



Instance-based Learning

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

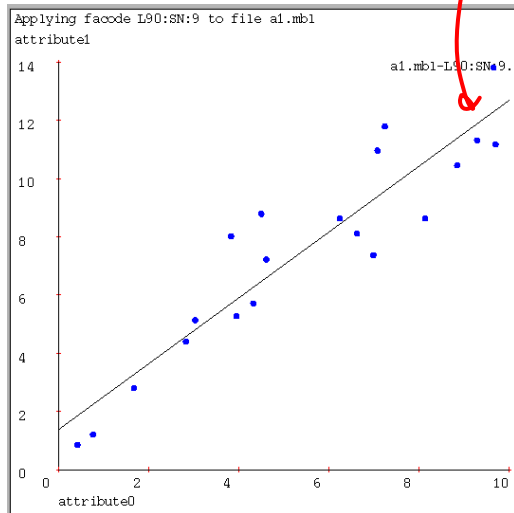
February 20th, 2006

Announcements

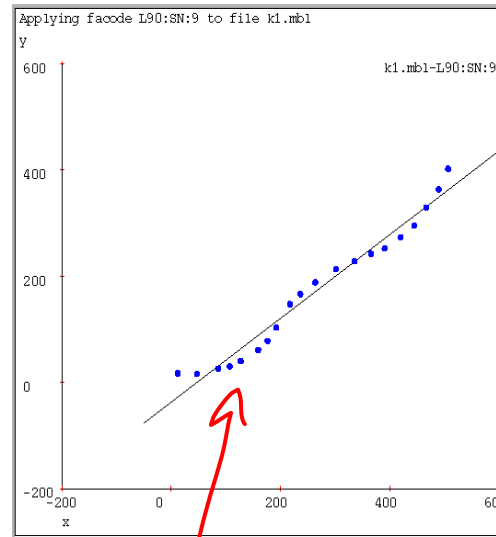


- Third homework
 - ☐ Out later today
 - ☐ Due March 1st

Why not just use Linear Regression?

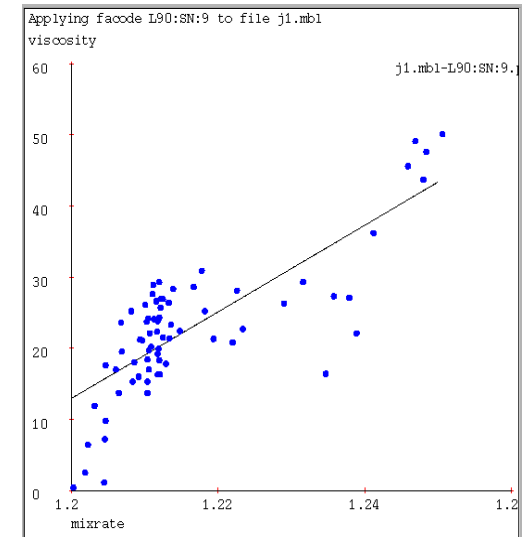


pretty good



missing trend

could more
basis functions

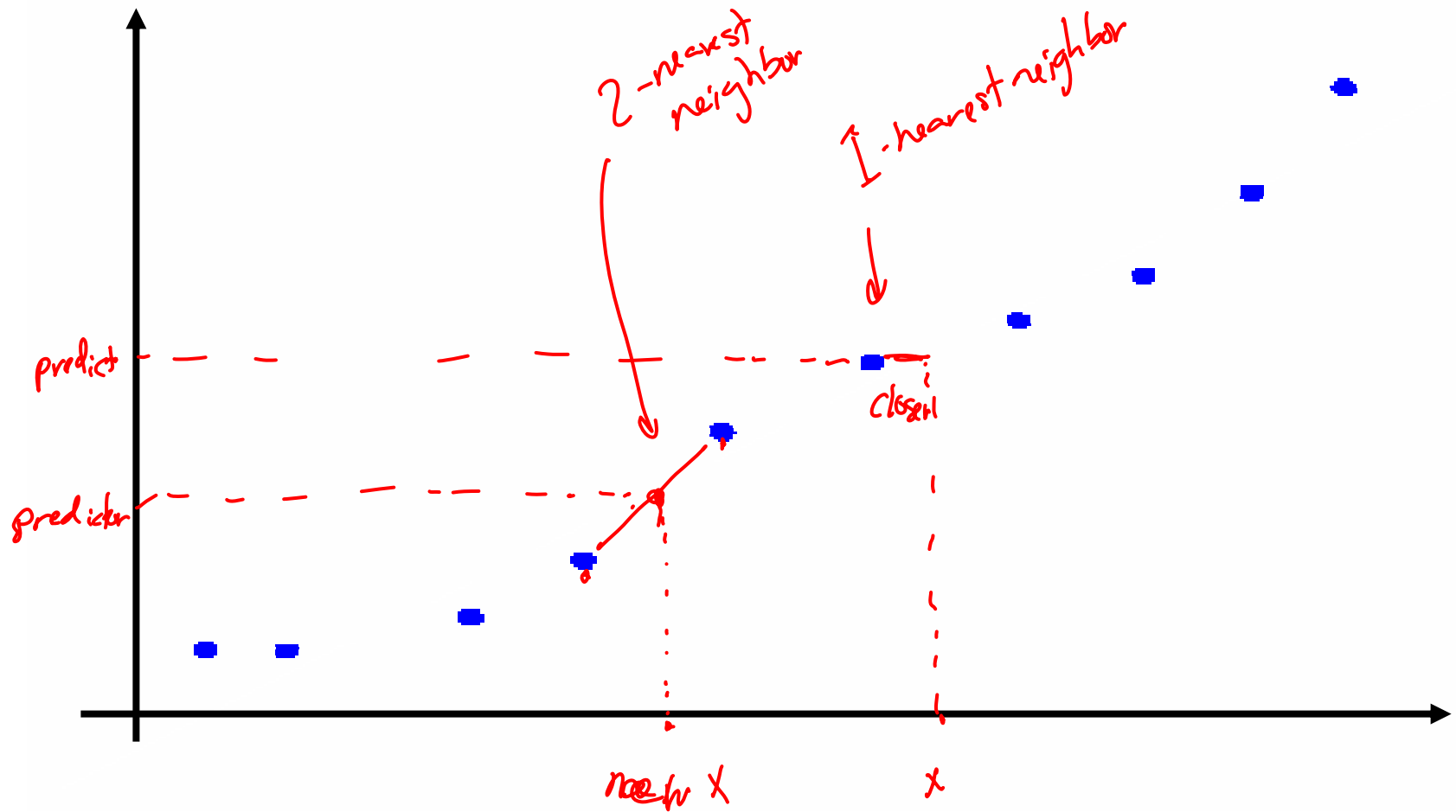


bad job!

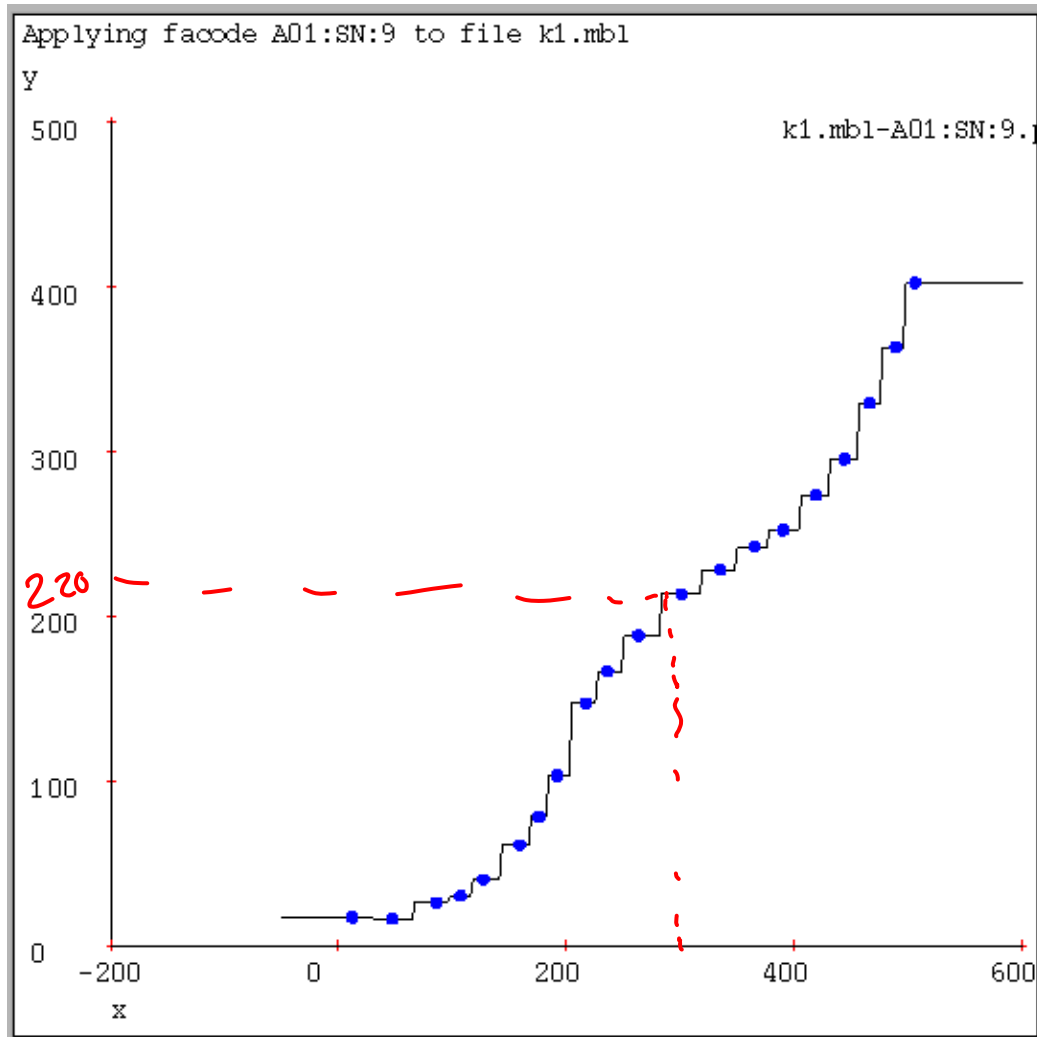
not sure what
do...

lots more basis
function

Using data to predict new data



1-Nearest neighbor



Univariate 1-Nearest Neighbor

Given datapoints (x_1, y_1) $(x_2, y_2) \dots (x_N, y_N)$, 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:

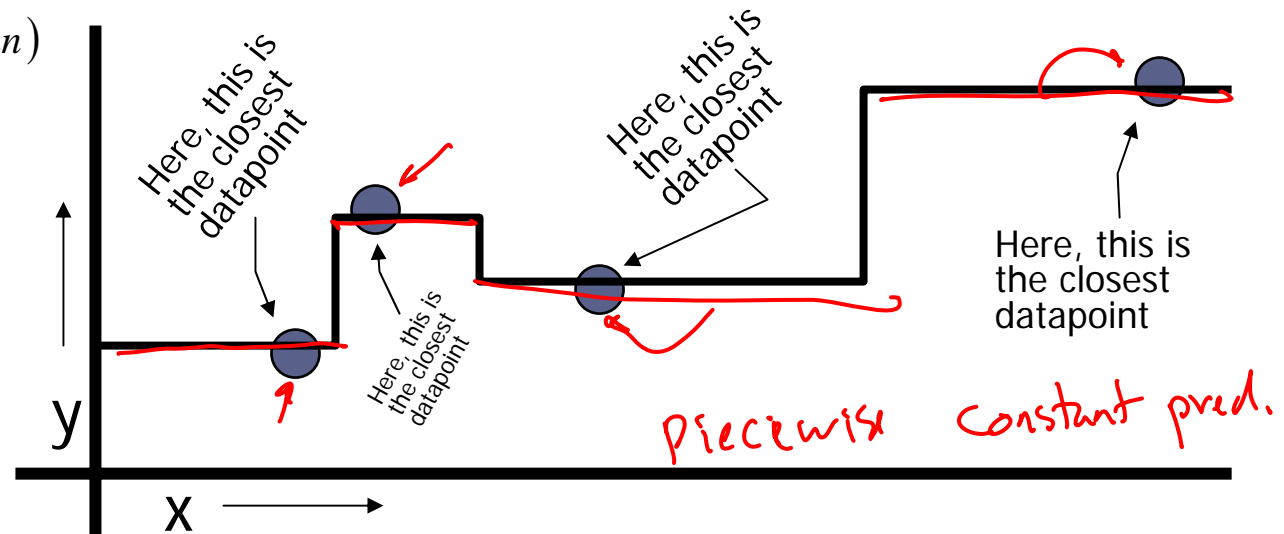
1. Find the closest x_i in our set of datapoints

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

closest in dataset

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

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

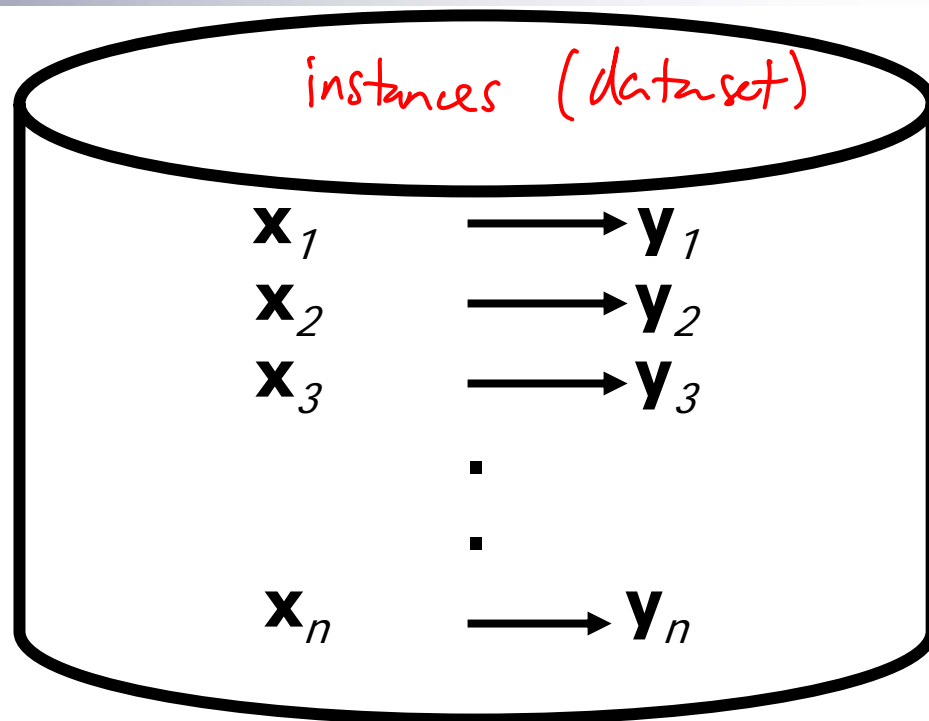


1-Nearest Neighbor is an example of....

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 *define "closest"*
- 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:

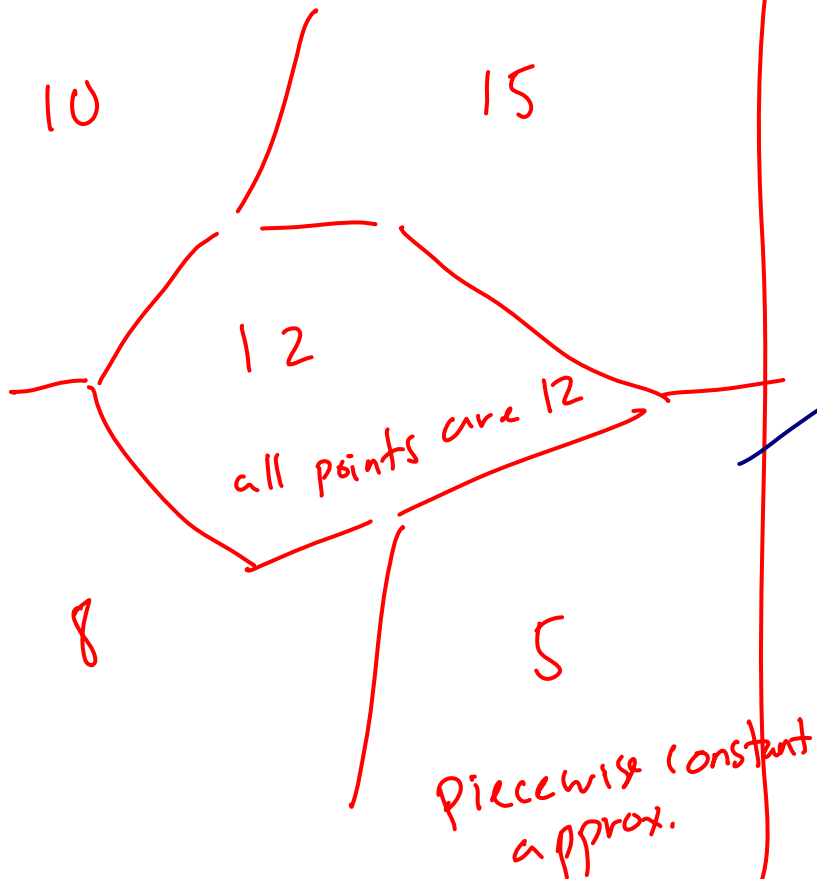
1. A distance metric *input x : $i^* = \arg \min_i \|x_i - x\|_2$*
Euclidian (and many more)
2. How many nearby neighbors to look at?
One
3. A weighting function (optional)
Unused
4. How to fit with the local points?
Just predict the same output as the nearest neighbor.

output y_{i^}*

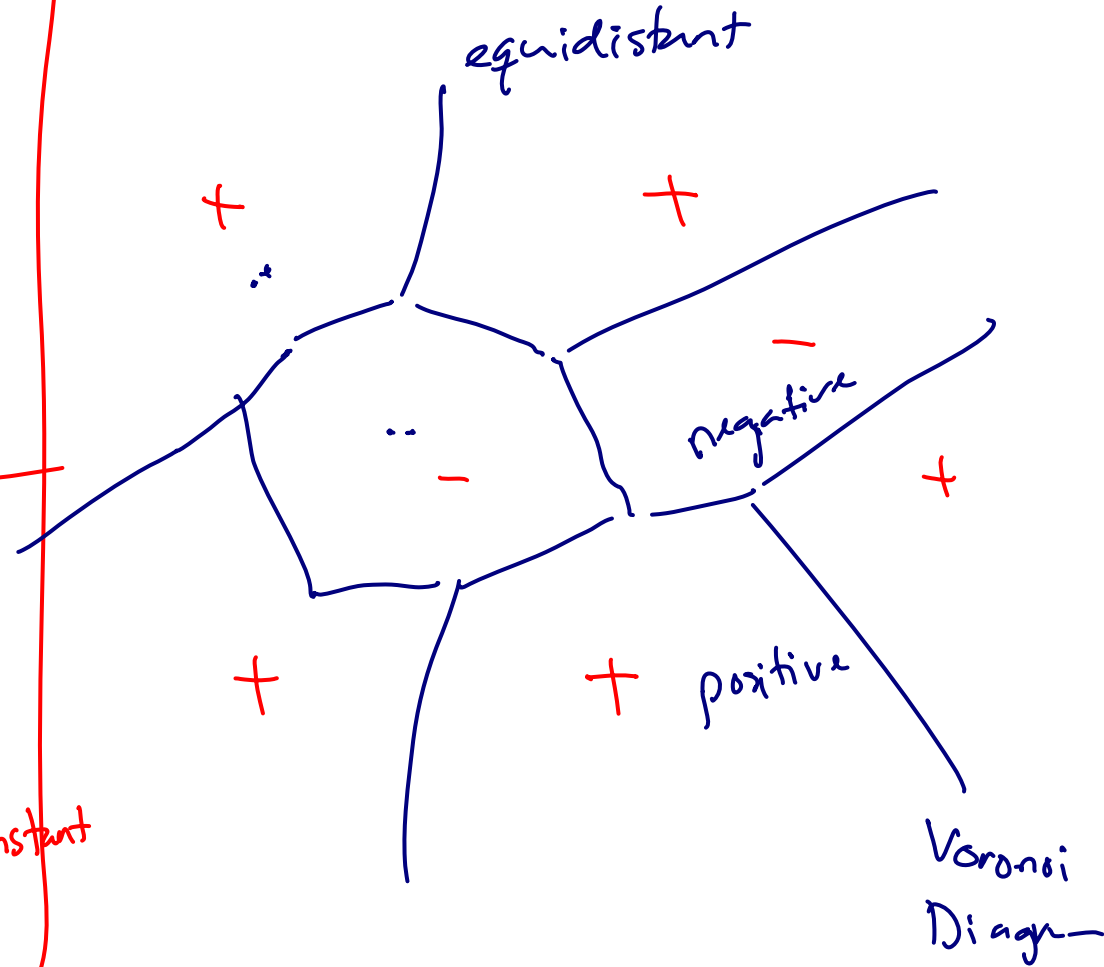
Multivariate 1-NN examples

nearest-neighbor

Regression



Classification

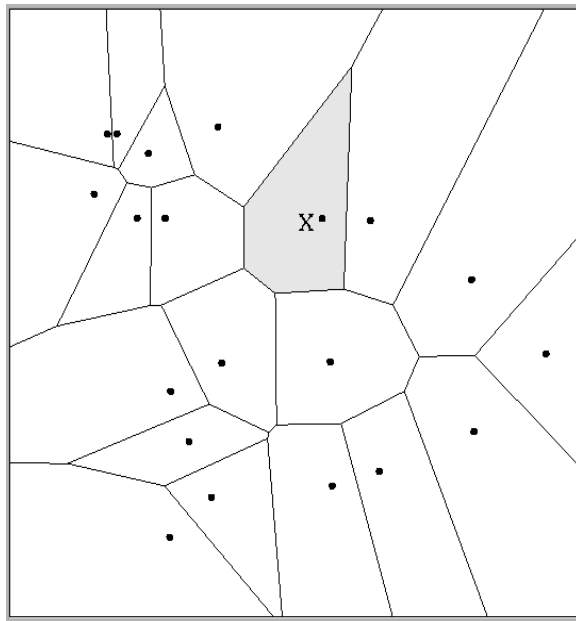


Multivariate distance metrics

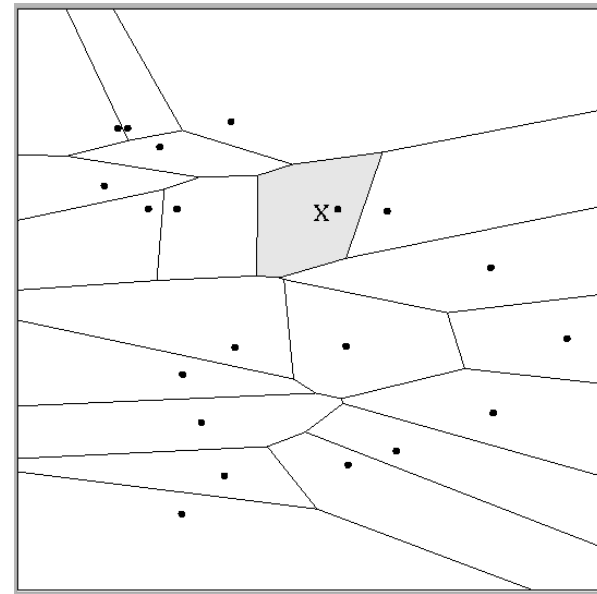
Suppose the input vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ are two dimensional:

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

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,

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

where

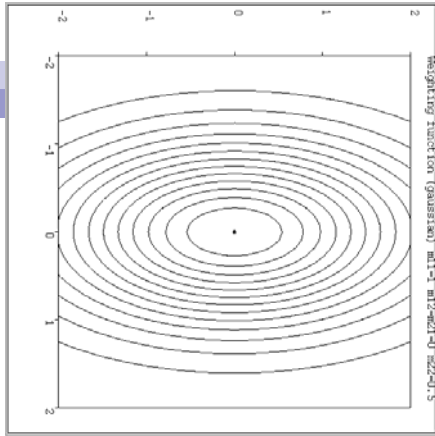
$$D(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Sigma (\mathbf{x} - \mathbf{x}')}$$

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

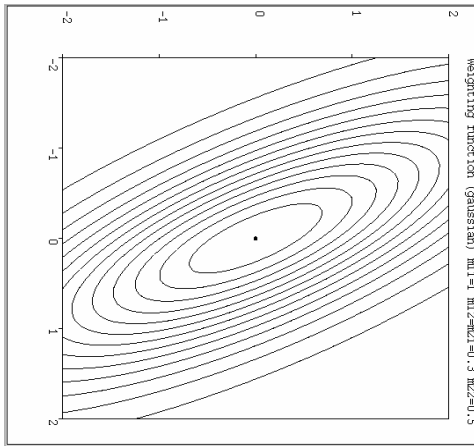
Other Metrics...

- Mahalanobis, Rank-based, Correlation-based,...

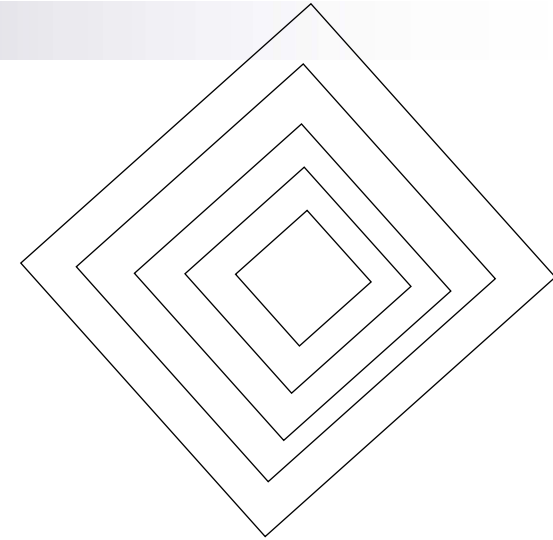
Notable distance metrics (and their level sets)



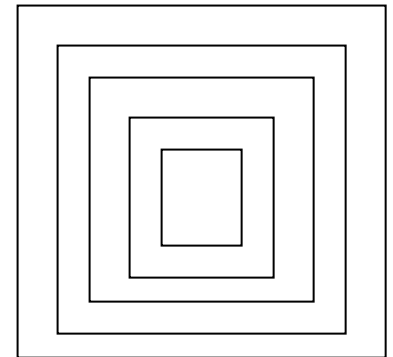
Scaled Euclidian (L_2)



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



L_1 norm (absolute)



L_∞ (*max*) norm

Consistency of 1-NN

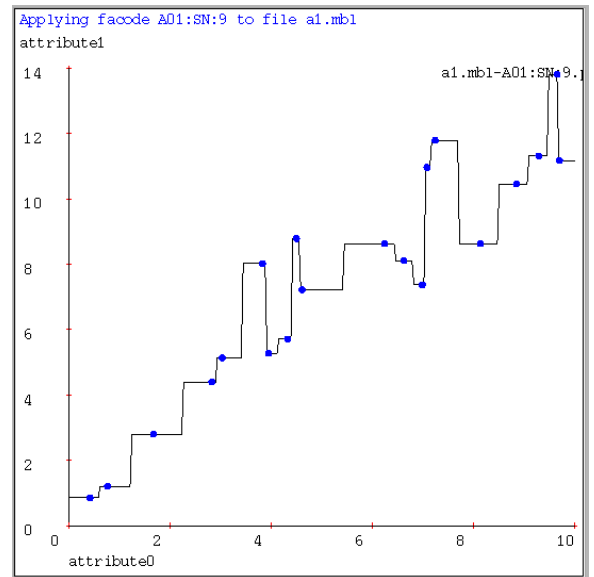
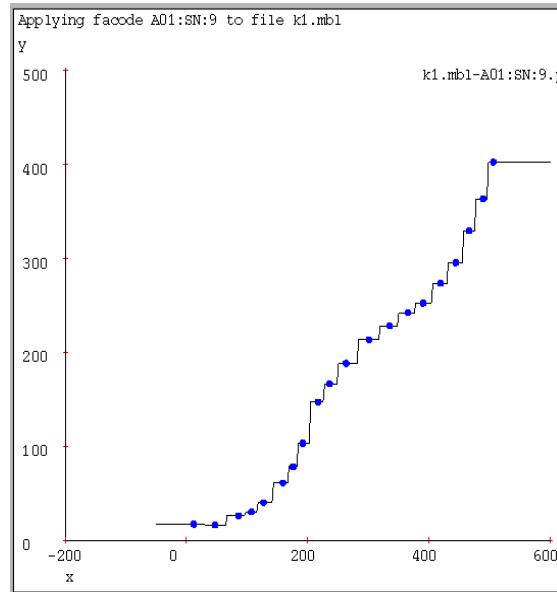
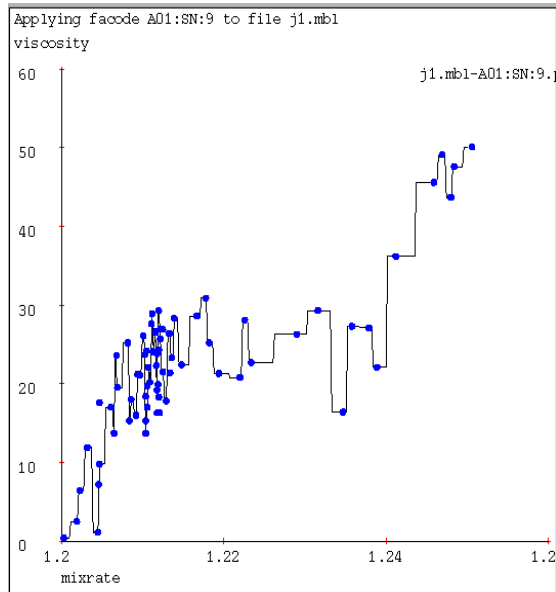
- Consider an estimator f_n trained on n examples
 - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if prediction error goes to zero as amount of data increases
 - e.g., for no noise data, consistent if:

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

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

What about variance???

1-NN overfits?



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

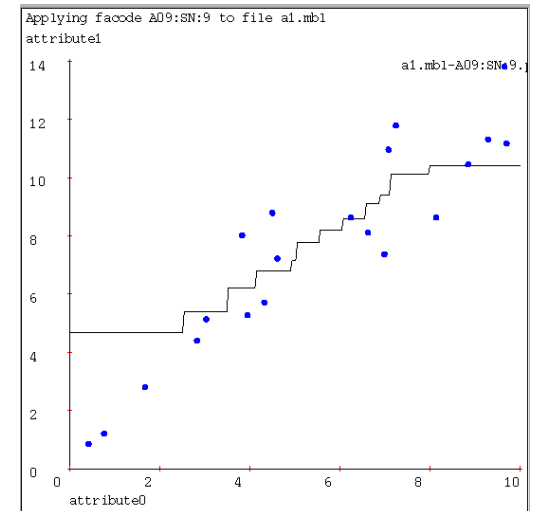
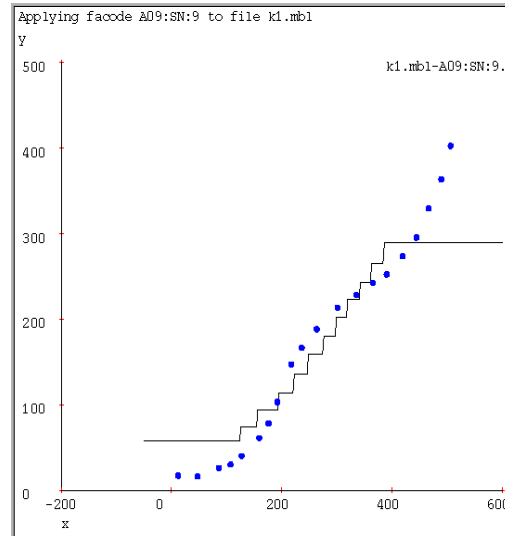
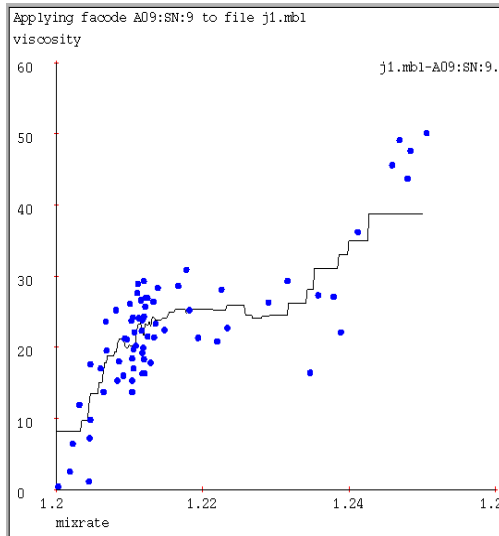
1. *A weighting function (optional)*

Unused

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

Just predict the average output among the k nearest neighbors.

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

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)*

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

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

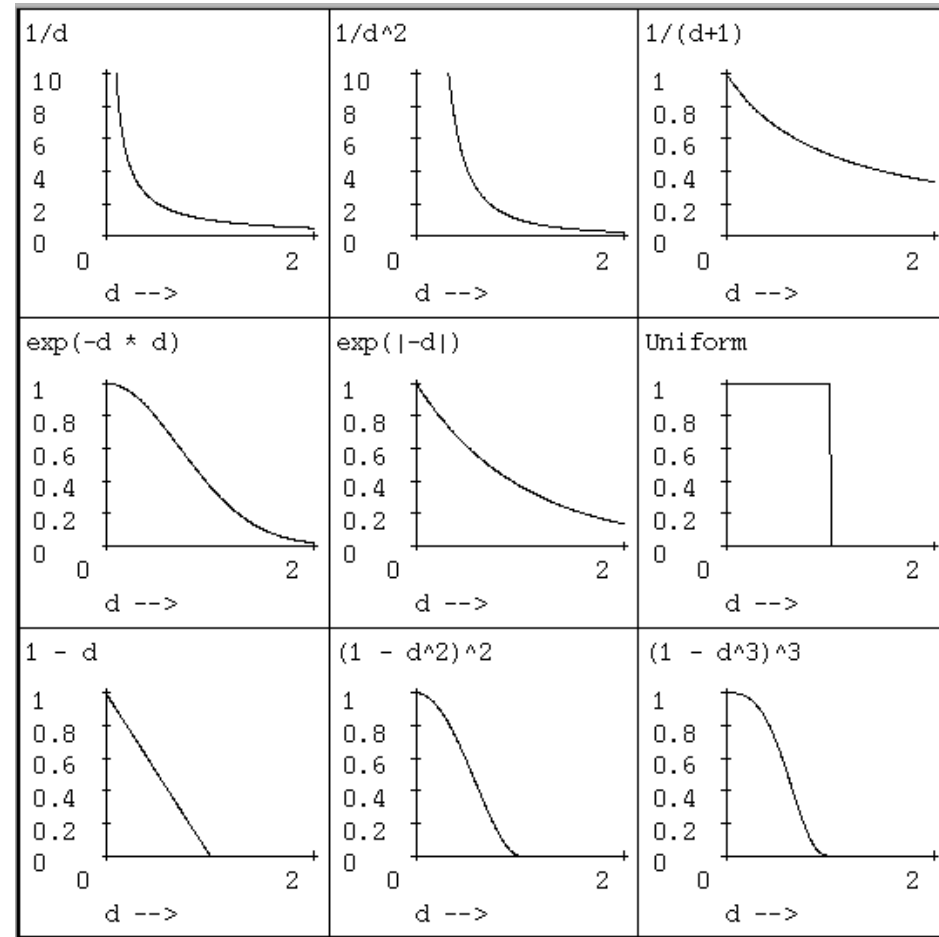
4. *How to fit with the local points?*

Predict the weighted average of the outputs:

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

Weighting functions

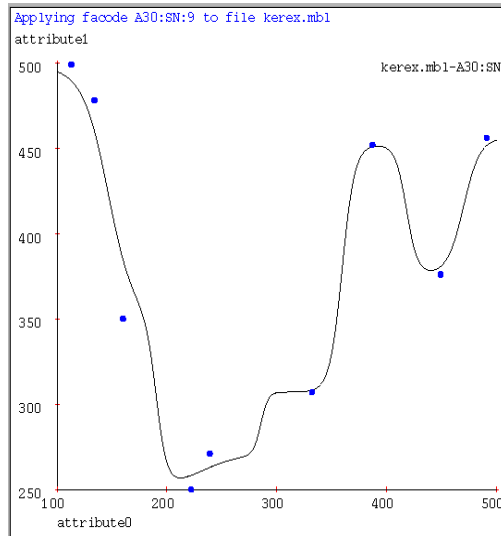
$$w_i = \exp(-D(x_i, \text{query})^2 / K_w^2)$$



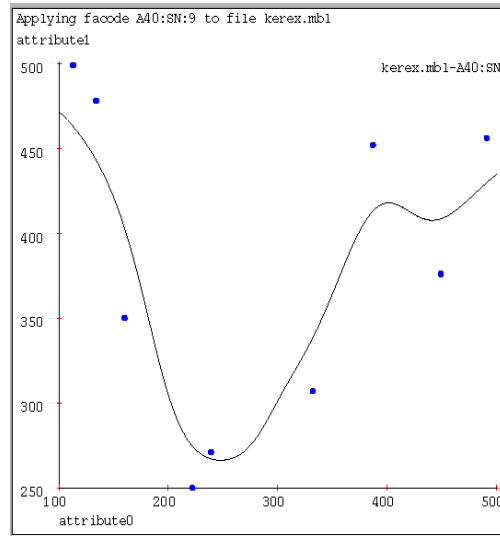
Typically optimize K_w
using gradient descent

(Our examples use Gaussian)

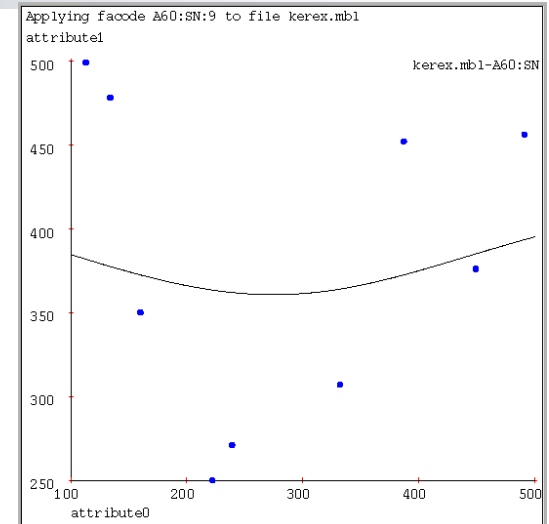
Kernel regression predictions



$K_W=10$



$K_W=20$

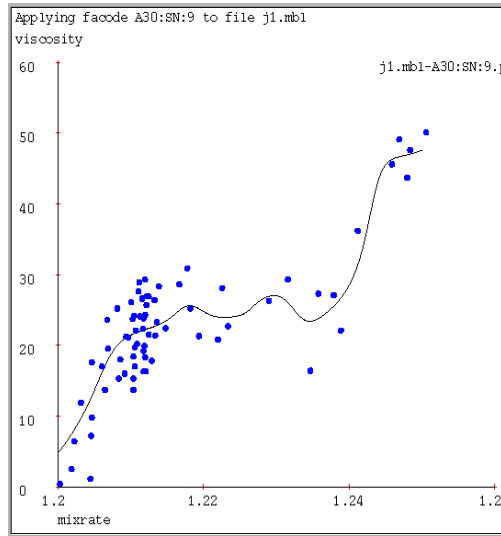


$K_W=80$

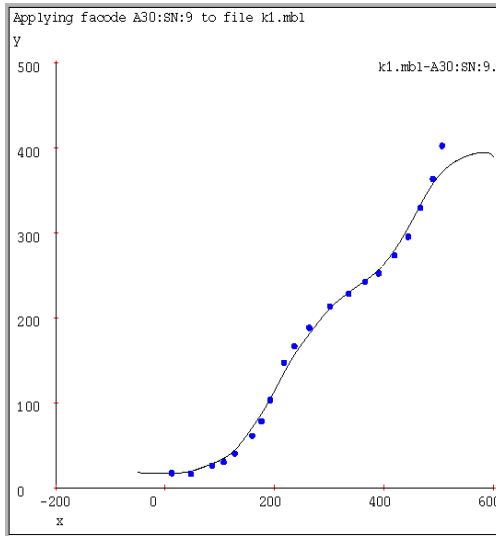
Increasing the kernel width K_W means further away points get an opportunity to influence you.

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

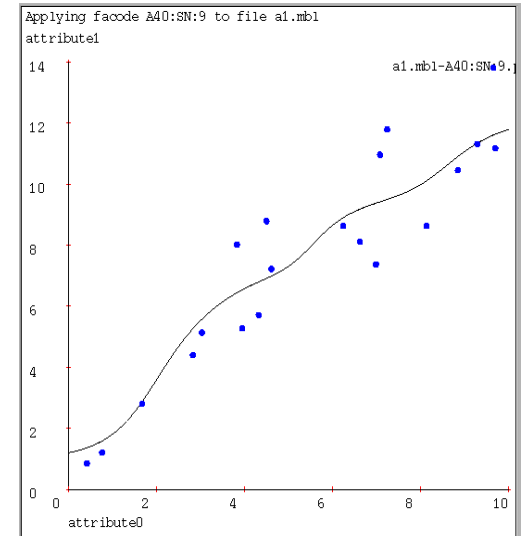
Kernel regression on our test cases



KW=1/32 of x-axis width.



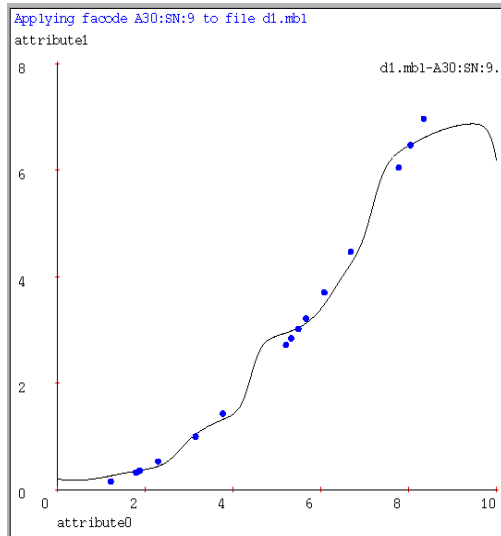
KW=1/32 of x-axis width.



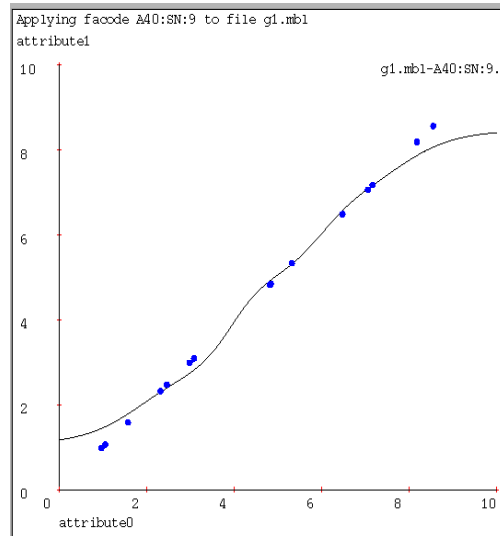
KW=1/16 axis width.

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

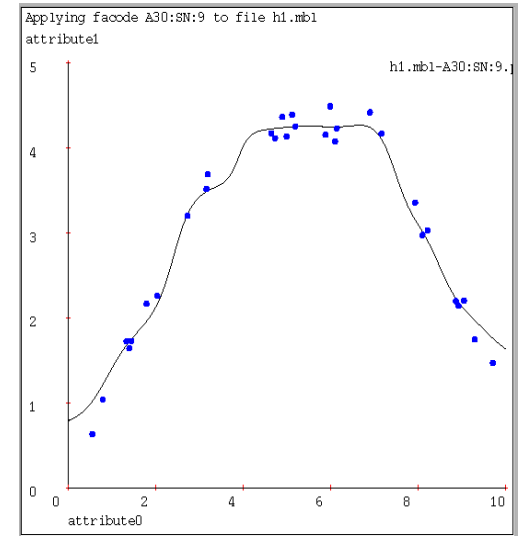
Kernel regression can look bad



KW = Best.



KW = Best.



KW = Best.

Time to try something more powerful...

Locally weighted regression



Kernel regression:

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

Locally weighted regression:

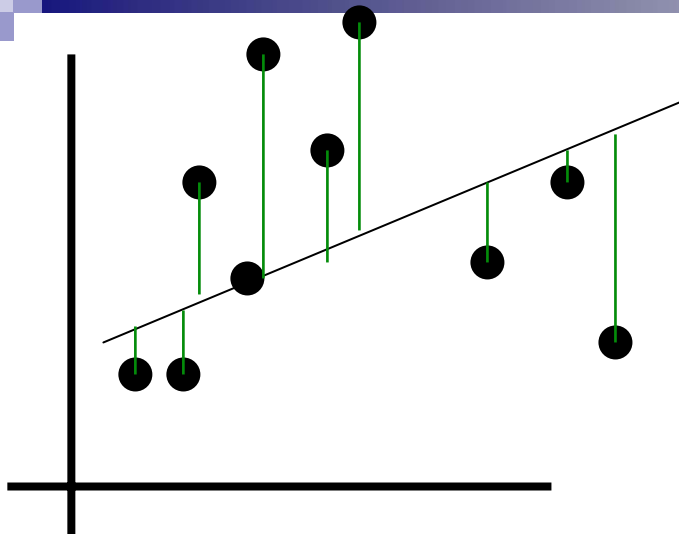
Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

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
 - $w_i = \exp(-D(x_i, \text{query})^2 / Kw^2)$
- *How to fit with the local points?*
General weighted regression:

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

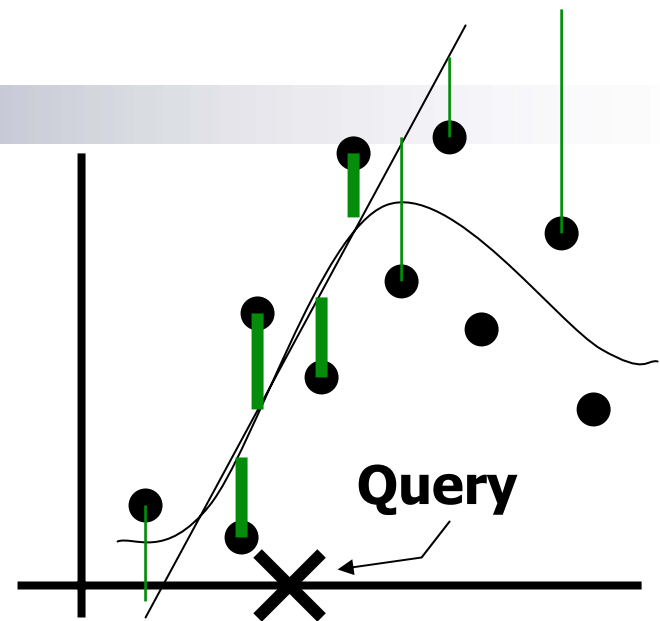
How LWR works



Linear regression

- Same parameters for all queries

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



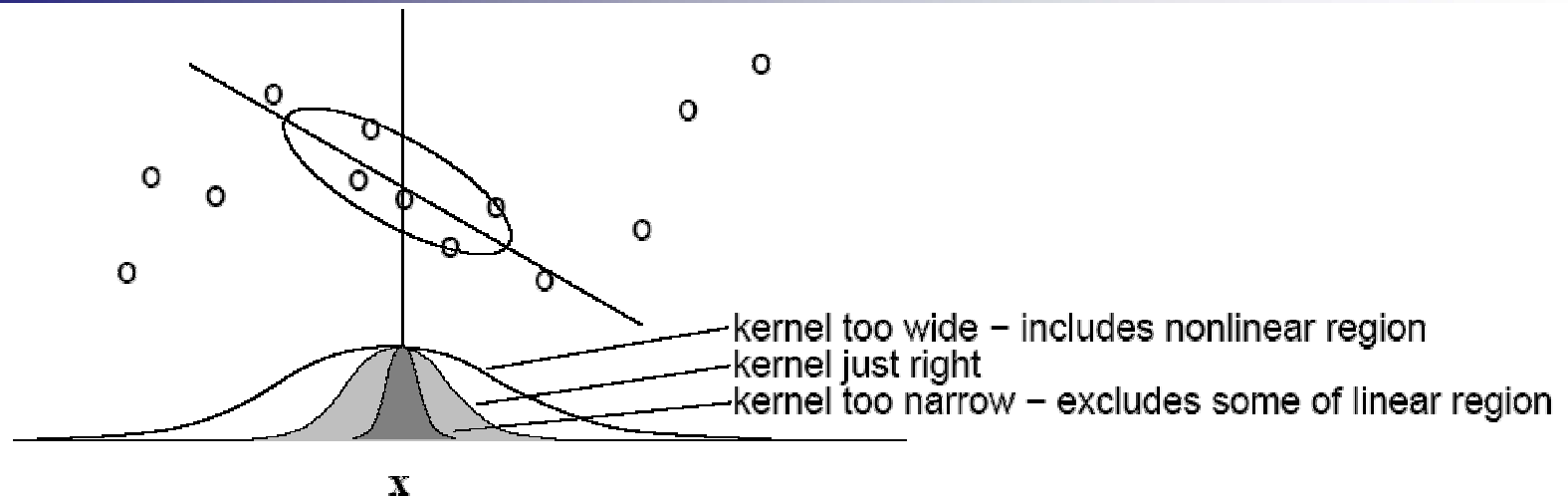
Locally weighted regression

- Solve weighted linear regression for each query

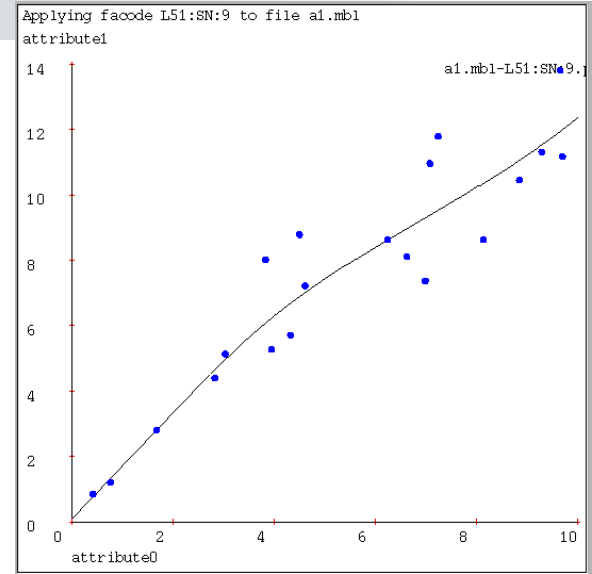
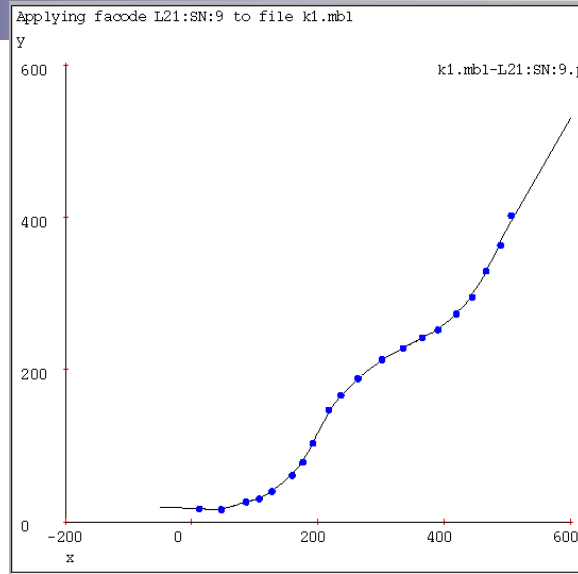
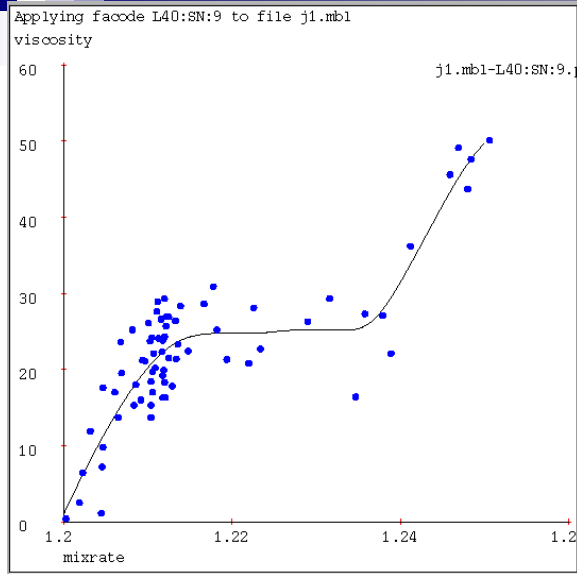
$$\hat{\beta} = (W X^T W X)^{-1} W X^T W Y$$

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

Another view of LWR



LWR on our test cases

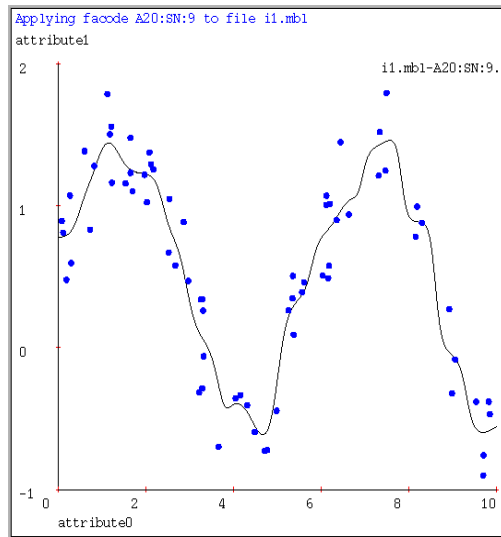


KW = 1/16 of x-axis width.

KW = 1/32 of x-axis width.

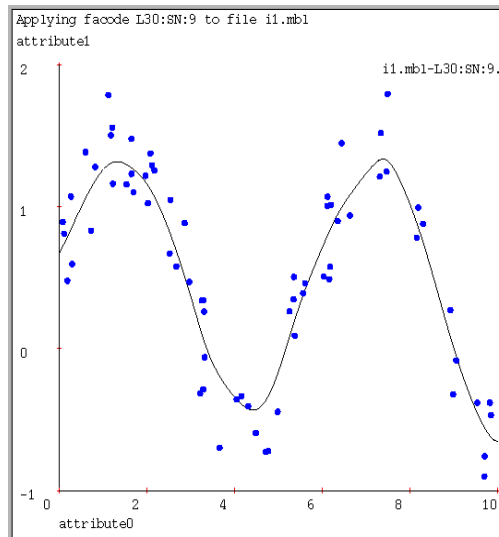
KW = 1/8 of x-axis width.

Locally weighted polynomial regression



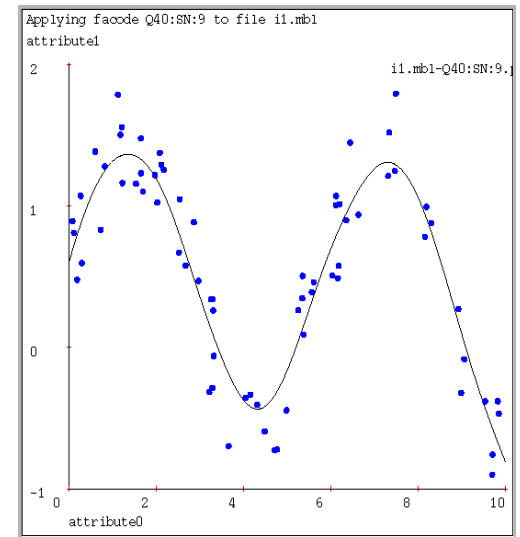
Kernel Regression
Kernel width K_W at optimal level.

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 retrieve 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



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
- 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>

**Two SVM tutorials linked in class website
(please, read both):**

- High-level presentation with applications (Hearst 1998)
- Detailed tutorial (Burges 1998)

Support Vector Machines

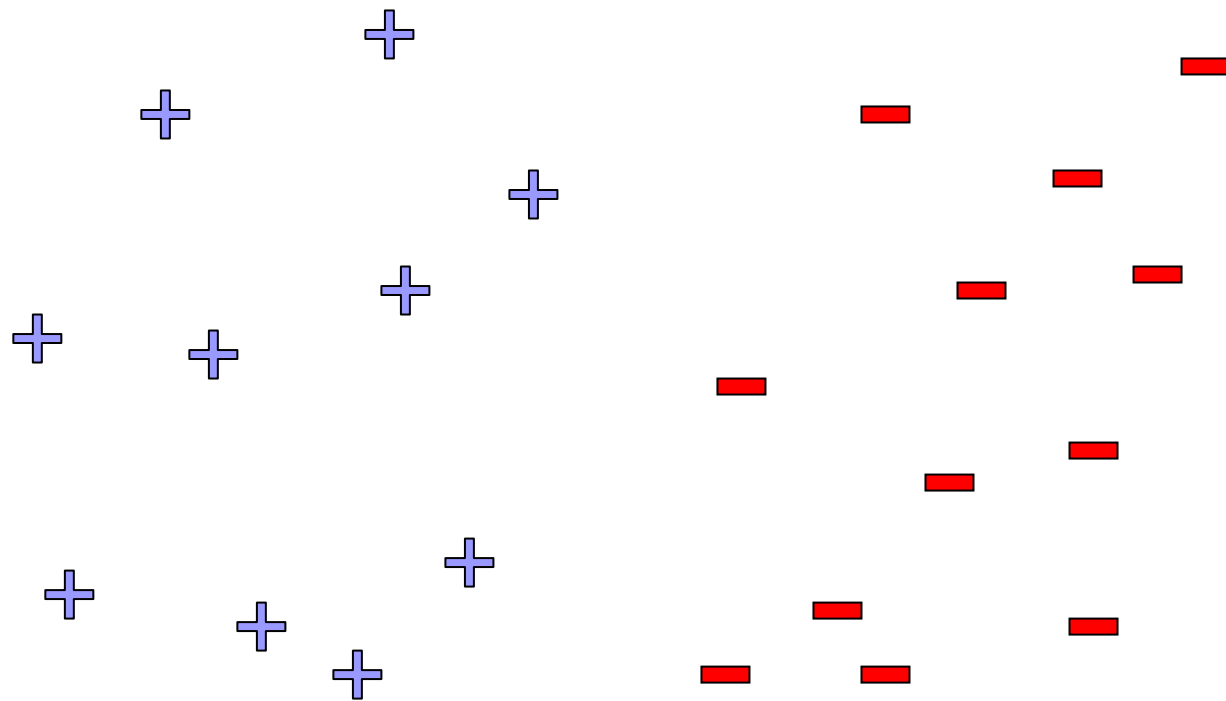
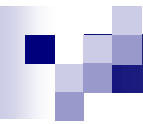
Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

February 20th, 2005

Linear classifiers – Which line is better?



Data:

$$\begin{aligned} &\langle x_1^{(1)}, \dots, x_1^{(m)}, y_1 \rangle \\ &\vdots \\ &\langle x_n^{(1)}, \dots, x_n^{(m)}, y_n \rangle \end{aligned}$$

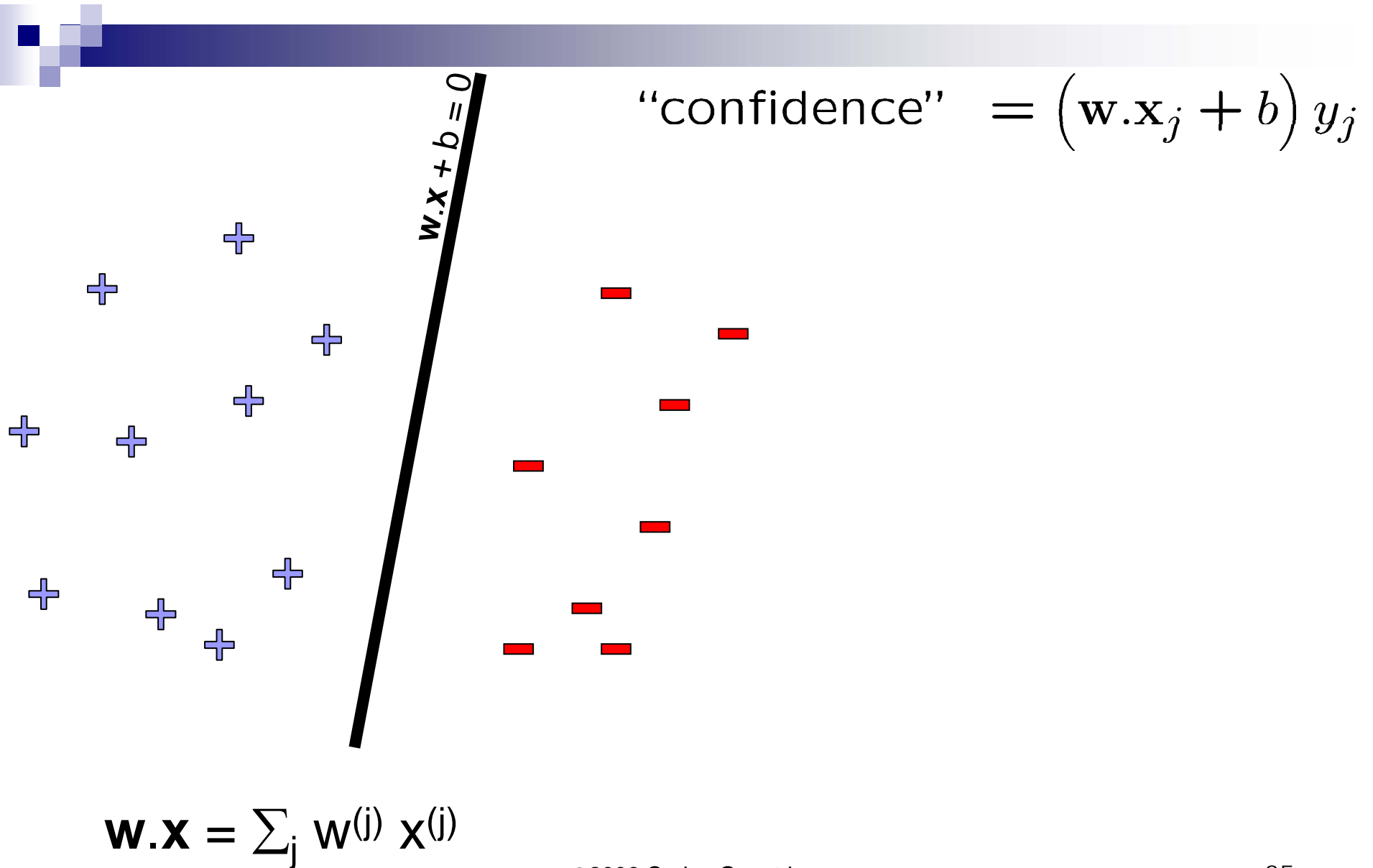
Example i:

$$\langle x_i^{(1)}, \dots, x_i^{(m)} \rangle \text{ — } m \text{ features}$$

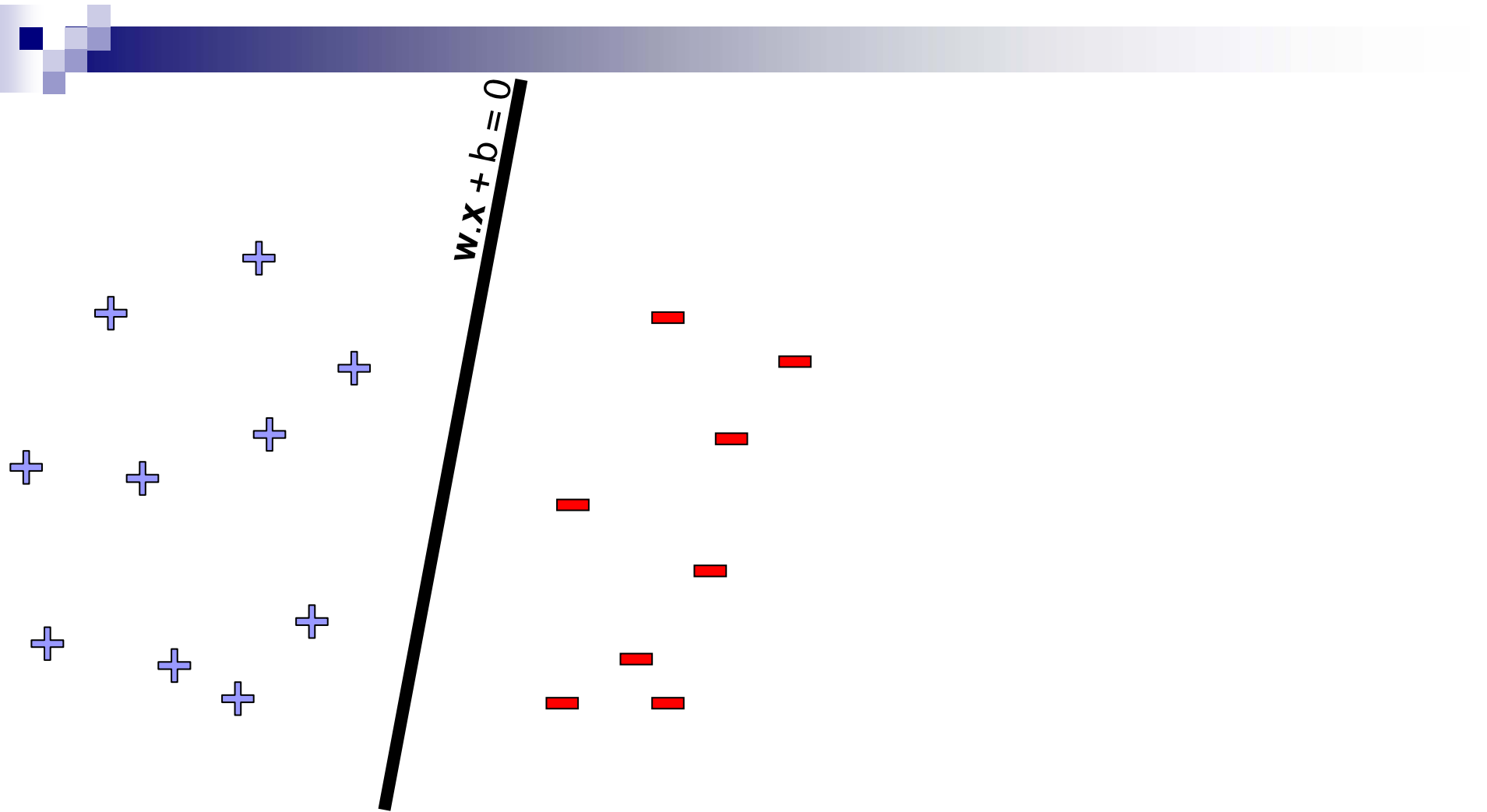
$$y_i \in \{-1, +1\} \text{ — class}$$

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

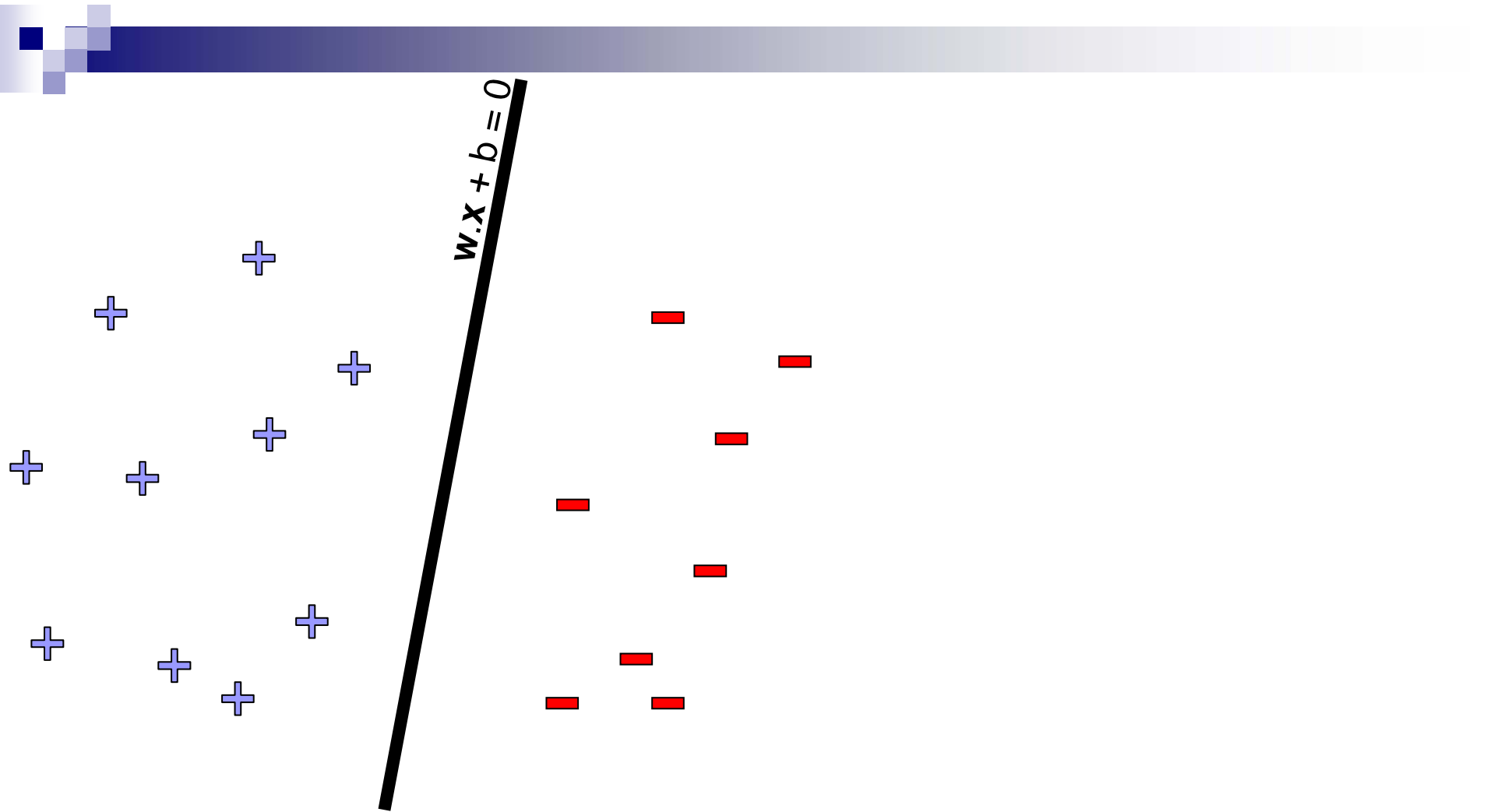
Pick the one with the largest margin!



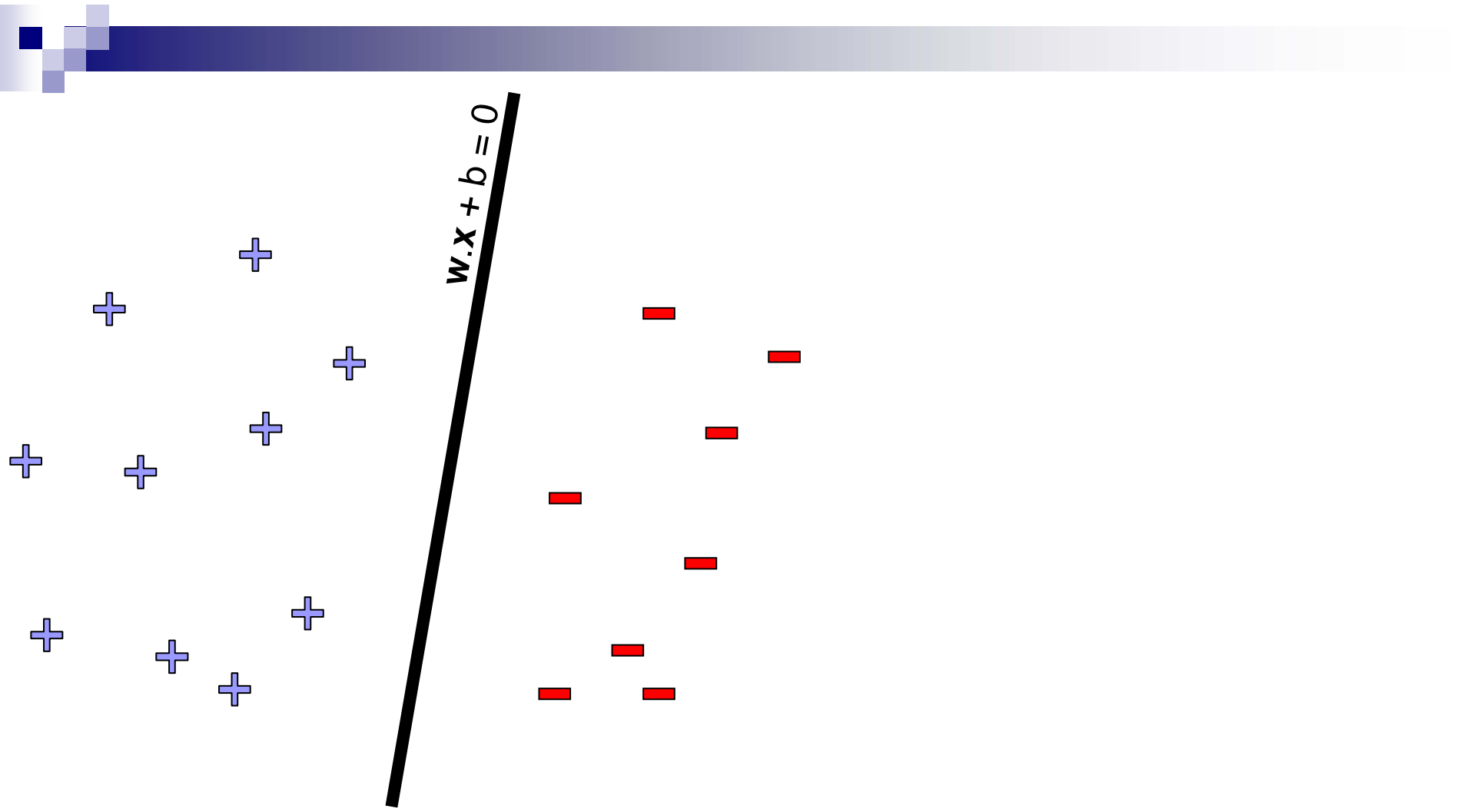
Maximize the margin



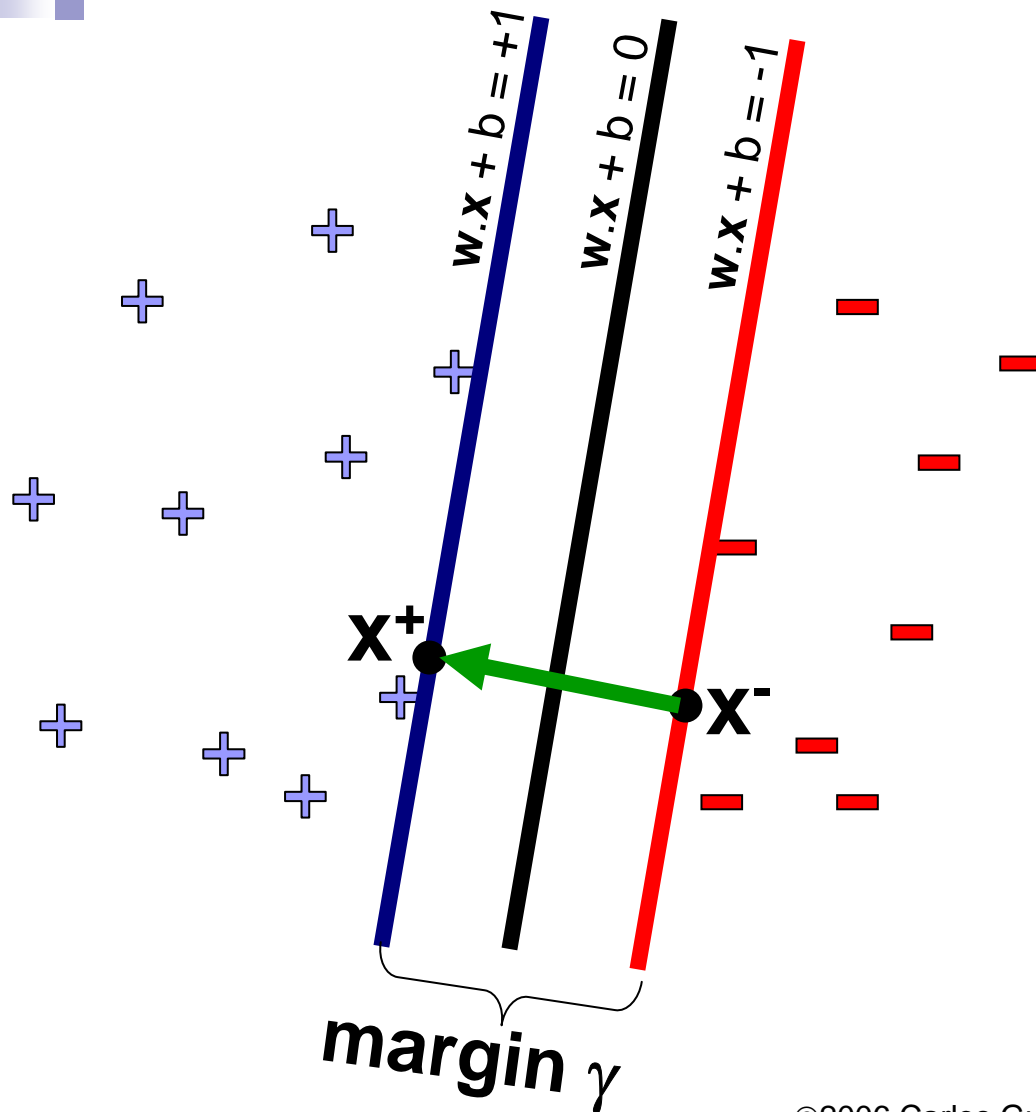
But there are a many planes...



Review: Normal to a plane

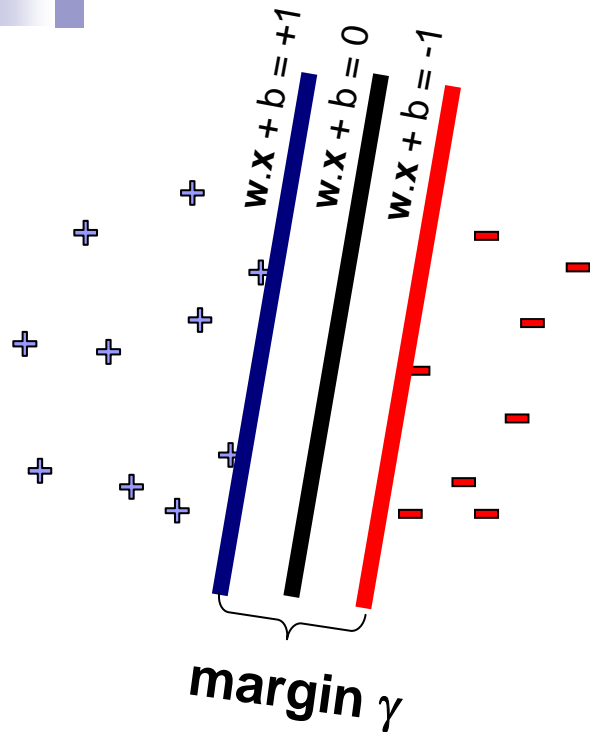


Normalized margin – Canonical hyperplanes



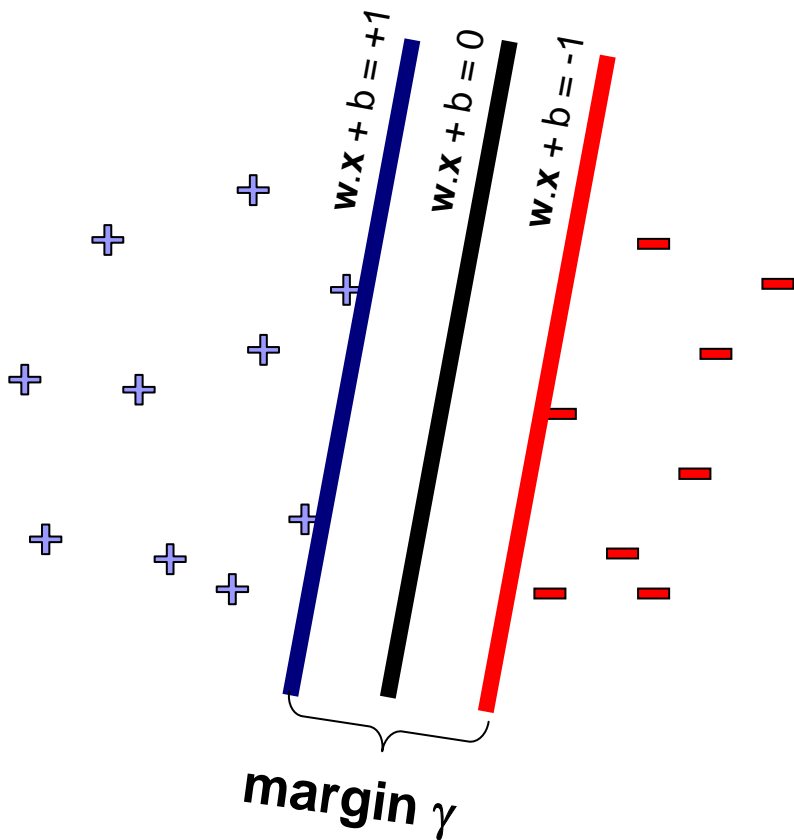
$$\gamma = \frac{2}{\sqrt{w \cdot w}}$$

Margin maximization using canonical hyperplanes



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

Support vector machines (SVMs)

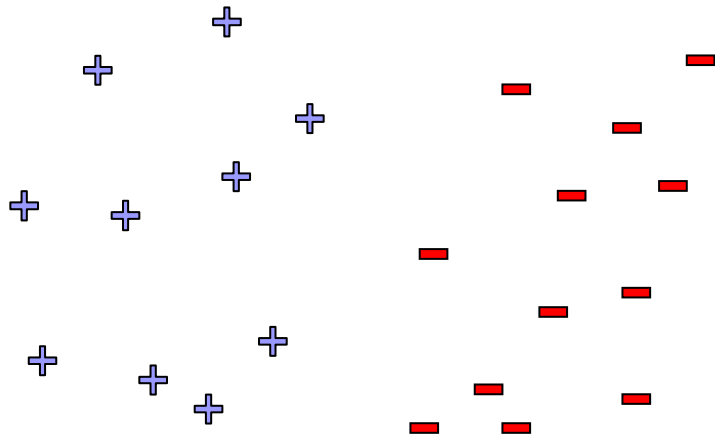


$$\text{minimize}_w \quad w \cdot w$$
$$\left(w \cdot x_j + b \right) y_j \geq 1, \quad \forall j$$

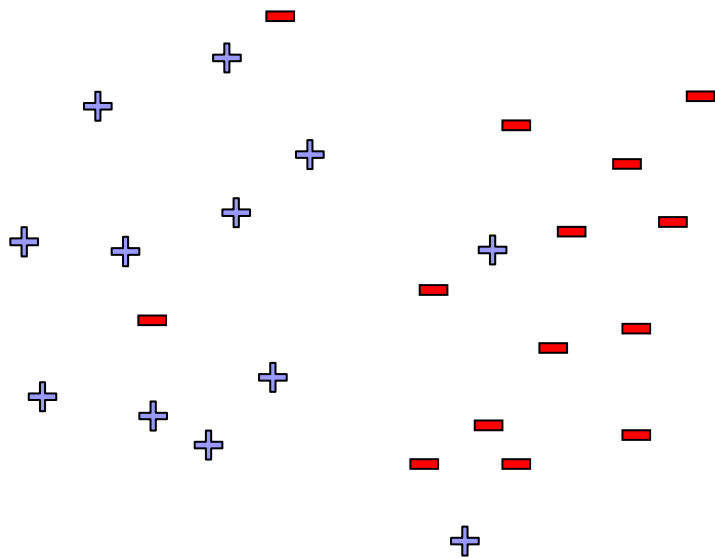
- Solve efficiently by quadratic programming (QP)
 - Well-studied solution algorithms
- Hyperplane defined by support vectors

What if the data is not linearly separable?

**Use features of features
of features of features....**



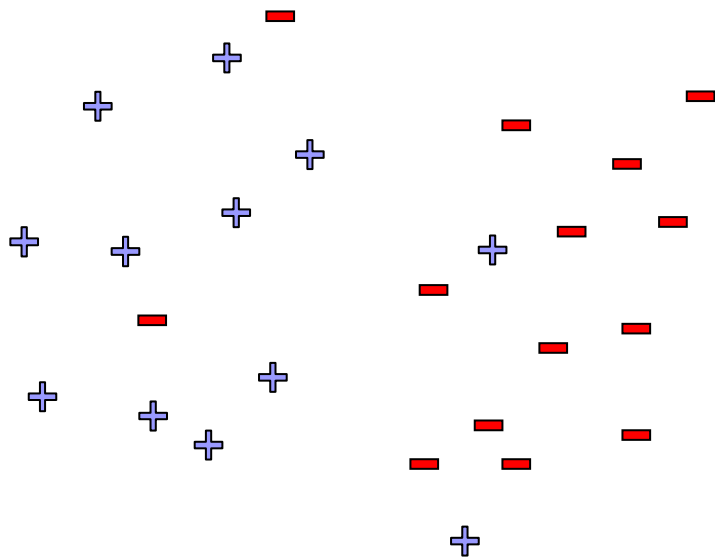
What if the data is still not linearly separable?



$$\begin{aligned} &\text{minimize}_{\mathbf{w}} \quad \mathbf{w} \cdot \mathbf{w} \\ &(\mathbf{w} \cdot \mathbf{x}_j + b) y_j \geq 1 \quad , \forall j \end{aligned}$$

- Minimize $\mathbf{w} \cdot \mathbf{w}$ and number of training mistakes
 - Tradeoff two criteria?
- Tradeoff #(mistakes) and $\mathbf{w} \cdot \mathbf{w}$
 - 0/1 loss
 - Slack penalty C
 - Not QP anymore
 - Also doesn't distinguish near misses and really bad mistakes

Slack variables – Hinge loss



$$\begin{aligned} &\text{minimize}_{\mathbf{w}} \quad \mathbf{w} \cdot \mathbf{w} \\ &(\mathbf{w} \cdot \mathbf{x}_j + b) y_j \geq 1 \quad , \forall j \end{aligned}$$

- If margin ≥ 1 , don't care
- If margin < 1 , pay linear penalty

Side note: What's the difference between SVMs and logistic regression?

SVM:

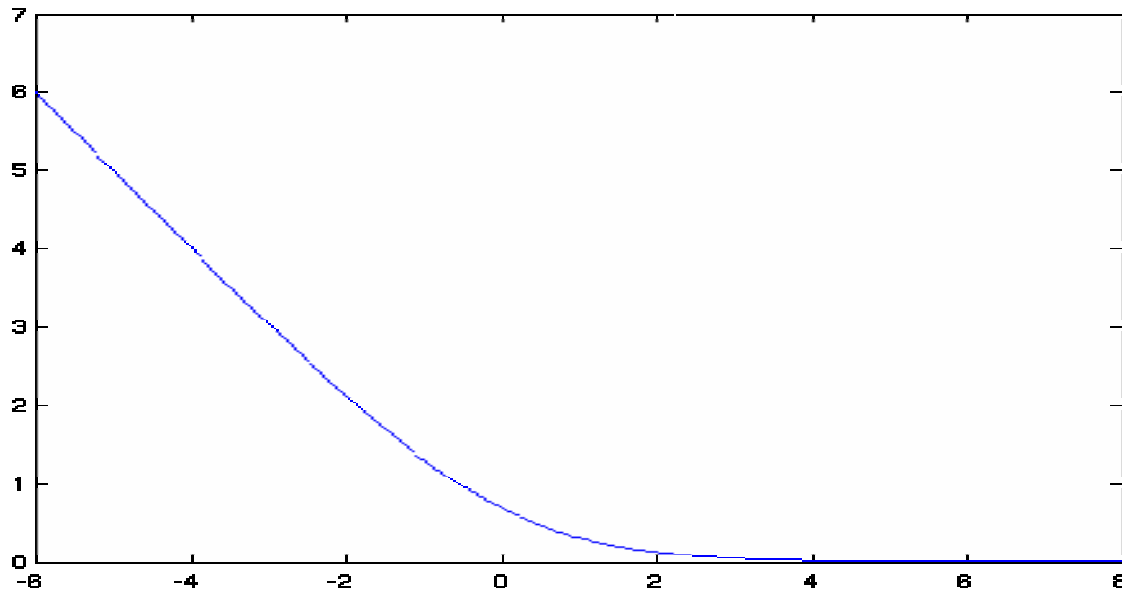
$$\begin{aligned} \text{minimize}_{\mathbf{w}} \quad & \mathbf{w} \cdot \mathbf{w} + C \sum_j \xi_j \\ & (\mathbf{w} \cdot \mathbf{x}_j + b) y_j \geq 1 - \xi_j, \quad \forall j \\ & \xi_j \geq 0, \quad \forall j \end{aligned}$$

Logistic regression:

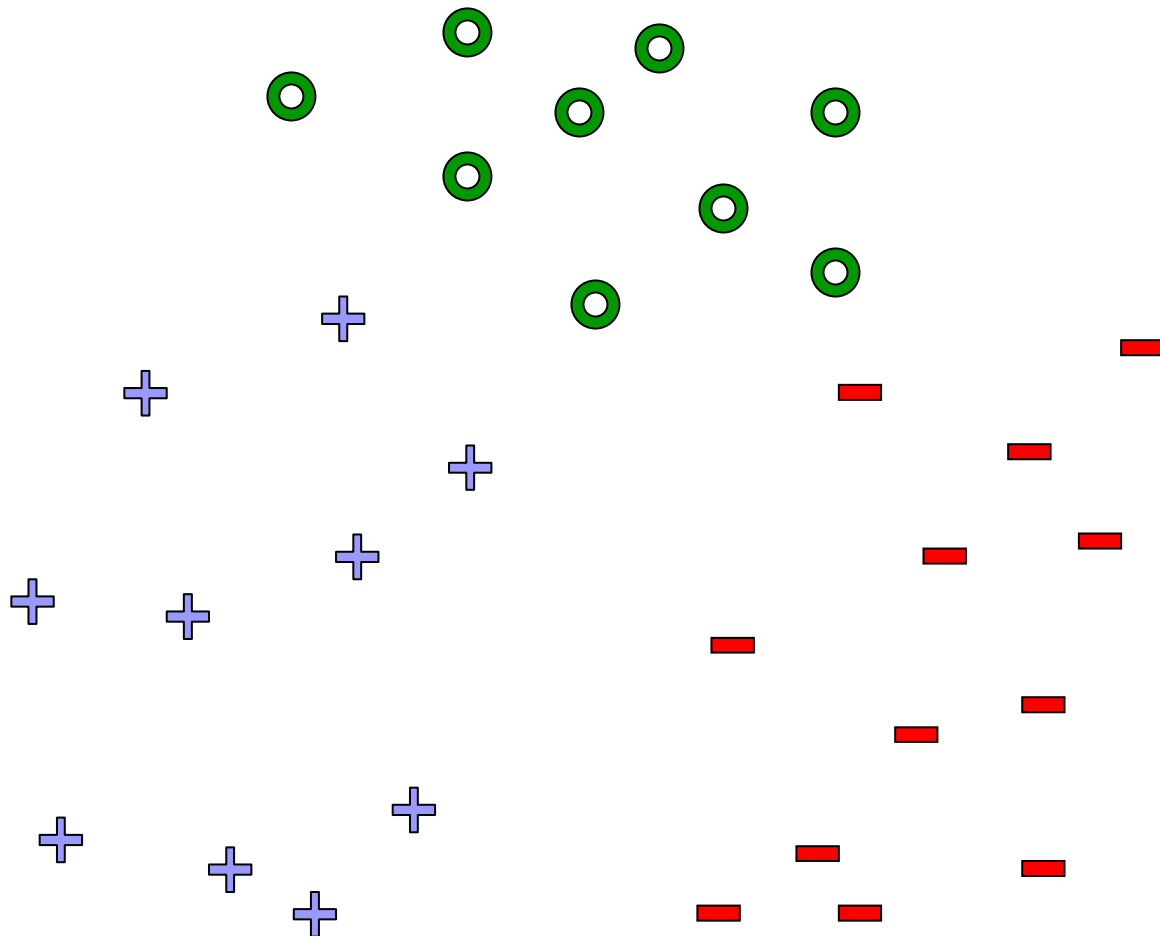
$$P(Y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

Log loss:

$$-\ln P(Y = 1 \mid \mathbf{x}, \mathbf{w}) = \ln(1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)})$$

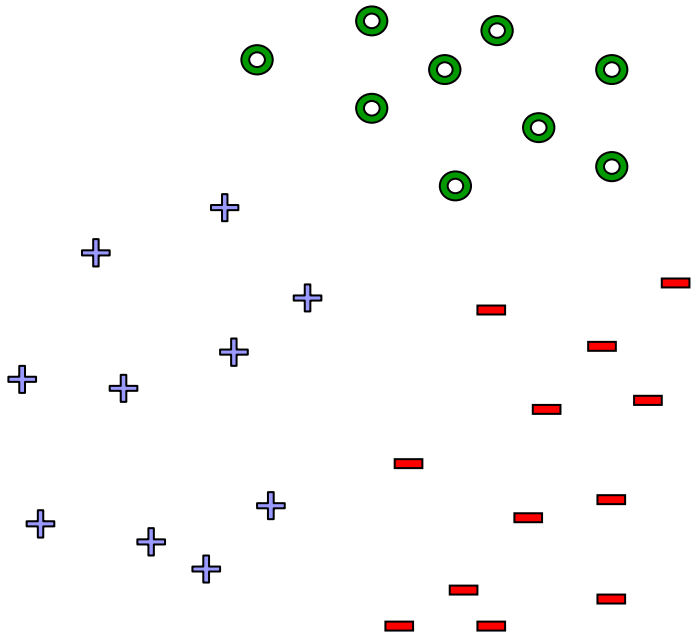


What about multiple classes?



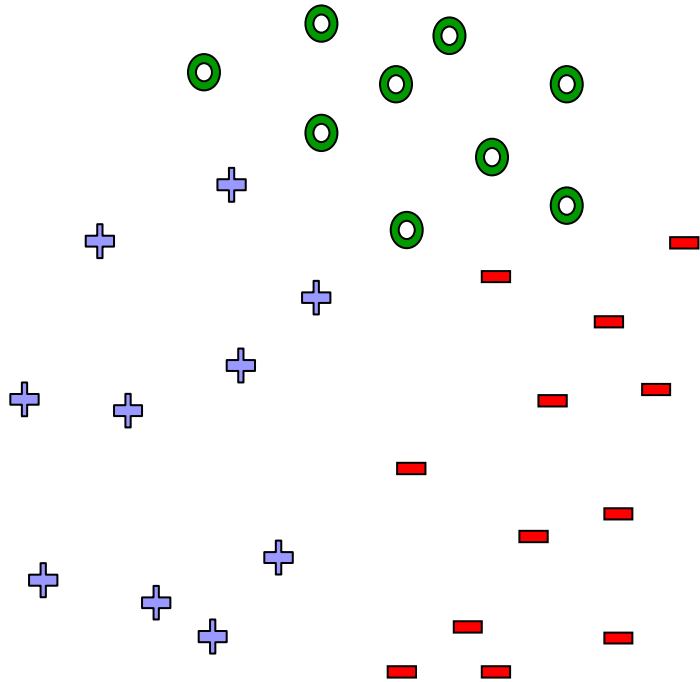
One against All

Learn 3 classifiers:



Learn 1 classifier: Multiclass SVM

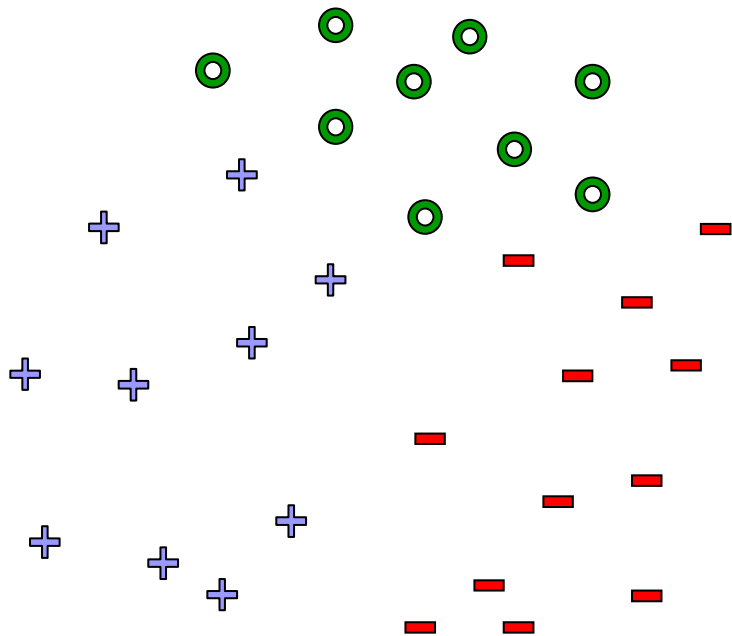
Simultaneously learn 3 sets of weights



$$\mathbf{w}^{(y_j)} \cdot \mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')} \cdot \mathbf{x}_j + b^{(y')} + 1, \quad \forall y' \neq y_j, \quad \forall j$$

Learn 1 classifier: Multiclass SVM

$$\begin{aligned} \text{minimize}_{\mathbf{w}} \quad & \sum_y \mathbf{w}^{(y)} \cdot \mathbf{w}^{(y)} + C \sum_j \xi_j \\ & \mathbf{w}^{(y_j)} \cdot \mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')} \cdot \mathbf{x}_j + b^{(y')} + 1 - \xi_j, \quad \forall y' \neq y_j, \quad \forall j \\ & \xi_j \geq 0, \quad \forall j \end{aligned}$$



What you need to know



- Maximizing margin
- Derivation of SVM formulation
- Slack variables and hinge loss
- Relationship between SVMs and logistic regression
 - 0/1 loss
 - Hinge loss
 - Log loss
- Tackling multiple class
 - One against All
 - Multiclass SVMs

Acknowledgment



- SVM applet:
 - <http://www.site.uottawa.ca/~gcaron/applets.htm>