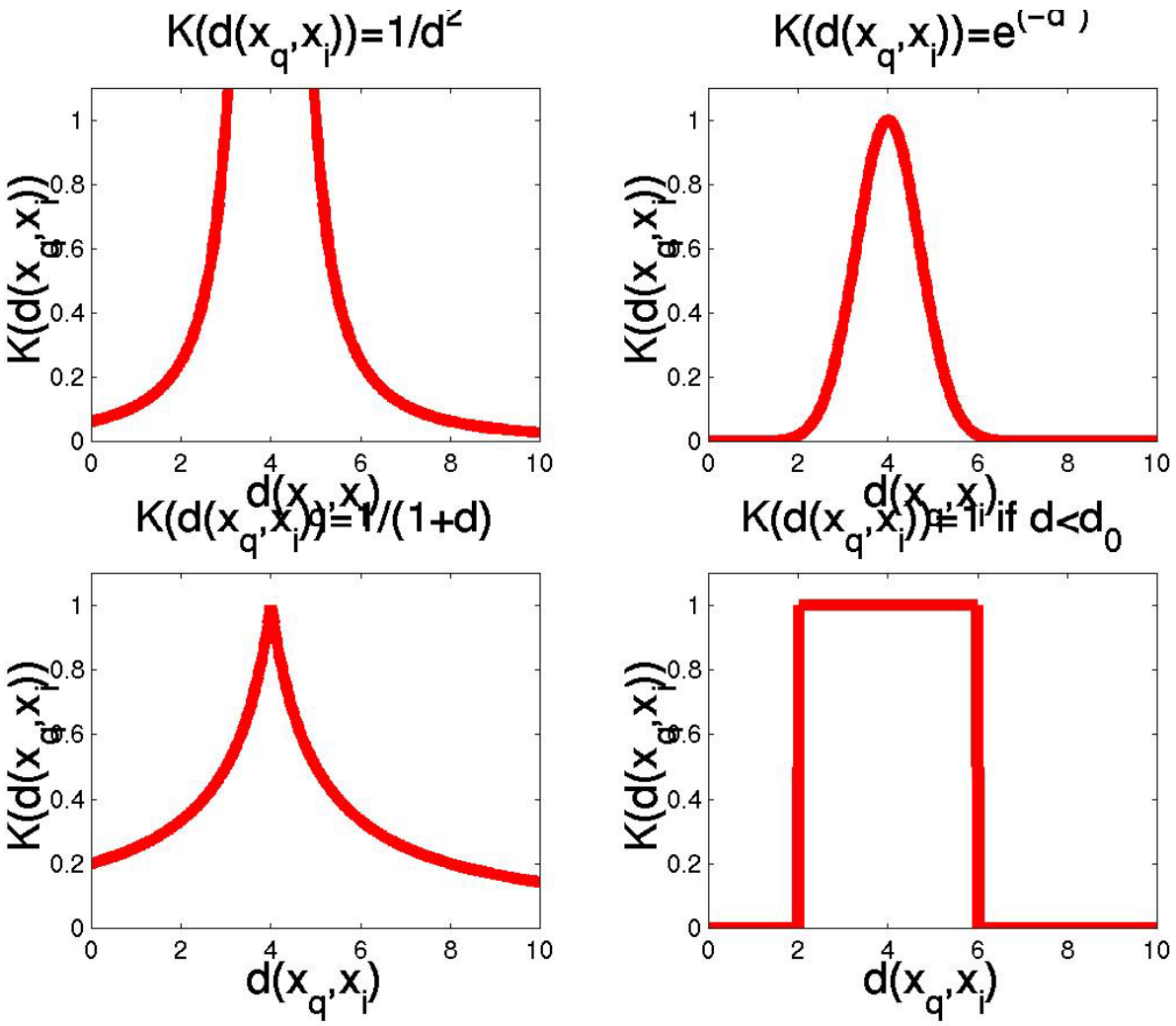
  Relevance of a data point $(x_i,f(x_i))$ is measured by calculating the distance $d(x_i,x_q)$ between the query x_q and the input vector x_i

- Weighting the error criterion:

  $E(x_q) = \sum_i (\hat{f}(x_q)-f(x_i))^2\, K(d(x_i,xq))$

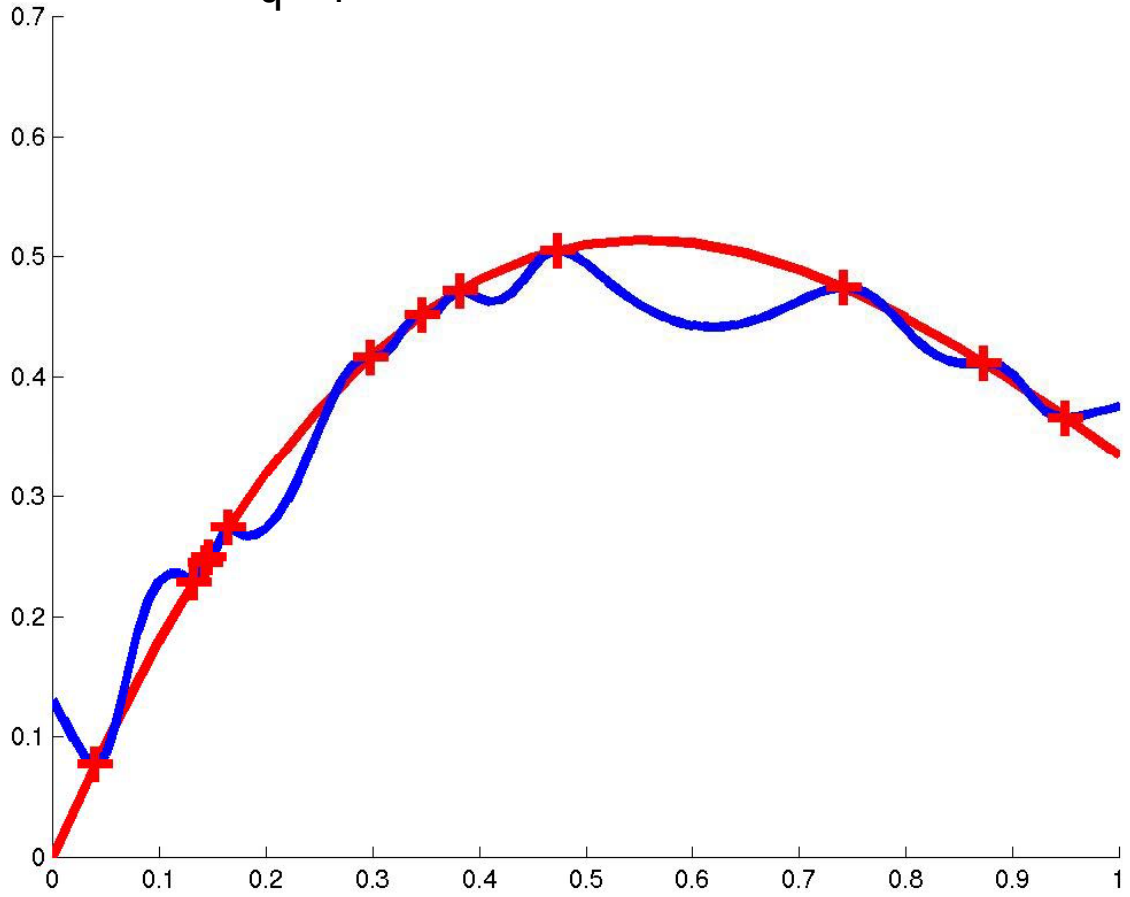
  the best estimate $\hat{f}(x_q)$ will minimize the cost $E(q)$, therefore $\partial E(q)/\partial \hat{f}(x_q)=0$

Kernel Functions



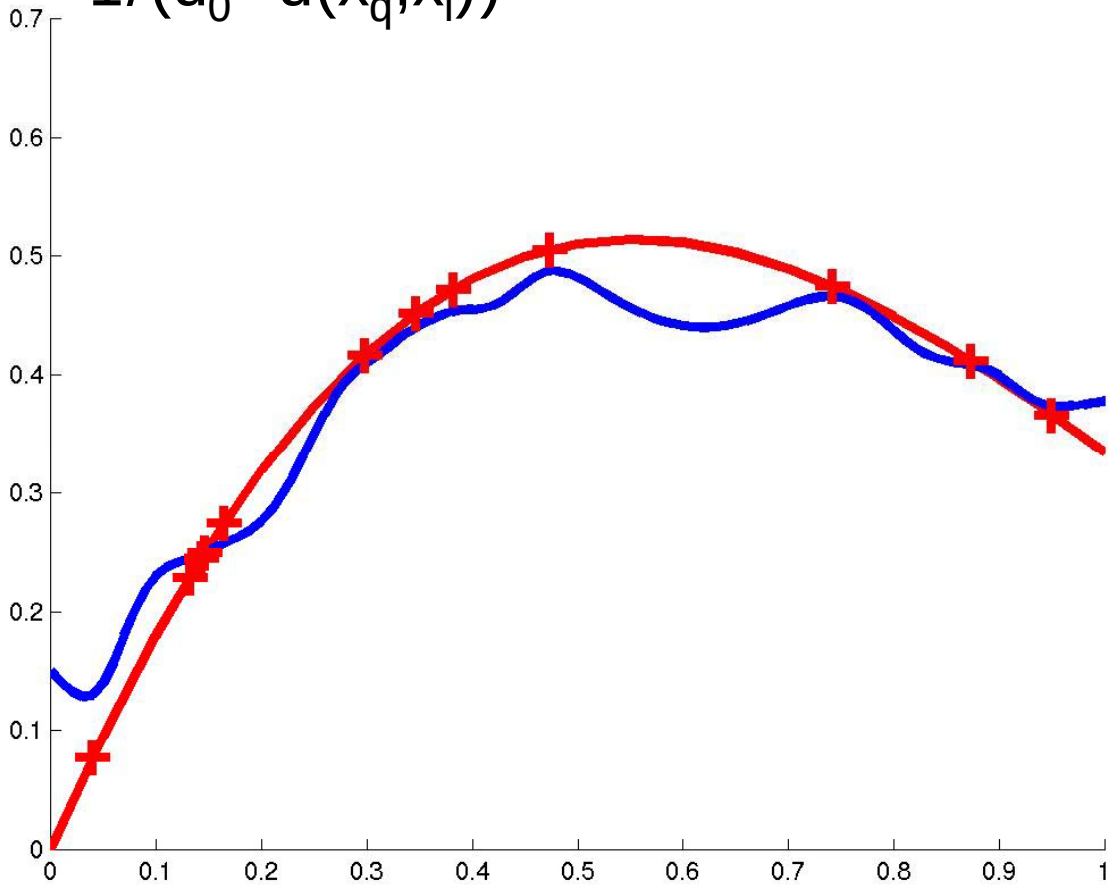
Distance Weighted NN

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



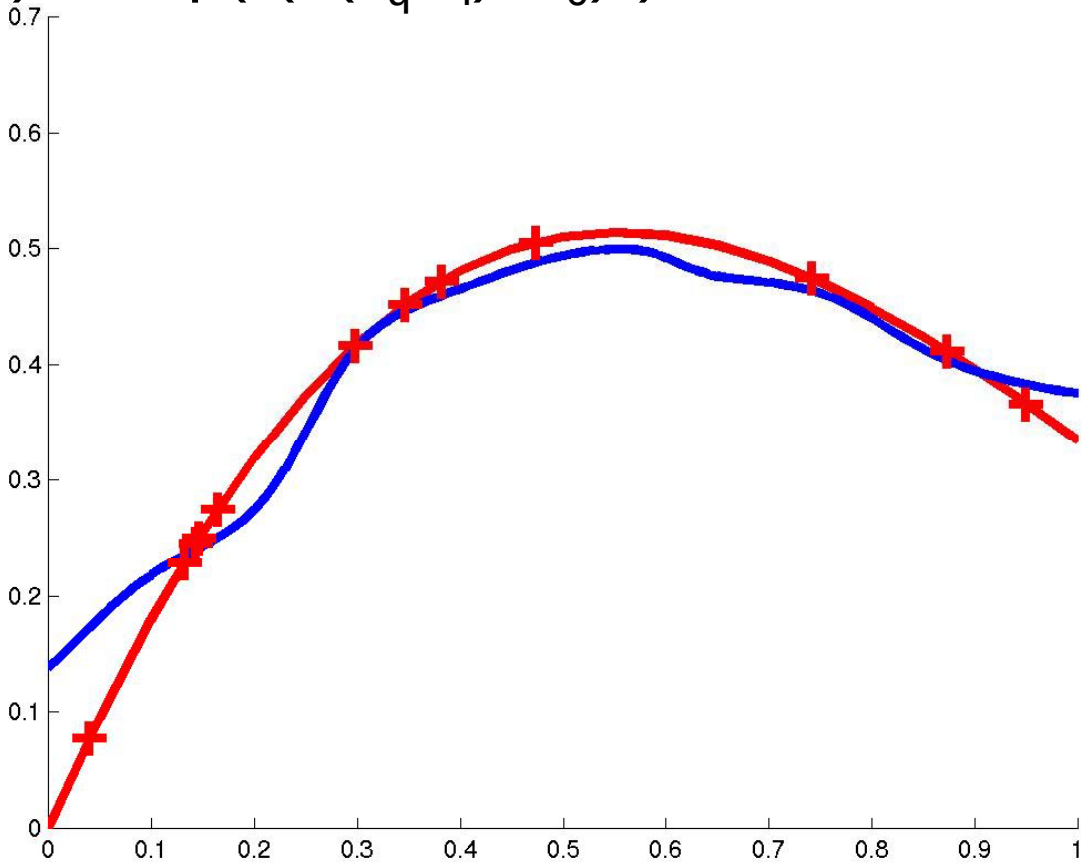
Distance Weighted NN

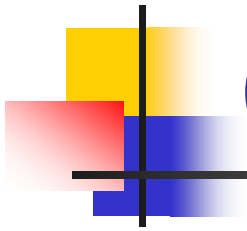
$K(d(x_q,x_i)) = 1/(d_0+d(x_q,x_i))^2$



Distance Weighted NN

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





Curse of Dimensionality

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

Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional

One approach:

- Stretch j-th 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 zj to zero eliminates this dimension alltogether (feature subset selection)

Linear Global Models

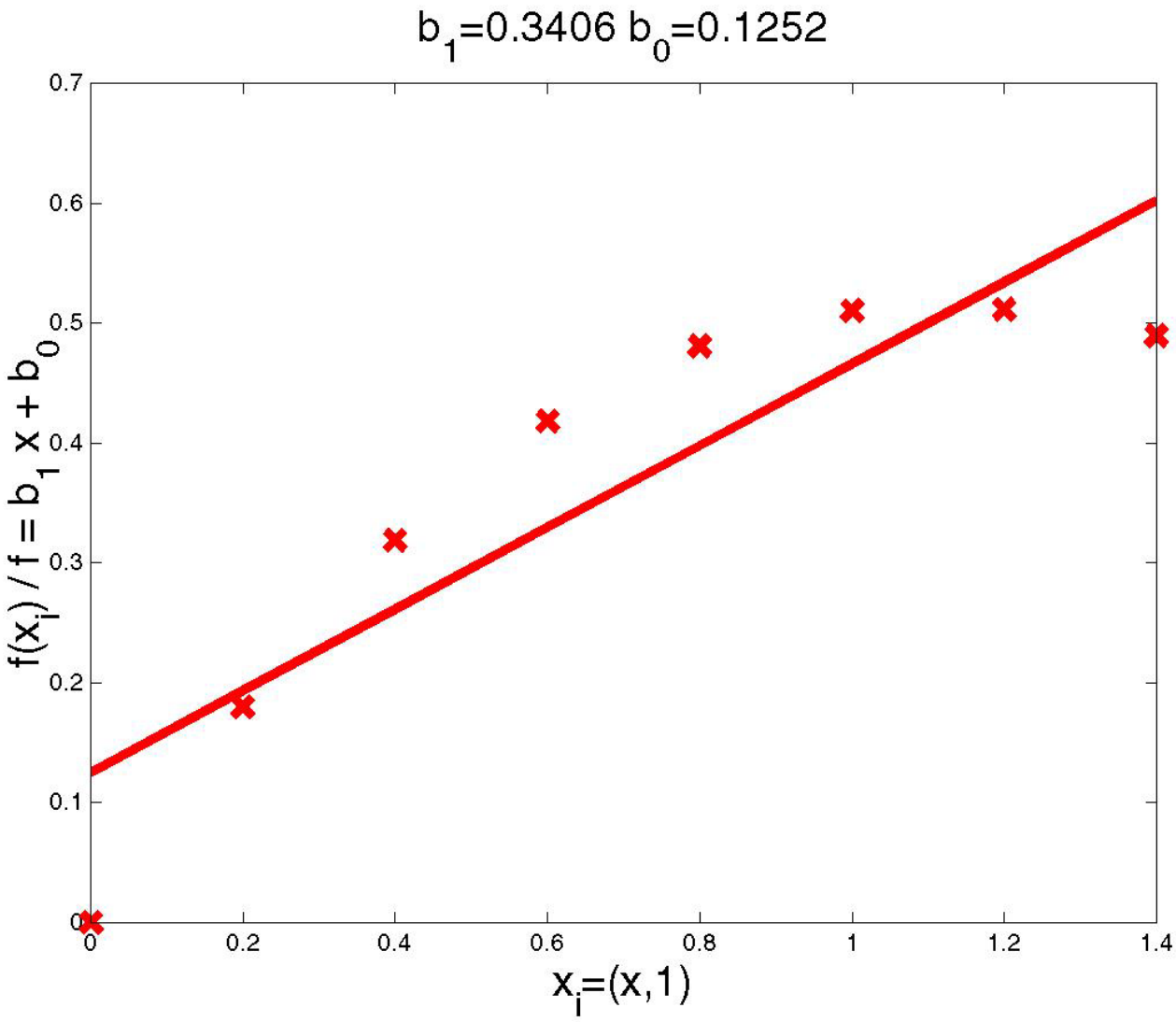
- The model is linear in the parameters w_k, which can be estimated using a least squares algorithm
  - $f\hat{}(x_i) = \sum_{k=1}^{D} \beta_k x_{ki}$ or $\mathbf{F(x)} = \mathbf{X}\,\beta$

Where $x_i=(x_1,\ldots,x_D)_i$, i=1..N, with D the input dimension and N the number of data points.

Estimate the w_k by minimizing the error criterion

- $E= \sum_{i=1}^{N} (f\hat{}(x_i) - y_i)^2$
- $(\mathbf{X}^T\mathbf{X})\,\beta = \mathbf{X}^T\,\mathbf{F(X)}$
- $\beta = \mathbf{(X}^T\,\mathbf{X)}^{-1}\,\mathbf{X}^T\,\mathbf{F(X)}$
- $\beta_k= \sum_{m=1}^{D} \sum_{n=1}^{N} (\sum_{l=1}^{D} x^T_{kl}\, x_{lm})^{-1}\, x^T_{mn}\, f(x_n)$

Linear Regression Example



Linear Local Models

- Estimate the parameters β_k such that they locally (near the query point x_q) match the training data either by
- weighting the data:

  $w_i = K(d(x_i, x_q))^{1/2}$ and transforming

  $z_i = w_i \, x_i$

  $v_i = w_i \, y_i$
- or by weighting the error criterion:

  $$E = \sum_{i=1}^{N} (x_i^T \beta - y_i)^2 \, K(d(x_i, x_q))$$

  still linear in β with LSQ solution

  $$\beta = \mathbf{((WX)^T \, WX)^{-1} \, (WX)^T \, WF(X)}$$

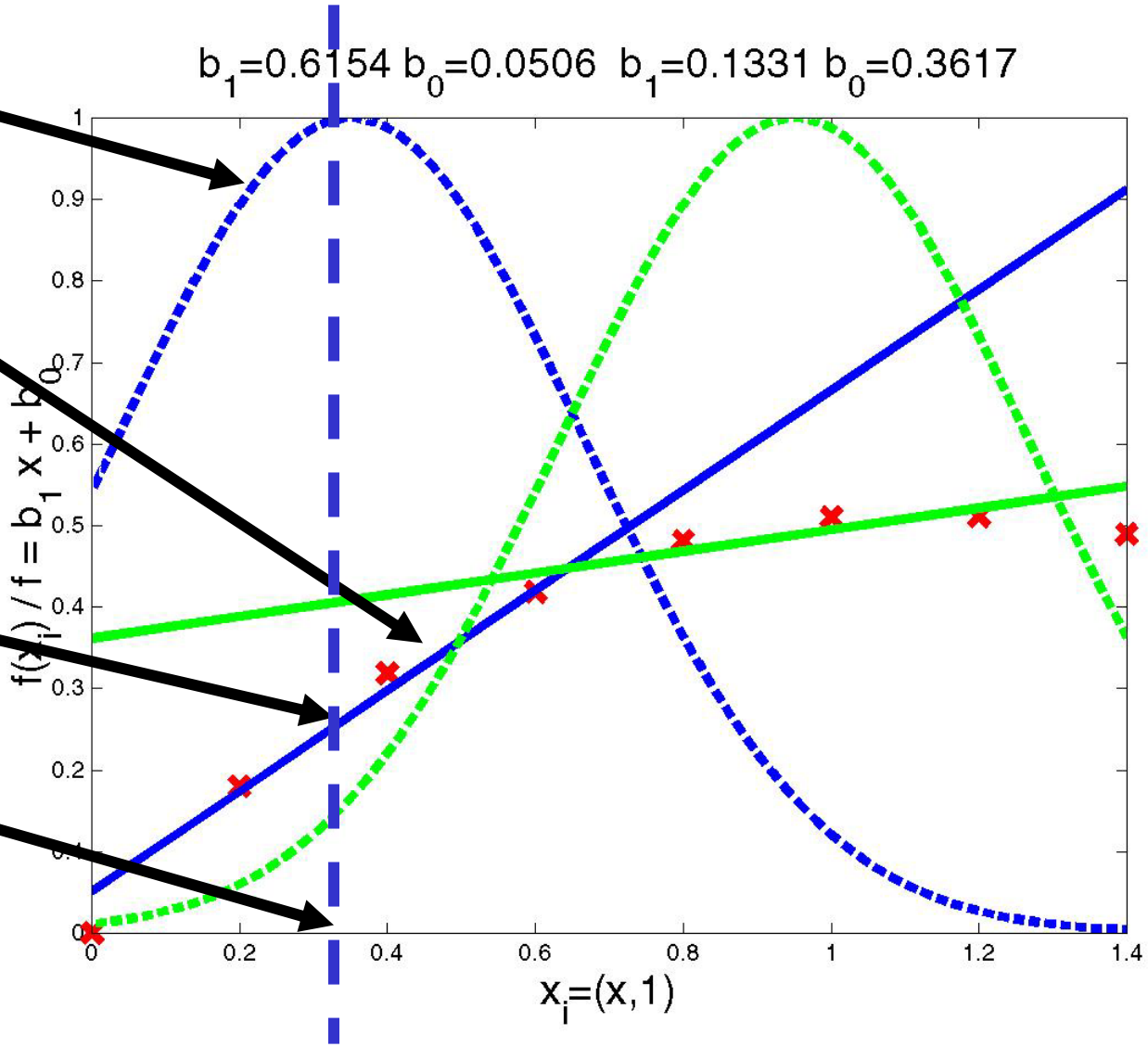
Linear Local Model Example

Kernel $K(x,x_q)$

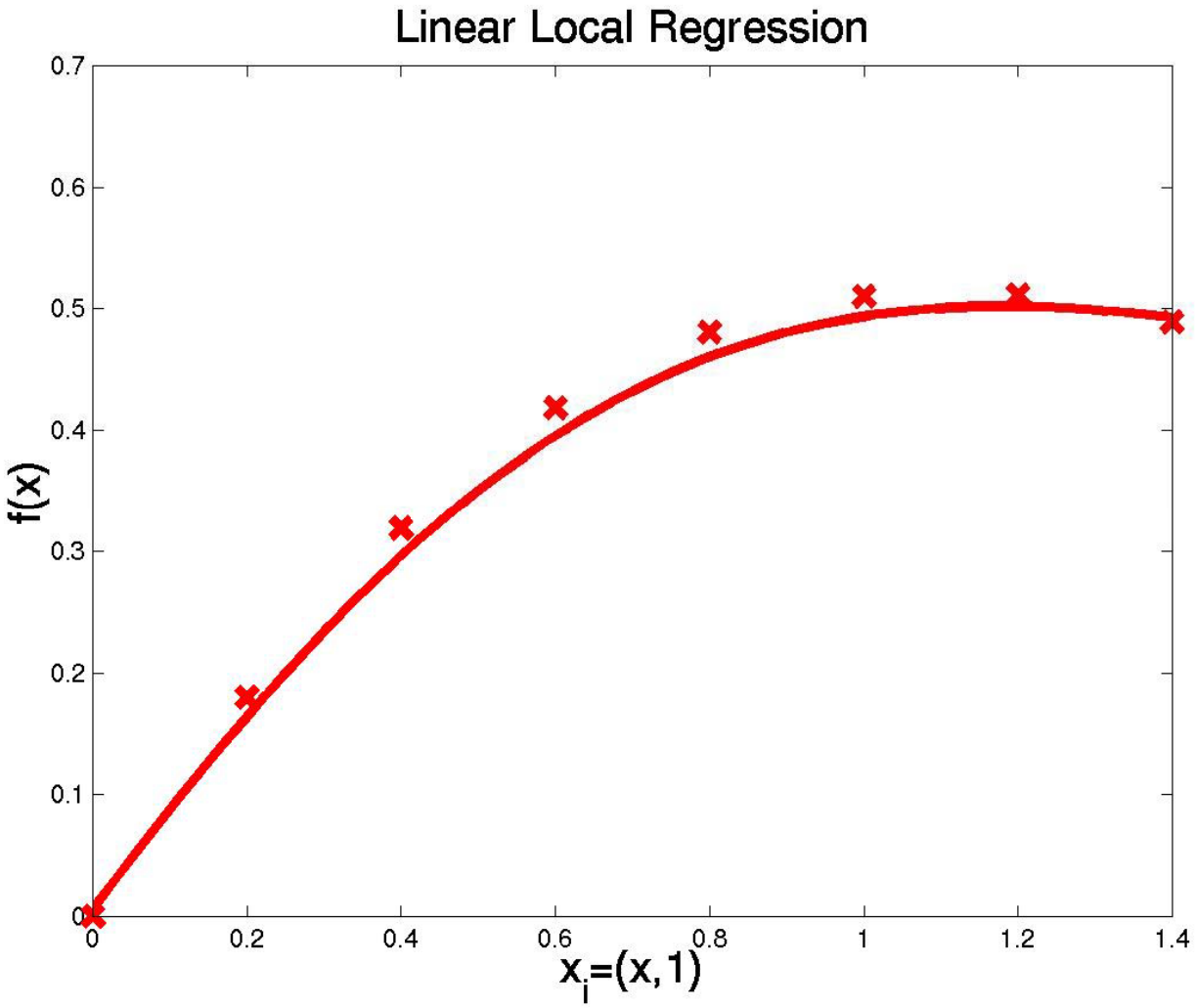
Local linear model:
$\hat{f}(x)=b_1x+b_0$

$\hat{f}(x_q)=0.266$

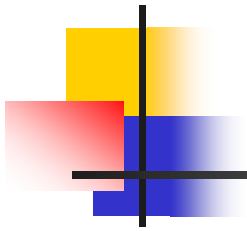
query point
$X_q=0.35$



Linear Local Model Example



Design Issues in Local Regression

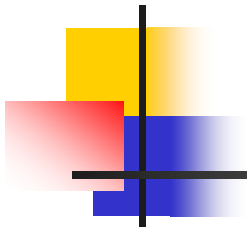


- Local model order (constant, linear, quadratic)
- Distance function d

  feature scaling: $d(x,q)=(\sum_{j=1}^{d} m_j(x_j-q_j)^2)^{1/2}$

  irrelevant dimensions $m_j=0$
- kernel function K
- smoothing parameter bandwidth h in K(d(x,q)/h)
  - h=|m| global bandwidth
  - h= distance to k-th nearest neighbor point
  - h=h(q) depending on query point
  - $h=h_i$ depending on stored data points

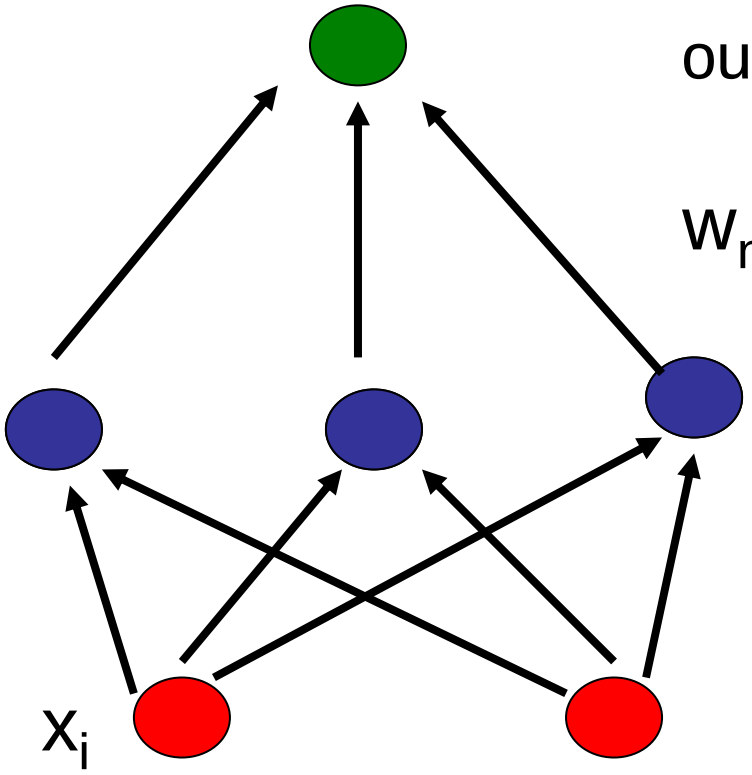
See paper by Atkeson [1996] ”Locally Weighted Learning”



Radial Basis Function Network

- Global approximation to target function in terms of linear combination of local approximations
- Used, e.g. for image classification
- 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”

Radial Basis Function Network



output f(x)

w_n linear parameters

Kernel functions

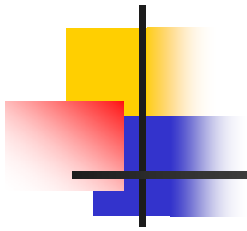
$K_n(d(x_n,x)) = \exp(-1/2\ d(x_n,x)^2/\sigma^2)$

input layer

x_i

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

Training Radial Basis Function Networks



- How to choose the center x_n for each Kernel function K_n?
  - scatter uniformly across instance space
  - use distribution of training instances (clustering)
- How to train the weights?
  - Choose mean x_n and variance σ_n for each K_n
    nonlinear optimization or EM
  - Hold K_n fixed and use local linear regression to compute the optimal weights w_n

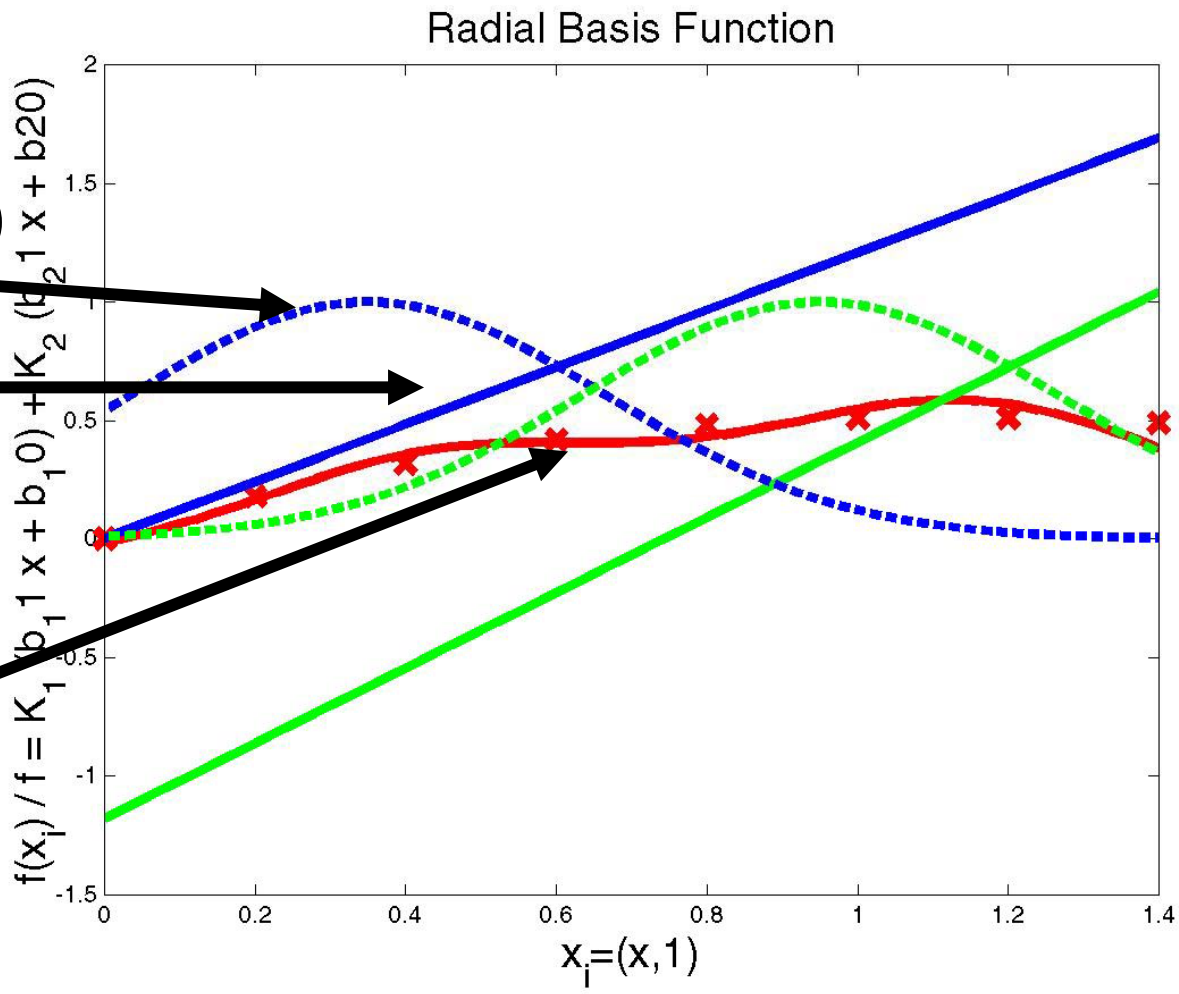
Radial Basis Network Example

$K_1(d(x_1,x))=$

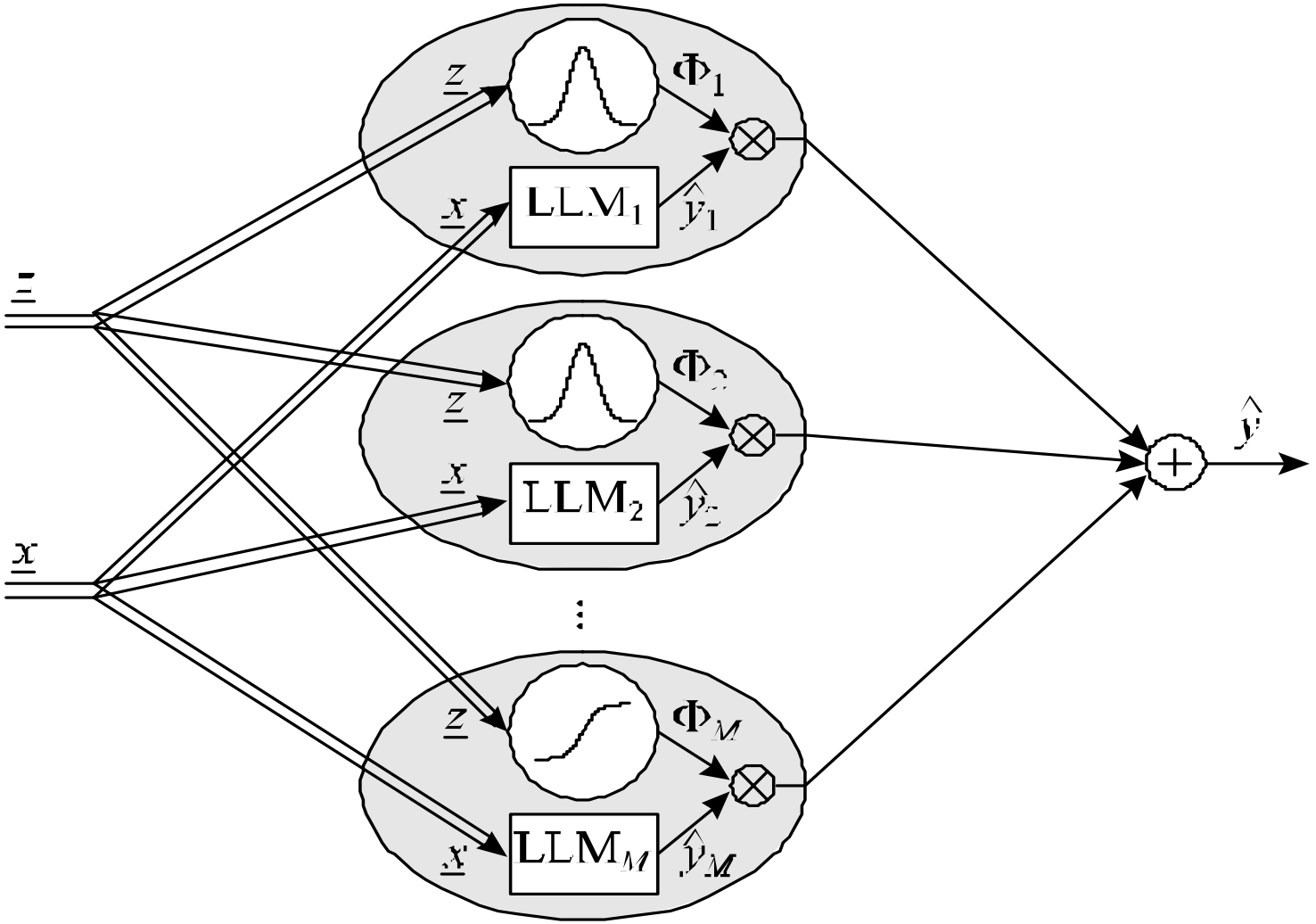
$\exp(-1/2\ d(x_1,x)^2/\sigma^2)$

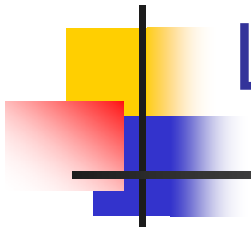
$w1\ x+\ w0$

$f\hat{}(x) = K_1\ (w_1\ x+\ w_0)$
$+\ K_2\ (w_3\ x\ +\ w_2)$



Local Linear Models





Local Linear Model Tree (LOLIMOT)

- incremental tree construction algorithm
- partitions input space by axis-orthogonal splits
- adds one local linear model per iteration

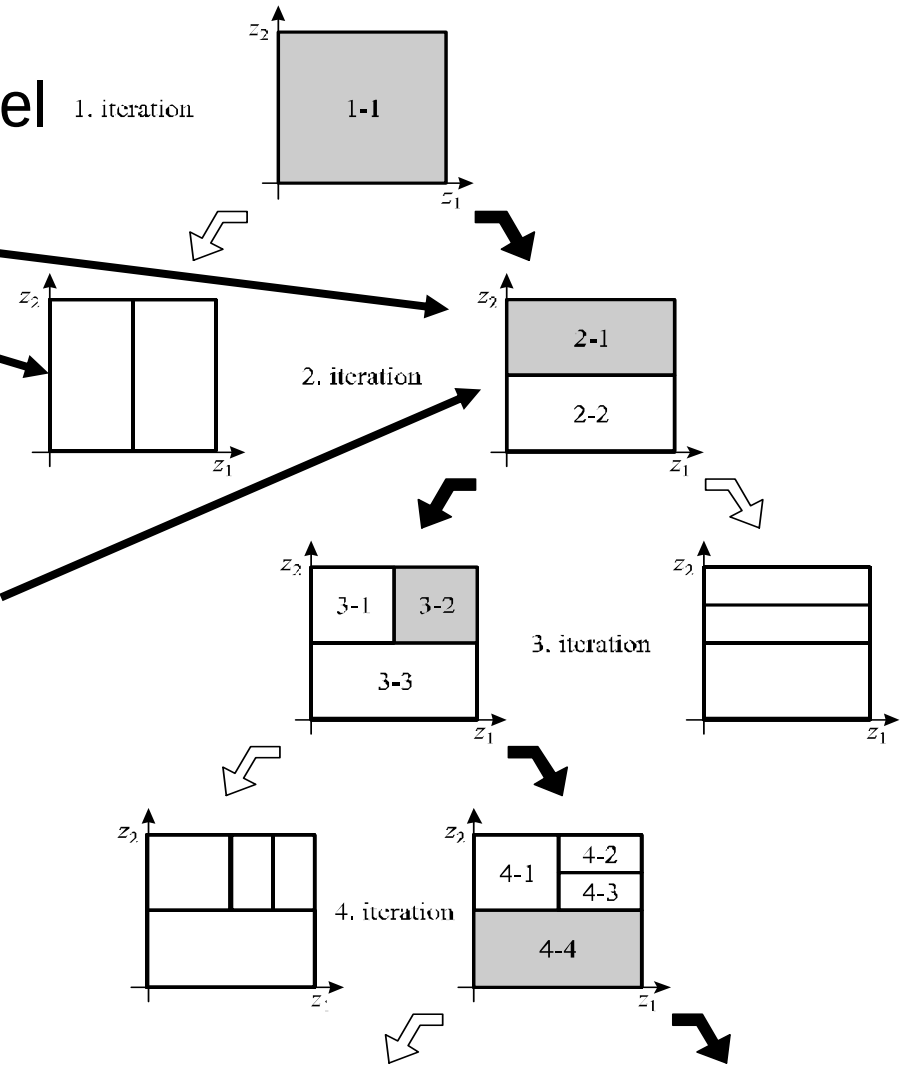
1. start with an initial model (e.g. single LLM)
2. identify LLM with worst model error E_i
3. check all divisions : split worst LLM hyper-rectangle in halves along each possible dimension
4. find best (smallest error) out of possible divisions
5. add new validity function and LLM
6. repeat from step 2. until termination criteria is met

LOLIMOT

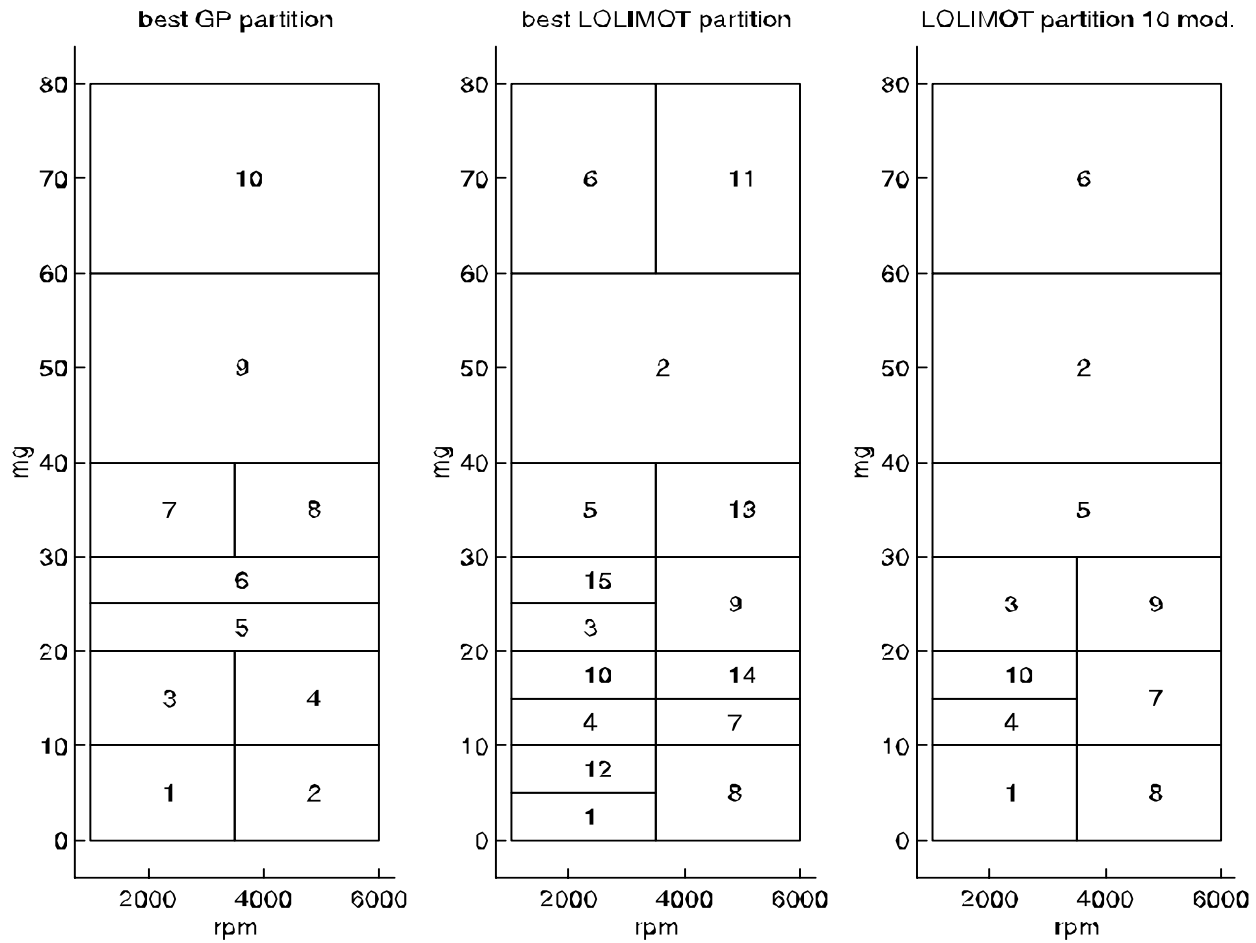
Initial global linear model

Split along x1 or x2

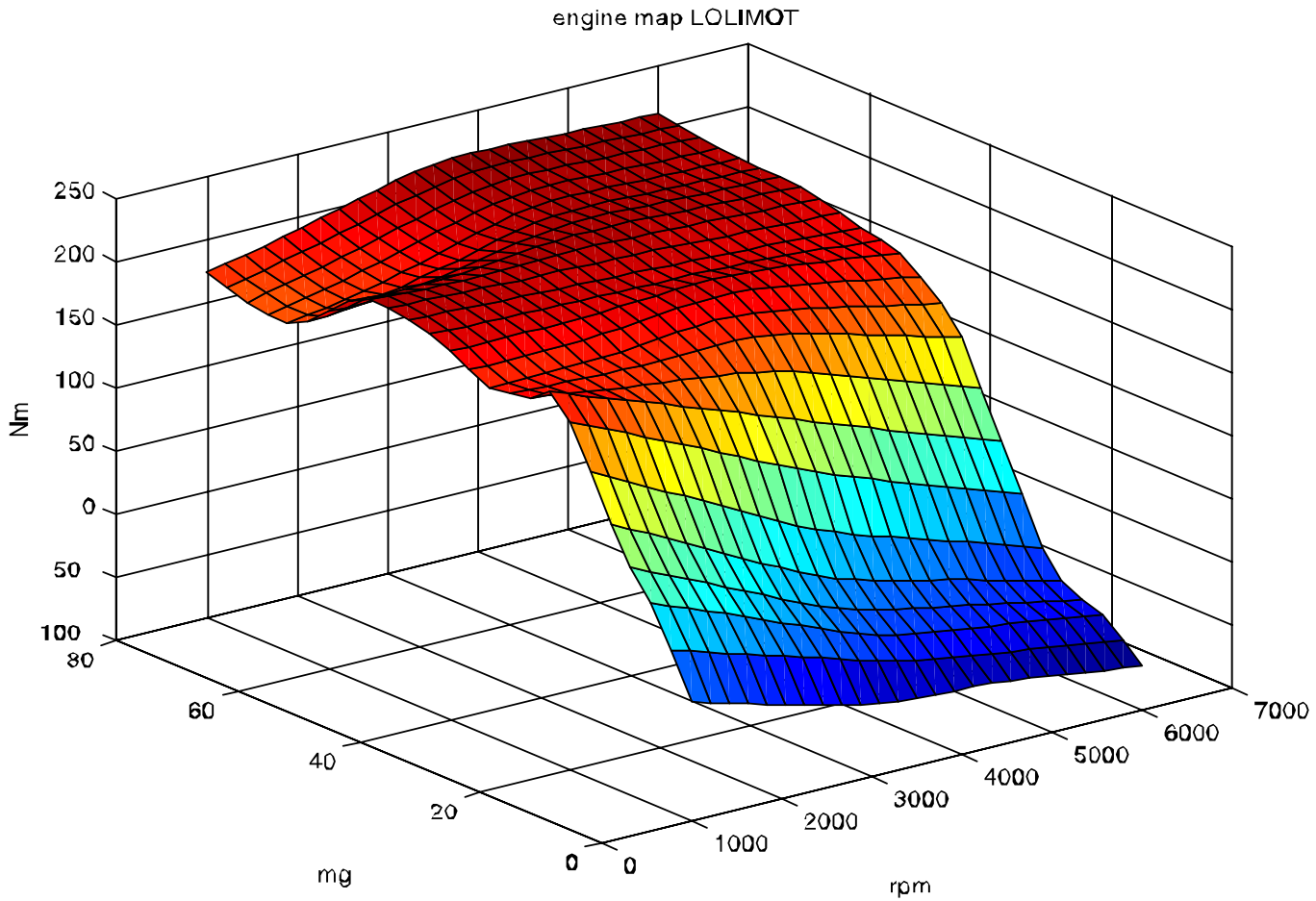
Pick split that minimizes model error (residual)

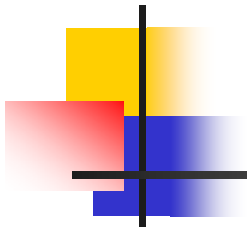


LOLIMOT Example



LOLIMOT Example





Lazy and Eager Learning

- Lazy: wait for query before generalizing
  - k-nearest neighbors, weighted linear regression
- Eager: generalize before seeing query
  - Radial basis function networks, decision trees, back-propagation, LOLIMOT
- Eager learner must create global approximation
- Lazy learner can create local approximations
- If they use the same hypothesis space, lazy can represent more complex functions (H=linear functions)