Instance-based Learning

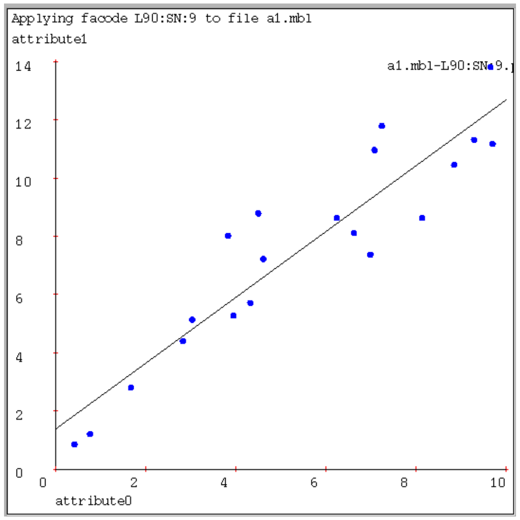
Machine Learning – 10701/15781

Carlos Guestrin

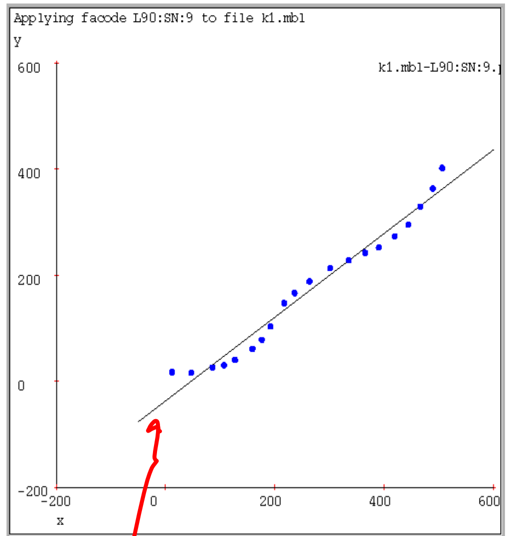
Carnegie Mellon University

February 19th, 2007

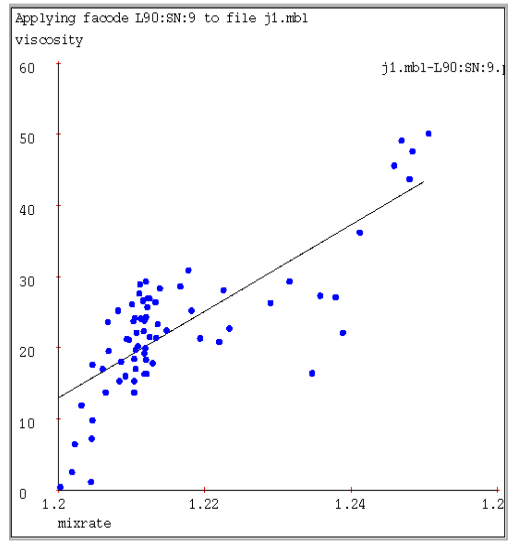
Why not just use Linear Regression?



looks OK?



missing trend



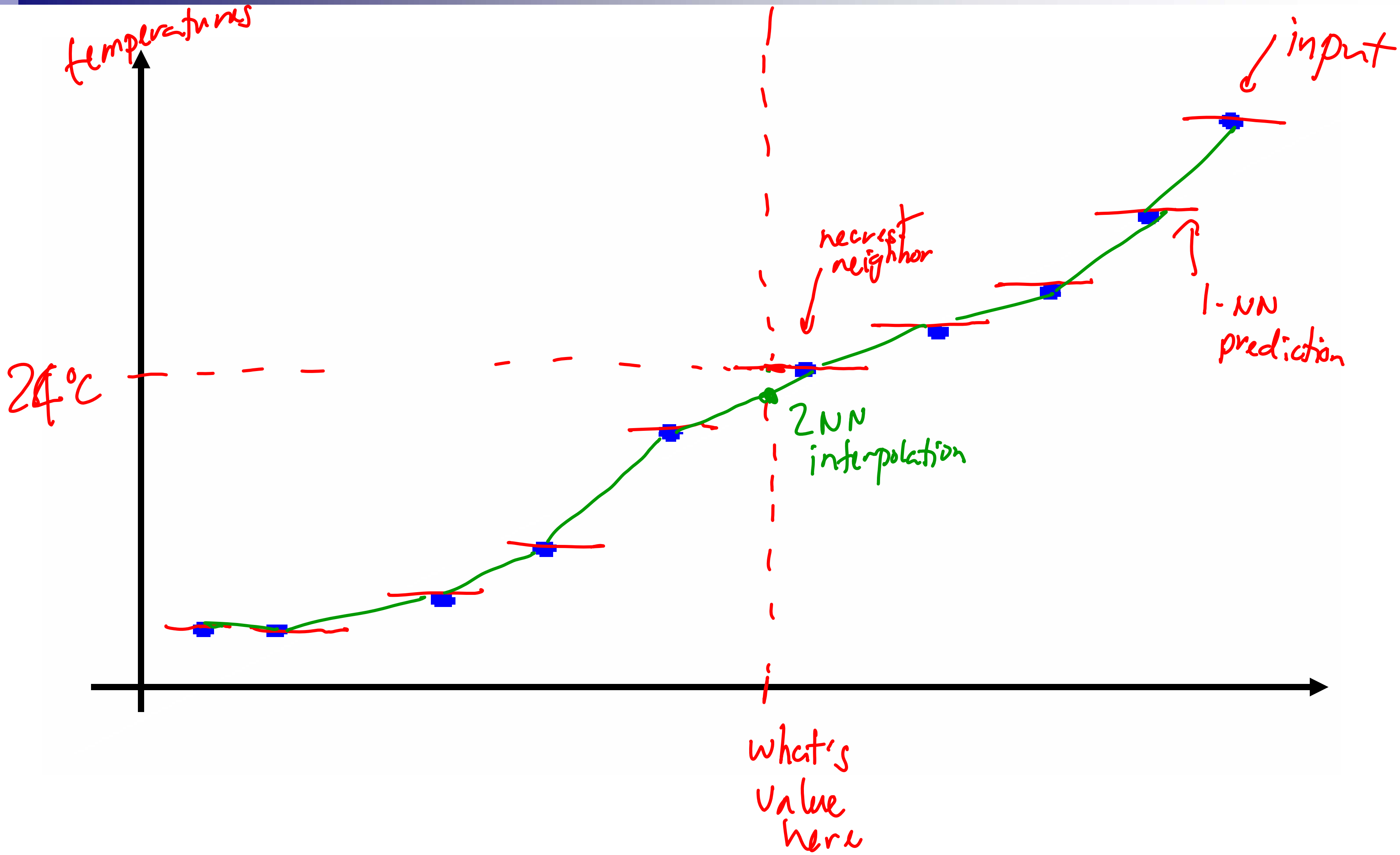
not doing so well...

add more basis fns.

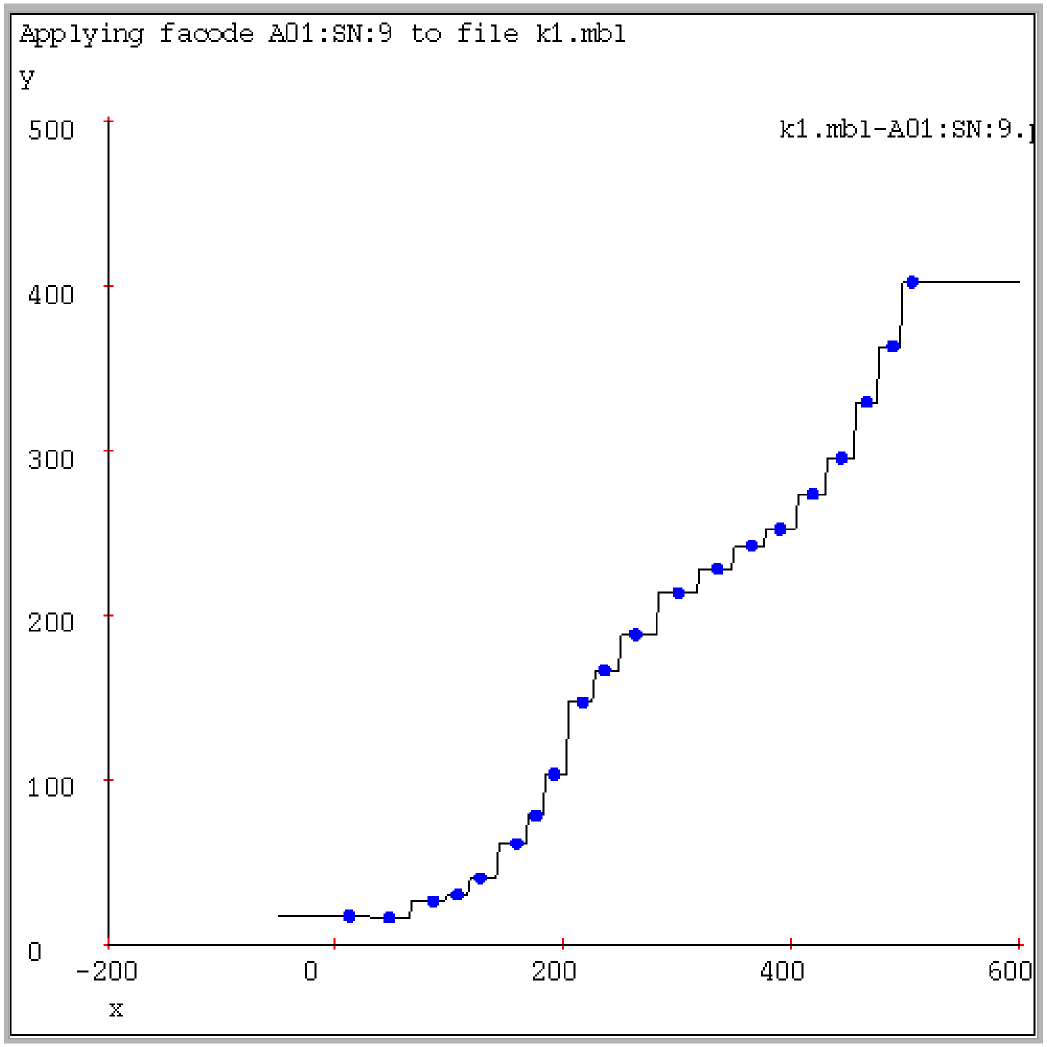
use some kind of non-linear regression

see later in semester

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 $y_i=f(x_i)$ for some unknown function f.

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

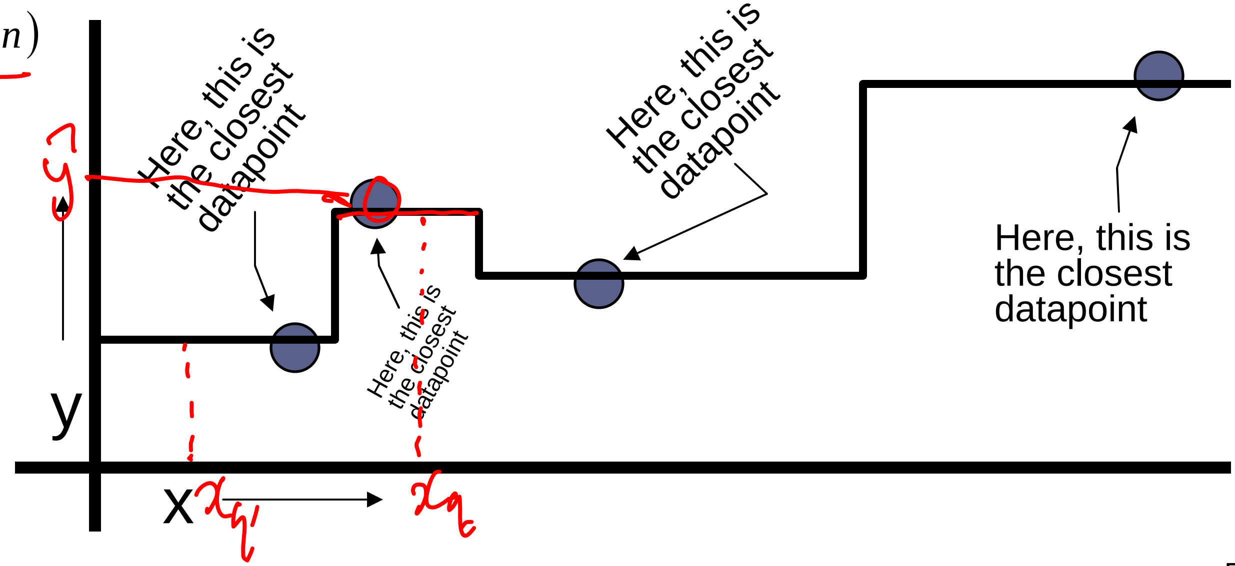
Nearest Neighbor:

1. Find the closest x_i in our set of datapoints

$$i(nn) = \underset{i \in \{1,\ldots,N\}}{\operatorname{argmin}} \left| x_i - x_q \right|$$

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

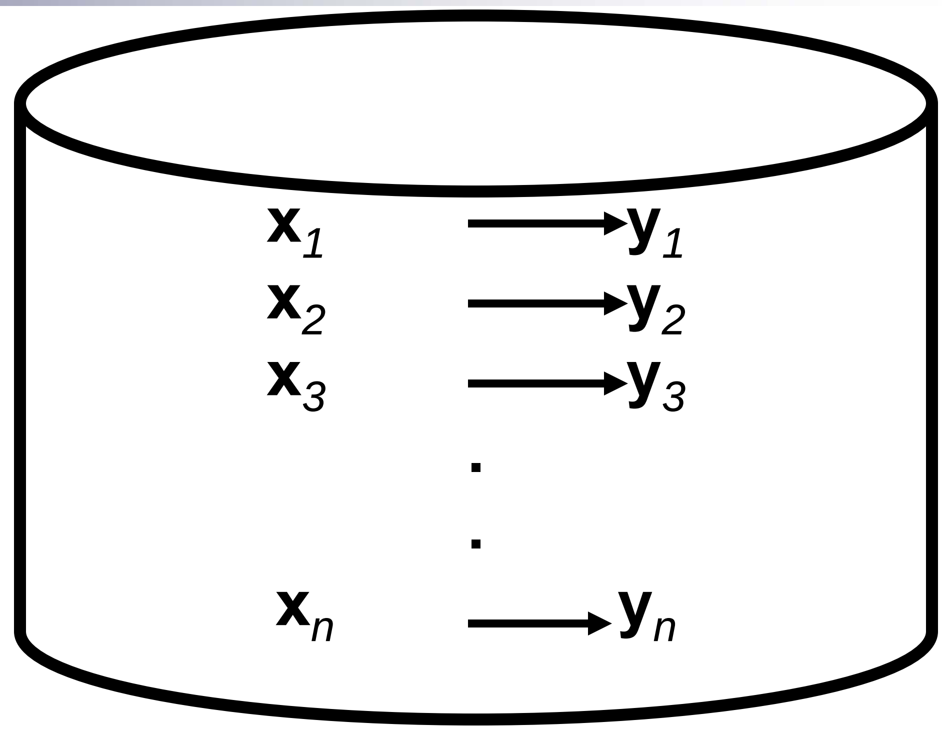
Here's a dataset with one input, one output and four datapoints.



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?
- A weighting function (optional)
- How to fit with the local points?

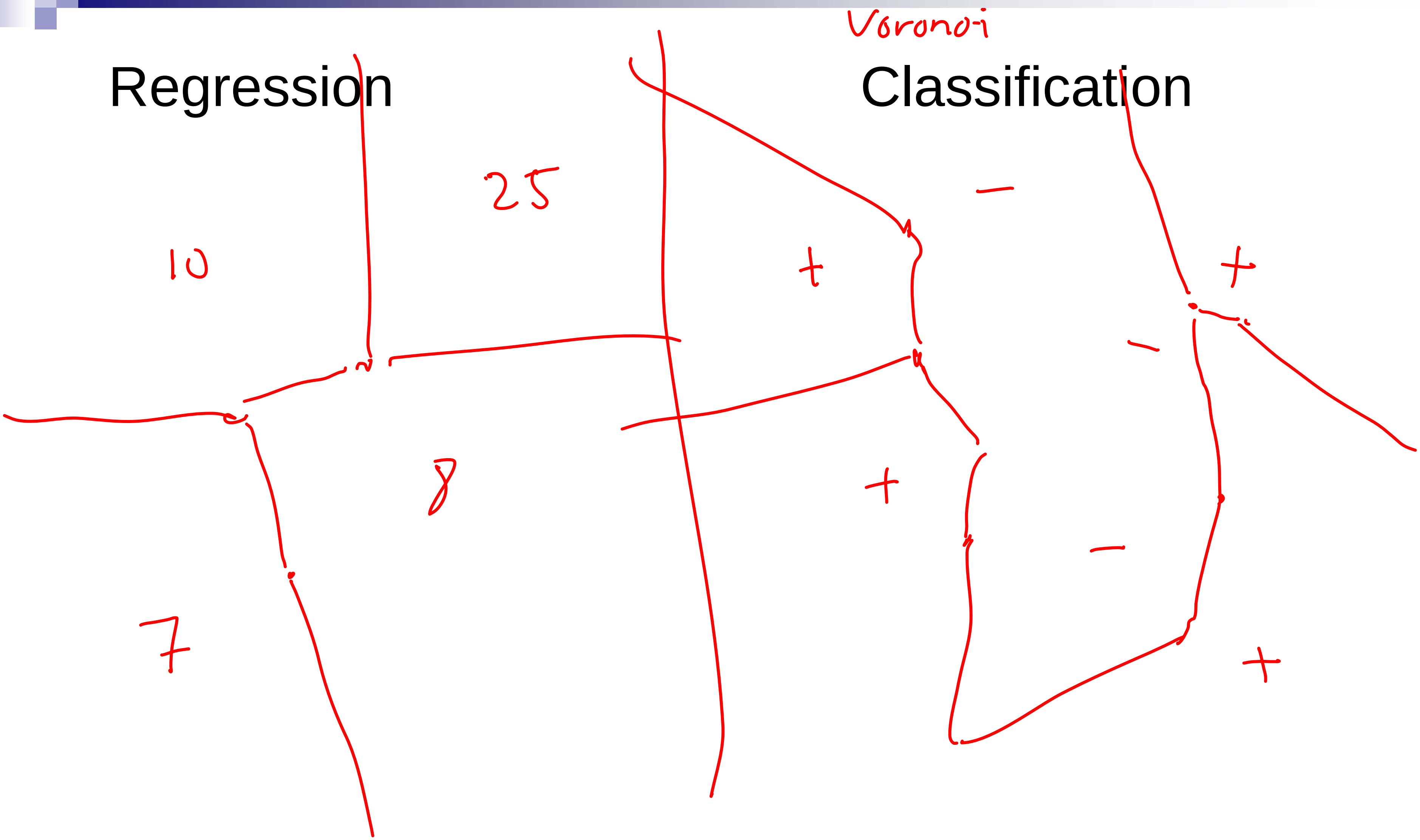
1-Nearest Neighbor

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 | Classification

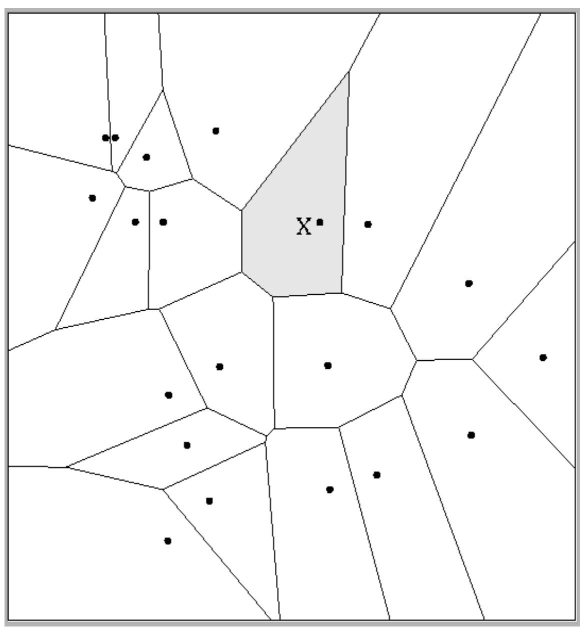


Multivariate distance metrics

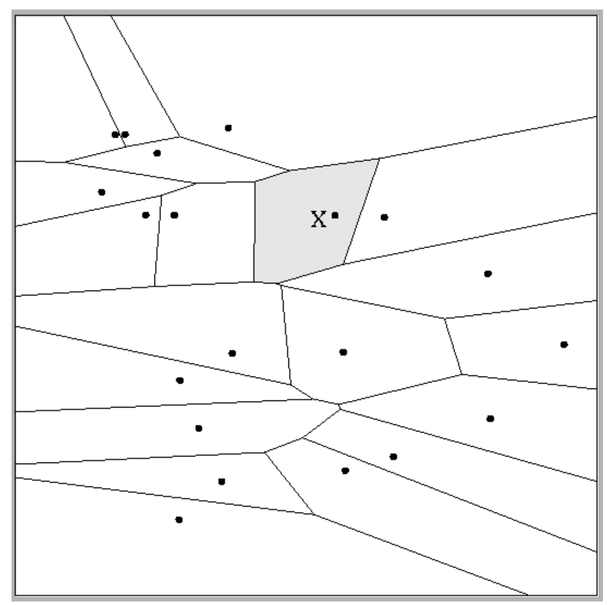
Suppose the input vectors x_1, x_2, …x_n are two dimensional:

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

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



x_2



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

$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (3x_{i2} - 3x_{j2})^2$ x_1

The relative scalings in the distance metric affect region shapes

Euclidean distance metric

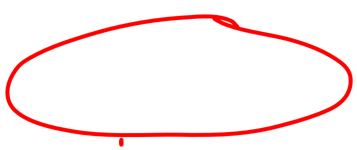
Or equivalently, $$D(\mathrm{x},\mathrm{x}') = \sqrt{\sum_i \sigma_i^{\,2}(x_i - x'_i)^2}$$

if
$\sigma_1^2 = 1$
$\sigma_2^2 = 1$

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

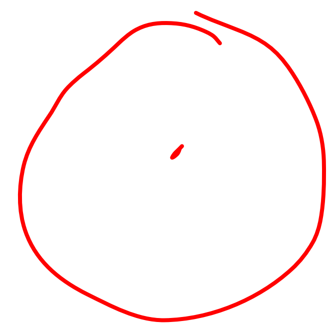
where

$\sigma_1^2 = 1$
$\sigma_2^2 = 3$



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

equidistant:

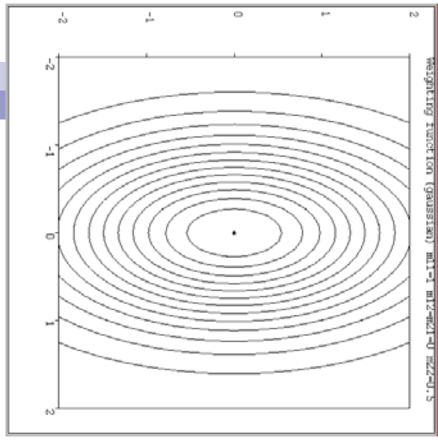


Other Metrics…

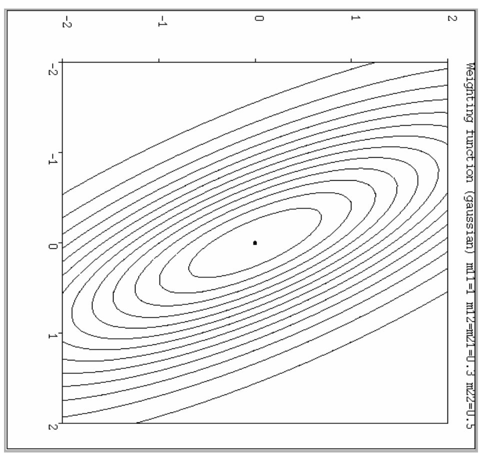
- Mahalanobis, Rank-based, Correlation-based,…

Notable distance metrics (and their level sets)

equidistant:

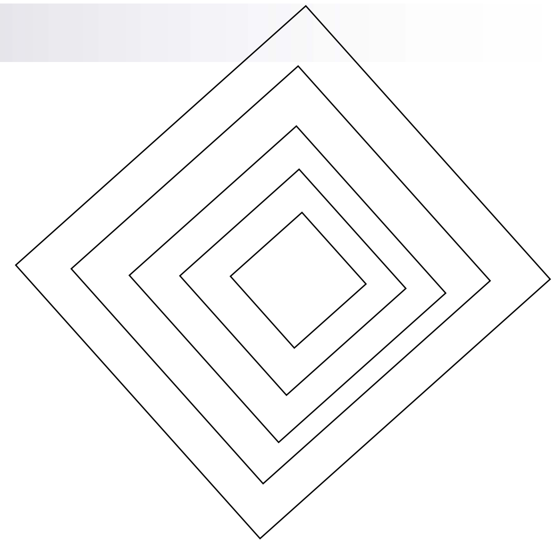


Scaled Euclidian (L_2)



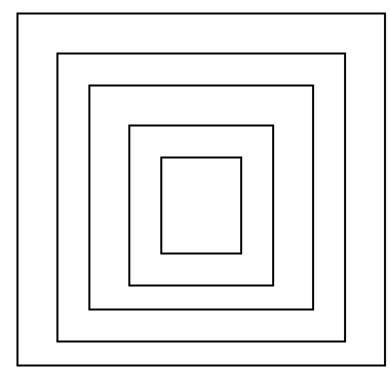
Mahalanobis (here, Σ on the previous slide is not necessarily diagonal, but is symmetric

$\|x_1 - x_2\|_1$
$= \sum_i |x_1^i - x_2^i|$



L_1 norm (absolute)

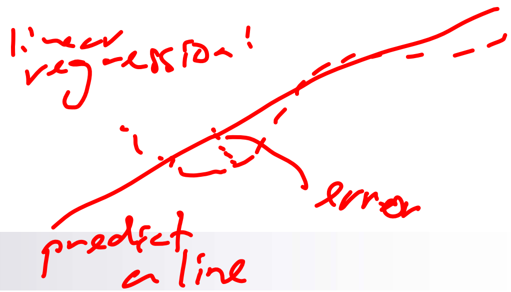
$\|x_1 - x_2\|_\infty$
$= \max_i |x_1^i - x_2^i|$



L_∞ (max) norm

with fixed # basis fns.

Consistency of 1-NN



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

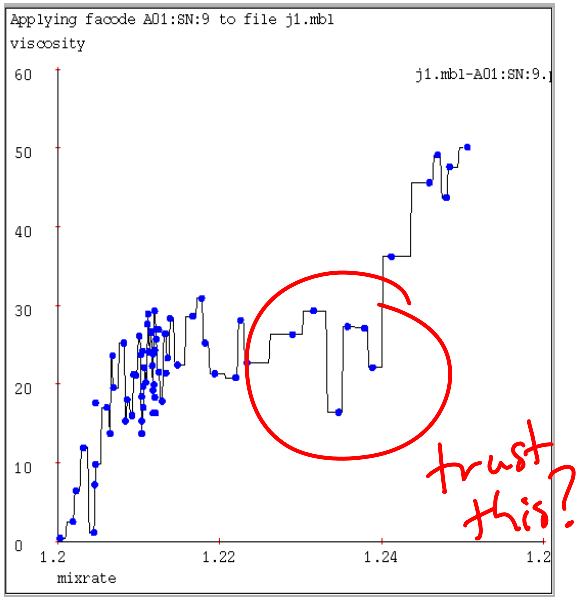
$$\lim_{n \to \infty} MSE(f_n) = 0$$

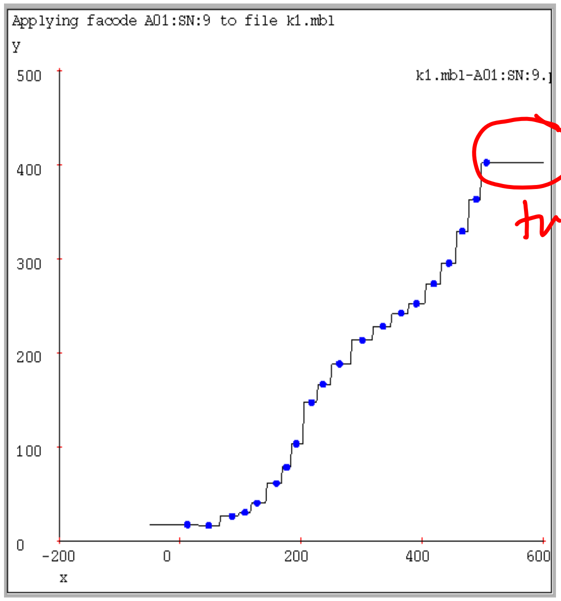
mean squared error

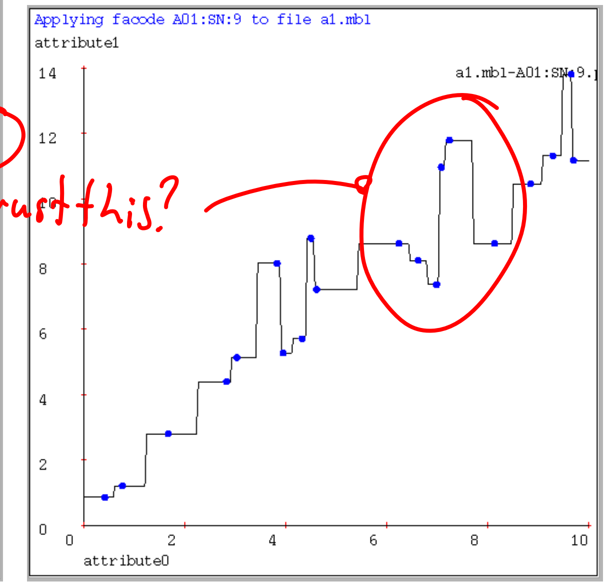
- Regression is not consistent!
  - Representation bias
- **1-NN is consistent** (under some mild fineprint)

What about variance???

1-NN overfits?







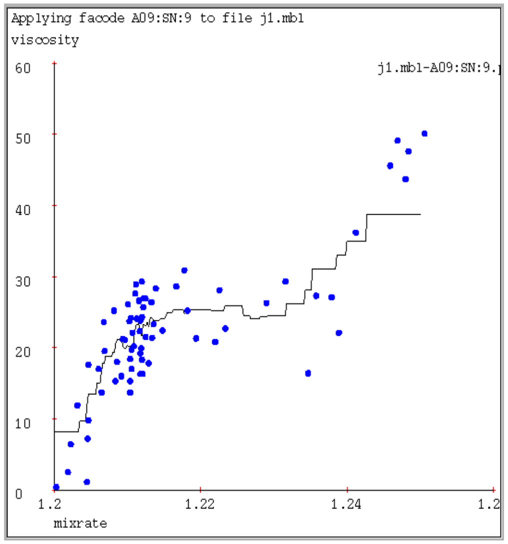
k-Nearest Neighbor

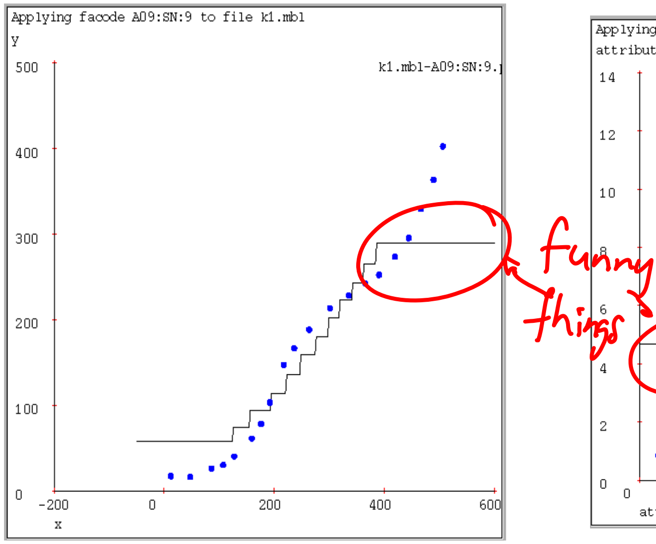
Four things make a memory based learner:

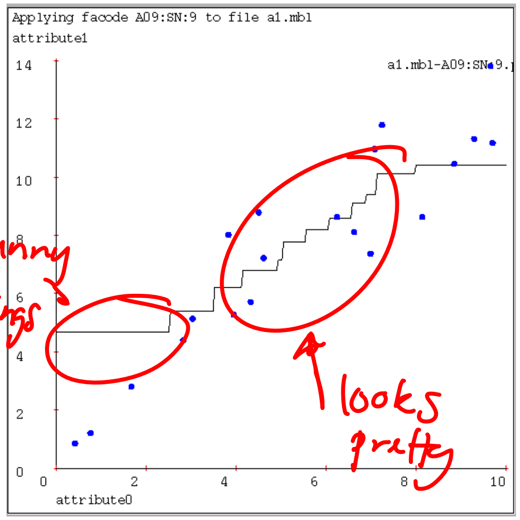
1. *A distance metric*
   - **Euclidian (and many more)**
2. *How many nearby neighbors to look at?*
   - **k**
1. *A weighting function (optional)*
   - **Unused**
2. *How to fit with the local points?*
   - **Just predict the average output among the k nearest neighbors.**

$$\hat{y} = \sum_{j \in KNN(x_q)} y^j$$

k-Nearest Neighbor (here k=9)







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

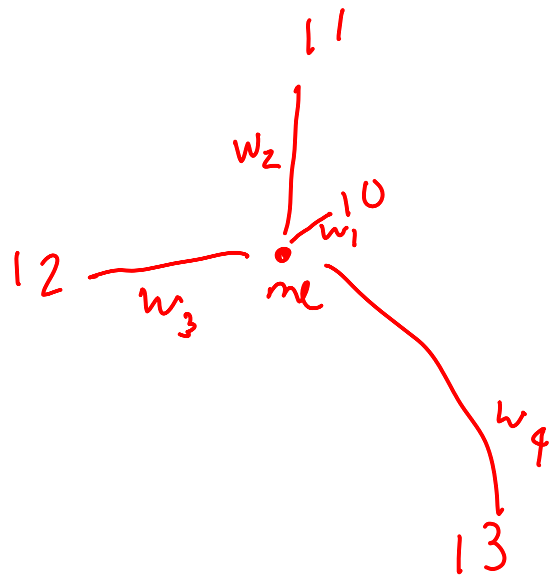
What can we do about all the discontinuities that k-NN gives us?

Weighted k-NNs

- Neighbors are not all the same

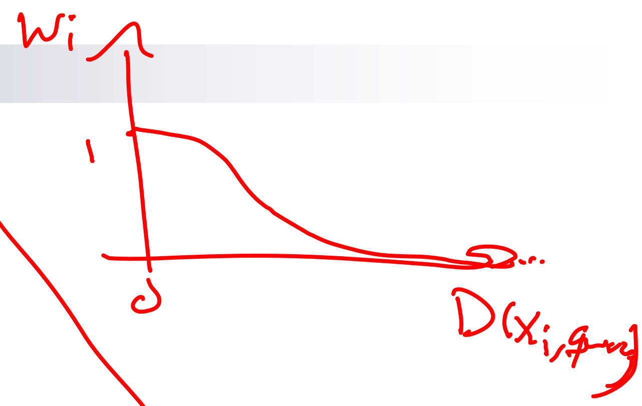
$w_1 + w_2 + w_3 + w_4 = 1$

$$\hat{y}_{me} = \frac{\sum_i w_i y_i}{\sum_i w_i}$$



e.g.: $w(x_1, x_2) = \dfrac{1}{D(x_1, x_2)}$

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)* one example:

   $\boldsymbol{w_i = exp(-D(x_i, query)^2 / K_w^2)}$

   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.

   wi not a pdf...

4. *How to fit with the local points?*

   Predict the weighted average of the outputs:

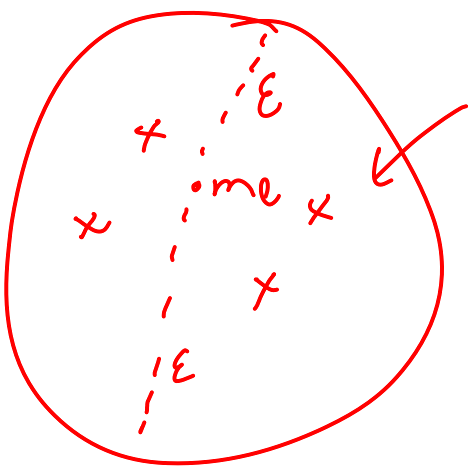
   predict = $\boldsymbol{\Sigma w_i y_i / \Sigma w_i}$

   normalizer

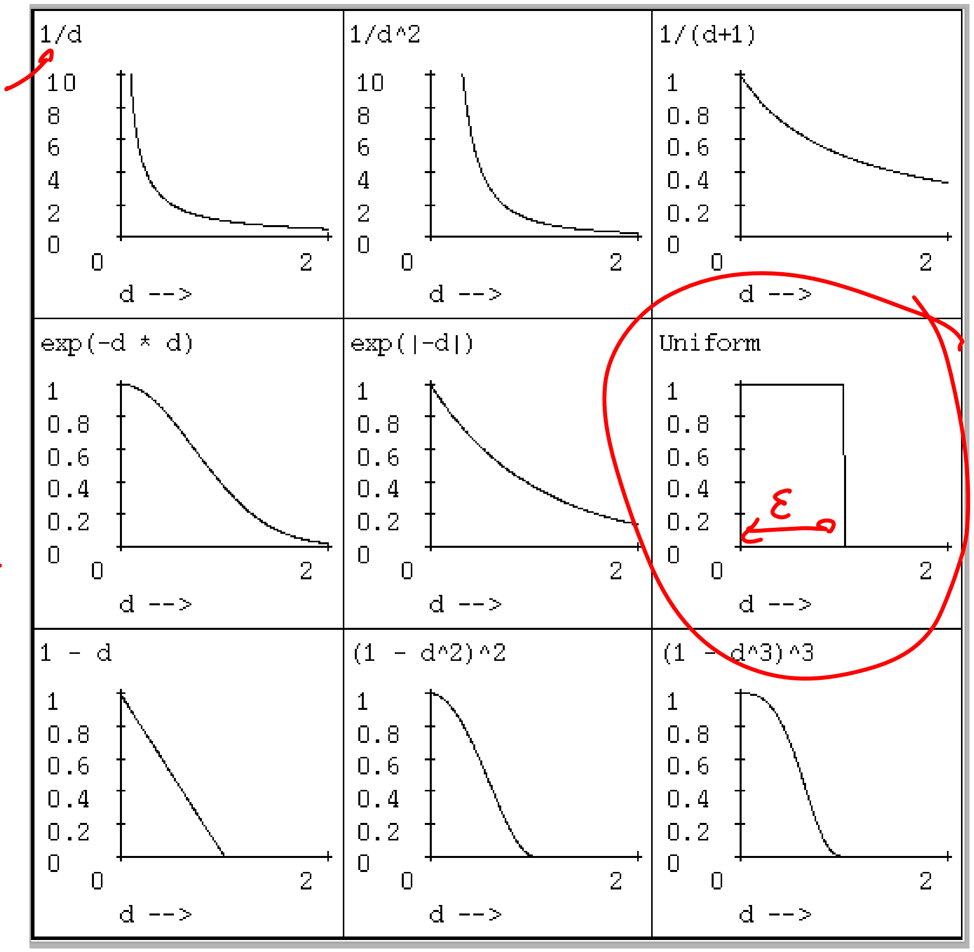
Weighting functions

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

many possibilities.
e.g., Uniform Kernel



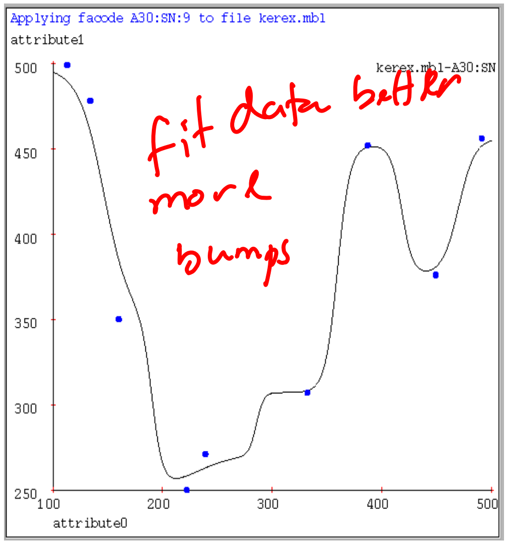
average of everyone in the circle

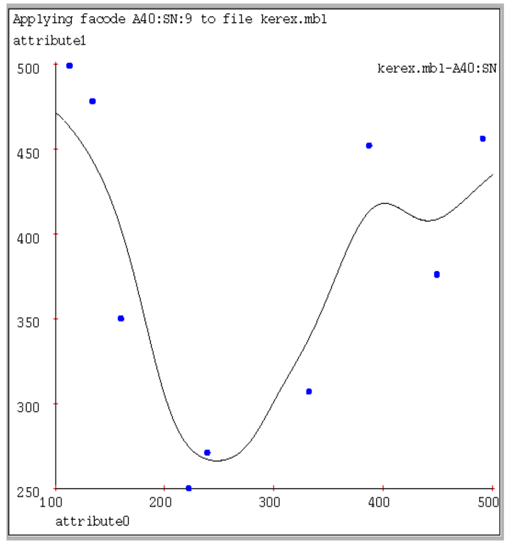


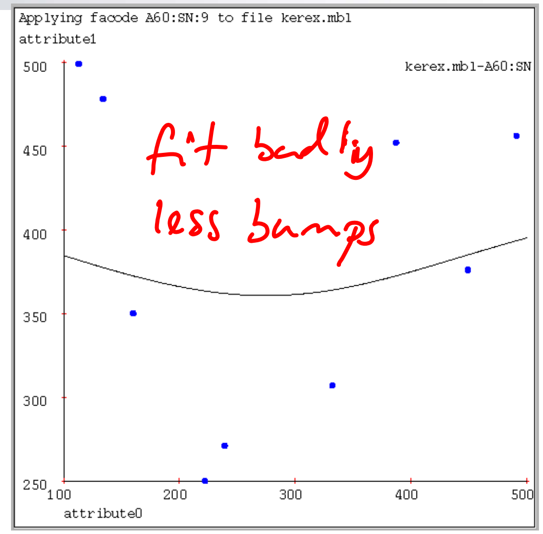
Typically optimize K_w using gradient descent

(Our examples use Gaussian)

Kernel regression predictions







K_W=10 — low bias, high variance

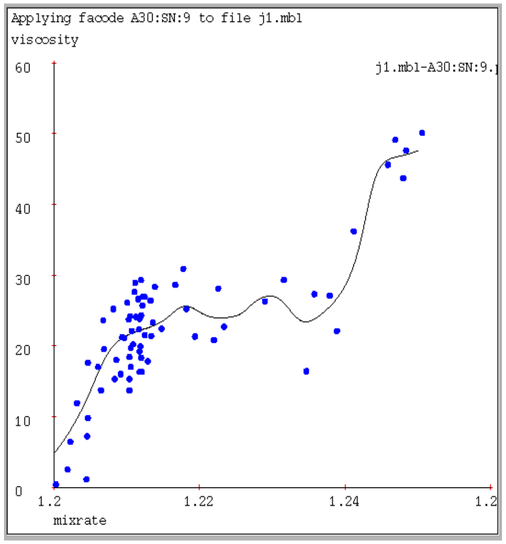
K_W=20

K_W=80 — low variance, high bias

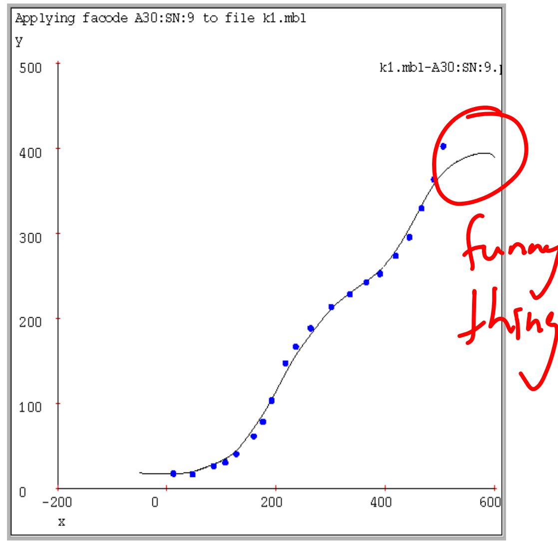
Increasing the kernel width K_w means further away points get an opportunity to influence you.

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

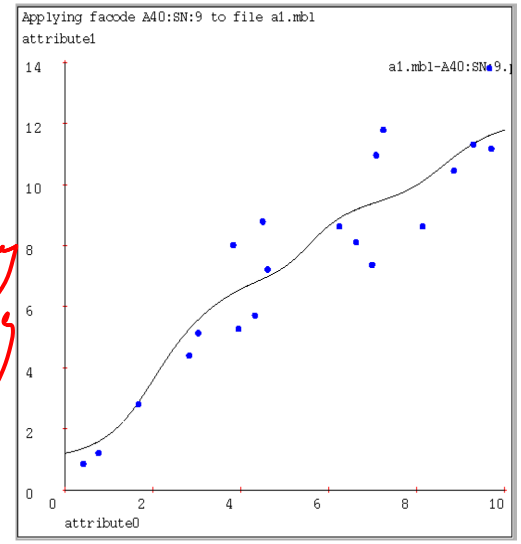
Kernel regression on our test cases



KW=1/32 of x-axis width.



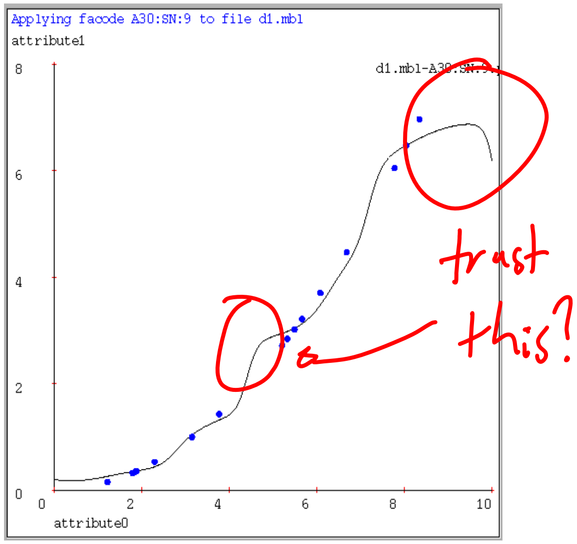
KW=1/32 of x-axis width.



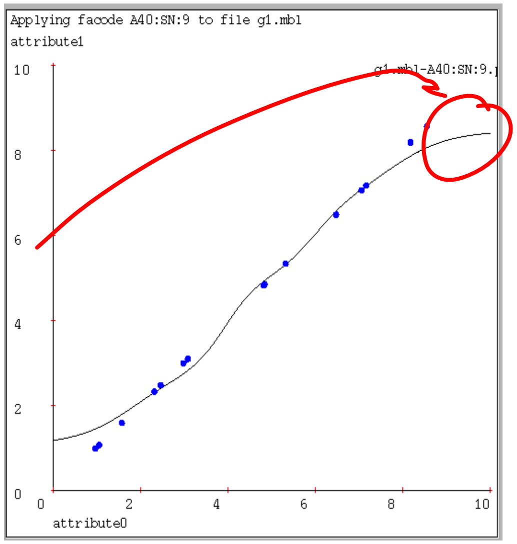
KW=1/16 axis width.

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

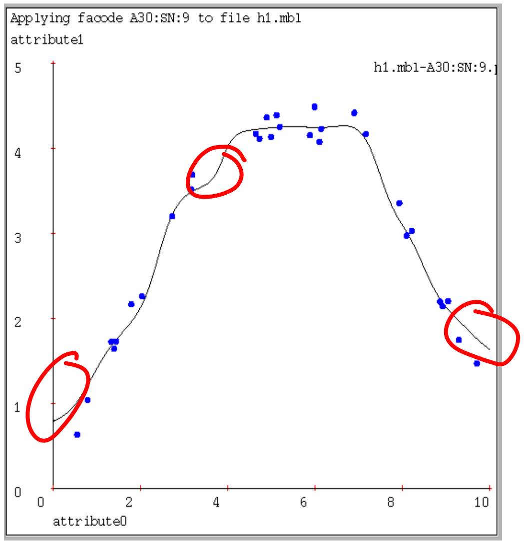
Kernel regression can look bad



KW = Best.



KW = Best.



KW = Best.

Time to try something more powerful…

Locally weighted regression

Kernel regression: equivalent to LWR if only using constant as a basis function

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**
    - e.g., ***wi = exp(-D(xi, query)² / Kw²)***
- *How to fit with the local points?*
  - **General weighted regression:**

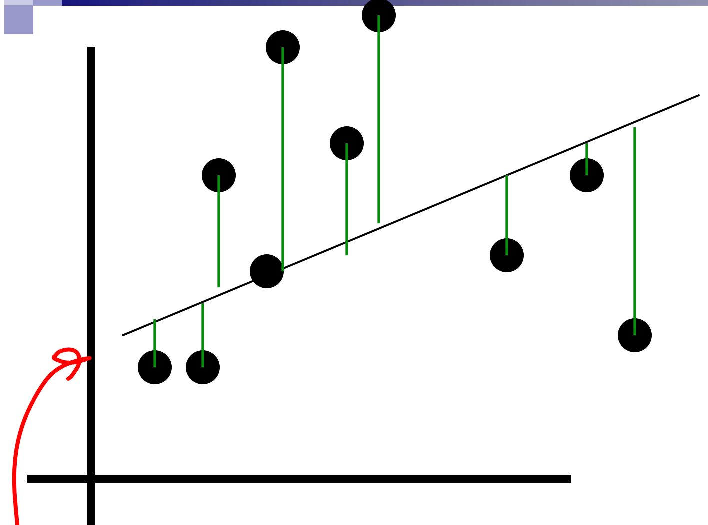
least squares like linear regression

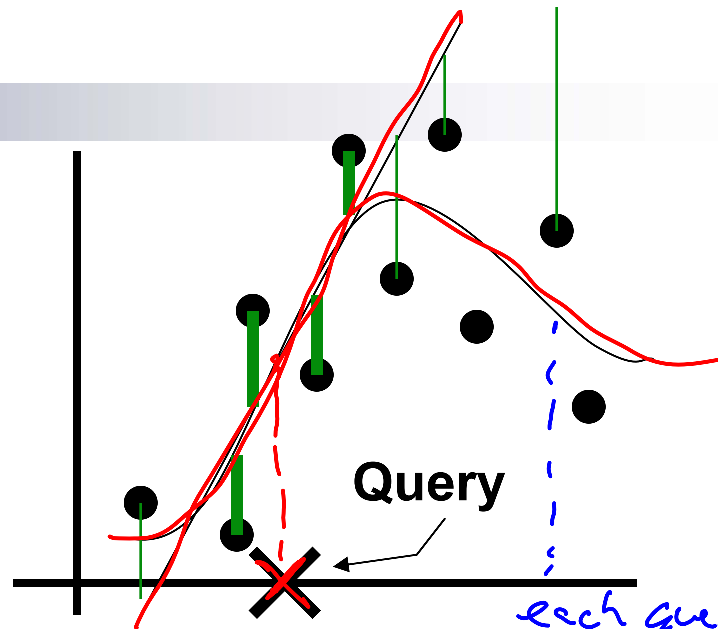
$$\hat{\beta} = \operatorname*{argmin}_{\beta} \sum_{k=1}^{N} w_k^{\,2} \left(y_k - \beta^T x_k \right)^2$$

weigh points near me

$\hat{y} = \hat{\beta}^T x_k$

How LWR works





each query point, new line

Linear regression

- Same parameters for all queries

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

Locally weighted regression

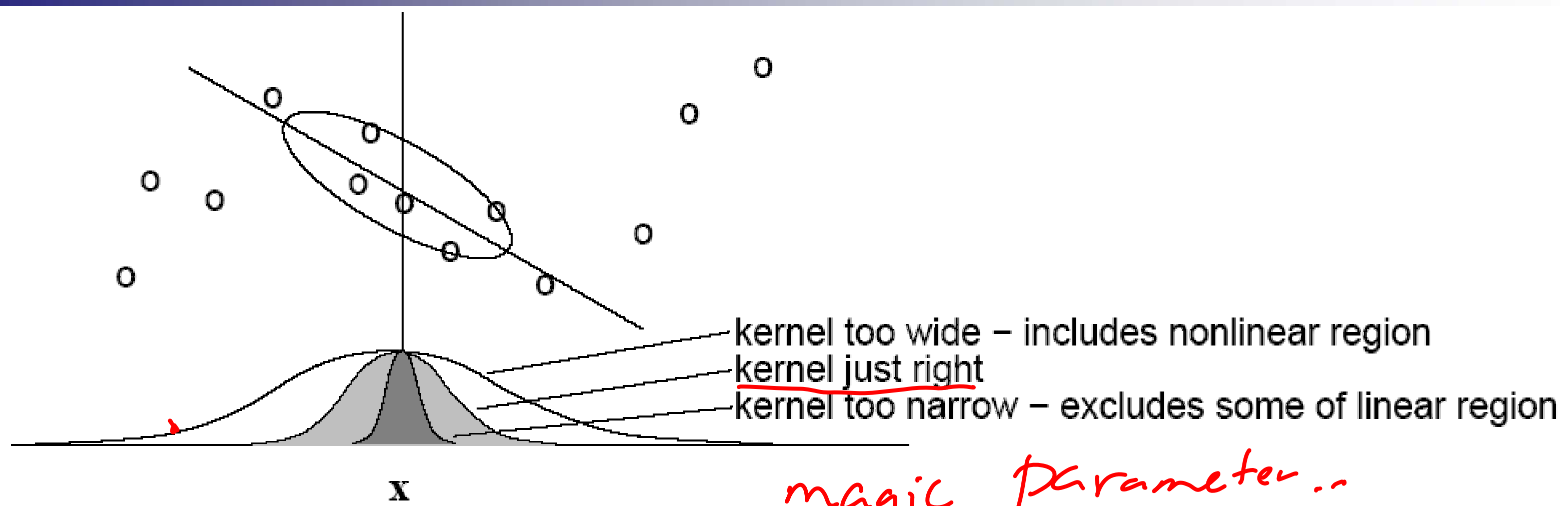
- Solve weighted linear regression for each query

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

weights depend on query point

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

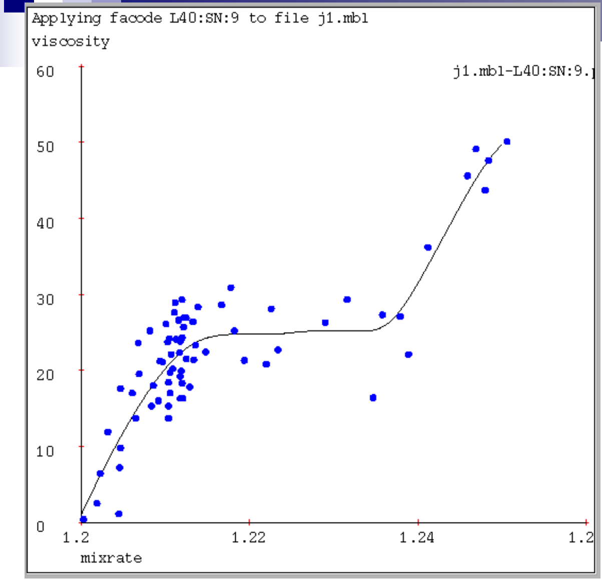
Another view of LWR



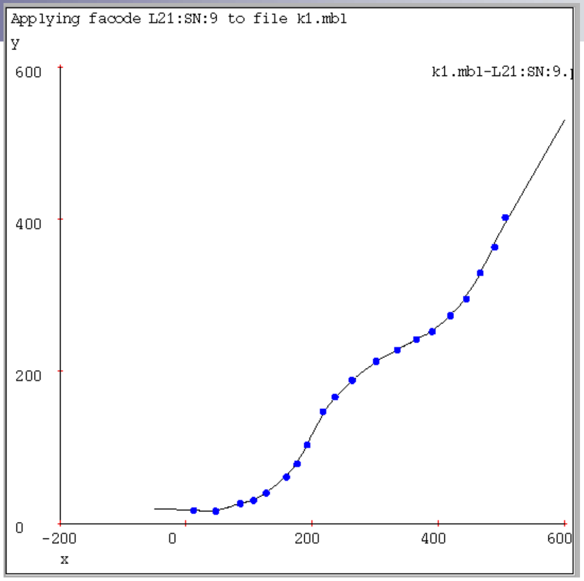
magic parameter..
typically with X validation

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

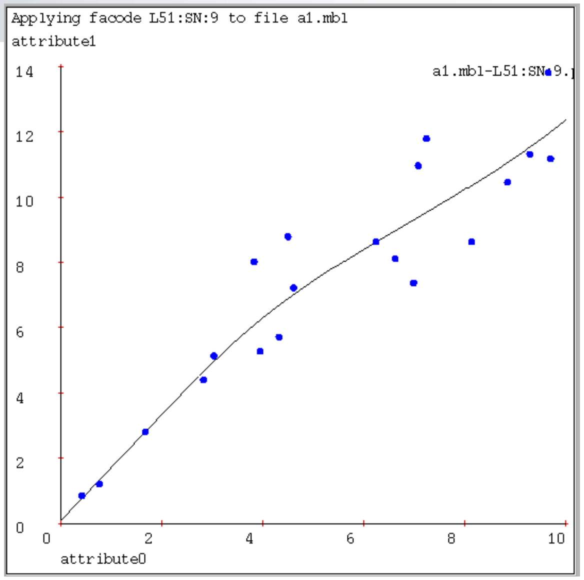
LWR on our test cases



KW = 1/16 of x-axis width.

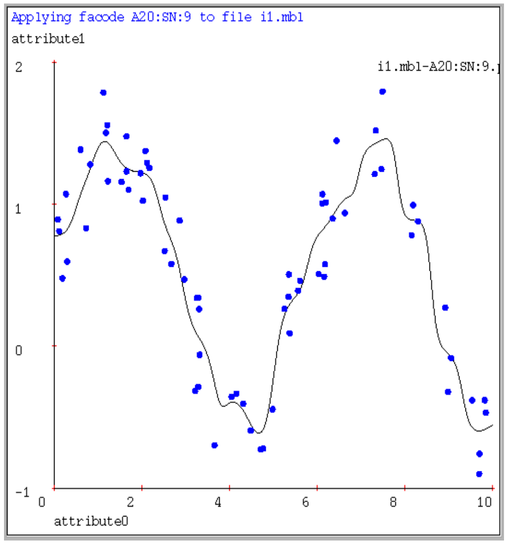


KW = 1/32 of x-axis width.



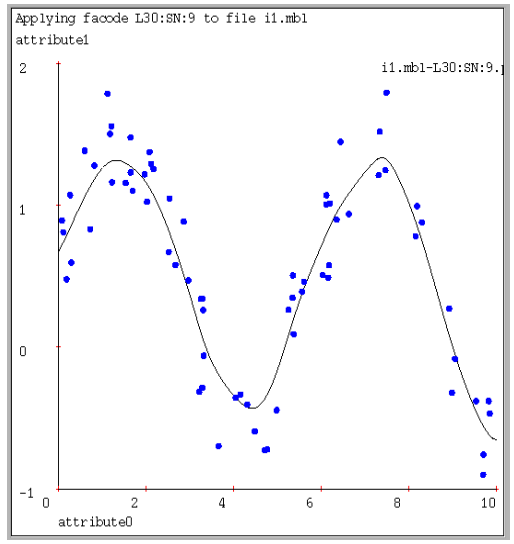
KW = 1/8 of x-axis width.

Locally weighted polynomial regression



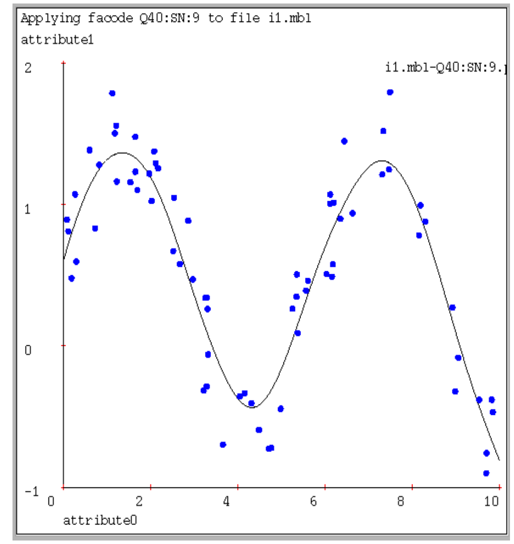
Kernel Regression
Kernel width K_W at optimal level.

KW = 1/100 x-axis



LW Linear Regression
Kernel width K_W at optimal level.

KW = 1/40 x-axis



LW Quadratic Regression
Kernel width K_W at optimal level.

KW = 1/15 x-axis

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

- Must store and retreve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!

    point
  - We'll see fast methods for dealing with large datasets

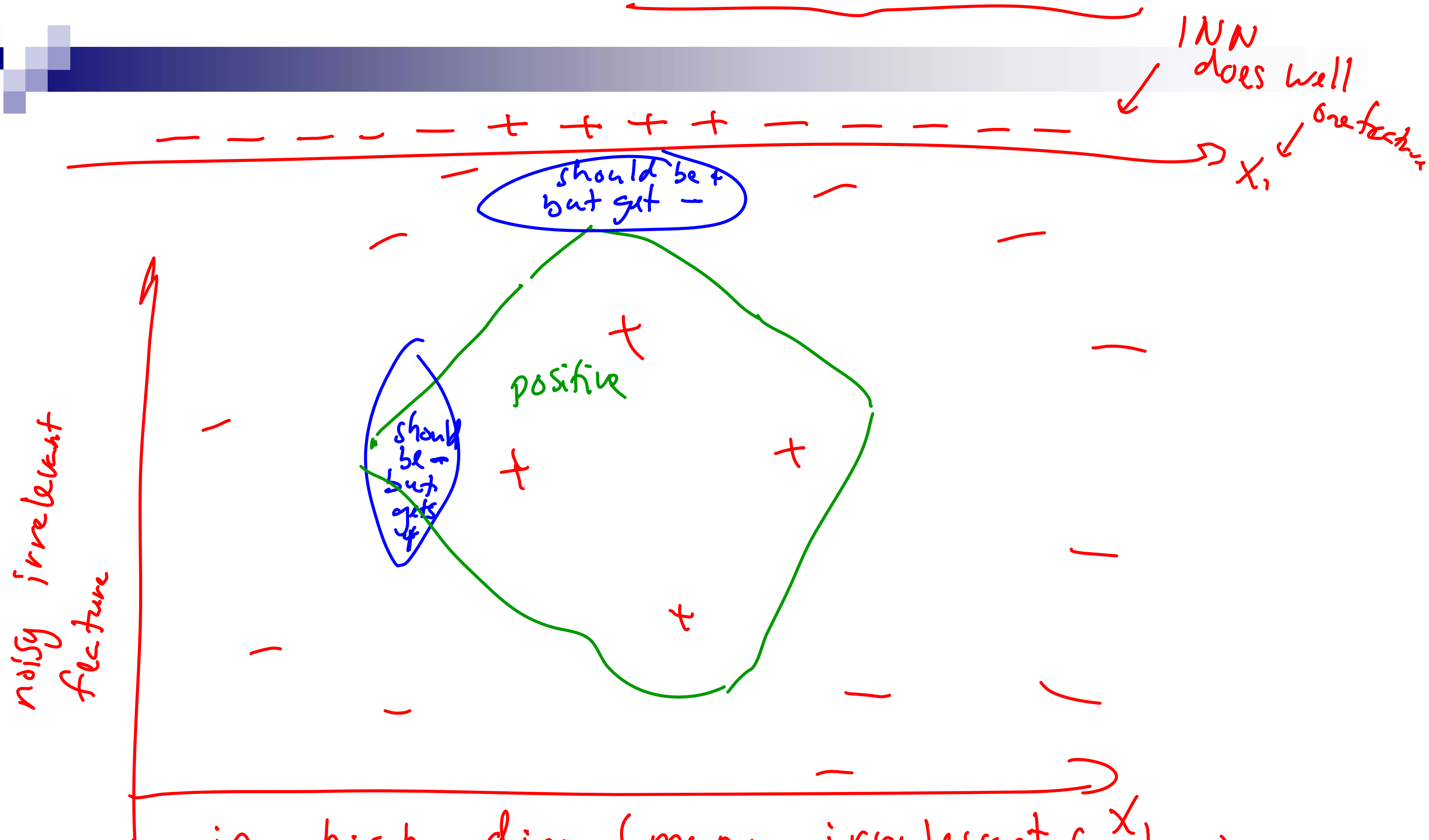
    KD-trees
- Instance-based learning often poor with noisy or irrelevant features

  Ball-trees ...

  ⋮

if: $P(x_i \mid y=+, x_{i-}) = P(x_i \mid y=-, x_{i-})$
x_i where x_{i-} is value of rest of features.

Curse of the irrelevant feature



in high dims (many irrelevant features)
every point is equally far apart

What you need to know about instance-based learning

- 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   pick k_w
  - Smoother than k-NN
- Locally weighted regression
  - Generalizes kernel regression, not just local average
- Curse of dimensionality
  - Must remember (very large) dataset for prediction
  - 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

Support Vector Machines

Machine Learning – 10701/15781

Carlos Guestrin

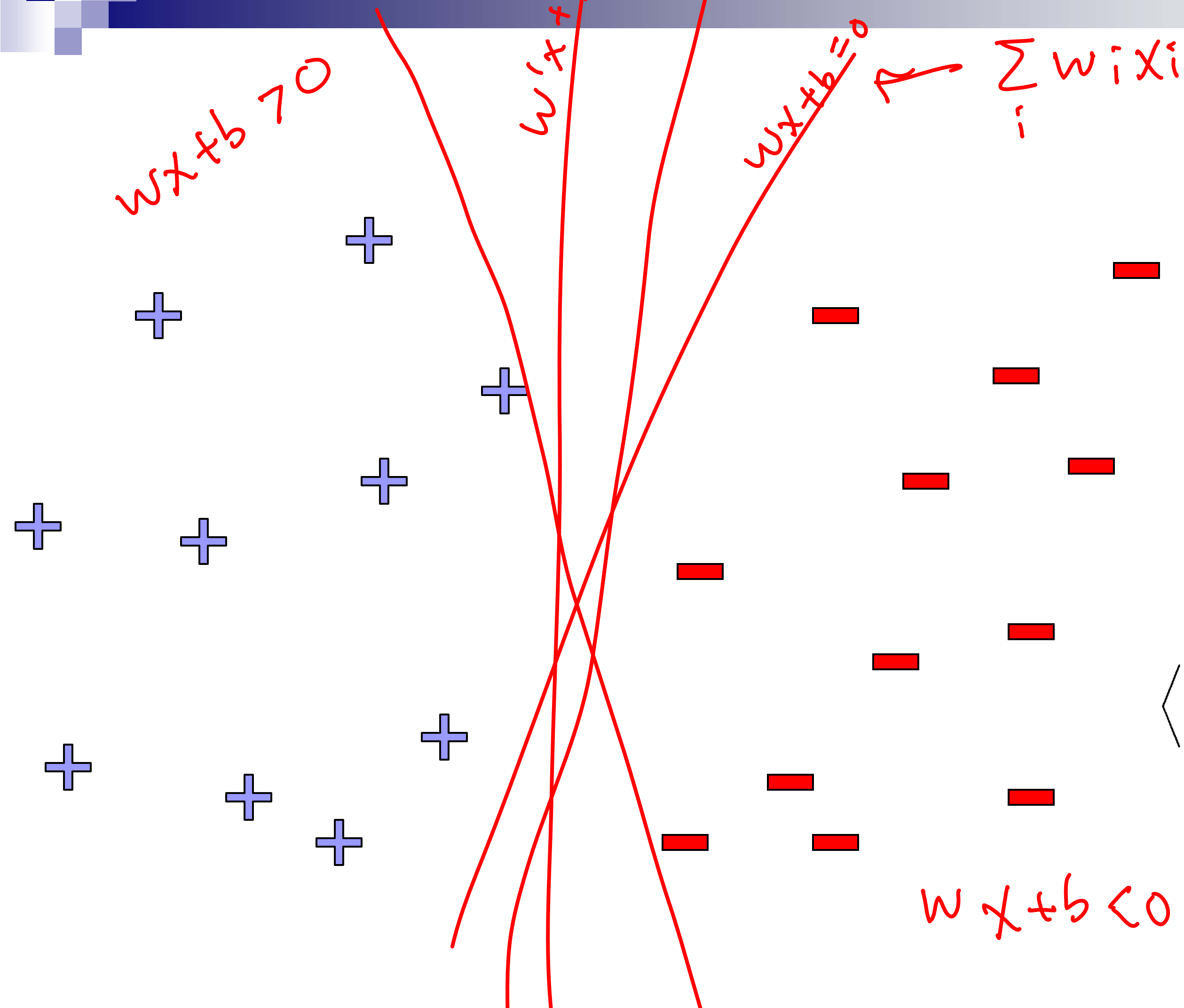
Carnegie Mellon University

February 19th, 2007

Linear classifiers – Which line is better?

constant

$\sum_i w_i x_i + b = 0$



Data:

m dimensions

$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \right\rangle$$
$$\vdots$$
$$\left\langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \right\rangle$$

Example i:

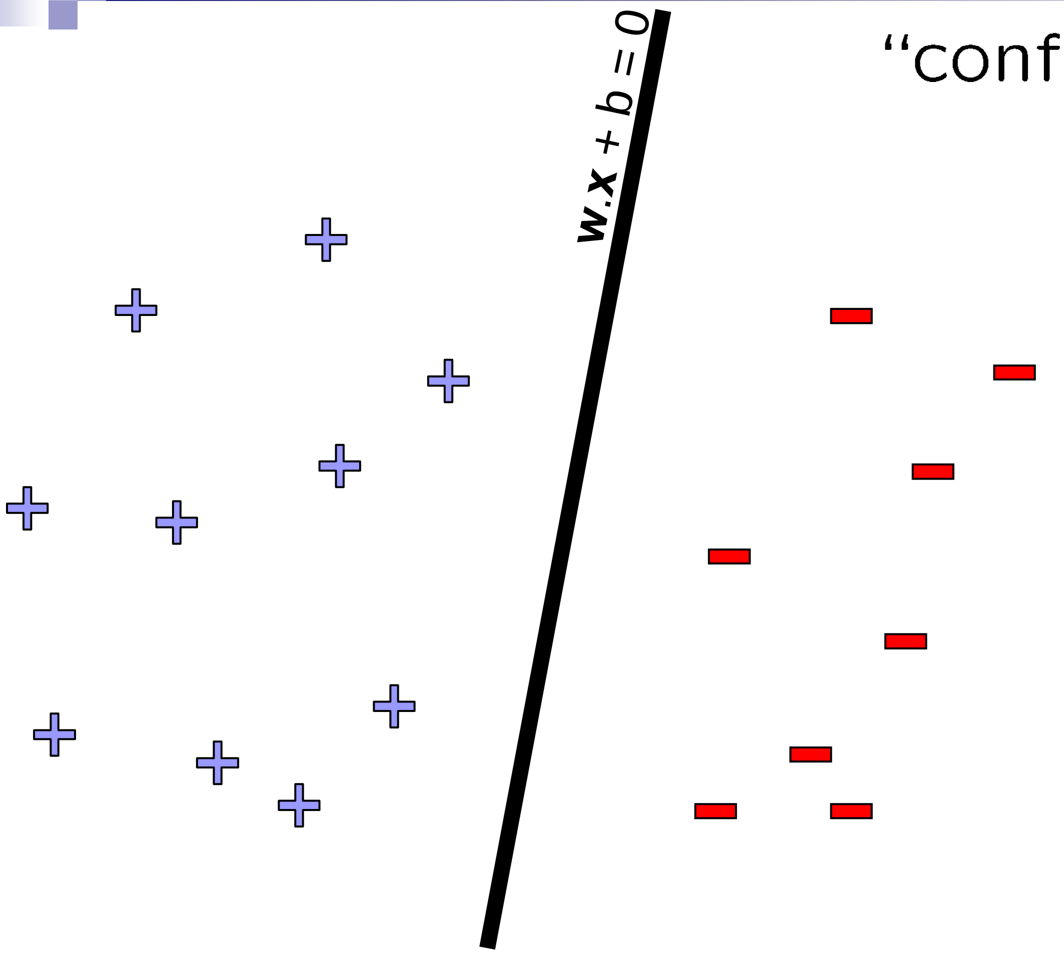
$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$

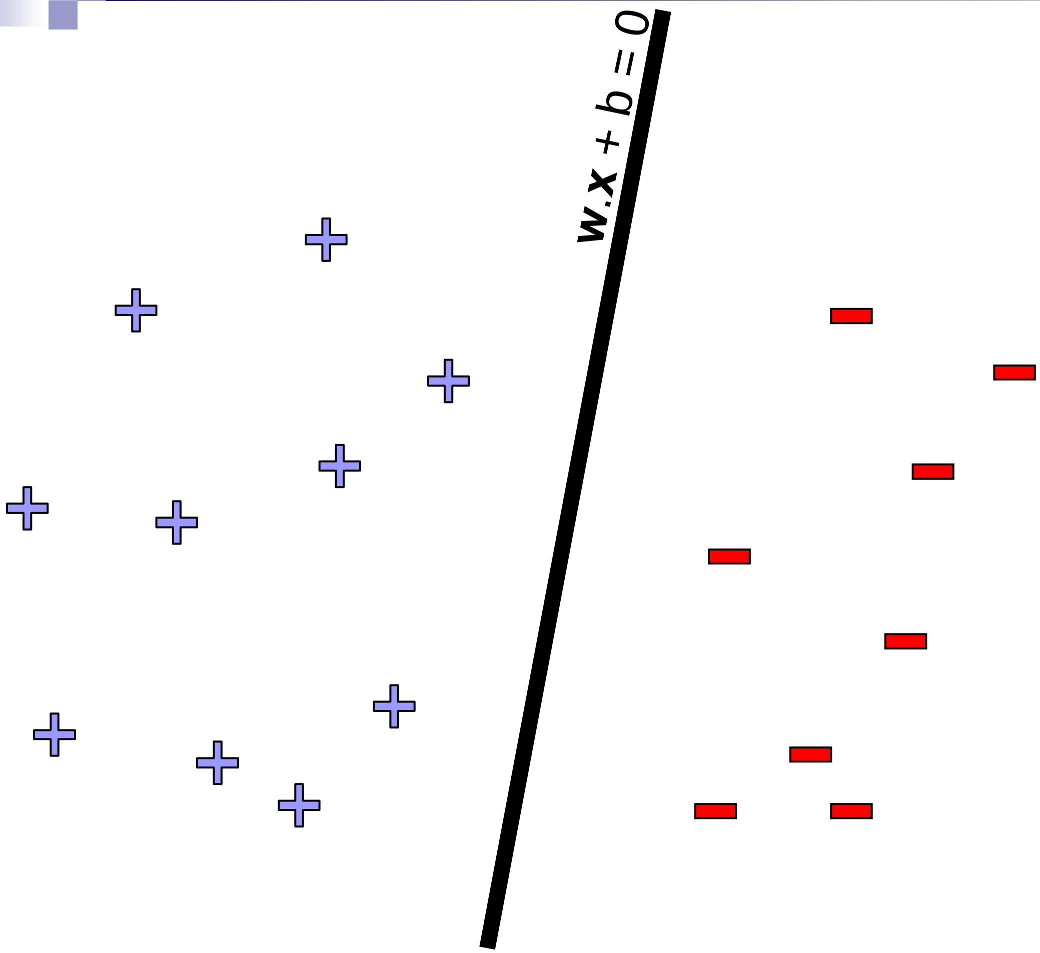
Pick the one with the largest margin!

"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

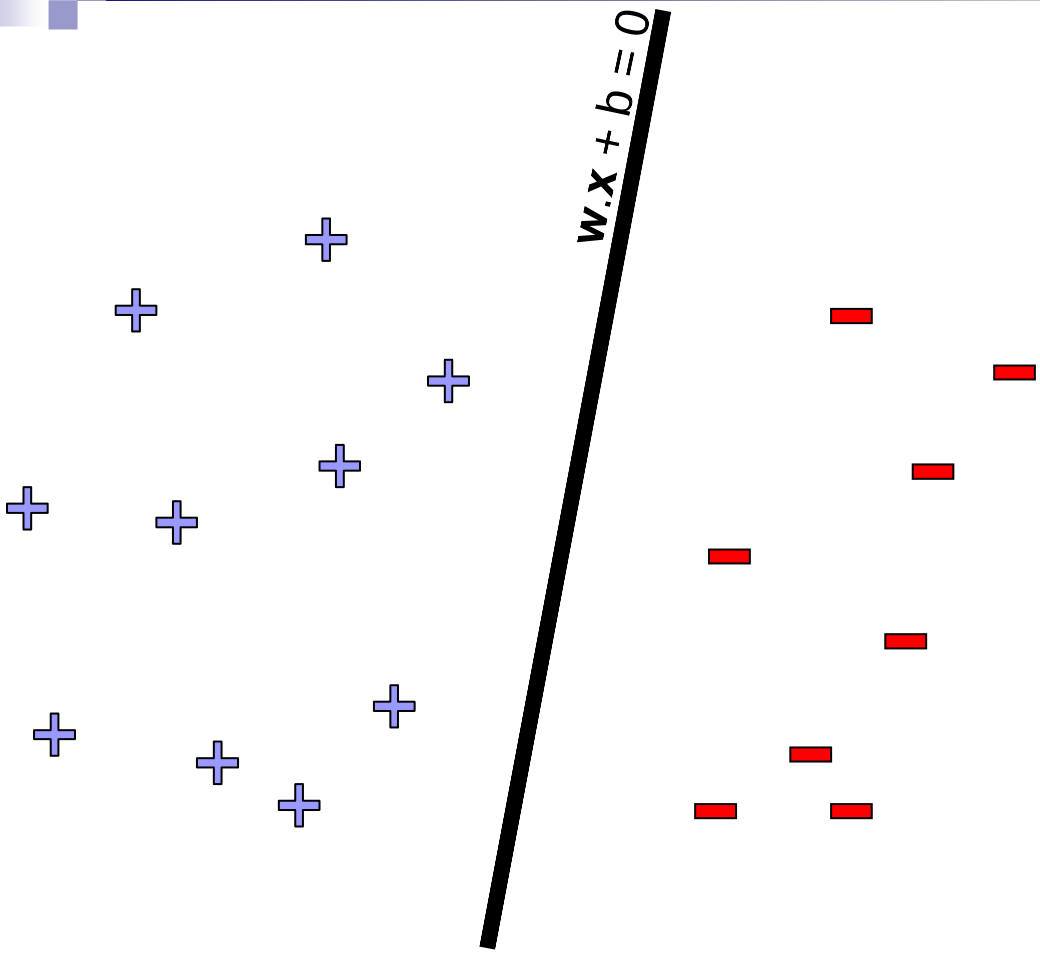


$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$

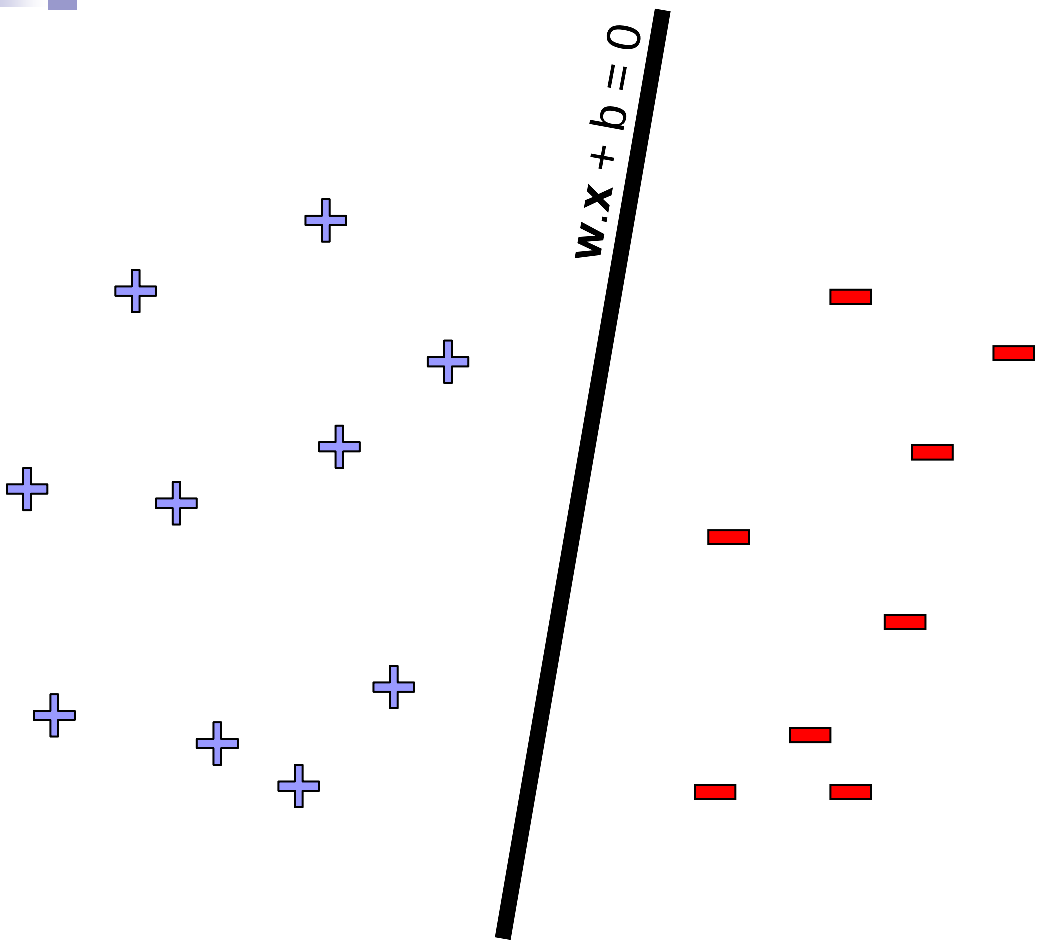
Maximize the margin



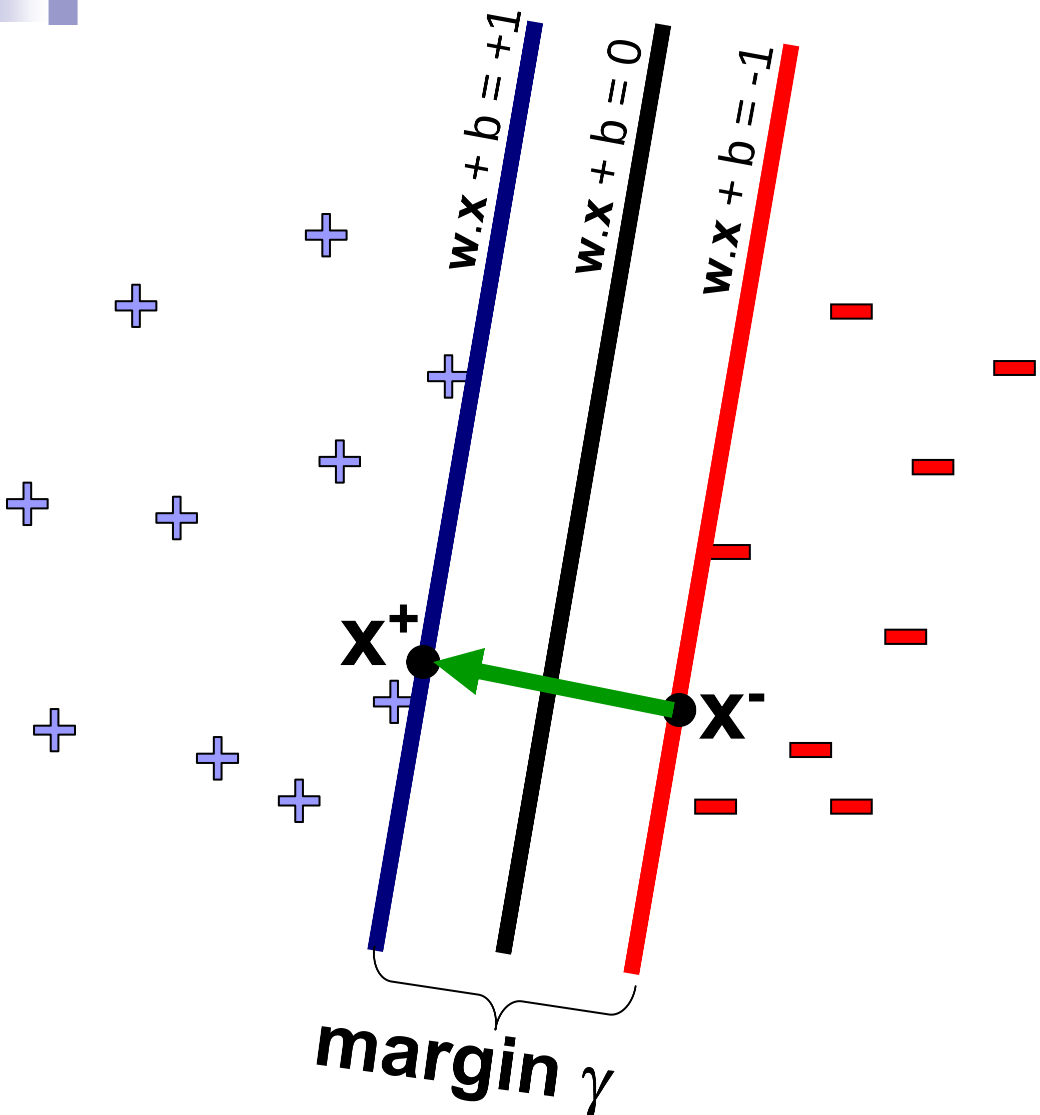
But there are a many planes…



Review: Normal to a plane

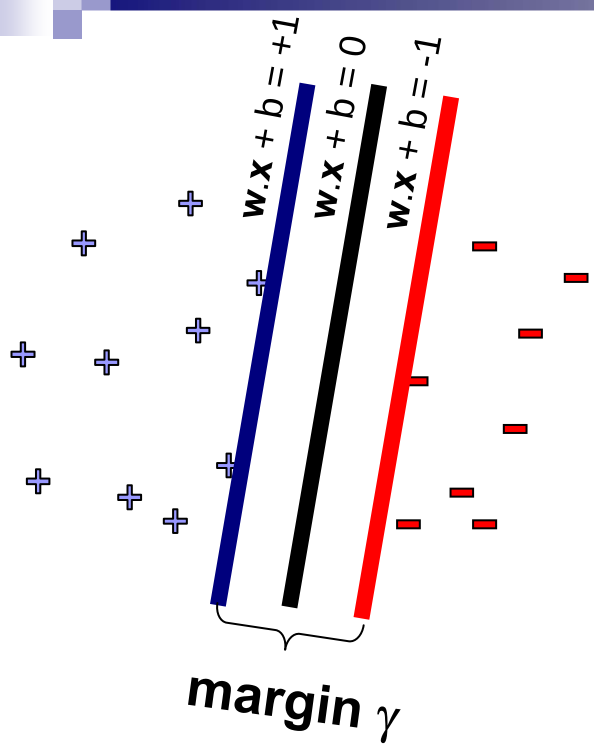


Normalized margin – Canonical hyperplanes



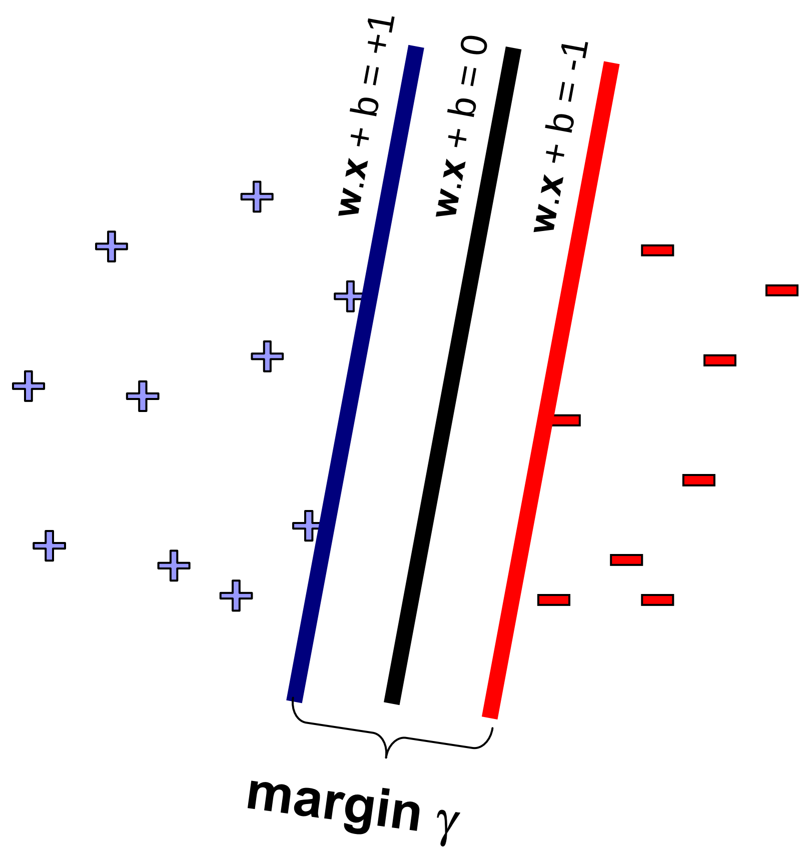
$$\gamma = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Margin maximization using canonical hyperplanes



$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \ \forall j \in \text{Dataset}$$

Support vector machines (SVMs)



$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \ \forall j$$

- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms

- Hyperplane defined by support vectors