Instance-based Learning

Machine Learning – 10701/15781

Carlos Guestrin
Carnegie Mellon University

February 20th, 2006

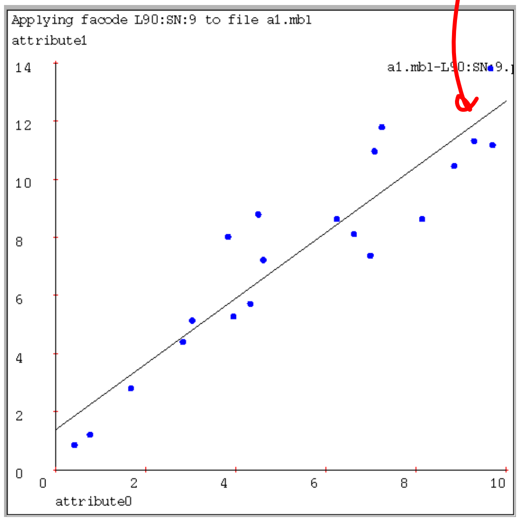
©2006 Carlos Guestrin

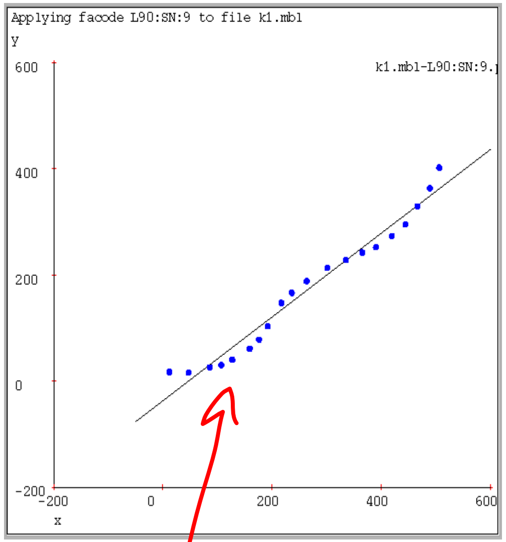
Announcements

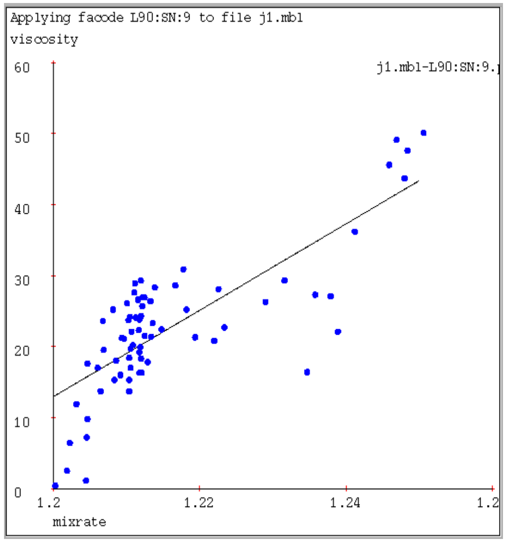
- Third homework
  - Out later today
  - Due March 1st

Why not just use Linear Regression?

$ax+b$







pretty good

missing trend

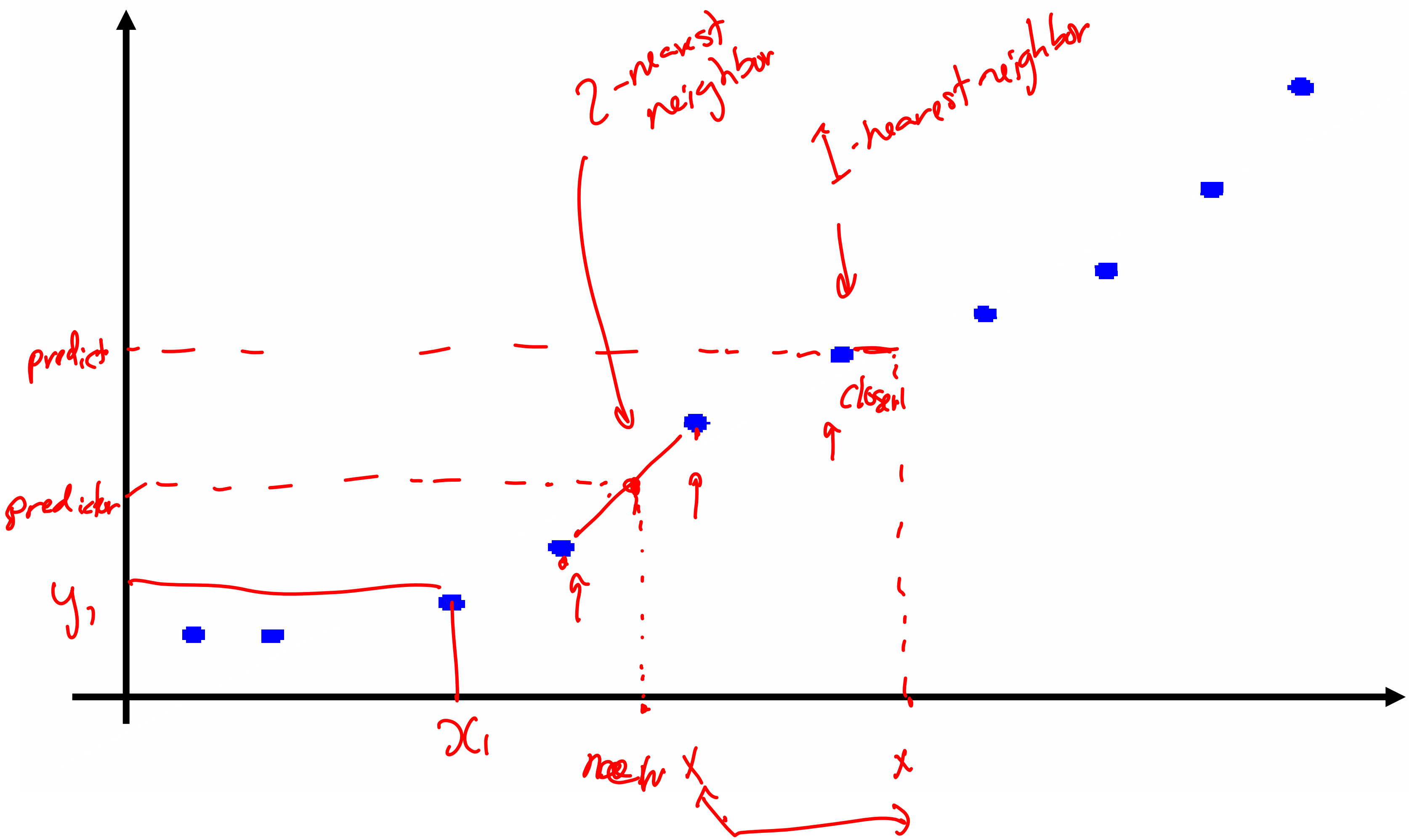
Could more basis functions

bad job!

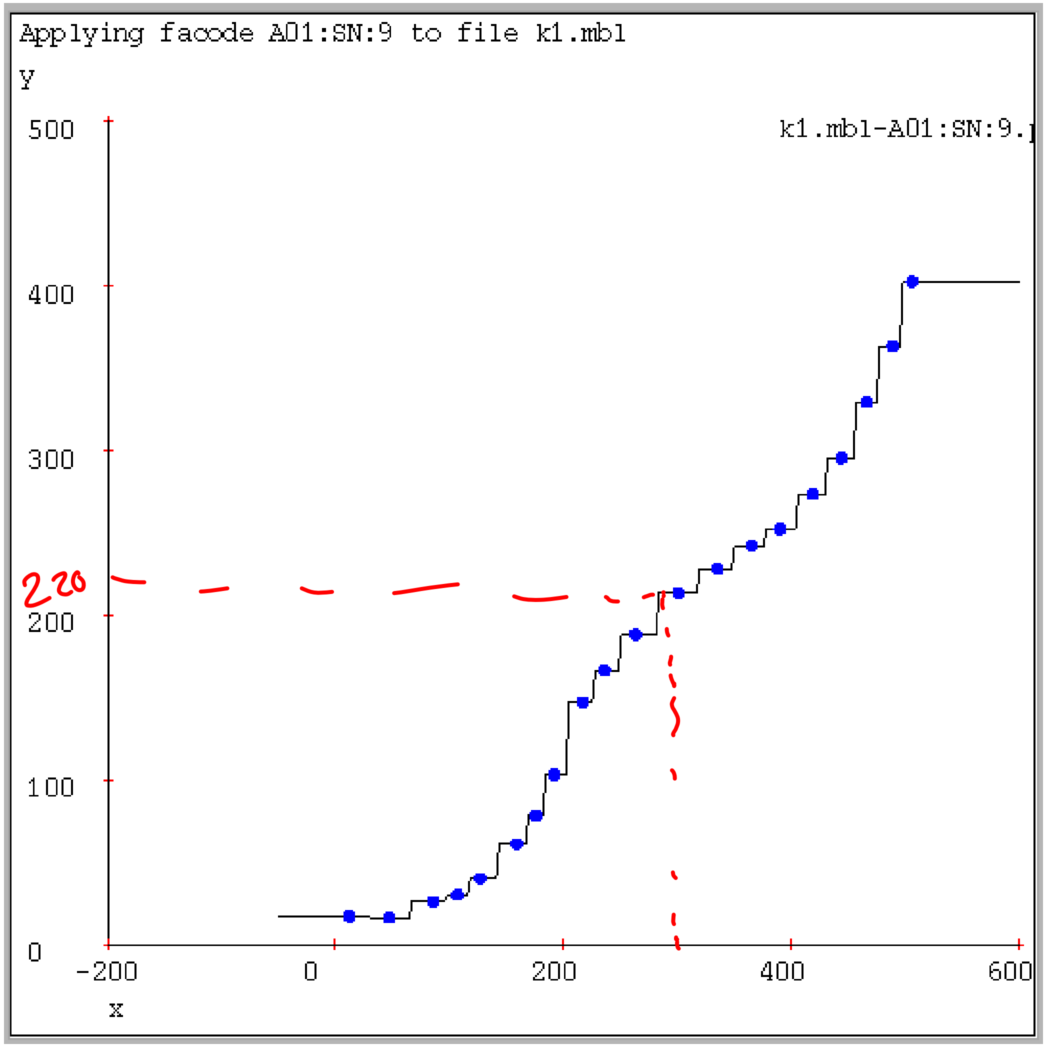
not sure what do...

lots more basis functions

Using data to predict new data



1-Nearest neighbor



Univariate 1-Nearest Neighbor

Given datapoints (x_1,y_1) (x_2,y_2)..(x_N,y_N),where we assume $y_i=f(\cancel{x_i})$ for some unknown function f.

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

Nearest Neighbor:

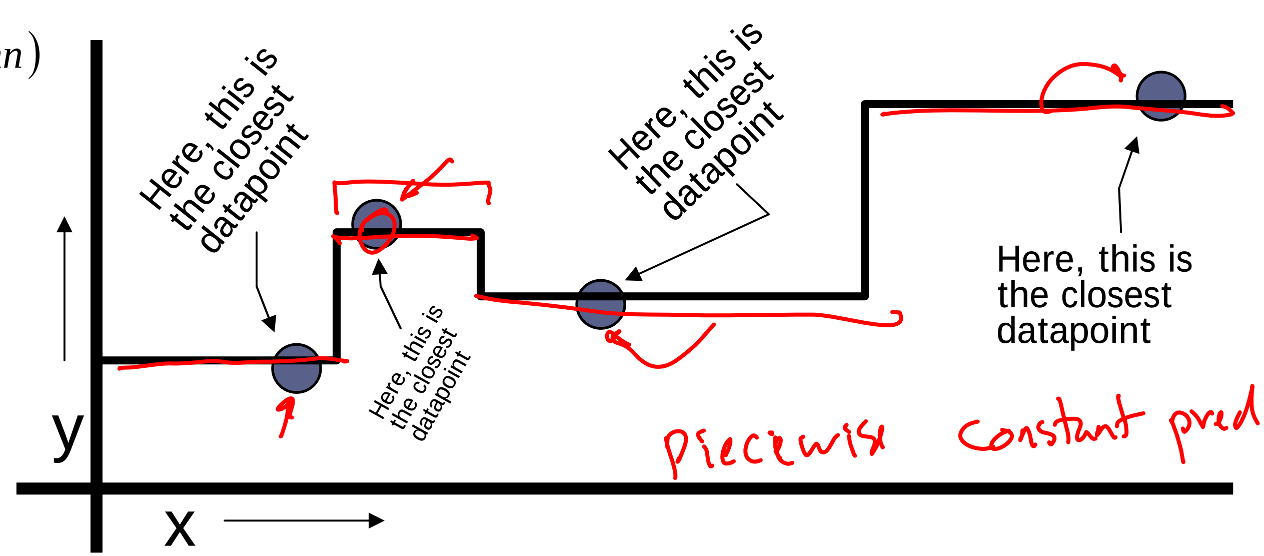
1. Find the closest x_i in our set of datapoints

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

closest in dataset

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

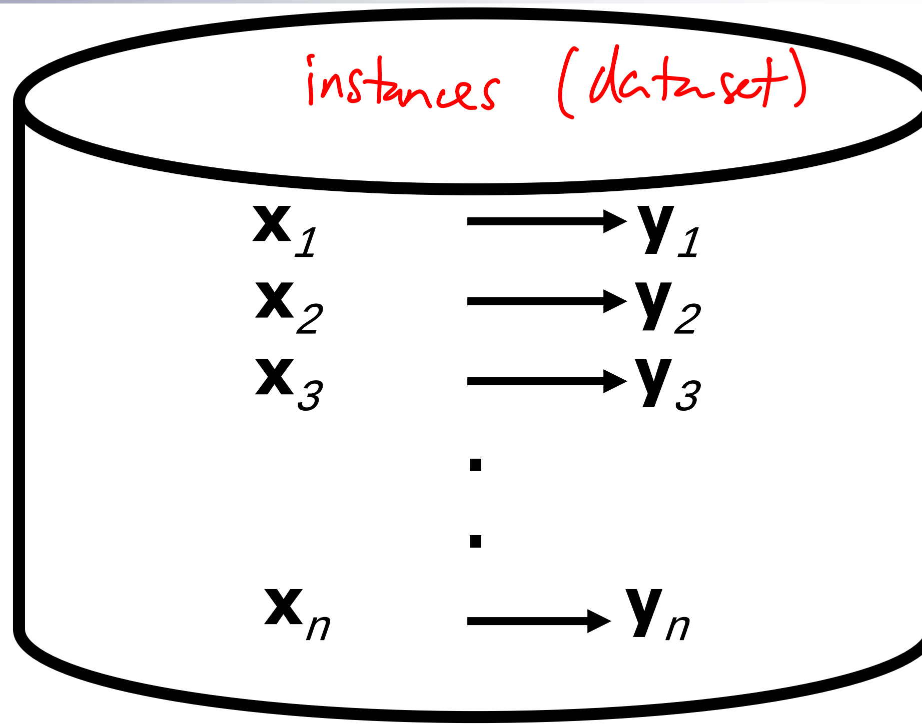
Here's a dataset with one input, one output and four datapoints.



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:

eucledicen distance

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?*

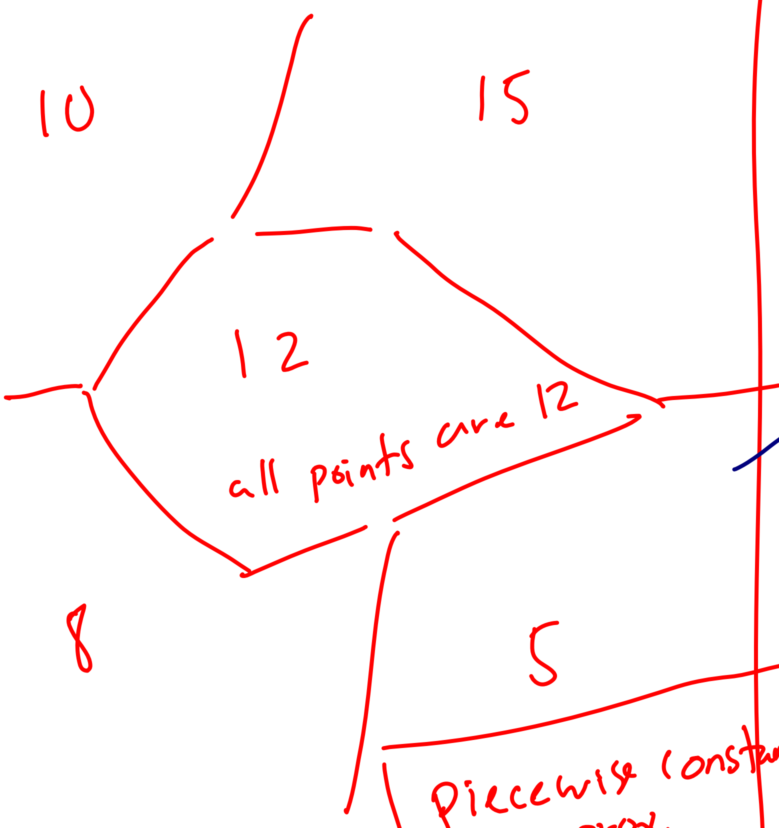
   Just predict the same output as the nearest neighbor.

output y_{i^*}

nearest-neighbor

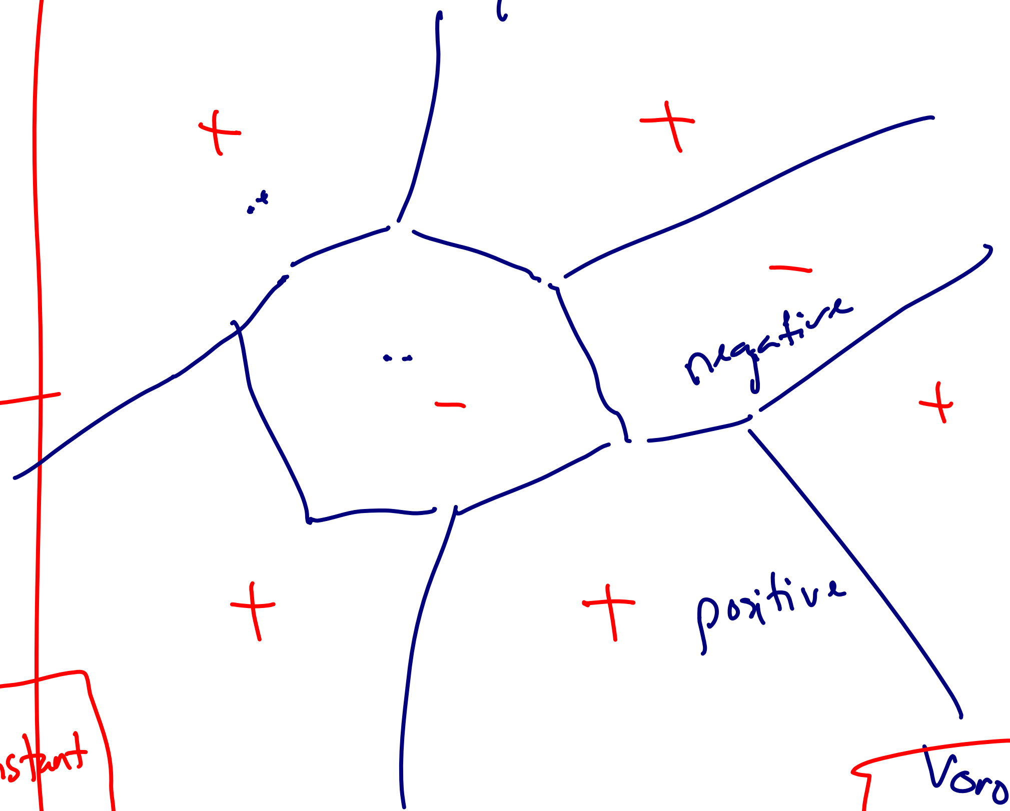
Multivariate 1-NN examples

Regression



Piecewise constant approx.

Classification



Voronoi Diagram

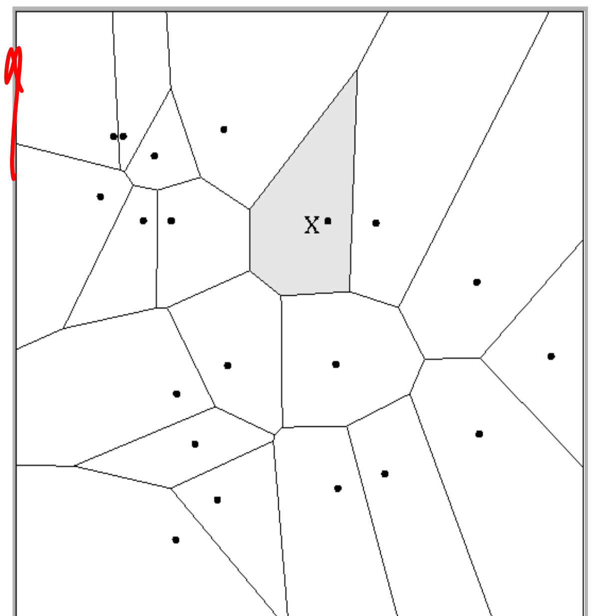
Multivariate distance metrics

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

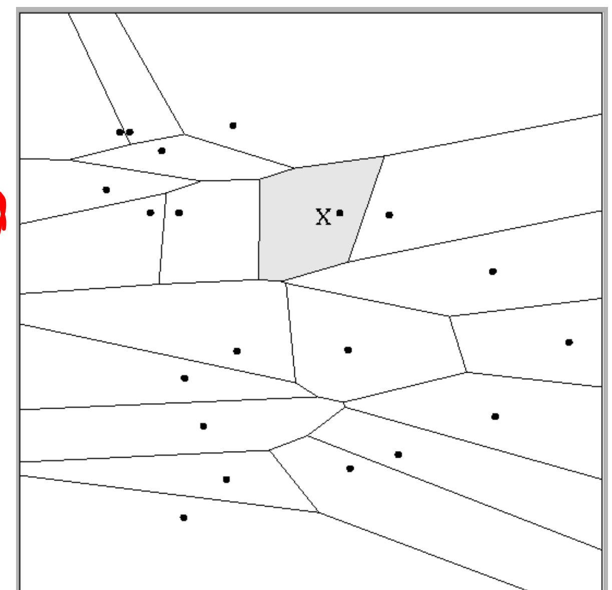
$\mathbf{x}_1 = (x_{11}, x_{12})$, $\mathbf{x}_2 = (x_{21}, x_{22})$, …$\mathbf{x}_N = (x_{N1}, x_{N2})$.

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

x2



x2



Euclidean

x1

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

x1

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

weight 1

The relative scalings in the distance metric affect region shapes.

Euclidean distance metric

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

weighted euclidean

Or equivalently,

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

where

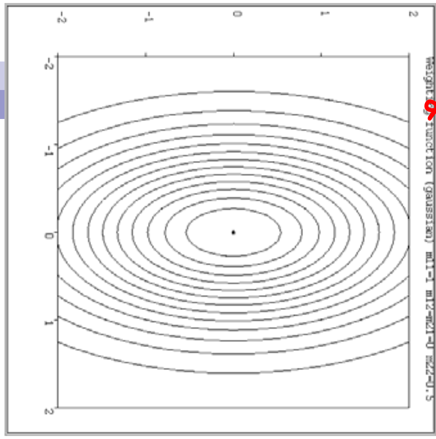
$$\sum = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{bmatrix}$$

reduce effect of features with different ranges / variance

Other Metrics…

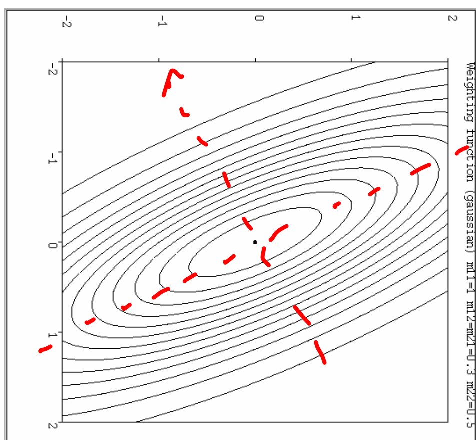
- Mahalanobis, Rank-based, Correlation-based,…

Notable distance metrics (and their level sets)



equi-distant points

Scaled Euclidian (L_2)

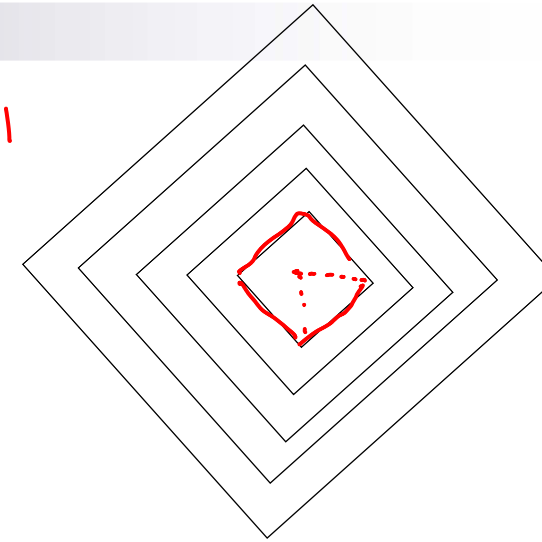


tilted Euclidian

correlated inputs

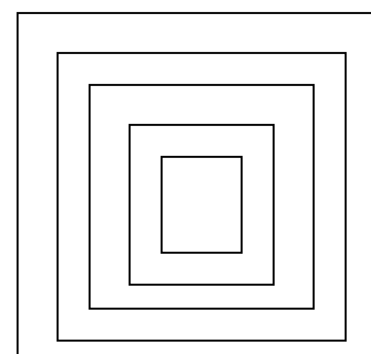
Mahalanobis
(here, Σ on the previous slide is not necessarily diagonal, but is symmetric

$\|x\|_1 = \sum_i |x_i|$



L_1 norm (absolute)

$\|x\|_\infty = \max_i |x_i|$



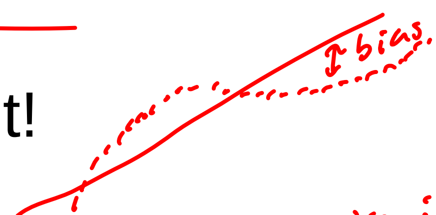
L_∞ (max) norm

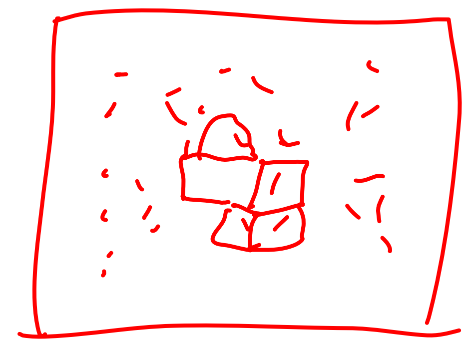
Consistency of 1-NN

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

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

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





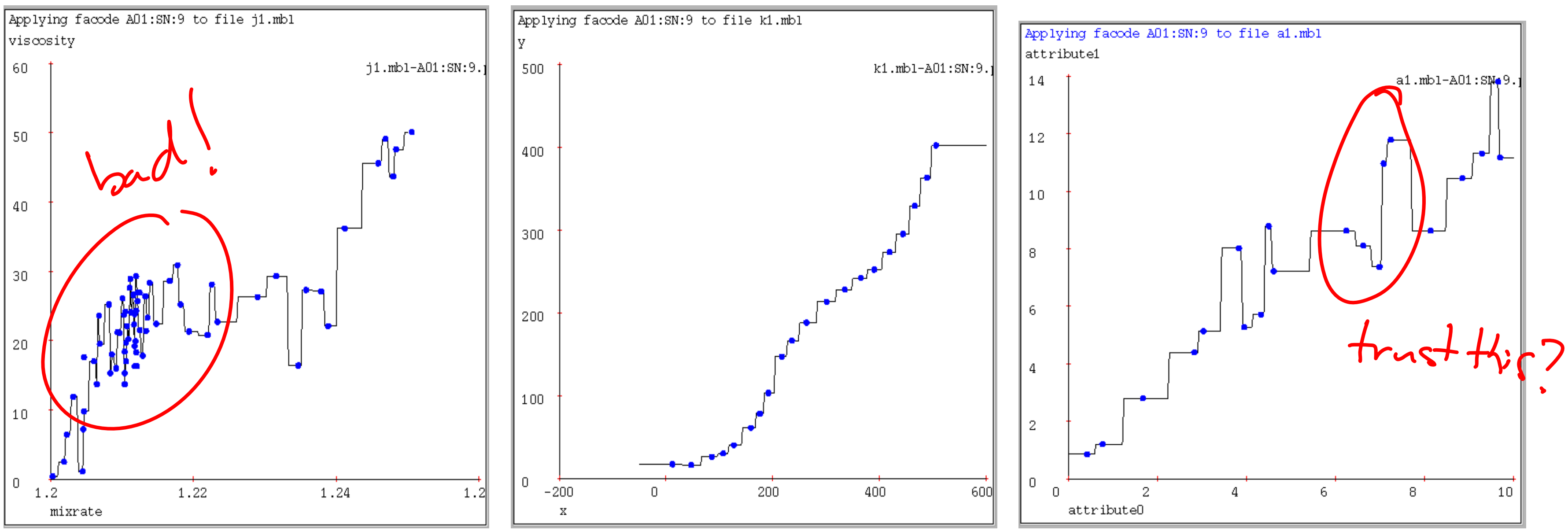
no noise in labels

as data → ∞, test error → 0

What about variance???

1-NN → lots variance

1-NN overfits?



good!

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

k neighbors to new x are x_1, y_1
x_2, y_2
$\vdots$

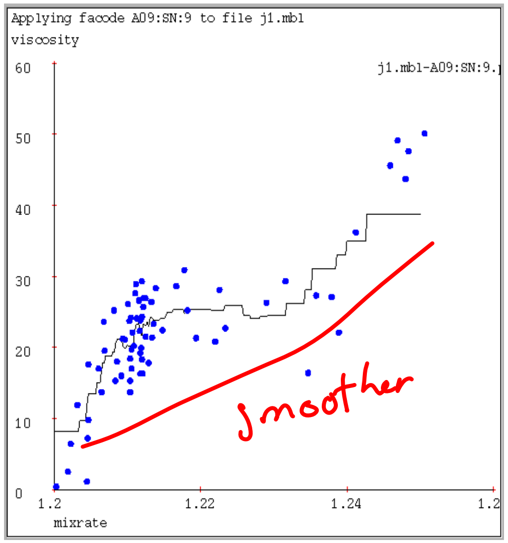
1. *A weighting function (optional)*
   Unused

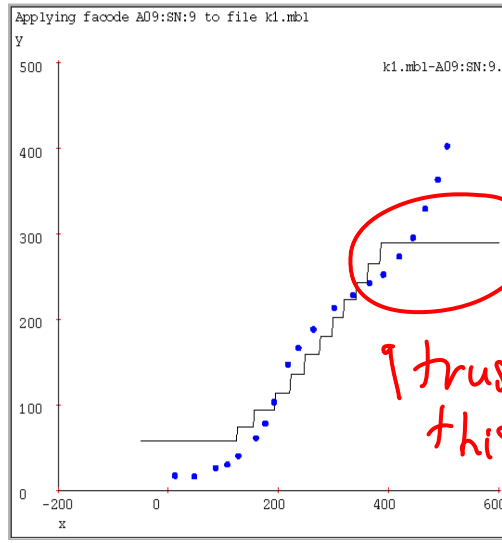
2. *How to fit with the local points?*
   Just predict the average output among the k nearest neighbors.

new x:

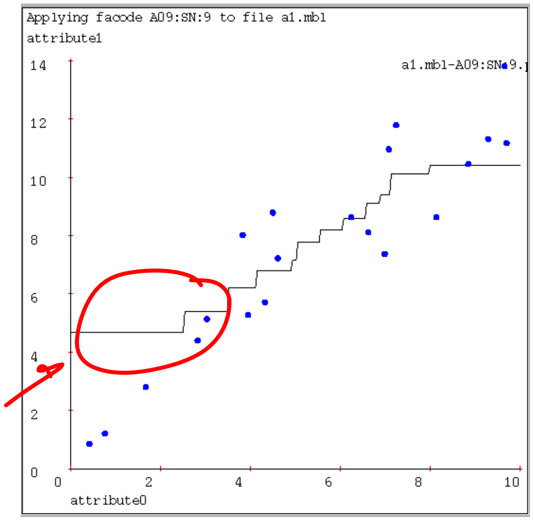
$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

k-Nearest Neighbor (here k=9)





I trust this?

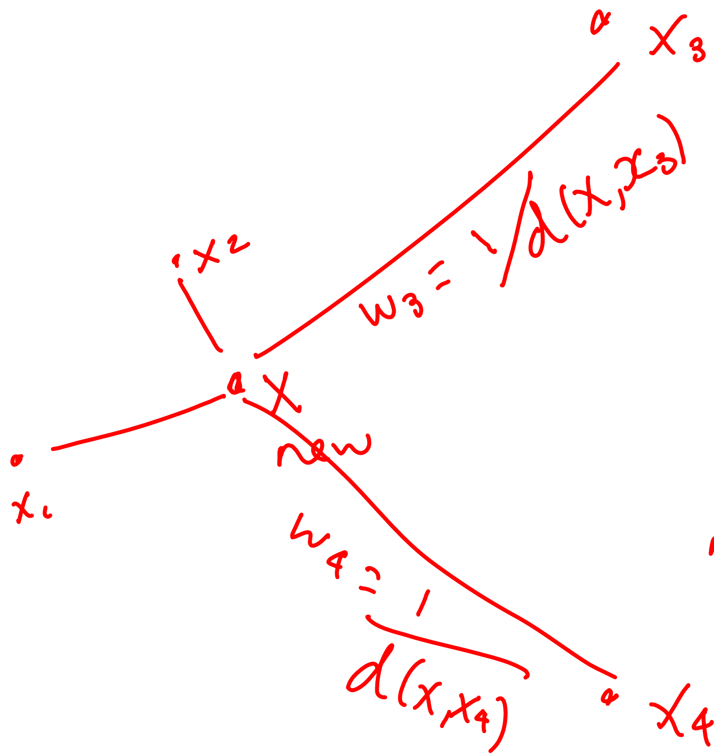


K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

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

Weighted k-NNs

- Neighbors are not all the same

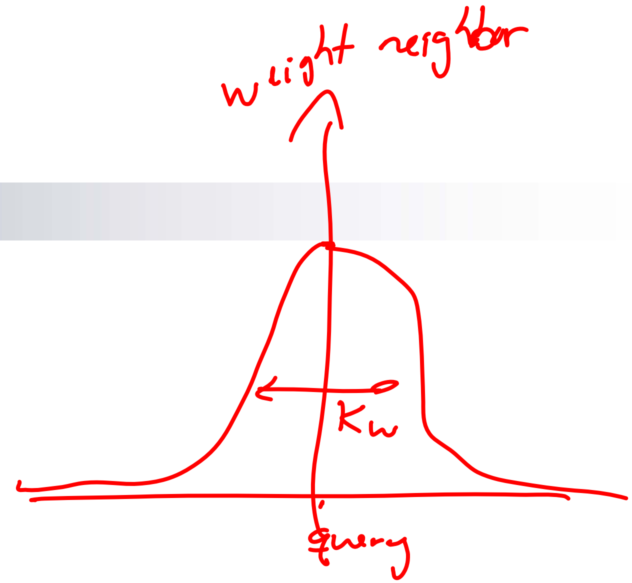


$$\hat{y} = \frac{\sum_{i=1}^{K} w_i y_i}{\sum_{i=1}^{K} w_i}$$

normalization

©2006 Carlos Guestrin

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)* Kernel function
   $w_i = \exp(-D(x_i, query)^2 / K_w^2)$
   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.
4. *How to fit with the local points?*
   Predict the weighted average of the outputs:
   predict = $\Sigma w_i y_i / \Sigma w_i$

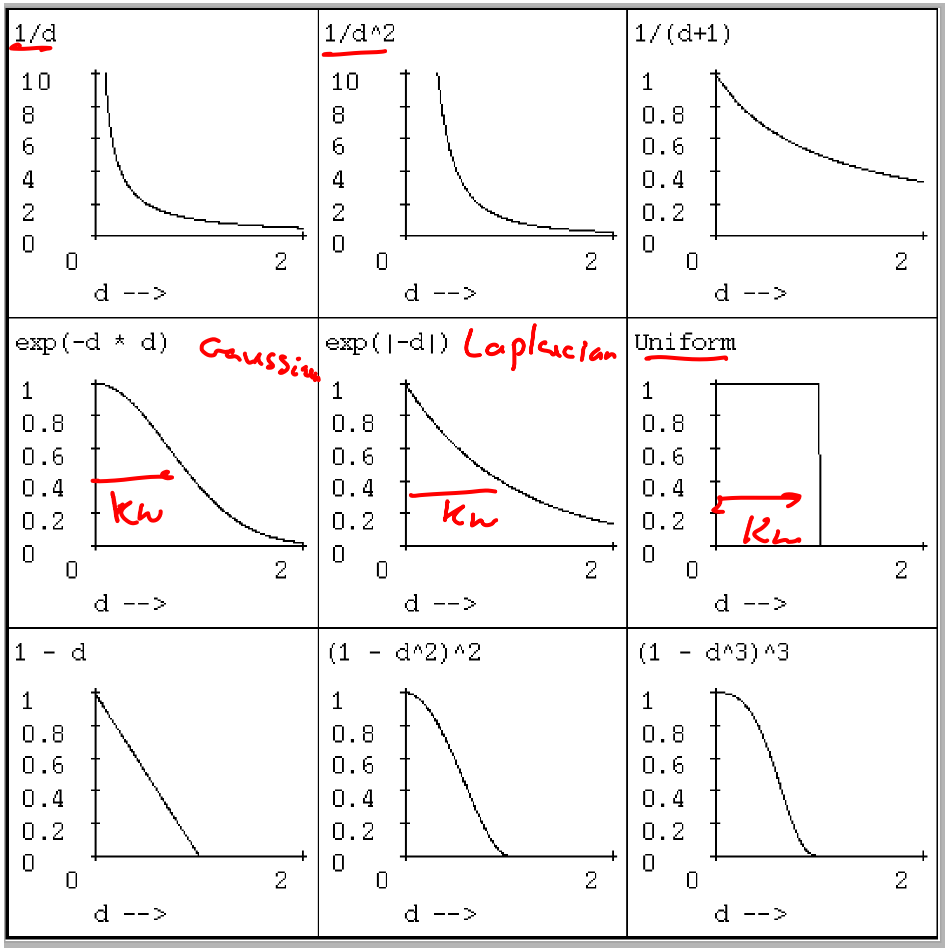
normalized weighted average

Weighting functions

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

typically:

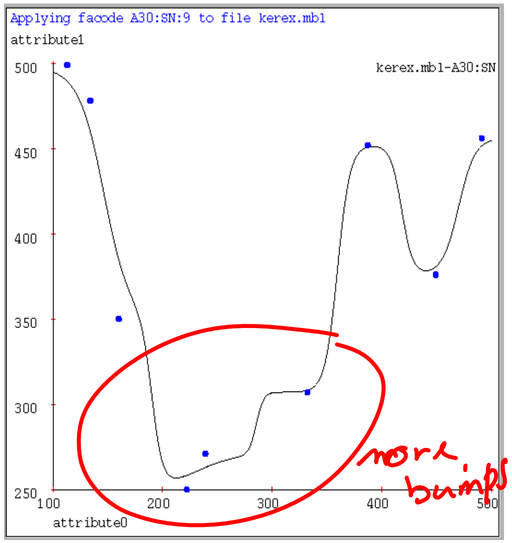
- symmetric
- decays with distance.



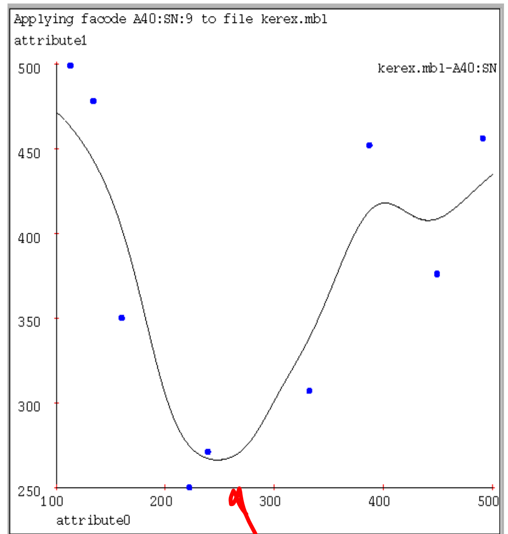
Typically optimize K_w using gradient descent

(Our examples use Gaussian)

Kernel regression predictions

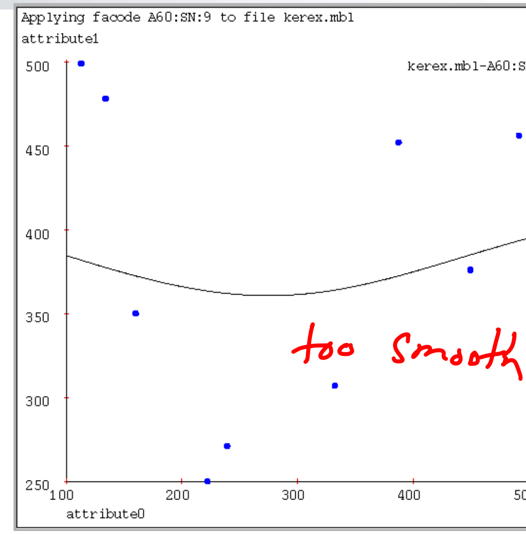


K_W=10 small



K_W=20

(almost) just right

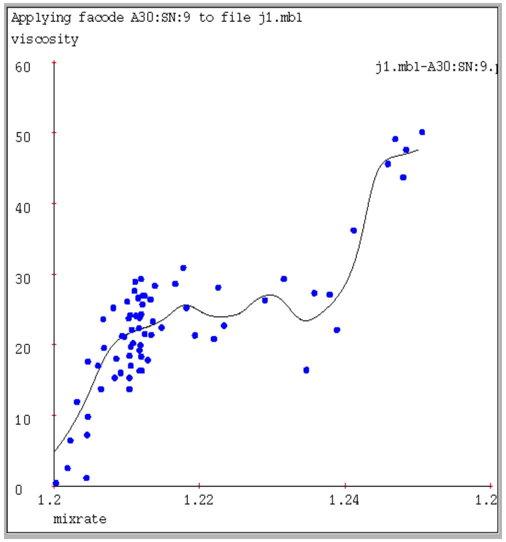


K_W=80 large

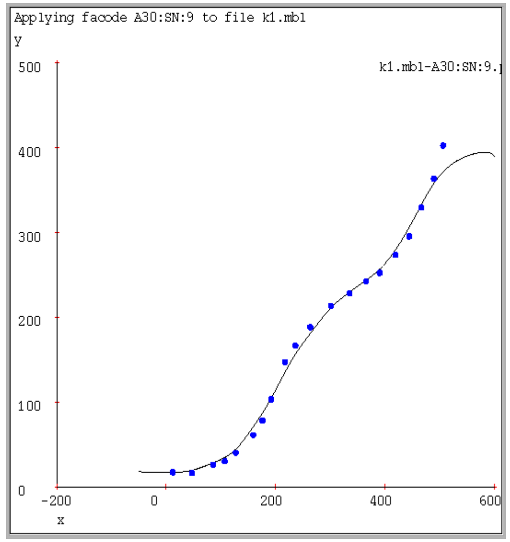
Increasing the kernel width K_w means further away points get an opportunity to influence you.

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

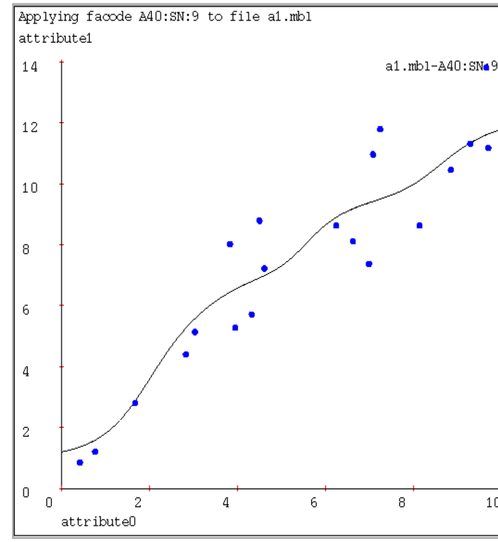
Kernel regression on our test cases



KW=1/32 of x-axis width.



KW=1/32 of x-axis width.



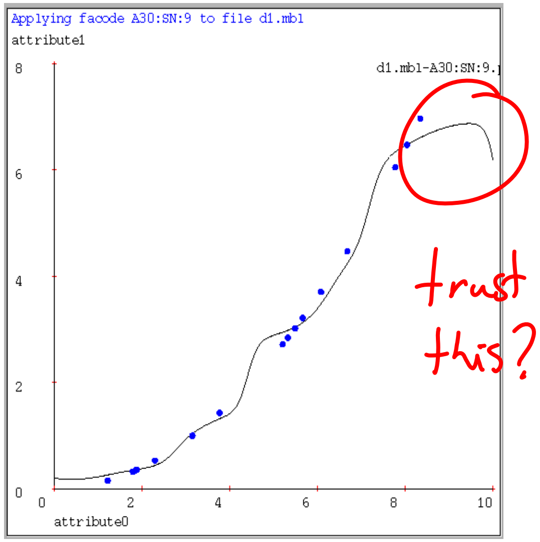
KW=1/16 axis width.

quite good

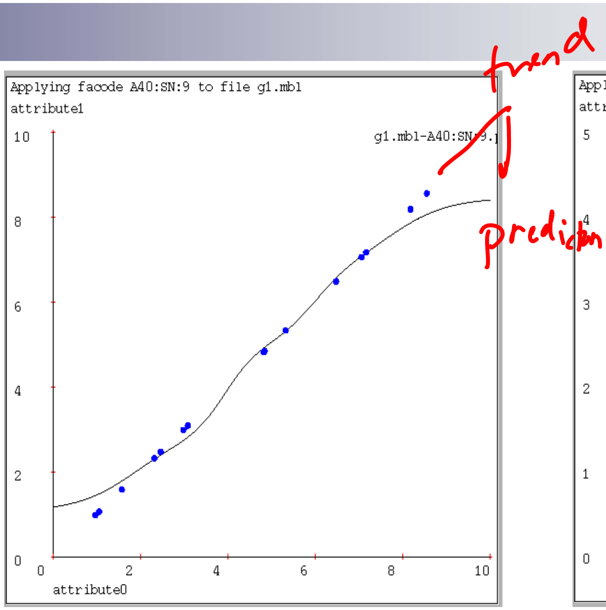
Choosing a good K_w is important. Not just for Kernel Regression, but for all the locally weighted learners we're about to see.

©2006 Carlos Guestrin

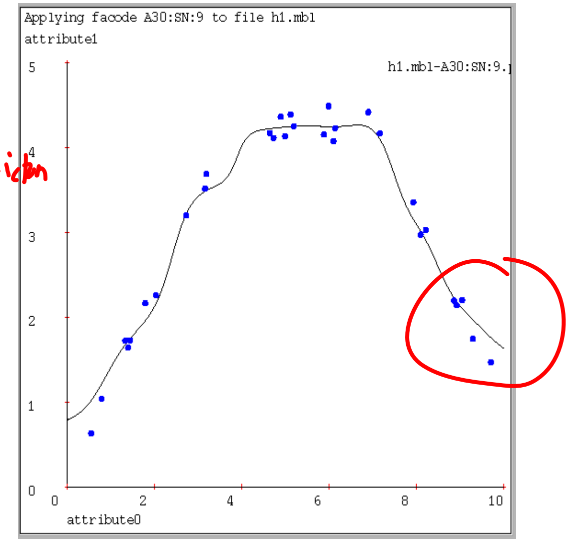
Kernel regression can look bad



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.

©2006 Carlos Guestrin

Locally weighted regression

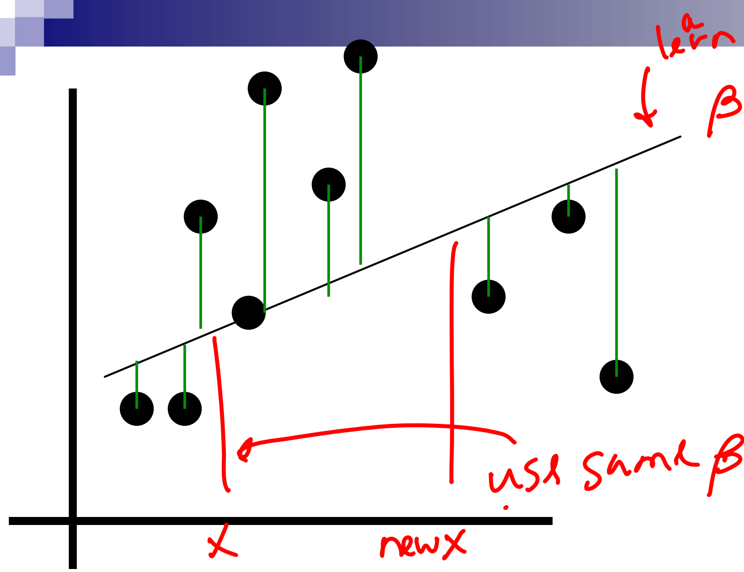
- **Four things make a memory based learner:**
- *A distance metric*
  - **Any**
- *How many nearby neighbors to look at?*
  - **All of them**
- *A weighting function (optional)*
  - **Kernels**
    - ***wi = exp(-D(xi, query)2 / Kw2)***
- *How to fit with the local points?*
  - **General weighted regression:**

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

weigh points

least squares (like linear regression)

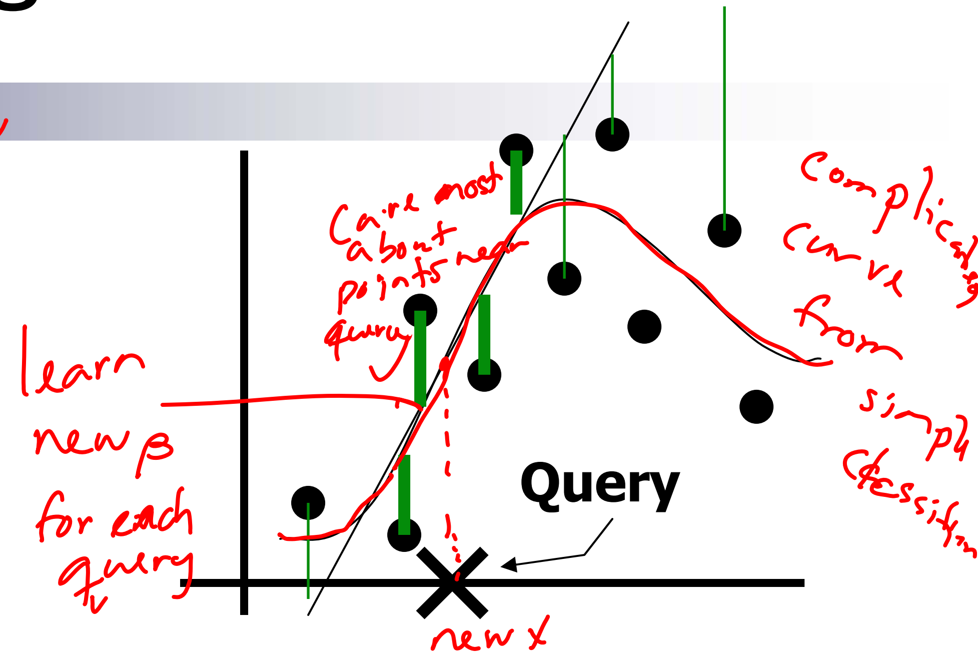
How LWR works



Linear regression

- Same parameters for all queries

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



Locally weighted regression

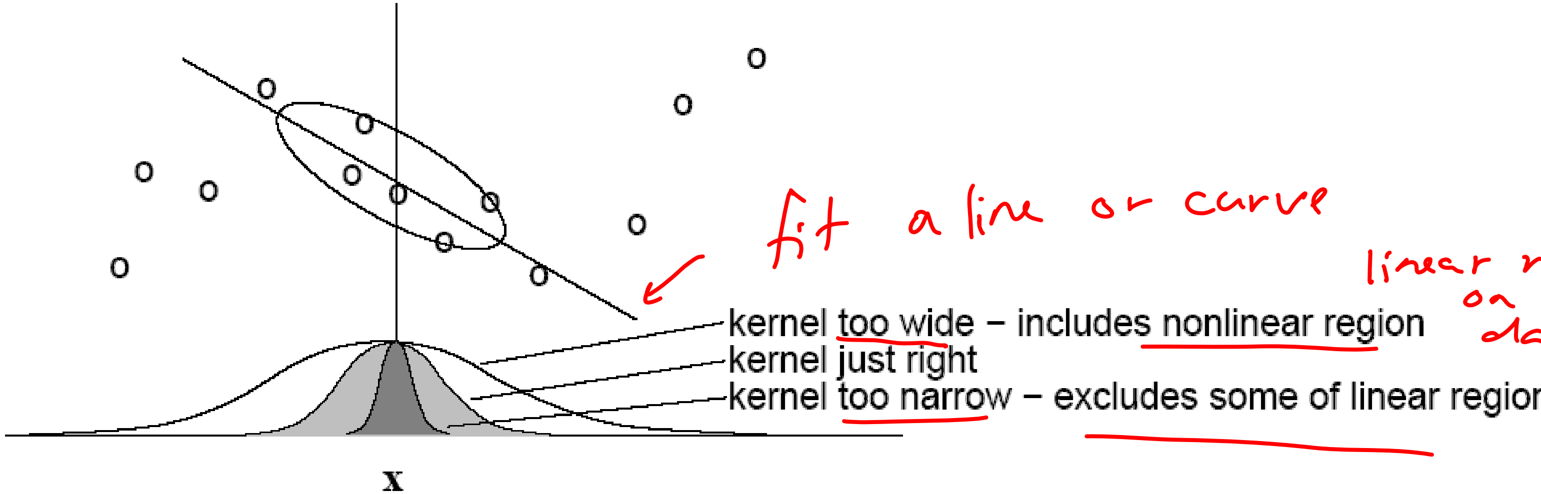
- Solve weighted linear regression for each query

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

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

©2006 Carlos Guestrin

Another view of LWR



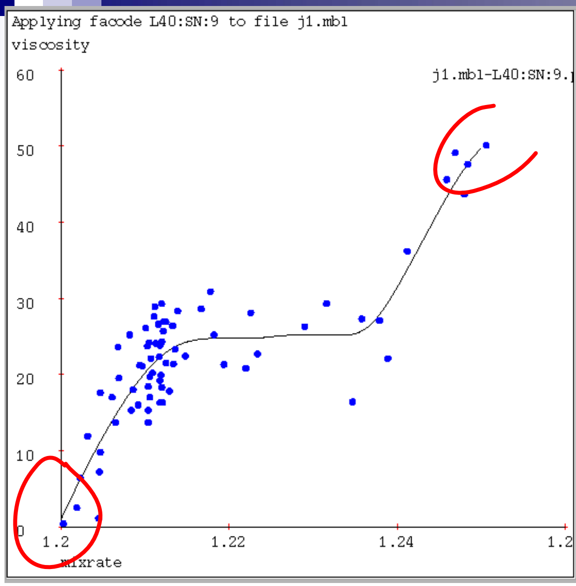
fit a line or curve

linear regression on non-linear data

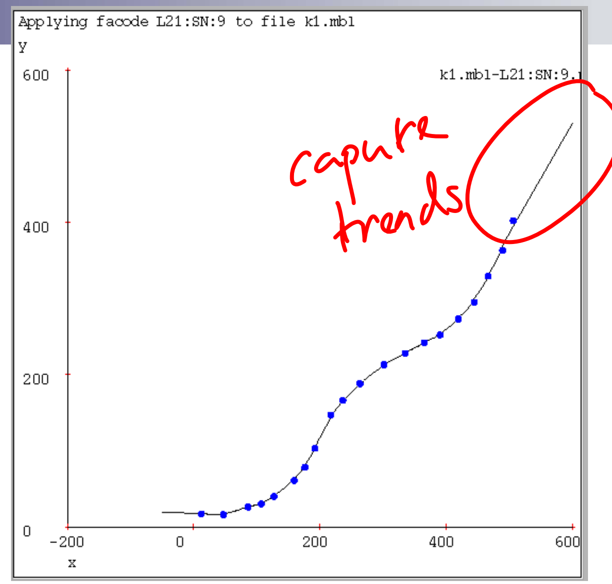
Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996) "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

©2006 Carlos Guestrin

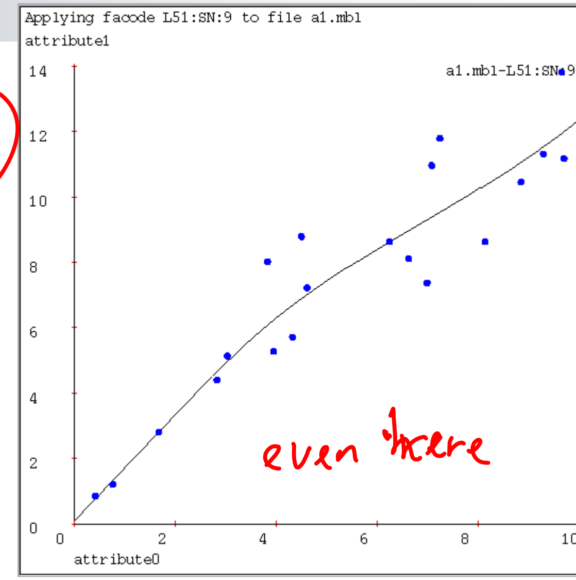
LWR on our test cases



KW = 1/16 of x-axis width.

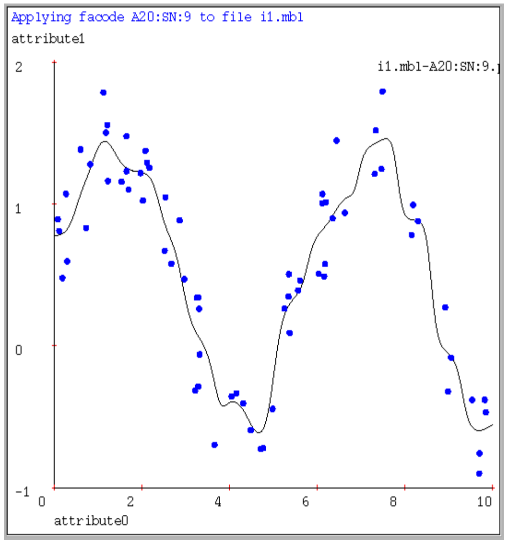


KW = 1/32 of x-axis width.



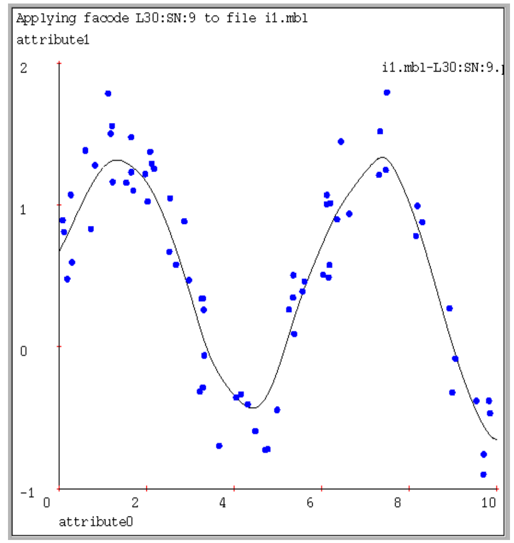
KW = 1/8 of x-axis width.

Locally weighted polynomial regression



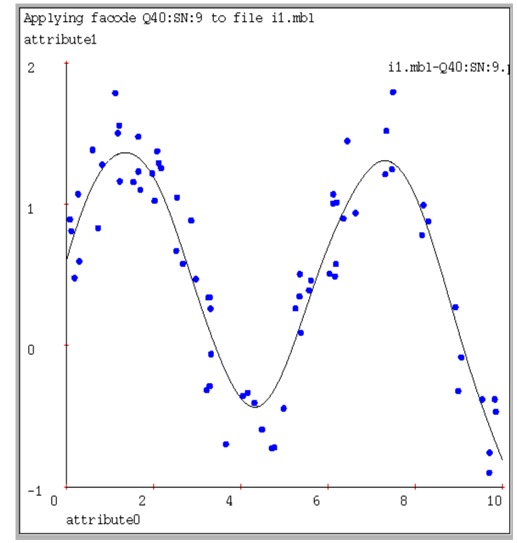
Kernel Regression
Kernel width K_W at optimal level.

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

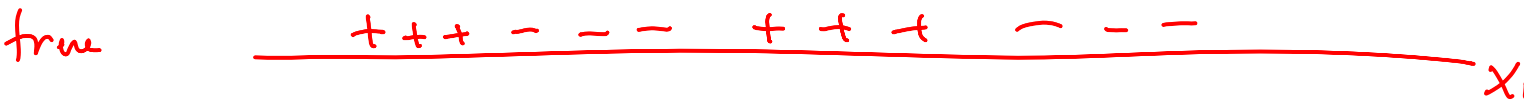
Local quadratic regression is easy: just add quadratic terms to the WXTWX matrix. As the regression degree increases, the kernel width can increase without introducing bias.

©2006 Carlos Guestrin

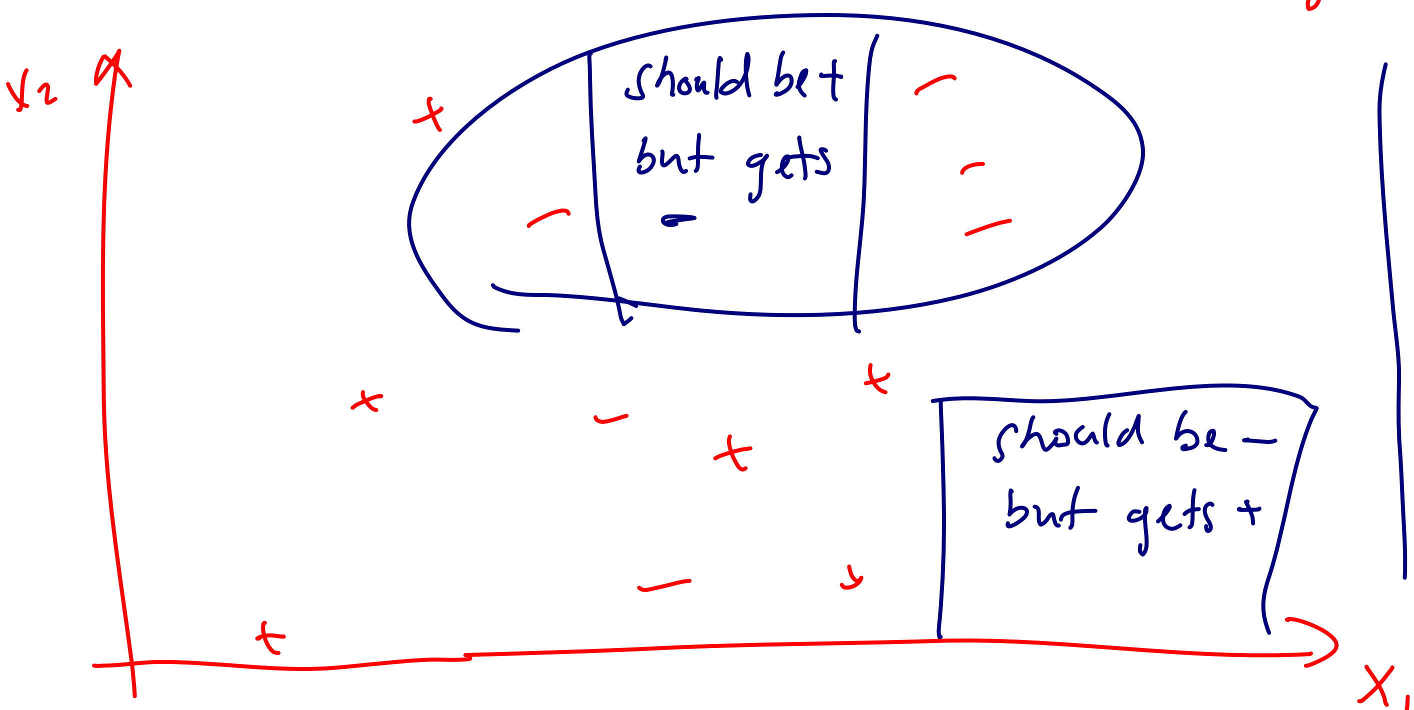
Curse of dimensionality for instance-based learning

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



add irrelevant feature x_2 with high variance



much much worse in high dims.

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

©2006 Carlos Guestrin

Acknowledgment

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

©2006 Carlos Guestrin

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

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

Support Vector Machines (SVMs)

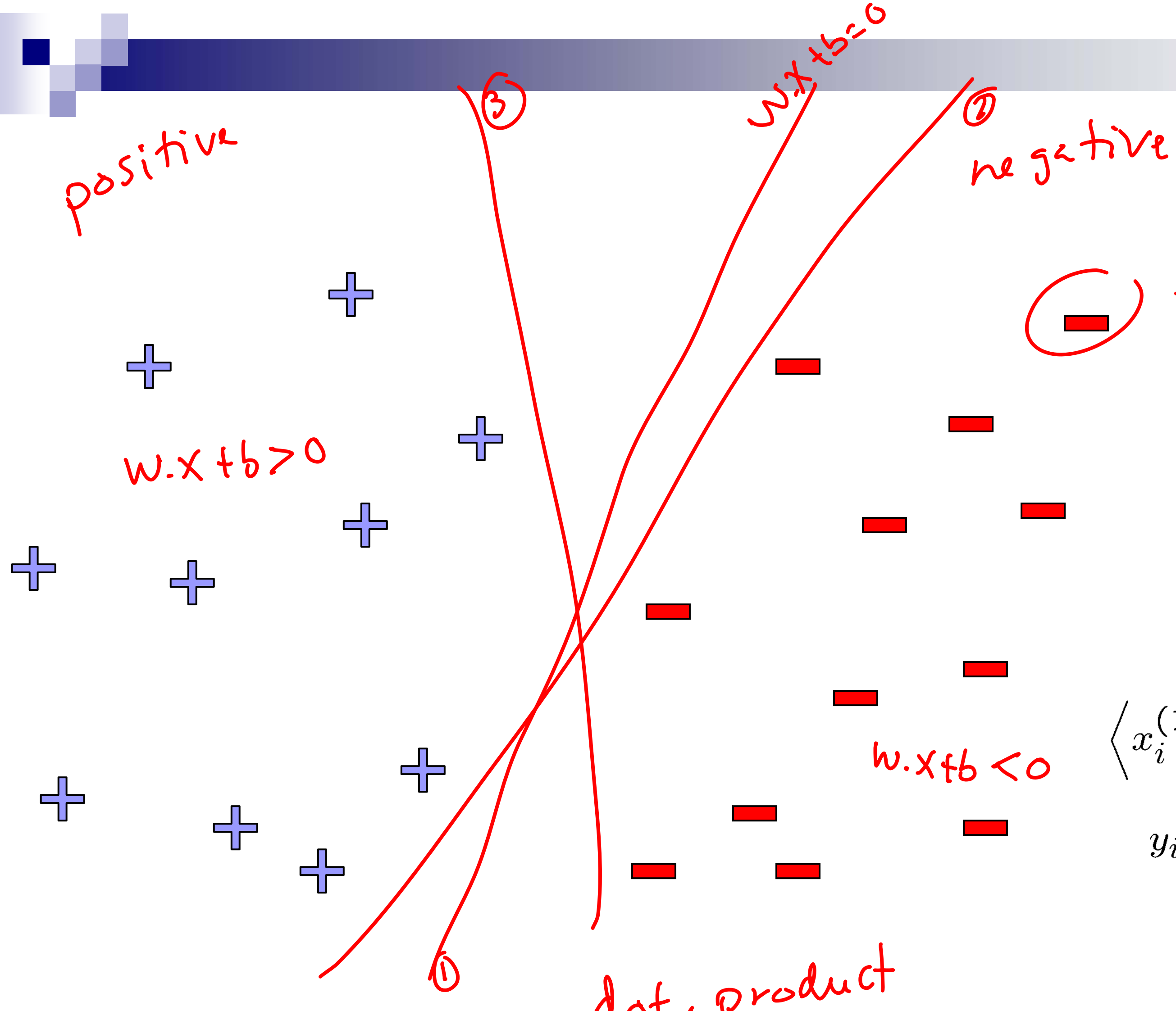
Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

February 20th, 2005

Linear classifiers – Which line is better?



Data:

m features

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

n data points

Example i:

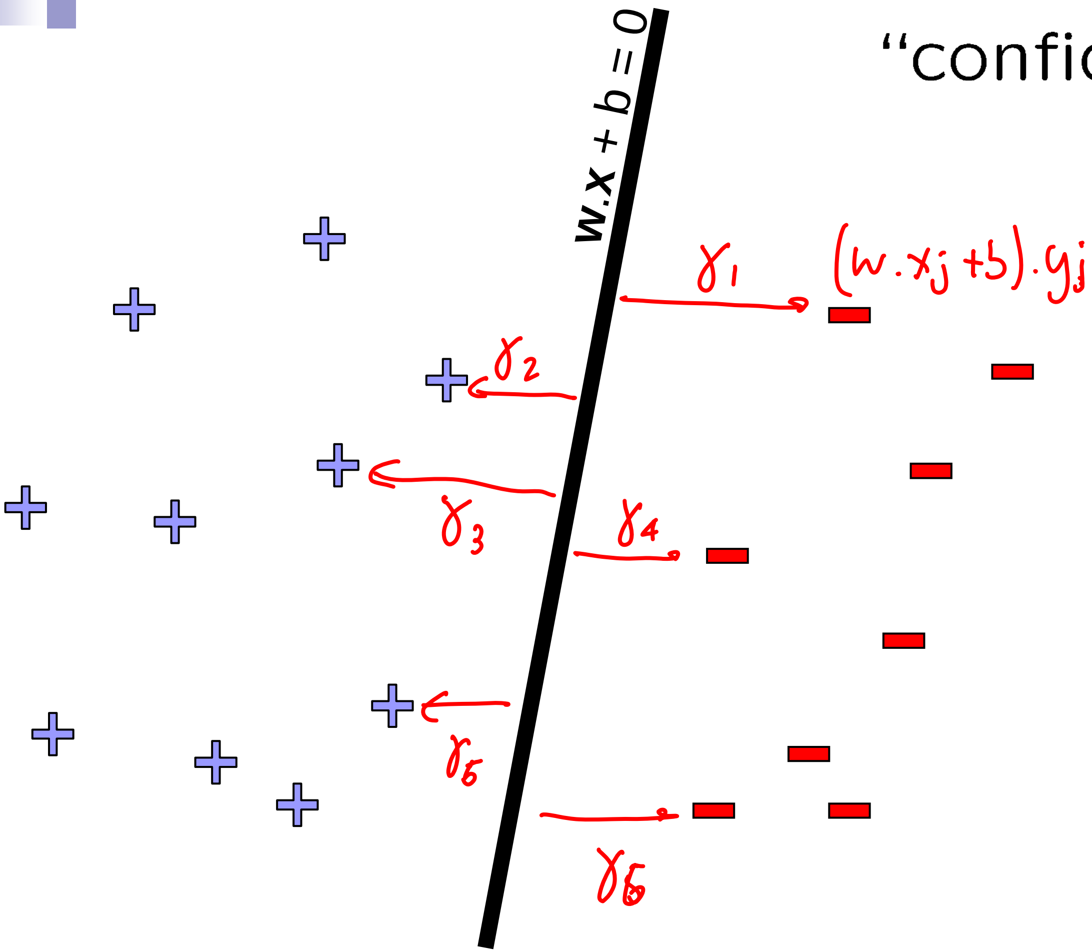
$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

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

©2006 Carlos Guestrin

Pick the one with the largest margin!



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

make $\gamma_1, \gamma_2, \gamma_3, \ldots, \gamma_n$ large

make smallest γ_i as large as possible!

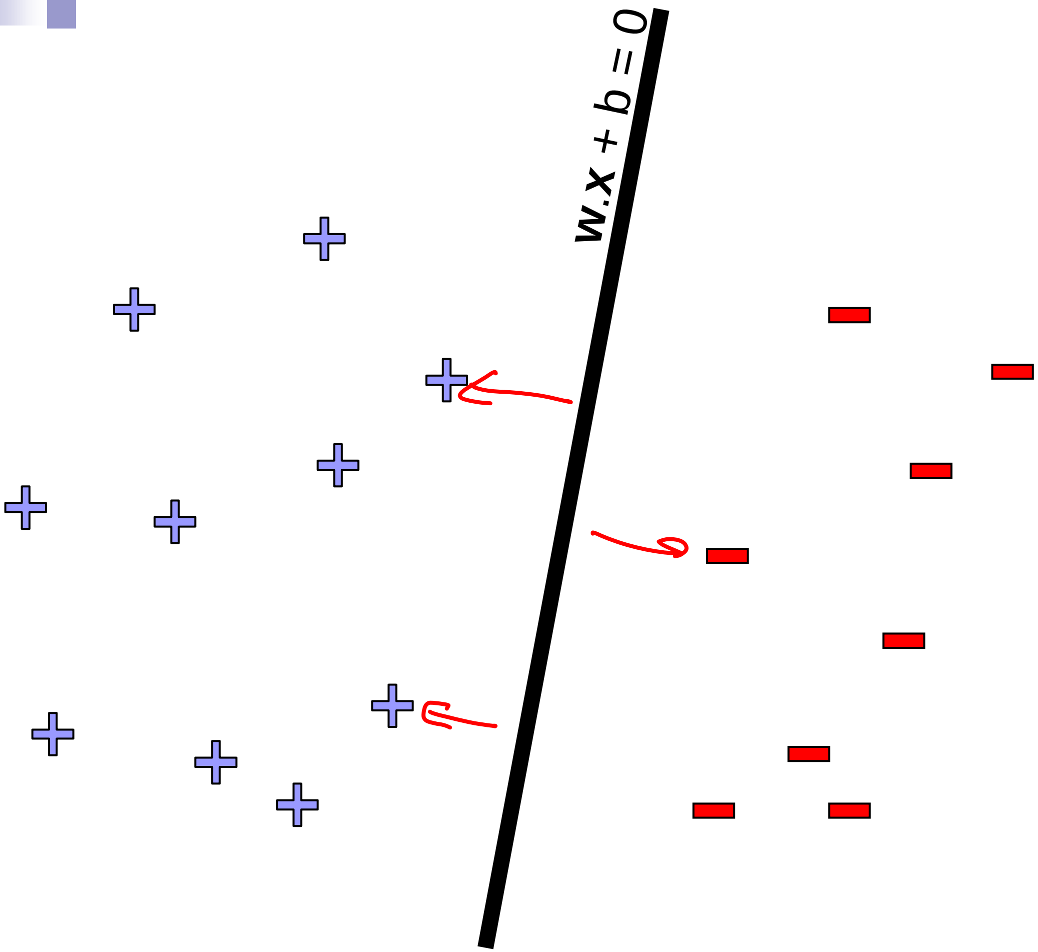
margin $\gamma = \min_i \gamma_i$

$$\max_{w,b} \gamma$$

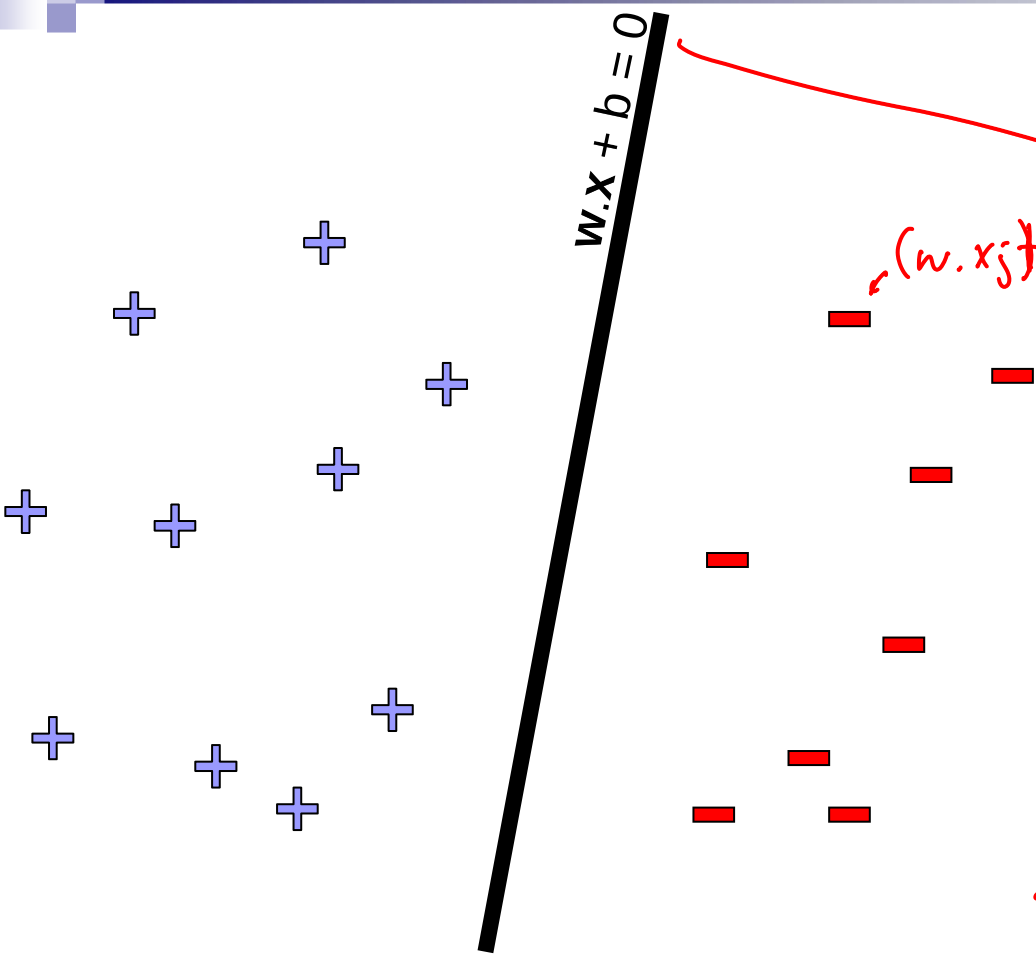
$$(w.x_j + b)\, y_j \geq \gamma \quad \forall j$$

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

Maximize the margin



But there are a many planes…



$(w.x_j+b)y_j$

say

$w.x + b = 0$

$\hookrightarrow 2w.x + 2b = 0$

what's confidence now?

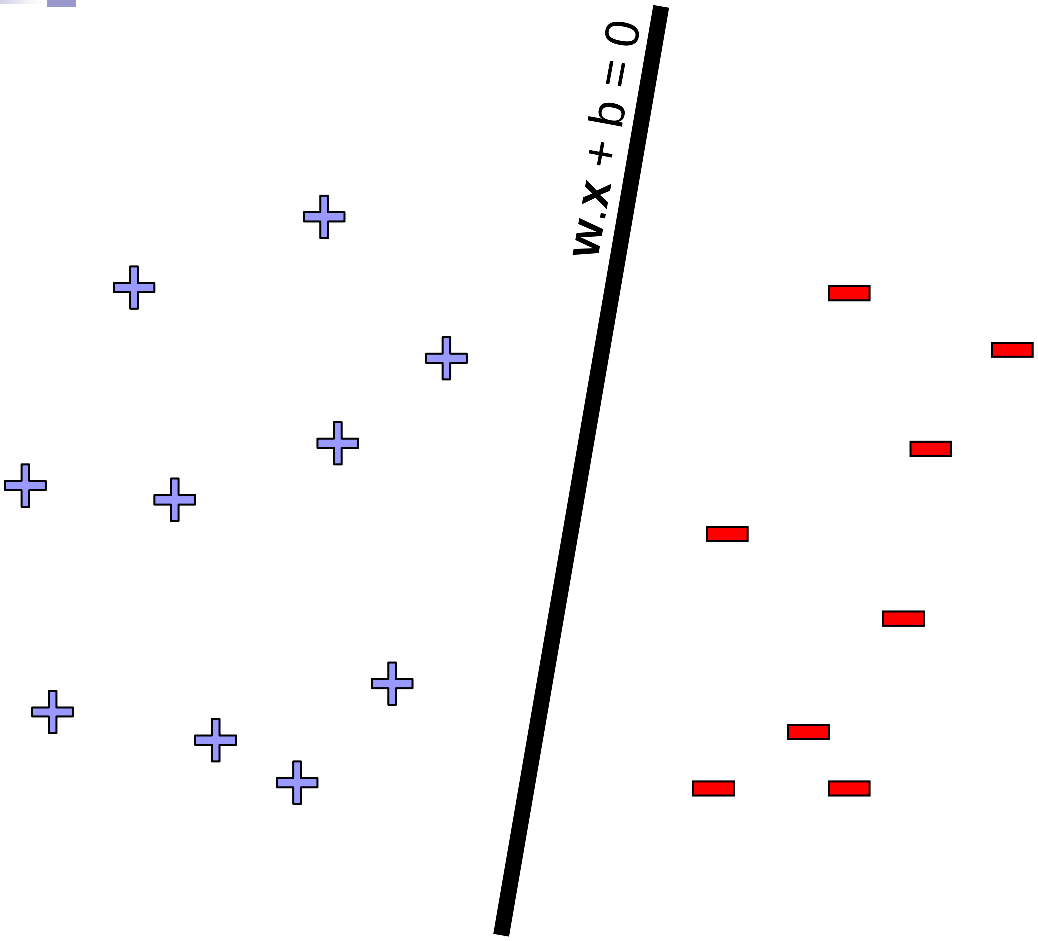
$(2w.x_j + 2b)\,y_j$

doubled "confidence" with no work!! :)

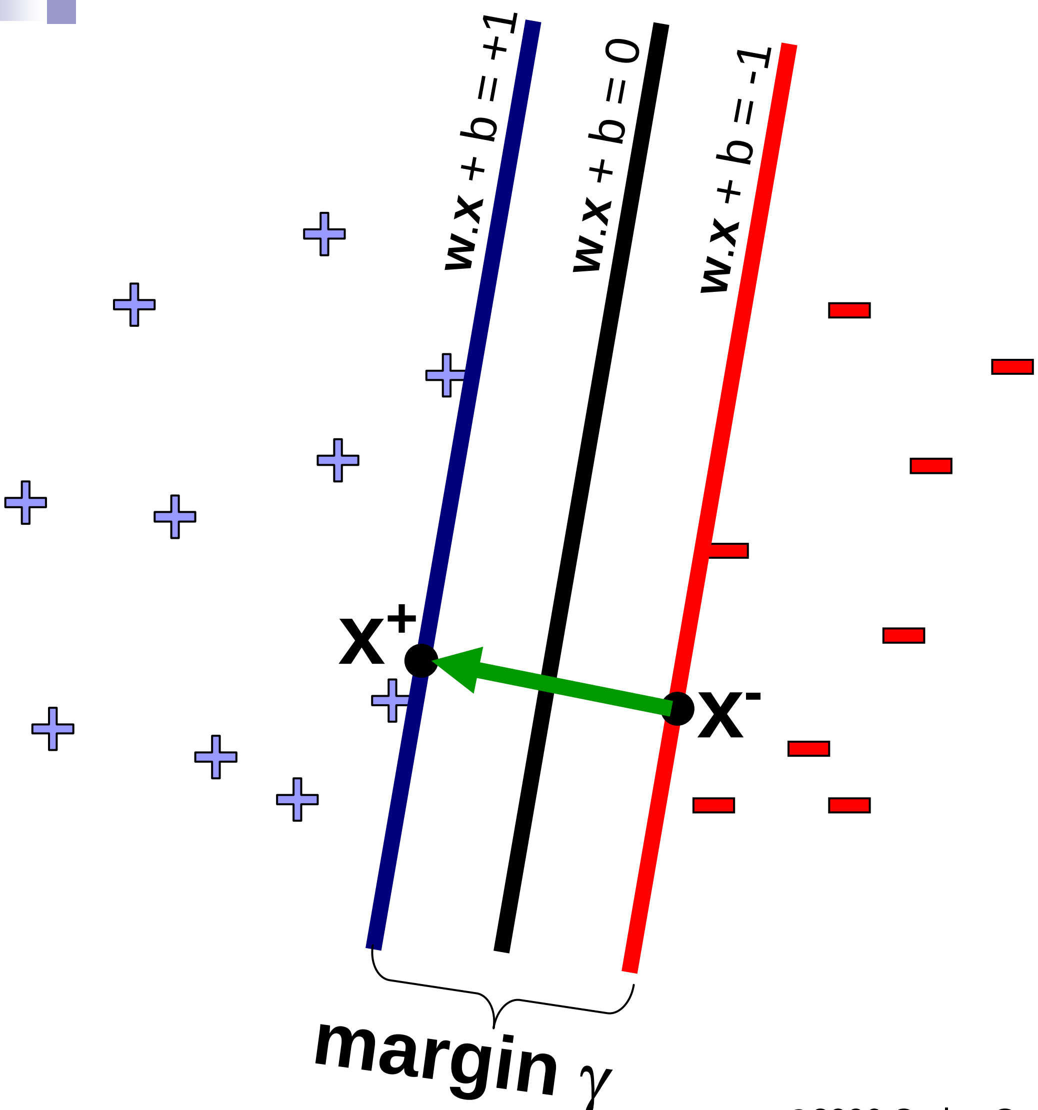
"confidence" $\rightarrow \infty$

as $||w|| \rightarrow \infty$

Review: Normal to a plane



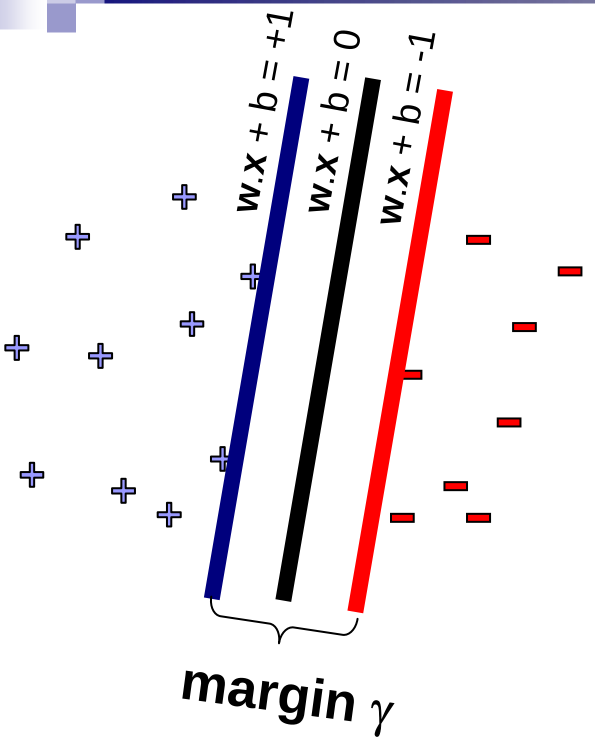
Normalized margin – Canonical hyperplanes



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

©2006 Carlos Guestrin

Margin maximization using canonical hyperplanes

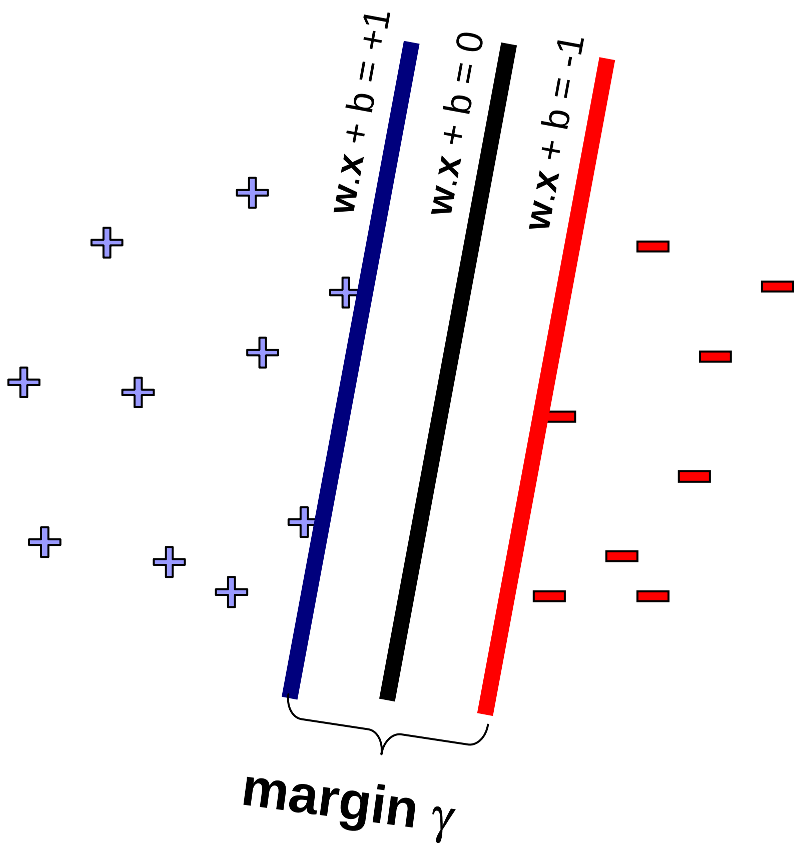


$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$

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

©2006 Carlos Guestrin

Support vector machines (SVMs)



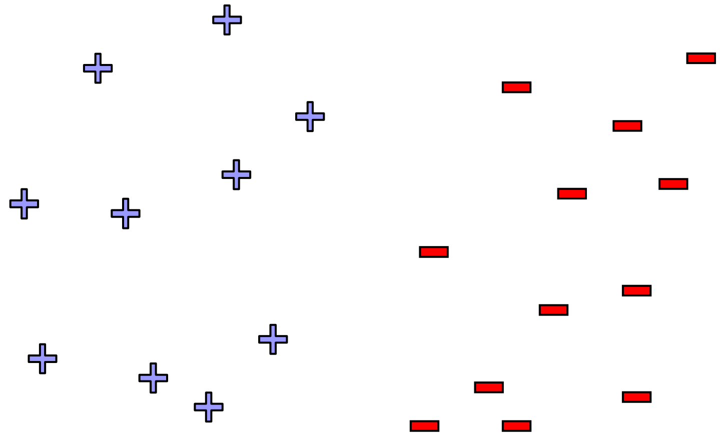
$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$

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

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

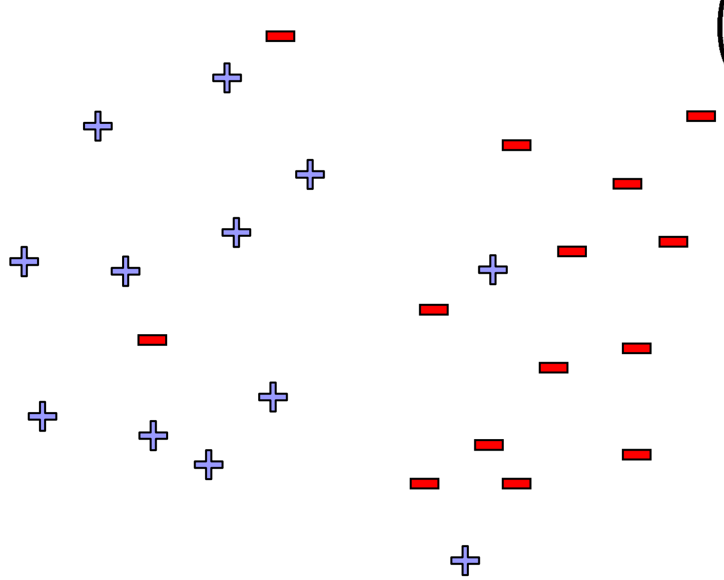
What if the data is not linearly separable?

Use features of features of features of features….



What if the data is still not linearly separable?

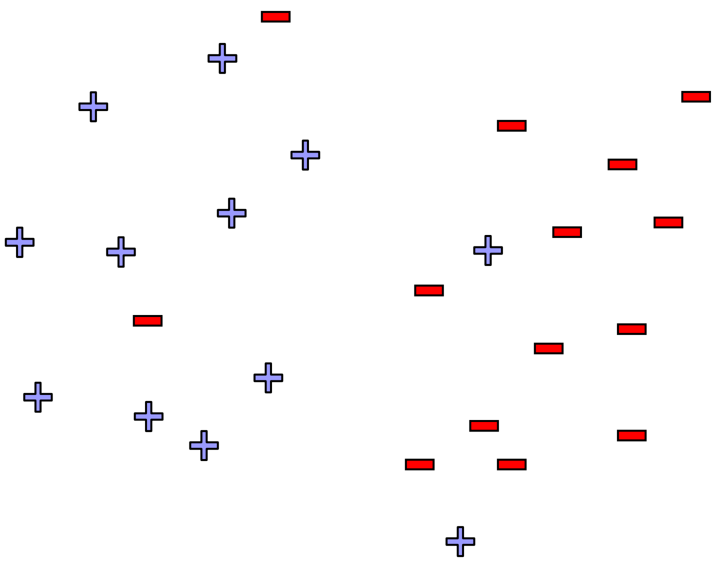
$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad , \forall j$$



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

Slack variables – Hinge loss

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



- If margin $\geq$ 1, don't care
- If margin < 1, pay linear penalty

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

SVM:

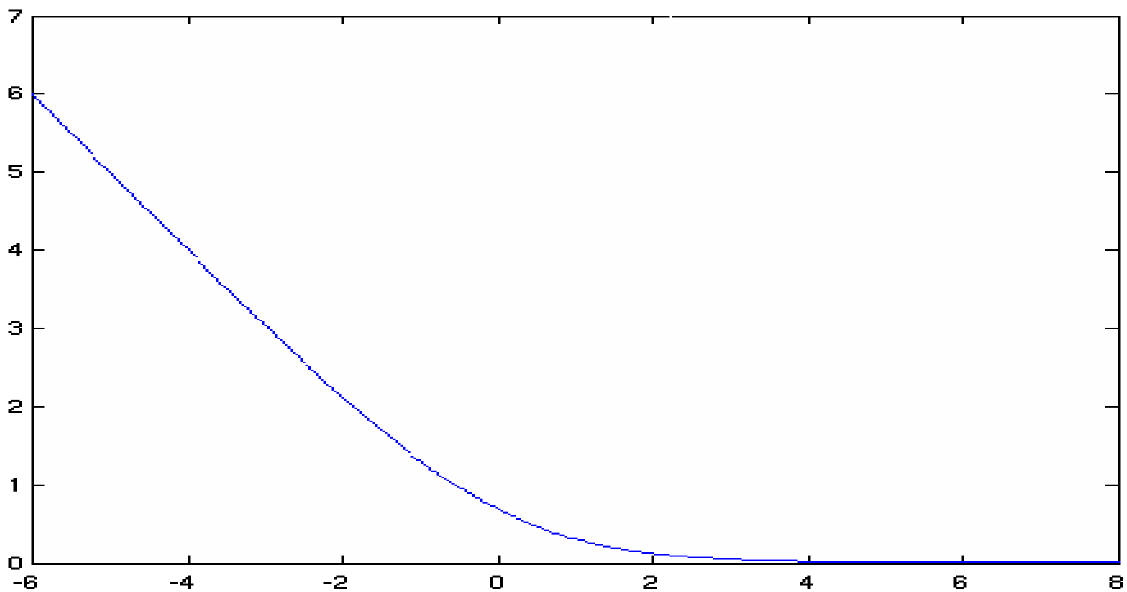
$$\text{minimize}_{\mathbf{w}} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

Logistic regression:

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

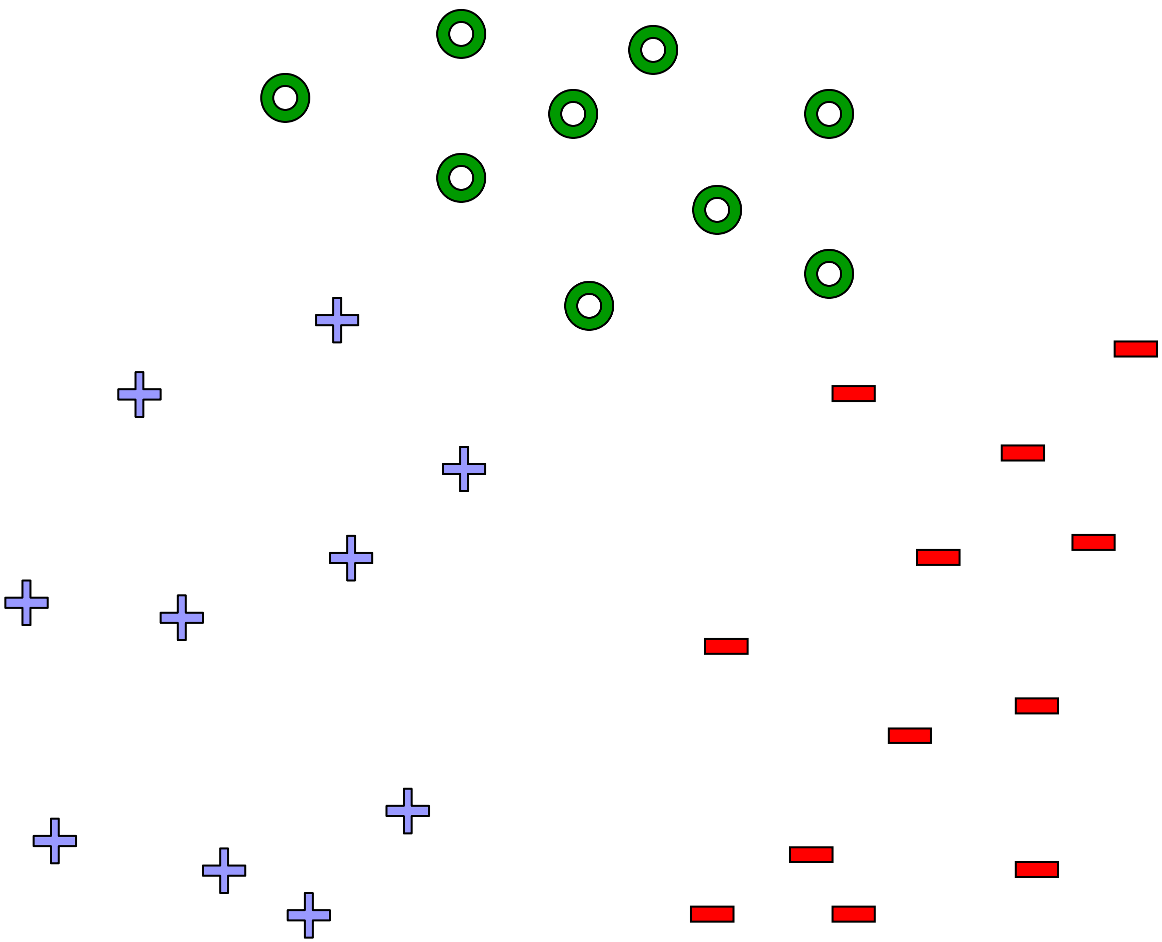
Log loss:

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

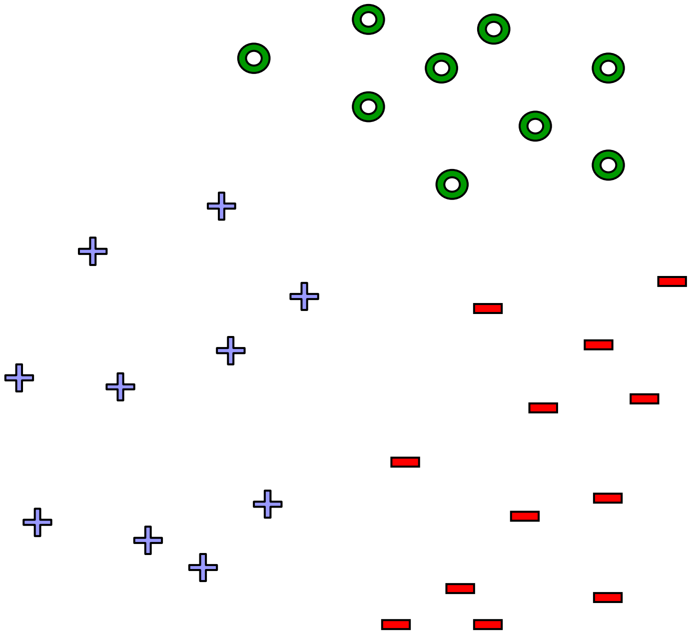


©2006 Carlos Guestrin

What about multiple classes?



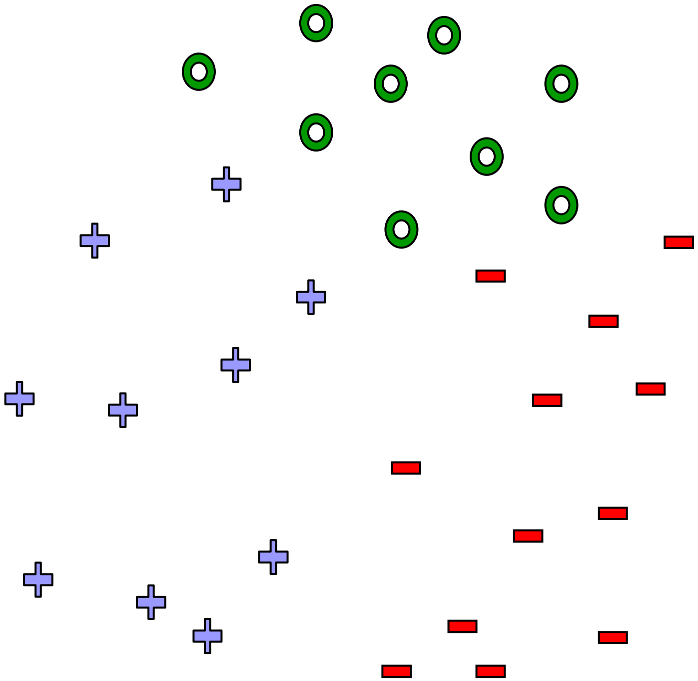
One against All



Learn 3 classifiers:

Learn 1 classifier: Multiclass SVM

Simultaneously learn 3 sets of weights



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

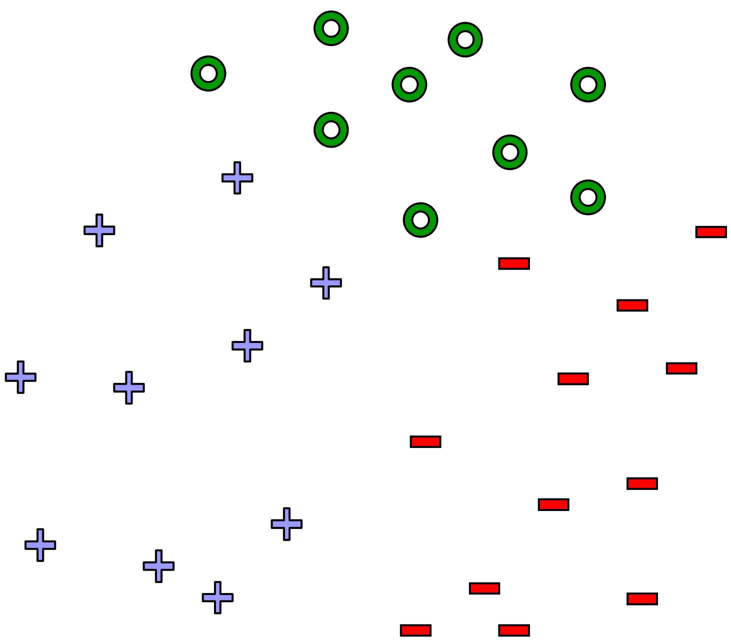
©2006 Carlos Guestrin

Learn 1 classifier: Multiclass SVM

$$\text{minimize}_{\mathbf{w}} \quad \sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C\sum_j \xi_j$$

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

$$\xi_j \geq 0, \ \forall j$$



©2006 Carlos Guestrin

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

©2006 Carlos Guestrin

Acknowledgment

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