Instance-based Learning

Machine Learning – 10701/15781

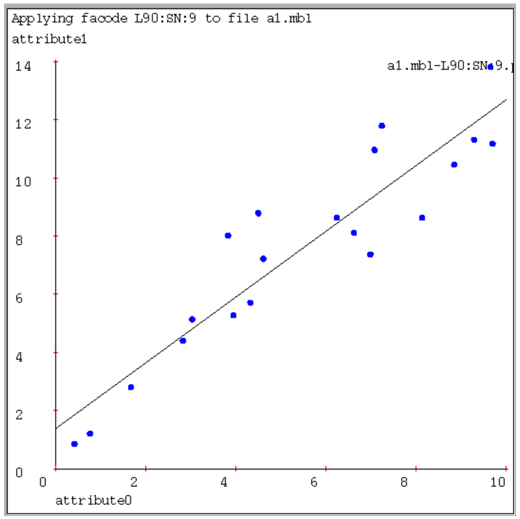
Carlos Guestrin
Carnegie Mellon University

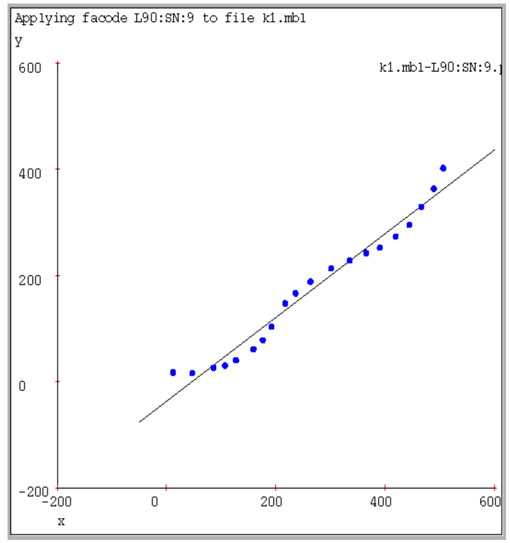
February 14th, 2005

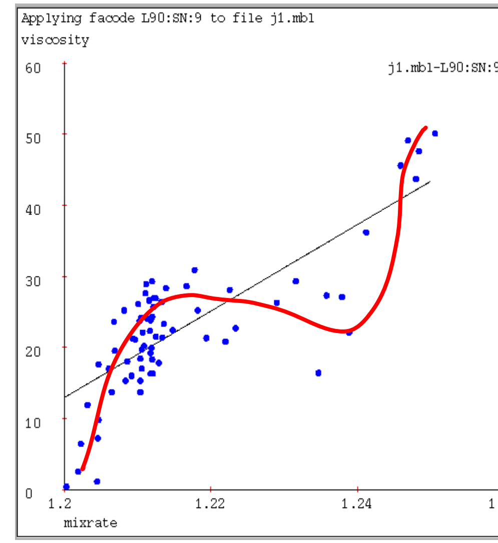
Announcements

- Reminder: Second homework due Monday 21st

Why not just use Linear Regression?



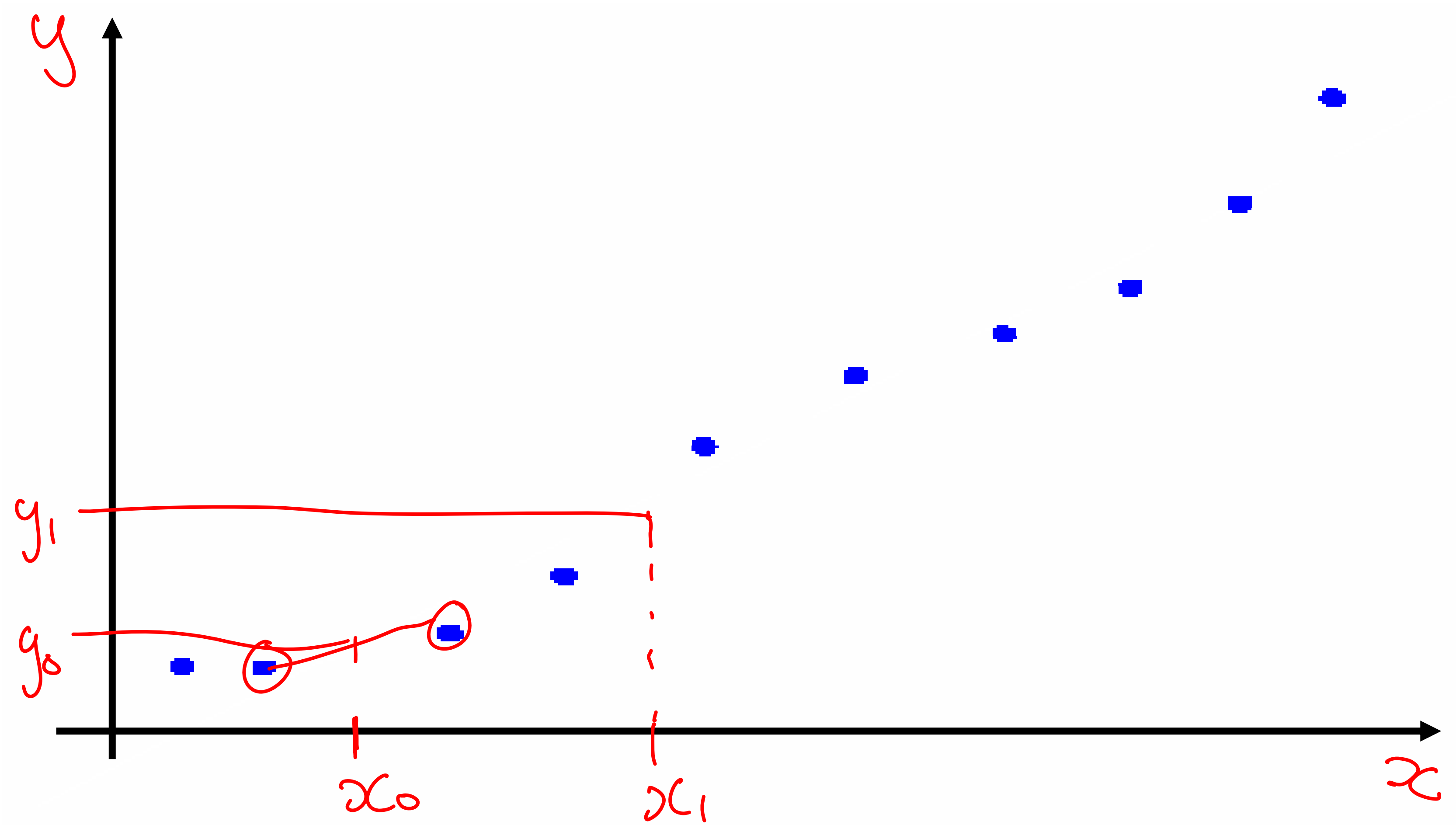




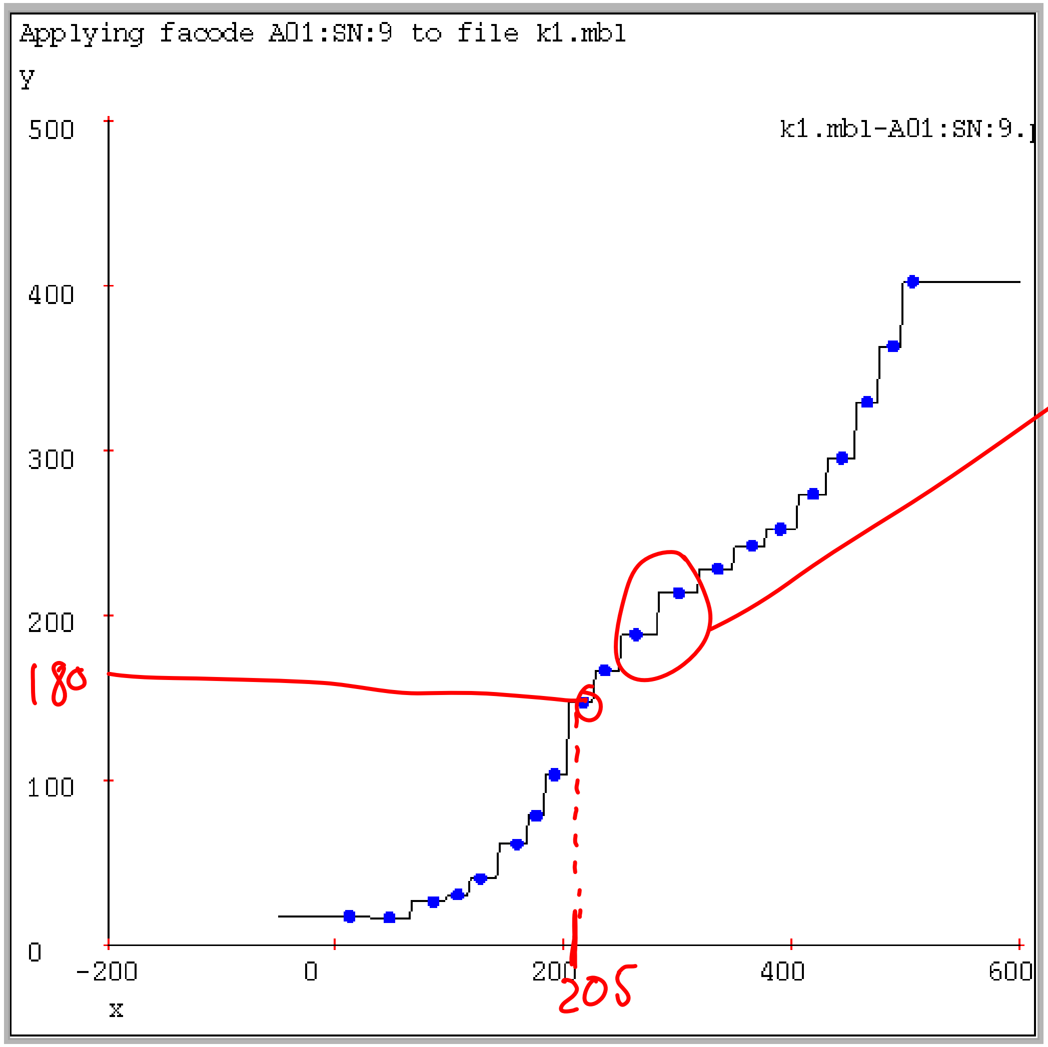
$y = ax + b$

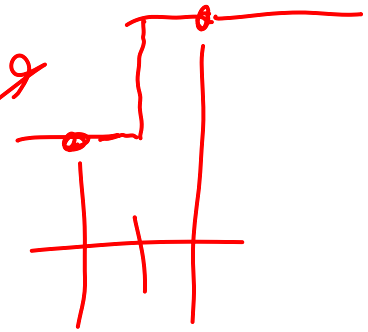
More basis functions

Using data to predict new data



Nearest neighbor





Univariate 1-Nearest Neighbor

Given datapoints (x_1,y_1) (x_2,y_2)..(x_N,y_N),where we assume $y_i=f(s_i)$ for some unknown function f.

Given query point x_q, your job is to predict $\hat{y} \approx f(x_q)$ query point

Nearest Neighbor:

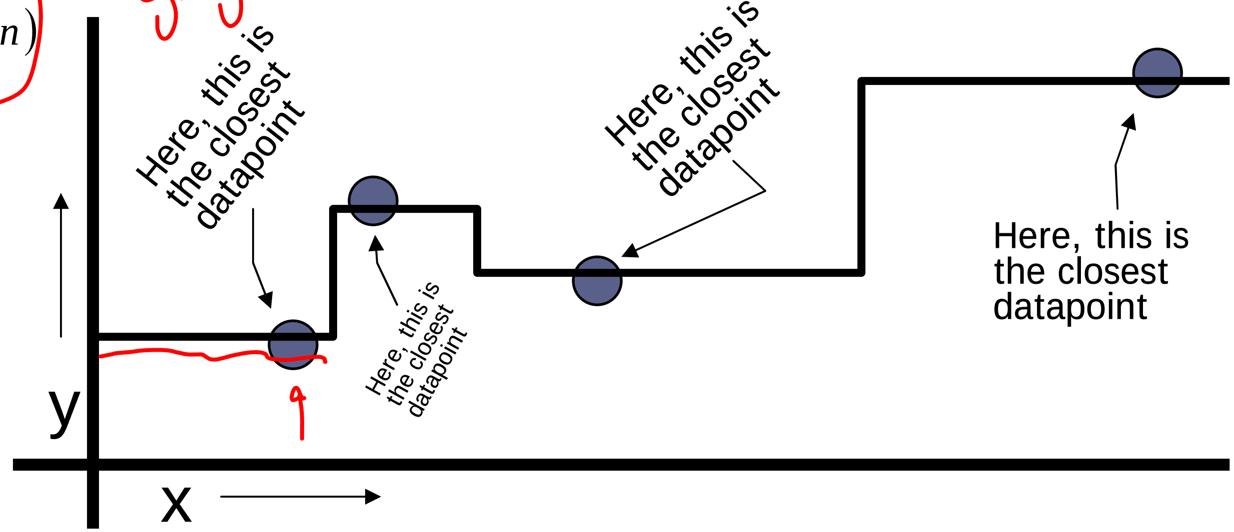
1. Find the closest x_i in our set of datapoints

$$i(nn) = \underset{i}{\operatorname{argmin}} \left| x_i - x_q \right|$$

closest guy

2. Predict $\hat{y} = y_{i(nn)}$

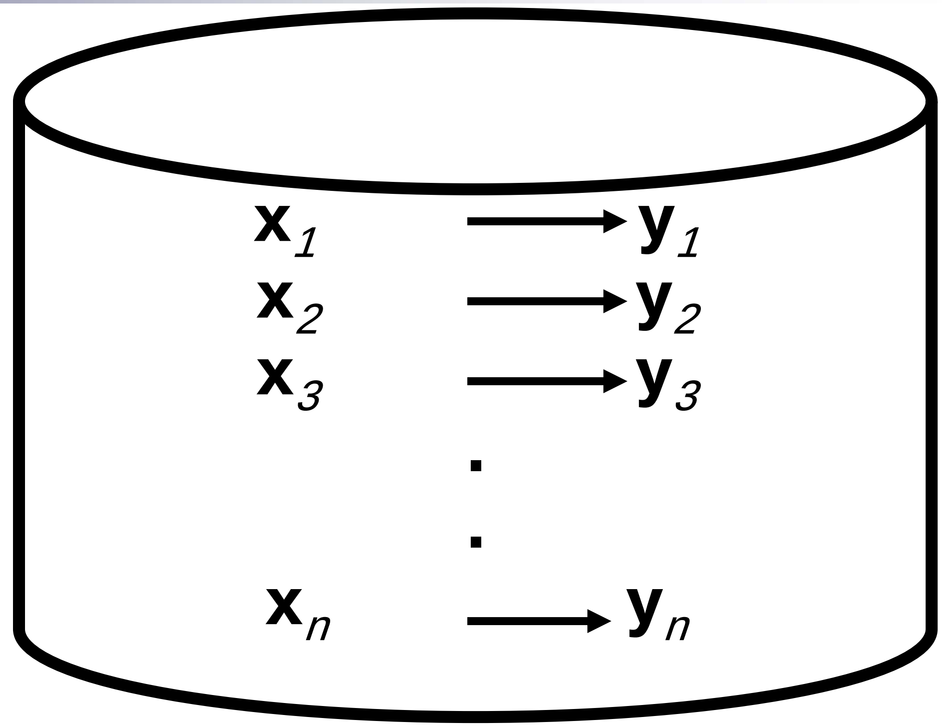
Here's a dataset with one input, one output and four datapoints.



1-Nearest Neighbor is an example of….
Instance-based learning

A function approximator that has been around since about 1910.

To make a prediction, search database for similar datapoints, and fit with the local points.



Four things make a memory based learner:

- A distance metric
- How many nearby neighbors to look at? ← multiple
- A weighting function (optional)
- How to fit with the local points?

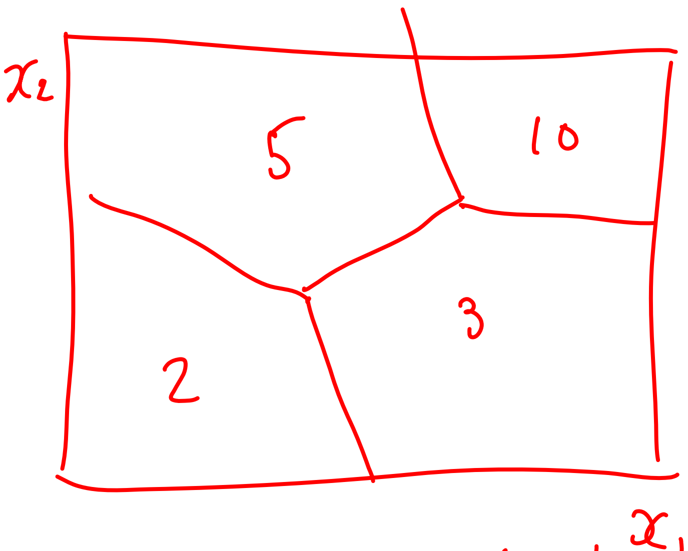
1- Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   One
3. *A weighting function (optional)*
   Unused
4. *How to fit with the local points?*
   Just predict the same output as the nearest neighbor.

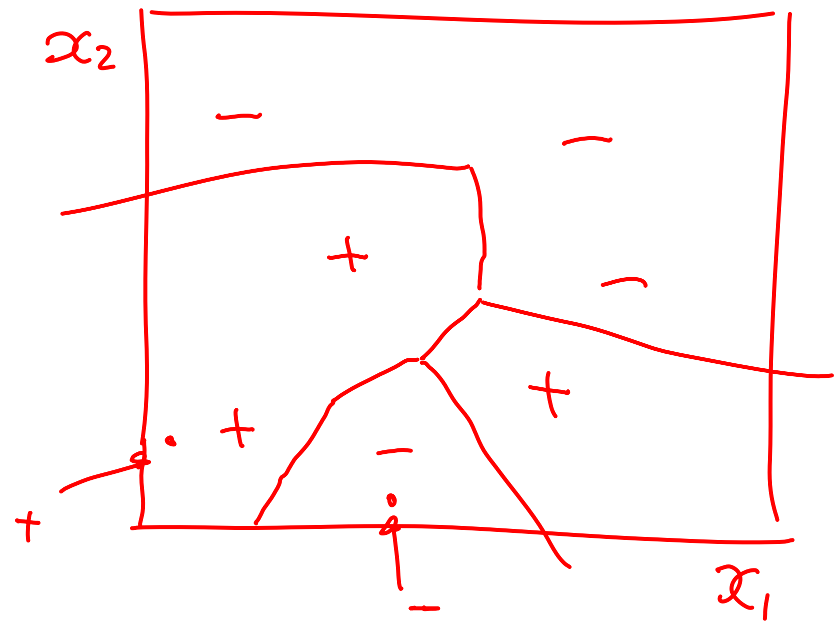
Multivariate 1-NN examples

Regression



piecewise constant function (predictor)

Classification



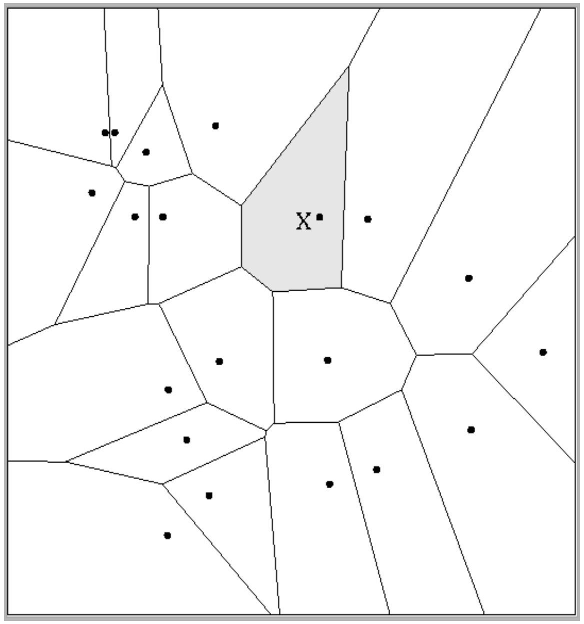
Voronoi diagram

Multivariate distance metrics

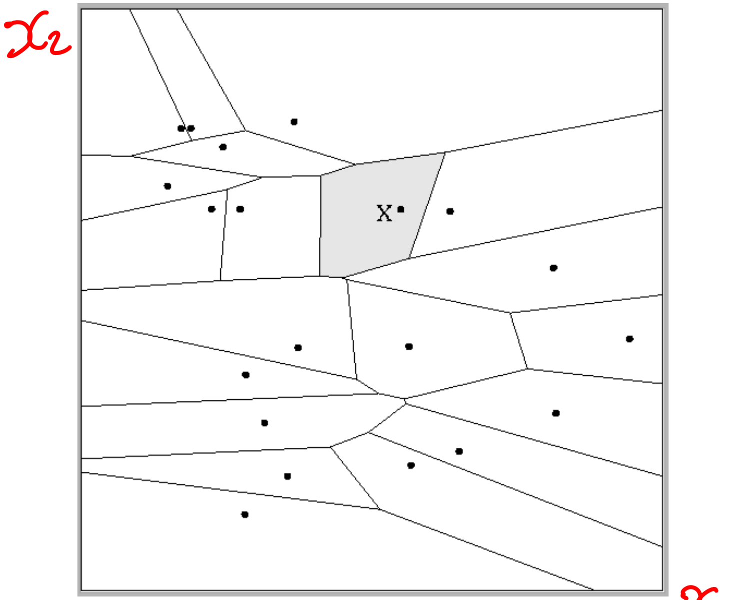
Suppose the input vectors x1, x2, …xn are two dimensional:

$\mathbf{x}_1 = (x_{11} , x_{12})$, $\mathbf{x}_2 = (x_{21} , x_{22})$, …$\mathbf{x}_N = (x_{N1} , x_{N2})$.

One can draw the nearest-neighbor regions in input space.



$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$



$Dist(\mathbf{x}_i,\mathbf{x}_j) =(x_{i1} - x_{j1})^2+(3x_{i2} - 3x_{j2})^2$

The relative scalings in the distance metric affect region shapes.

Euclidean distance metric

Or equivalently,

$$D(\mathrm{x},\mathrm{x}') = \sqrt{\sum_i \underbrace{\sigma_i^{\,2}}_{\text{weight}} \underbrace{\left(x_i - x'_i\right)^2}_{\text{squared coord. diff.}}}$$

where

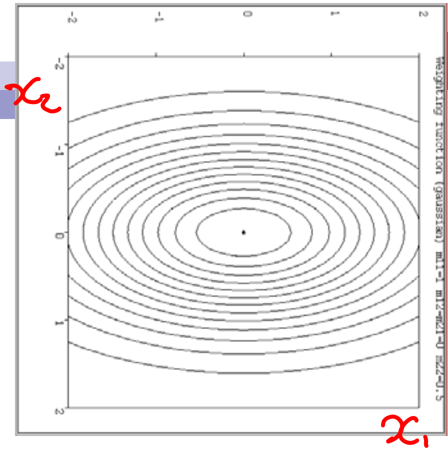
$$D(\mathrm{x},\mathrm{x}') = \sqrt{(\mathrm{x}-\mathrm{x}')^T \sum (\mathrm{x}-\mathrm{x}')}$$

$$\sum = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{bmatrix}$$

Other Metrics…

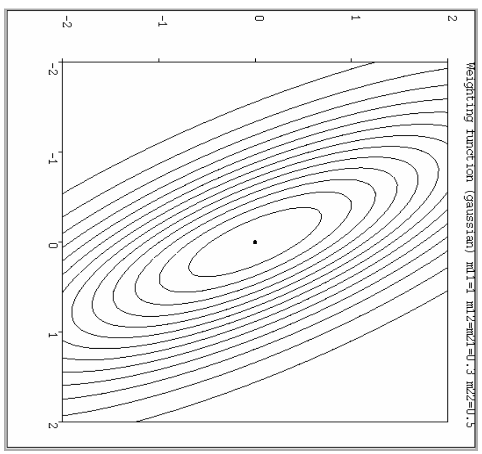
- Mahalanobis, Rank-based, Correlation-based,…

Notable distance metrics (and their level sets)



Scaled Euclidian (L_2)

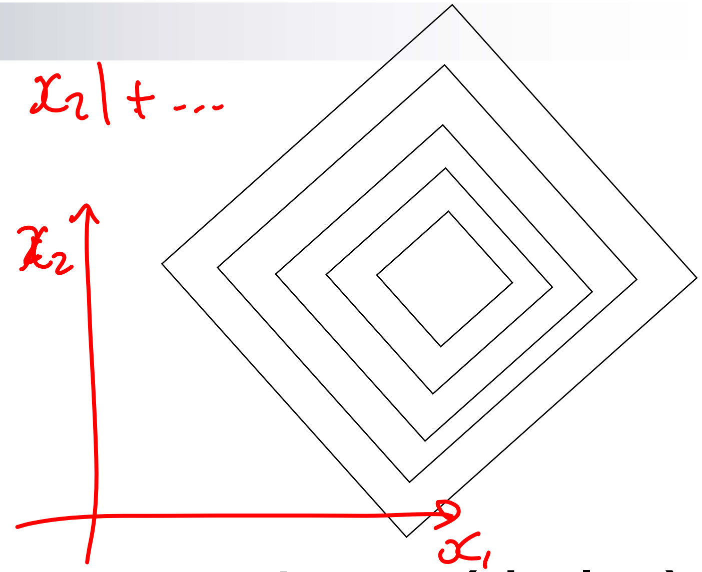
independent weights



Mahalanobis
(here, Σ on the previous slide is not necessarily diagonal, but is symmetric

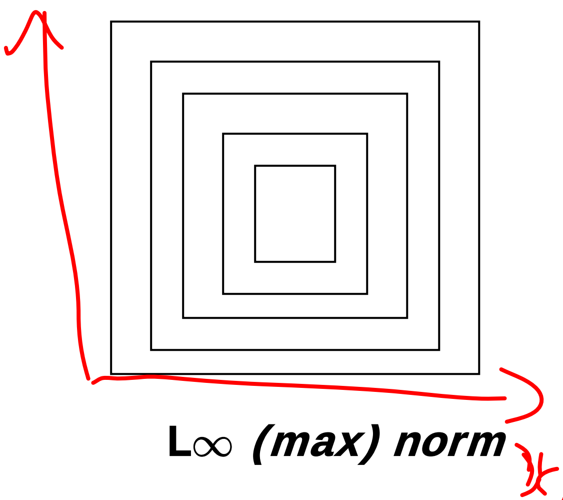
correlations between features

$|x_1| + |x_2| + \ldots$



L_1 norm (absolute)

$\max_i |x_i|$

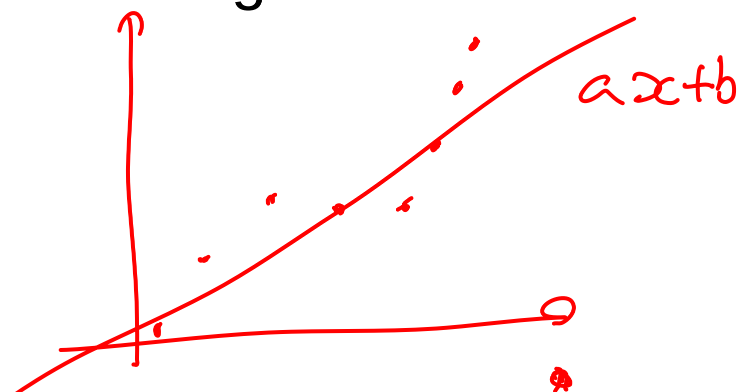


L_∞ (max) norm

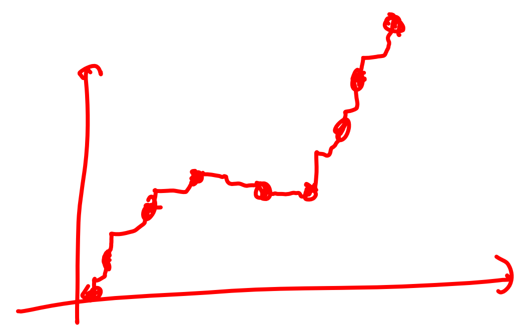
Consistency of 1-NN

- Consider an estimator f_n trained on *n* examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if prediction error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:

$$\lim_{n \to \infty} MSE(f_n) = 0$$

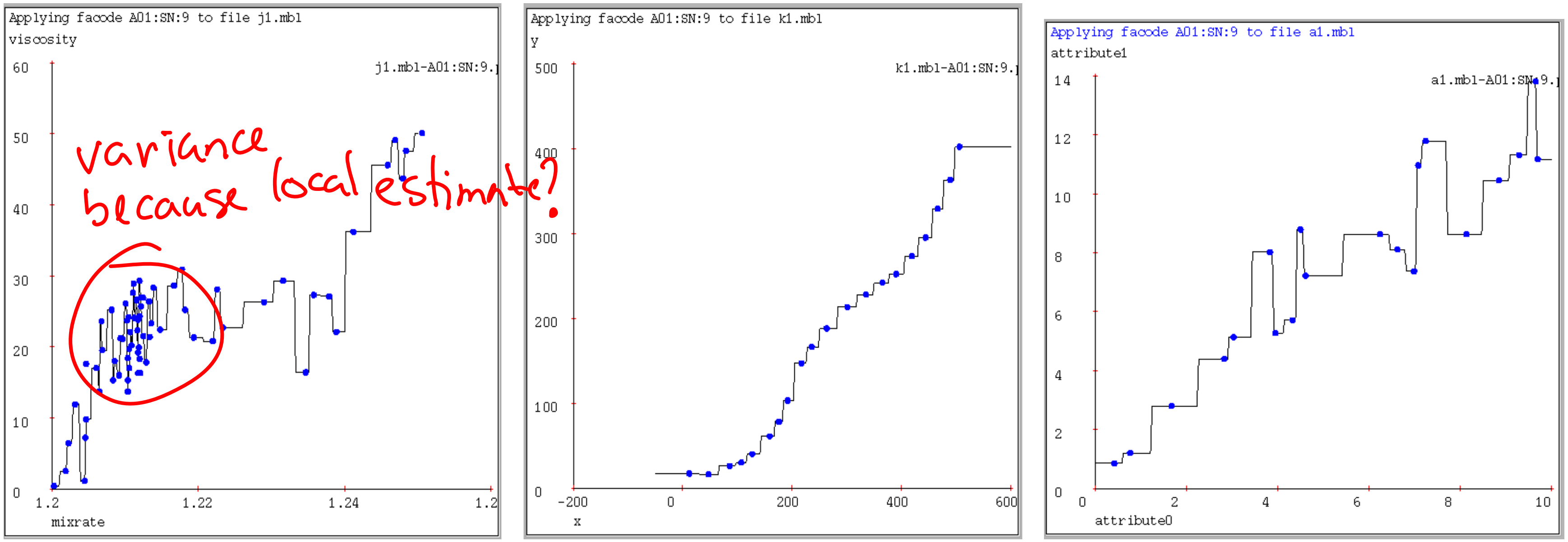


- Regression is not consistent!
  - Representation bias
- **1-NN is consistent** (under some mild fineprint)



What about variance???

1-NN overfits?



fit the noise

In some cases, 1-NN need exponentially large dataset.

k-Nearest Neighbor

Four things make a memory based learner:

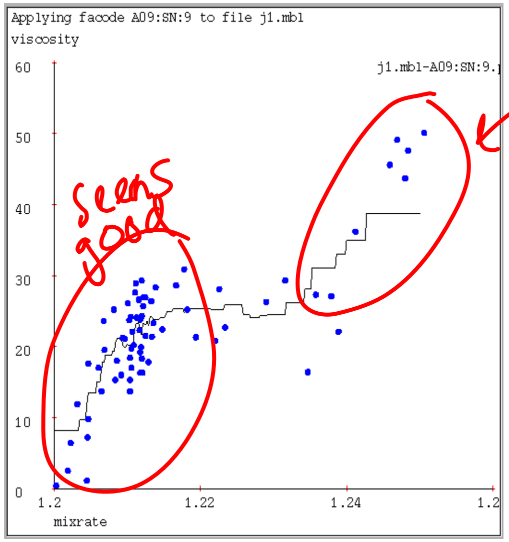
1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   k
1. *A weighting function (optional)*
   Unused

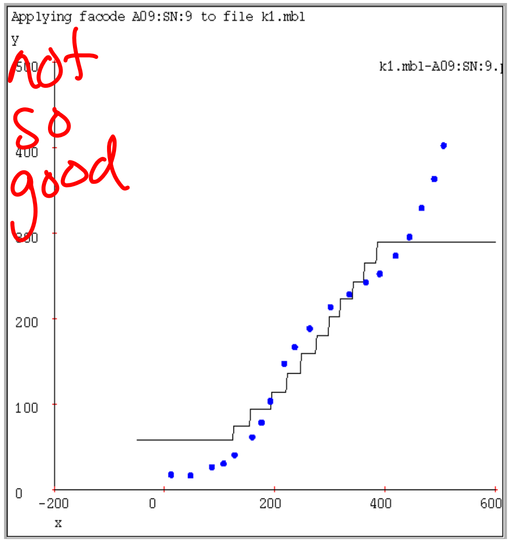
2. *How to fit with the local points?*
   Just predict the average output among the k nearest neighbors.

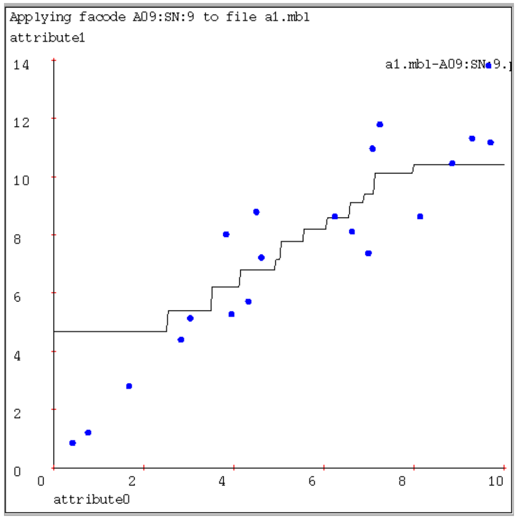
query x_q , $KNN(x_q)$:

$$\hat{y} = \frac{\sum_{i \in KNN(x_q)} y_i}{k}$$

k-Nearest Neighbor (here k=9)





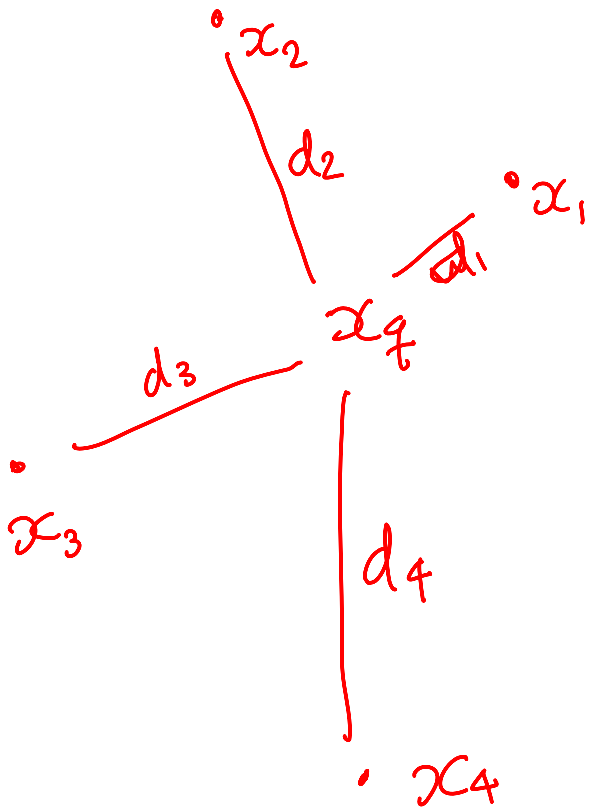


K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

What can we do about all the discontinuities that k-NN gives us?

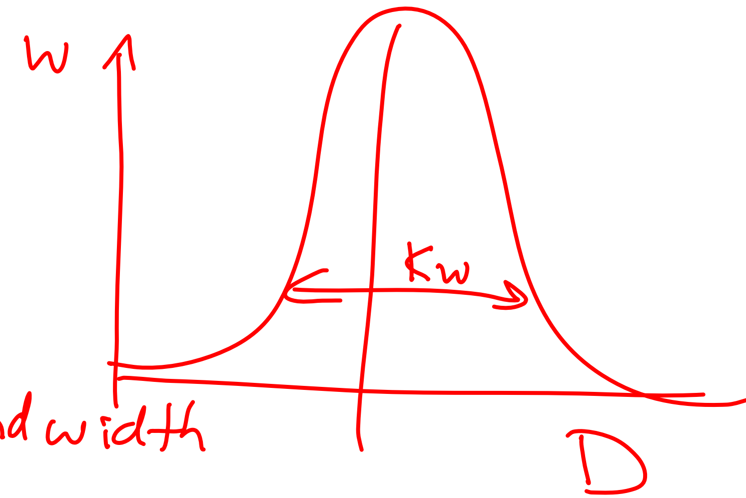
Weighted k-NNs

- Neighbors are not all the same



$$\hat{y}_q = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

Kernel regression



Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   $\boldsymbol{w_i = exp(-D(x_i, query)^2 / K_w^2)}$

   bandwidth

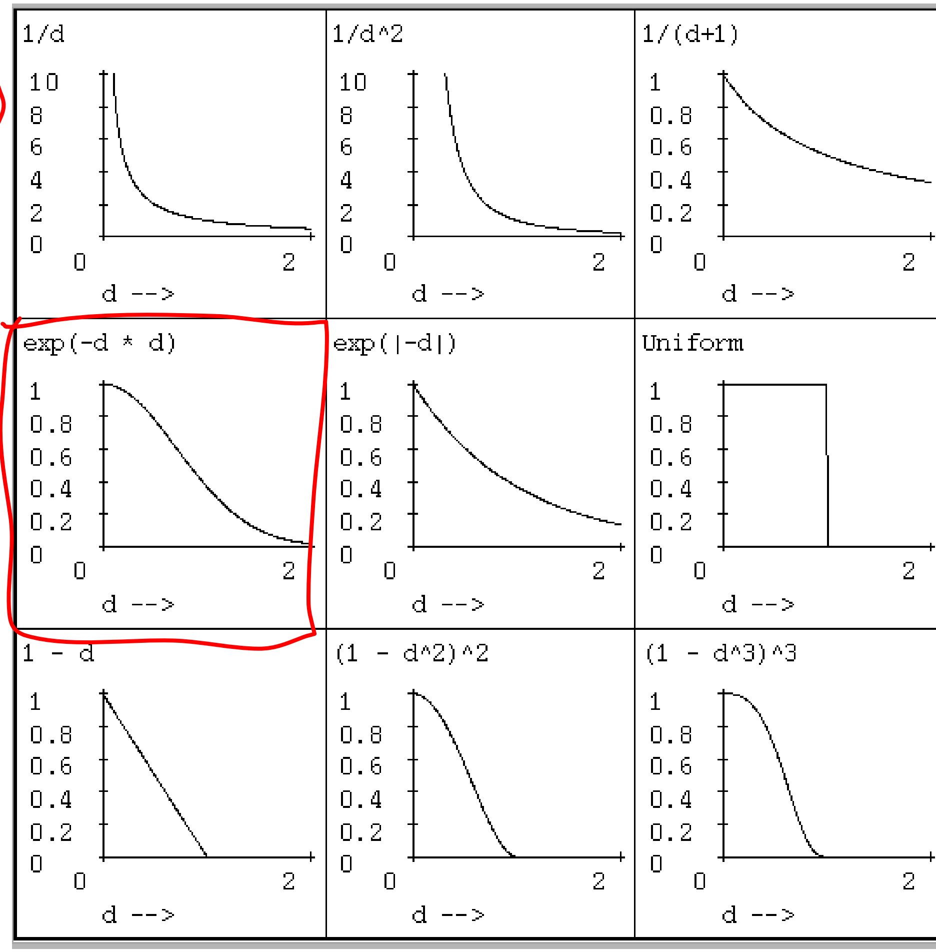
   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.
4. *How to fit with the local points?*
   Predict the weighted average of the outputs:
   $\boldsymbol{predict = \Sigma w_i y_i / \Sigma w_i}$

Weighting functions

often, behave similarly in practice

$w_i = exp(-D(x_i, query)^2 / K_w^2)$

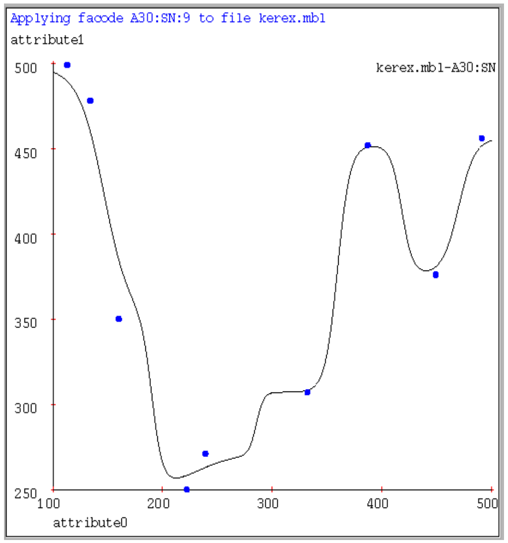
$w_i = \frac{1}{d(x_i, x_j)}$



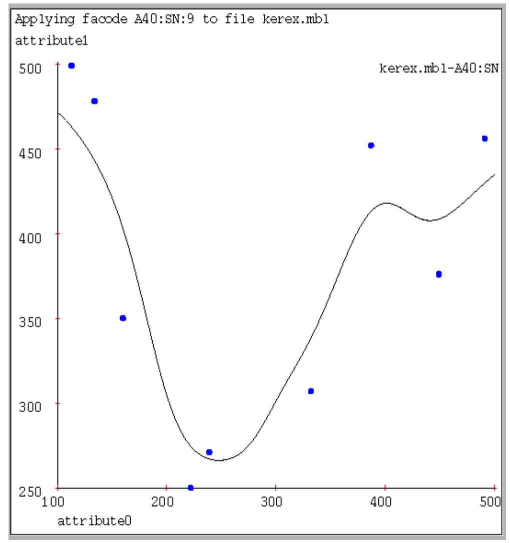
(Our examples use Gaussian)

Typically optimize K_w using gradient descent

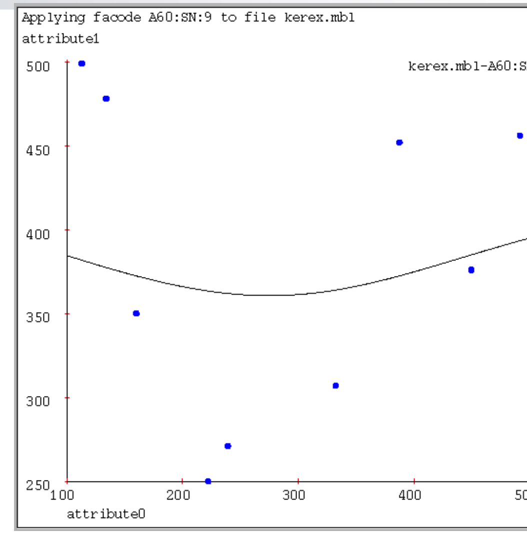
Kernel regression predictions



K_W=10



K_W=20



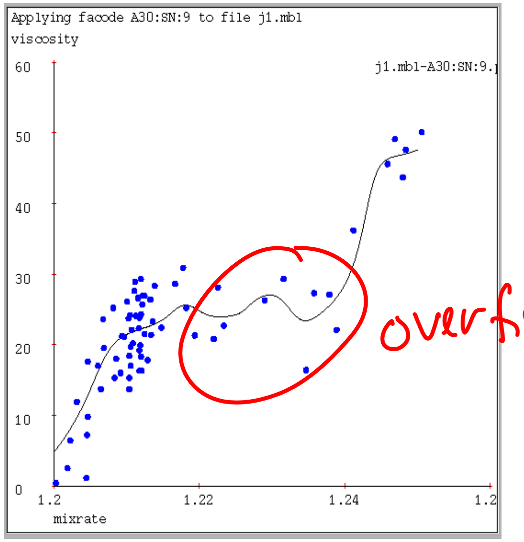
K_W=80

more bias

Increasing the kernel width K_w means further away points get an opportunity to influence you.

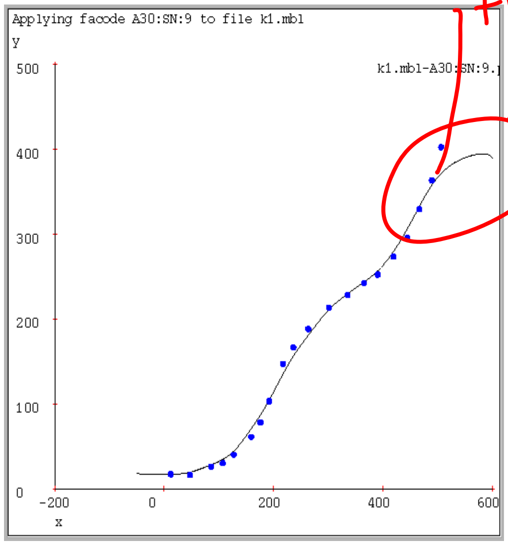
As $K_w \rightarrow \infty$, the prediction tends to the global average.

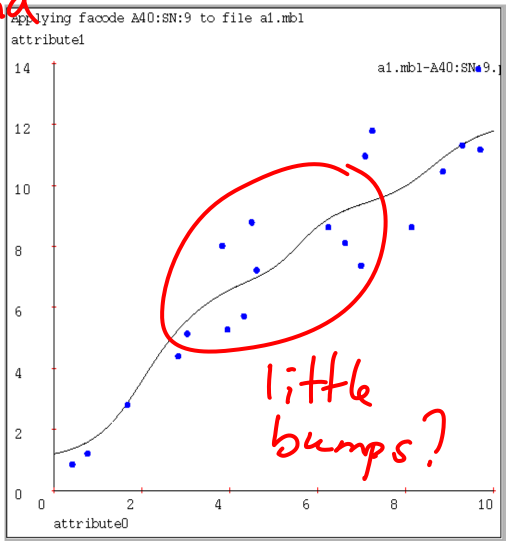
Kernel regression on our test cases



overfit?

trend





KW=1/32 of x-axis width.

looks pretty

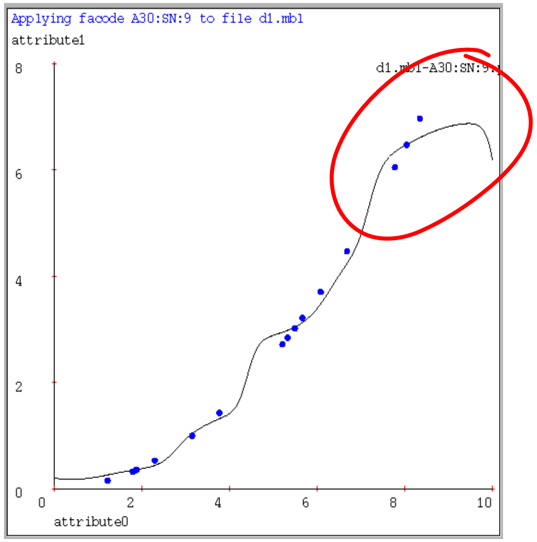
KW=1/32 of x-axis width.

KW=1/16 axis width.

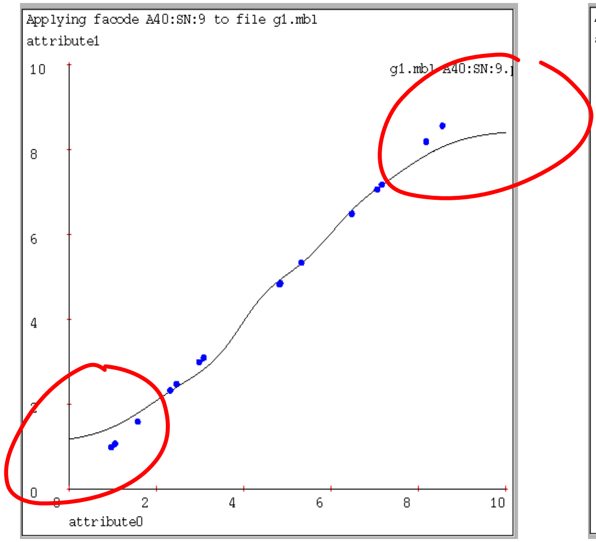
really good

Choosing a good K_w is important. Not just for Kernel Regression, but for all the locally weighted learners we're about to see.

Kernel regression can look bad

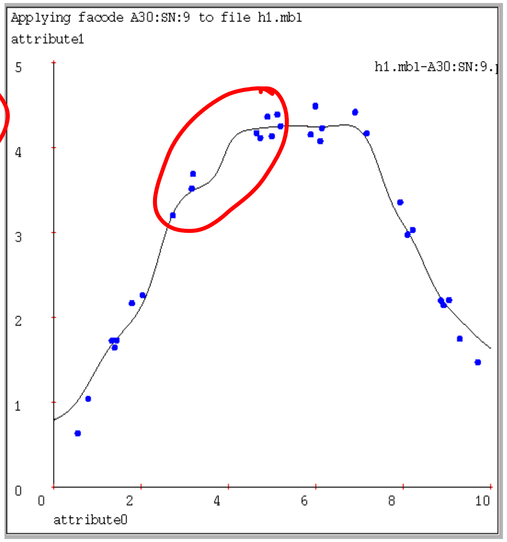


KW = Best.



KW = Best.

ax+b
better



KW = Best.

Time to try something more powerful…

Locally weighted regression

$$\hat{y} = \frac{\sum w_i y_i}{\sum w_i}$$

Kernel regression:

Take a very very conservative function approximator called AVERAGING. Locally weight it.

Locally weighted regression:

Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

Locally weighted regression

- **Four things make a memory based learner:**
- *A distance metric*
  - **Any**
- *How many nearby neighbors to look at?*
  - **All of them**
- *A weighting function (optional)*
  - **Kernels**
    - ***wi = exp(-D(xi, query)2 / Kw2)***
- *How to fit with the local points?*

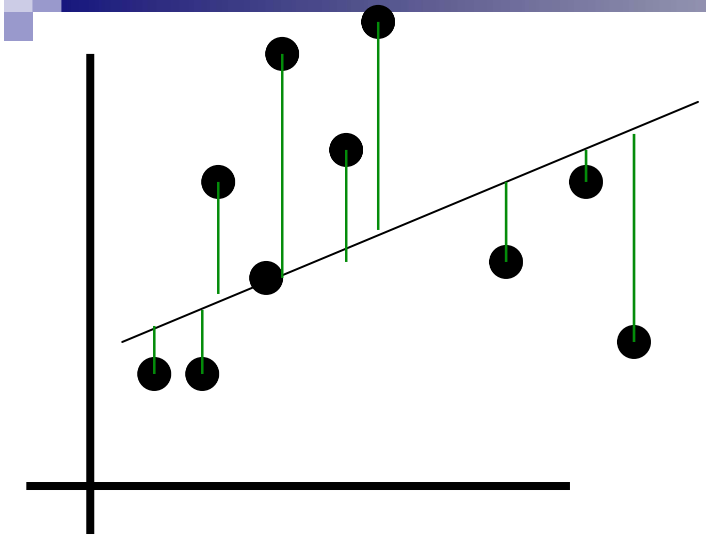
General weighted regression:

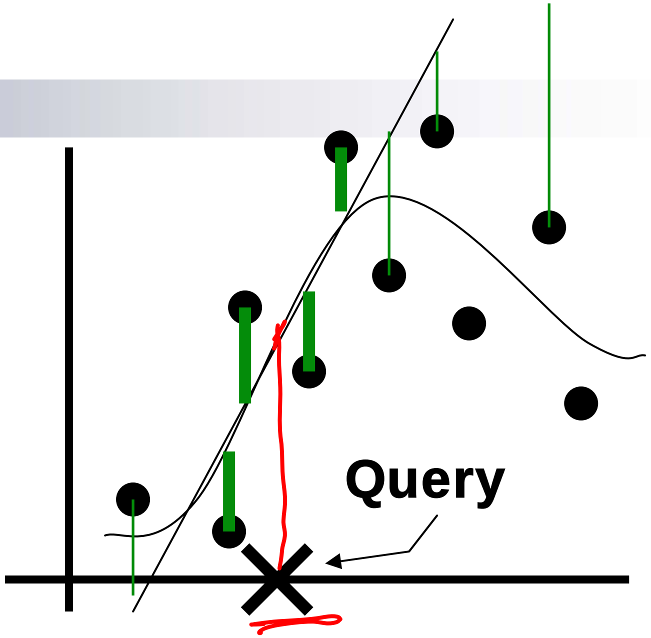
$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{k=1}^{N} w_k^{\,2} \left(y_k - \beta^T x_k \right)^2$$

for input x_q

Kernel wrt x_q

How LWR works





Linear regression

- Same parameters for all queries

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y$$

very similar

wi by kernels

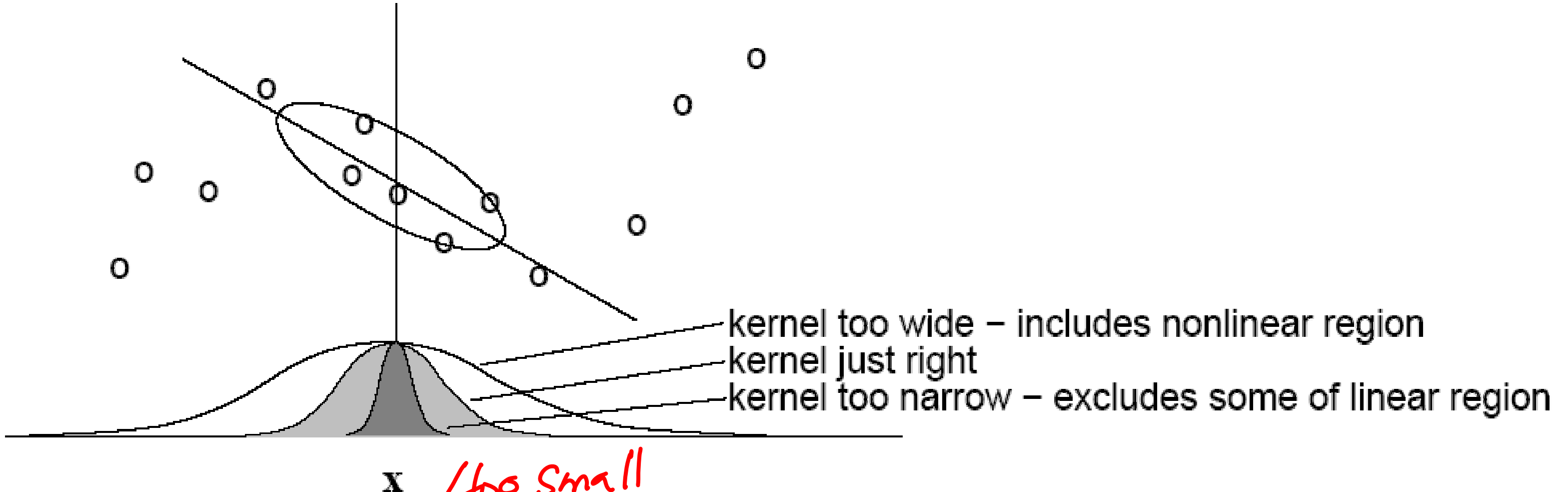
Locally weighted regression

- Solve weighted linear regression for each query

$$\hat{\beta} = \left(WX^T WX\right)^{-1} WX^T WY$$

$$W = \begin{pmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_n \end{pmatrix}$$

Another view of LWR



too small:

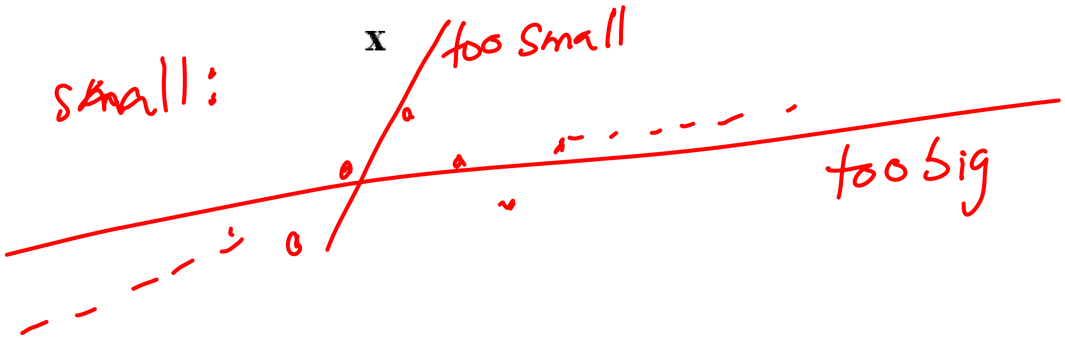
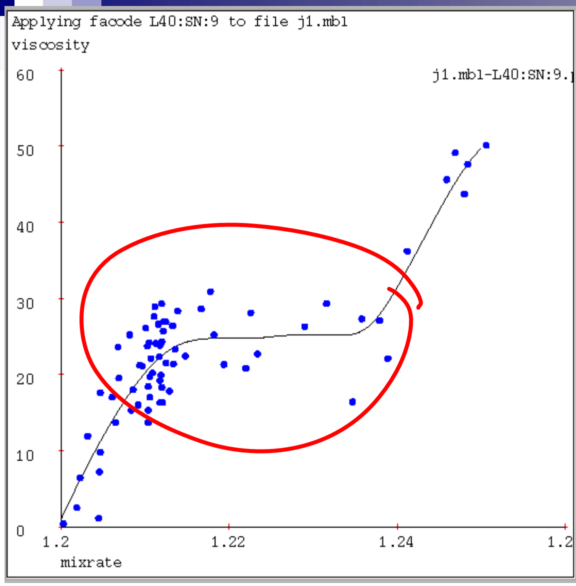
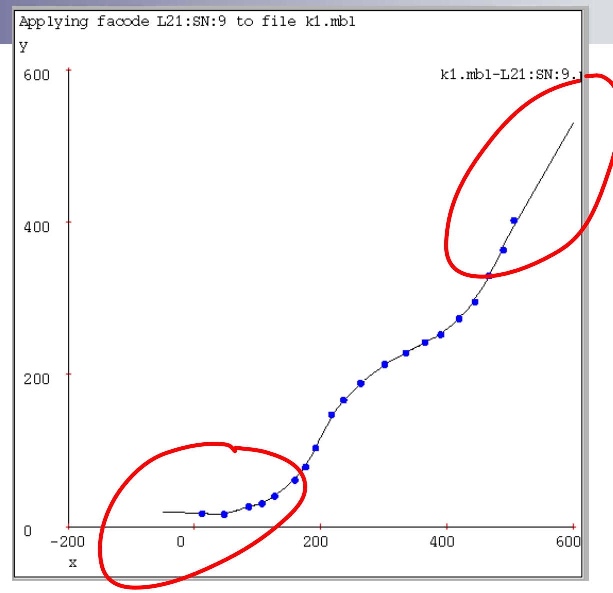


Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996) "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

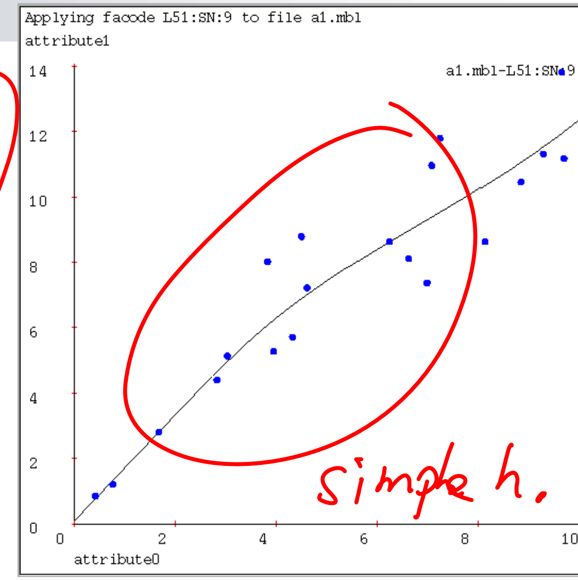
LWR on our test cases



KW = 1/16 of x-axis width.



KW = 1/32 of x-axis width.



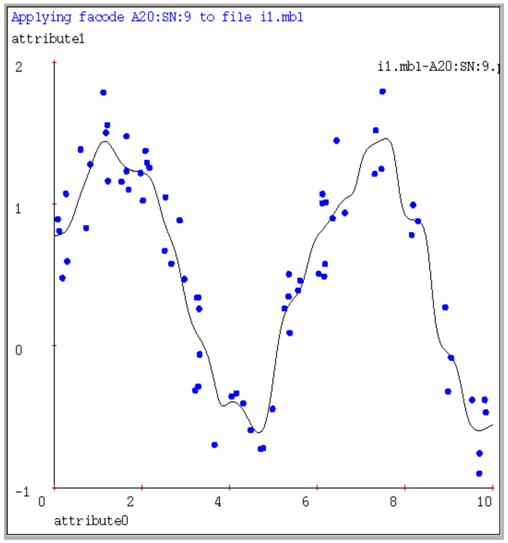
KW = 1/8 of x-axis width.

gaussian kernels

ax+b regression

Locally weighted polynomial regression

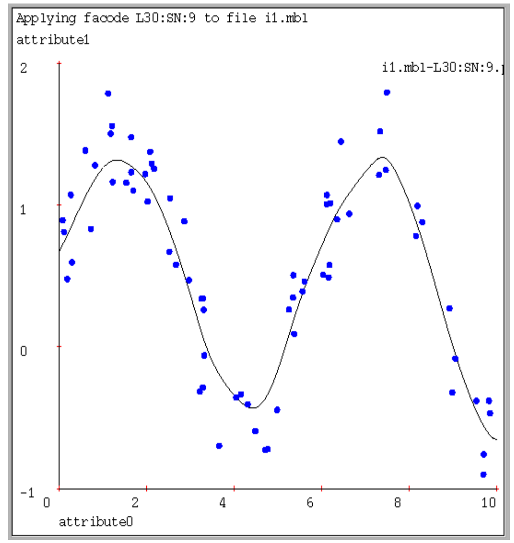
a



Kernel Regression
Kernel width K_W at optimal level.

KW = 1/100 x-axis

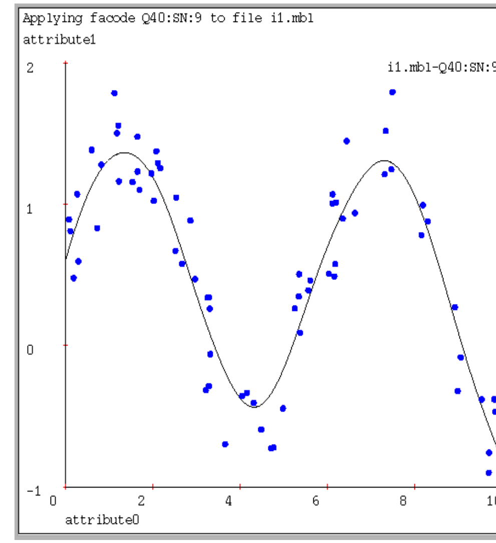
$a+bx$



LW Linear Regression
Kernel width K_W at optimal level.

KW = 1/40 x-axis

$a+bx+cx^2$



LW Quadratic Regression
Kernel width K_W at optimal level.

KW = 1/15 x-axis

increase bandwidth

Local quadratic regression is easy: just add quadratic terms to the WXTWX matrix. As the regression degree increases, the kernel width can increase without introducing bias.

Curse of dimensionality for instance-based learning

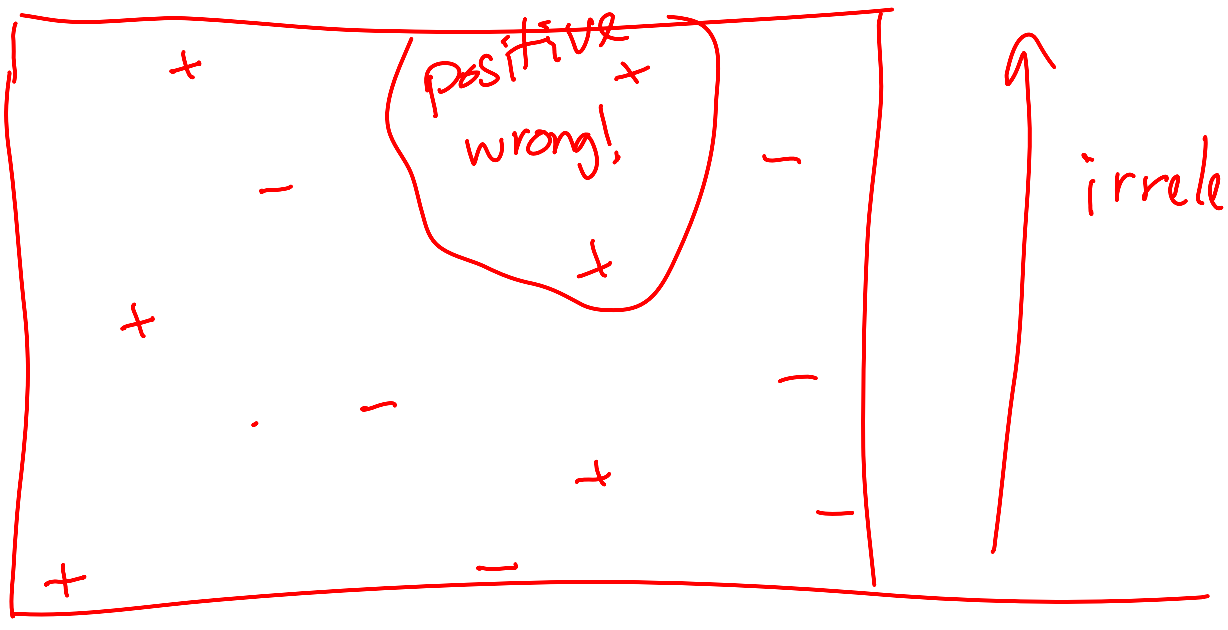
practical problem

- Must store and retreve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - We'll see fast methods for dealing with large datasets
- Instance-based learning often poor with noisy or irrelevant features

Curse of the irrelevant feature

1D + + + - - - - - + + + + - - - - NN well

add 1 irrelevand d



What you need to know

- k-NN
  - Simplest learning algorithm
  - With sufficient data, very hard to beat “strawman” approach
  - Picking k?
- Kernel regression
  - Set k to n (number of data points) and optimize weights by gradient descent
  - Smoother than k-NN
- Locally weighted regression
  - Generalizes kernel regression, not just local average
- Curse of dimensionality
  - Tackling large datasets
  - Irrelevant features often killers for instance-based approaches

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials