Instance Based Learning

- *k*-Nearest Neighbor
- Locally weighted regression
- Lazy and eager learning

Some Vocabulary

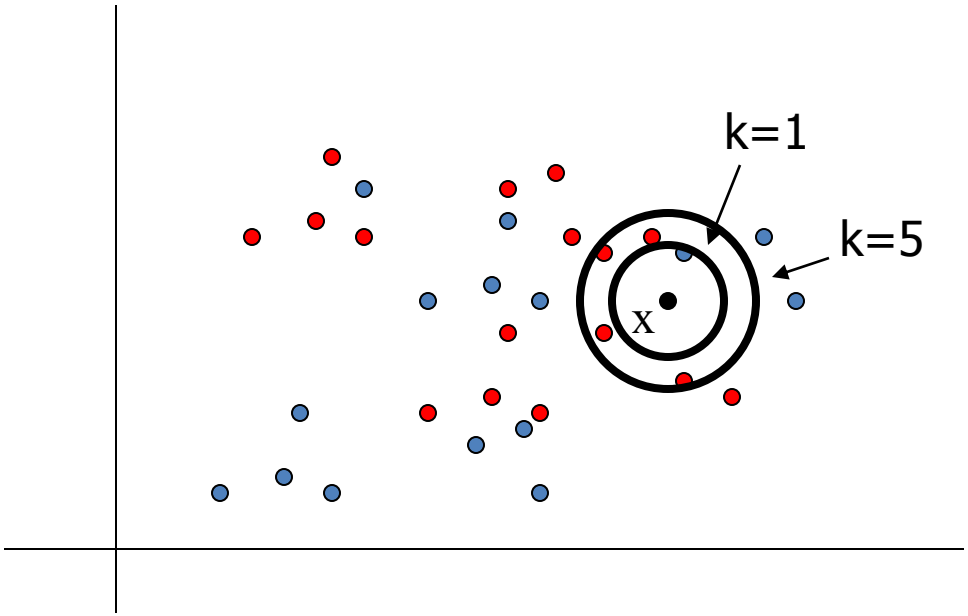
- **Parametric vs. Non-parametric:**
  - **parametric**:
    - A particular functional form is assumed, e.g., multivariate normal, naïve Bayes.
    - Advantage of simplicity – easy to estimate and interpret
    - may have high bias because the real data may not obey the assumed functional form.
  - **non-parametric:**
    - distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form.
- Other terms: Instance-based, Memory-based, Lazy, Case-based, kernel methods…

Nearest Neighbor Algorithm

- Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example **x** by finding the training example $(\mathbf{x}^i, y^i)$ that is *nearest* to **x**
  - Guess the class $y = y^i$

K-Nearest Neighbor Methods

- To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class

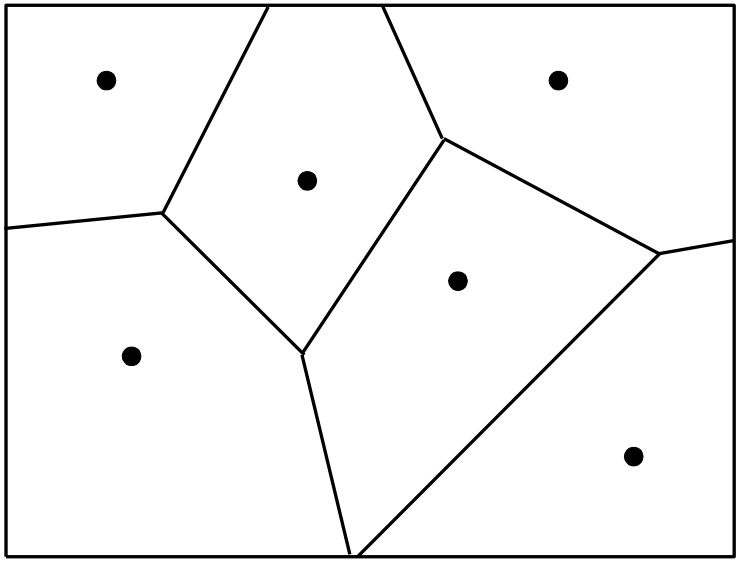


common values for k: 3, 5

Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.

1-NN Decision Surface



- Each line segment is equidistant between two points of opposite classes. The more examples that are stored, the more complex the decision boundaries can become.

Instance-Based Learning

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

Nearest neighbor (1-Nearest neighbor):

- Given query instance x_q, locate nearest example x_n, estimate

$$\hat{f}(x_q) \leftarrow f(x_n)$$

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)

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

Distance-Weighted *k*-NN

Might want to 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

$\rightarrow$ Shepard's method

Nearest Neighbor

When to Consider

- Instance 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 lose information

Disadvantages

- Slow at query time
- Easily fooled by irrelevant attributes

Issues

- Distance measure
  - Most common: Euclidean
- Choosing k
  - Increasing k reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

Distance

- Notation: object with p measurements

$$x^i = (x^i_1, x^i_2, \ldots, x^i_p)$$

- Most common distance metric is *Euclidean* distance:

$$d_E(x^i, x^j) = \left(\sum_{k=1}^{p} (x^i_k - x^j_k)^2 \right)^{\frac{1}{2}}$$

- efficiency trick: using squared Euclidean distance gives same answer, avoids computing square root

- ED makes sense when different measurements are commensurate; each is variable measured in the same units.

- If the measurements are different, say length and weight, it is not clear.

Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of x_k :

$$\hat{\sigma}_k = \left(\frac{1}{n} \sum_{i=1}^{n} \left(x_k^i - \bar{x}_k \right)^2 \right)^{\frac{1}{2}}$$

where x_k is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^i$$

Weighted Euclidean distance

Finally, if we have some idea of the relative importance of each variable, we can weight them:

$$d_{WE}(i,j) = \left(\sum_{k=1}^{p} w_k (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

One option: weight each feature by its mutual information with the class.

Other Distance Metrics

- Minkowski or L_λ metric:

$$d(i,j) = \left(\sum_{k=1}^{p} (x_k(i) - x_k(j))^{\lambda} \right)^{\frac{1}{\lambda}}$$

- Manhattan, city block or L_1 metric:

$$d(i,j) = \sum_{k=1}^{p} \left| x_k(i) - x_k(j) \right|$$

- L_∞

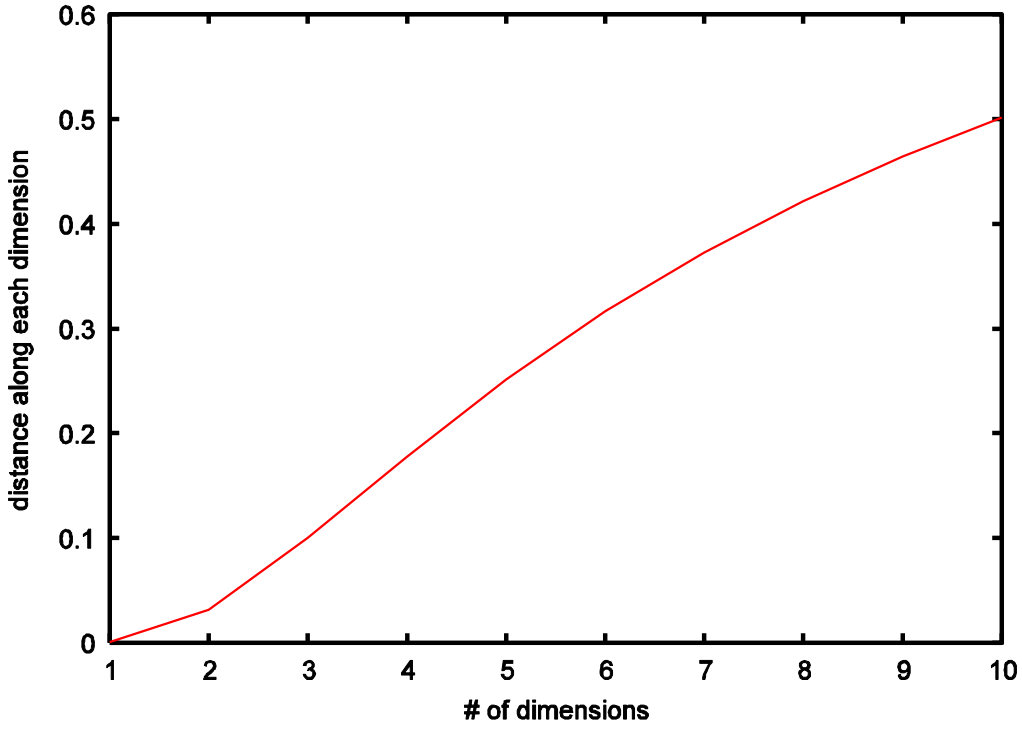
$$d(i,j) = \max_k \left| x_k(i) - x_k(j) \right|$$

The Curse of Dimensionality

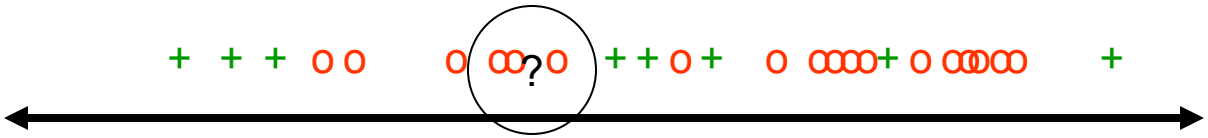
- Nearest neighbor breaks down in high-dimensional spaces because the "neighborhood" becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.
- Suppose our query point is at the origin.
  - 1D –
    - On a one dimensional line, we must go a distance of 5/5000 = 0.001 on average to capture the 5 nearest neighbors
  - 2D –
    - In two dimensions, we must go sqrt(0.001) to get a square that contains 0.001 of the volume
  - D –
    - In d dimensions, we must go $(0.001)^{1/d}$

Curse of Dimensionality cont.

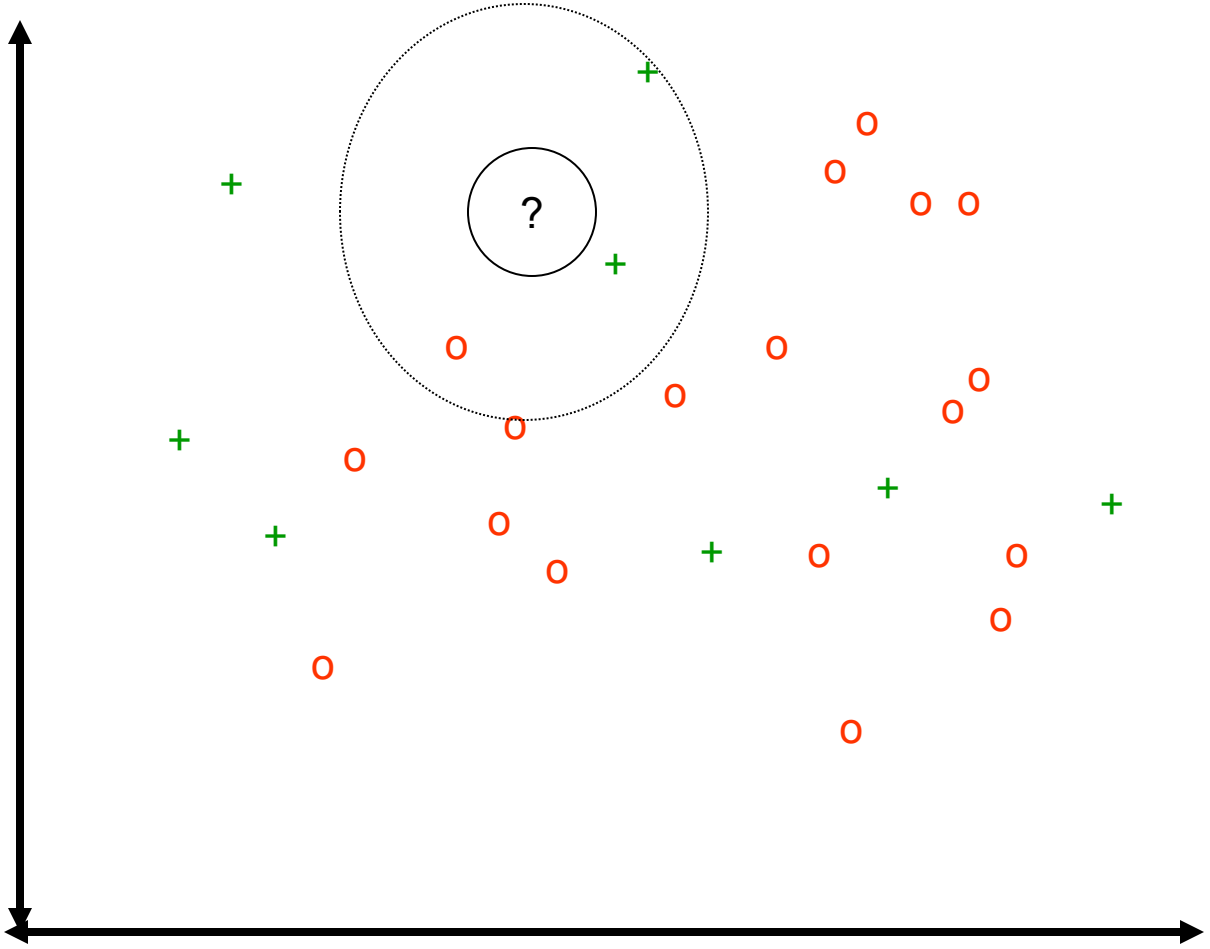
- With 5000 points in 10 dimensions, we must go 0.501 distance along each attribute in order to find the 5 nearest neighbors!



K-NN and irrelevant features



K-NN and irrelevant features

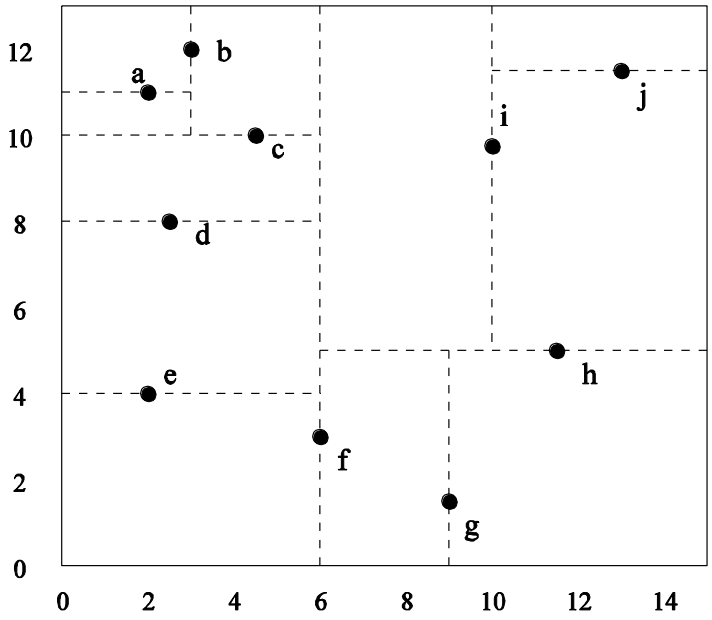


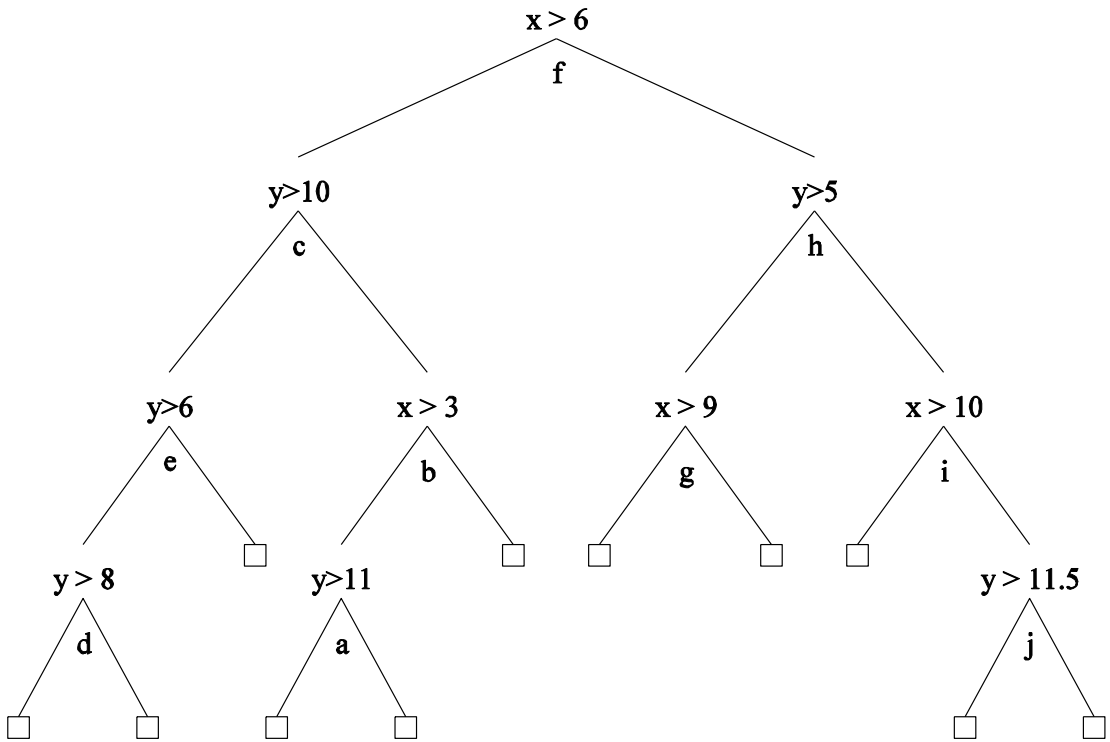
Efficient Indexing: Kd-trees

- A kd-tree is similar to a decision tree, except that we split using the median value along the dimension having the highest variance, and points are stored
- A kd-tree is a tree with the following properties
  - Each node represents a rectilinear region (faces aligned with axes)
  - Each node is associated with an axis aligned plane that cuts its region into two, and it has a child for each sub-region
  - The directions of the cutting planes alternate with depth

Kd-trees

- A kd-tree is similar to a decision tree, except that we split using the median value along the dimension having the highest variance, and points are stored





Edited Nearest Neighbor

- Storing all of the training examples can require a huge amount of memory. Select a subset of points that still give good classifications.
  - **Incremental deletion**. Loop through the training data and test each point to see if it can be correctly classified given the other points. If so, delete it from the data set.
  - **Incremental growth.** Start with an empty data set. Add each point to the data set only if it is not correctly classified by the points already stored.

KNN Advantages

- Easy to program
- No optimization or training required
- Classification accuracy can be very good; can outperform more complex models

Nearest Neighbor Summary

- Advantages
  - variable-sized hypothesis space
  - Learning is extremely efficient
    - however growing a good kd-tree can be expensive
  - Very flexible decision boundaries
- Disadvantages
  - distance function must be carefully chosen
  - Irrelevant or correlated features must be eliminated
  - Typically cannot handle more than 30 features
  - Computational costs: Memory and classification-time computation

Locally Weighted Linear Regression: LWLR

- Idea:
  - k-NN forms local approximation for each query point x_q
  - Why not form an explicit approximation $\hat{f}$ for region surrounding x_q
    - Fit linear function to k nearest neighbors
    - Fit quadratic, ...
    - Thus producing ``piecewise approximation'' to $\hat{f}$
      - Minimize error over k nearest neighbors of x_q
      - Minimize error entire set of examples, weighting by distances
      - Combine two above

LWLR: Continued

- Linear regression $\hat{f}(x) = w_0 + \sum_{i=1}^{n} w_i a_i(x)$

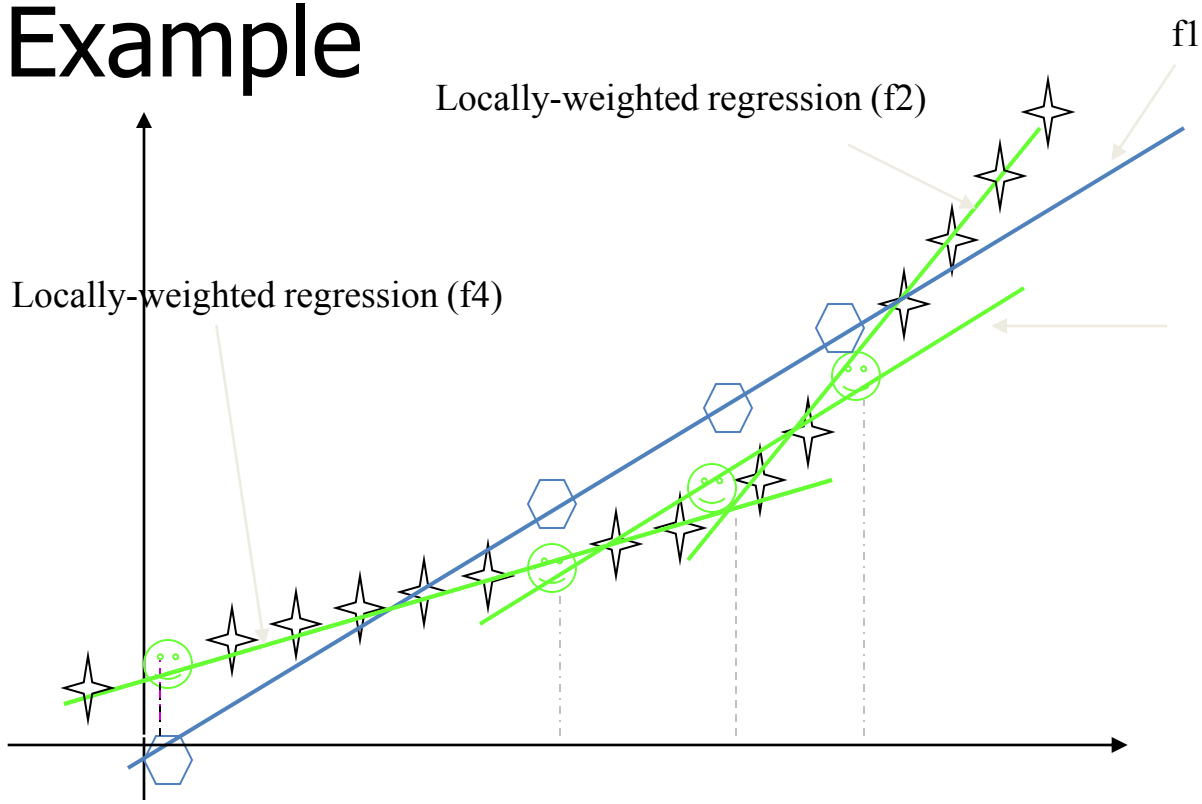
$$Error = \frac{1}{2}\sum_{x \in D}\left(f(x) - \hat{f}(x)\right)^2$$

$$Error1 = \frac{1}{2}\sum_{x \in k\ nearest\ nbrs\ of\ xq}\left(f(x) - \hat{f}(x)\right)^2$$

$$Error2 = \frac{1}{2}\sum_{x \in D}\left(f(x) - \hat{f}(x)\right)^2 K(d(x_q, x))$$

$$Error3 = \frac{1}{2}\sum_{x \in k\ nearest\ neighbors\ of\ xq}\left(f(x) - \hat{f}(x)\right)^2 K(d(x_q, x))$$

LWR Example



Training data

Predicted value using simple regression

Predicted value using locally weighted (piece-wise) regression

Yike Guo, Advanced Knowledge Management, 2000

Lazy and Eager Learning

Lazy: wait for query before generalizing

- k-Nearest Neighbor

- Eager: generalize before seeing query
  - ID3, Backpropagation, etc.

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 functions

What you need to know

- Instance-based learning
  - non-parametric
  - trade decreased learning time for increased classification time
- Issues
  - appropriate distance metrics
  - curse of dimensionality
  - efficient indexing

Given the following data:

instance	x_1	x_2	class
1	0.25	0.25	+
2	0.25	0.75	+
3	0.50	0.25	-
4	0.50	0.75	-
5	0.75	0.50	-
6	0.75	1.00	+
7	0.25	0.55	?
8	0.75	0.80	?

How does the kNN algorithm classify instances 7 and 8, with $k = 1$ and $k = 3$? You can use simple majority voting and the Manhattan distance.