Instance-based Learning

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

October 15th, 2007

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.

Consistency of 1-NN

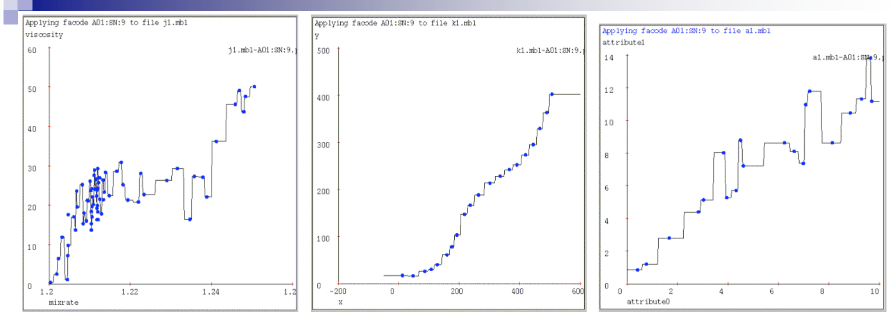
- Consider an estimator f_n trained on n examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if true error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:

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

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

What about variance???

1-NN overfits?

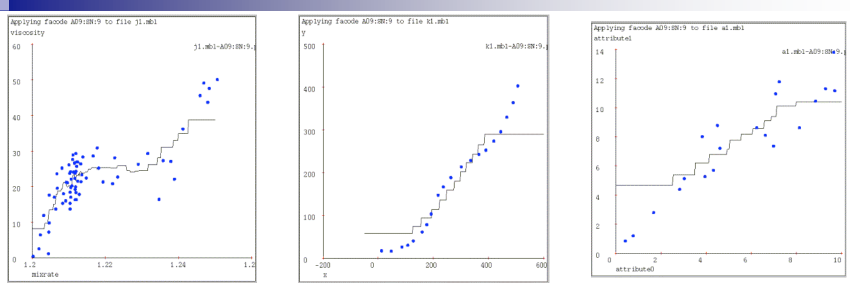


k-Nearest Neighbor

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

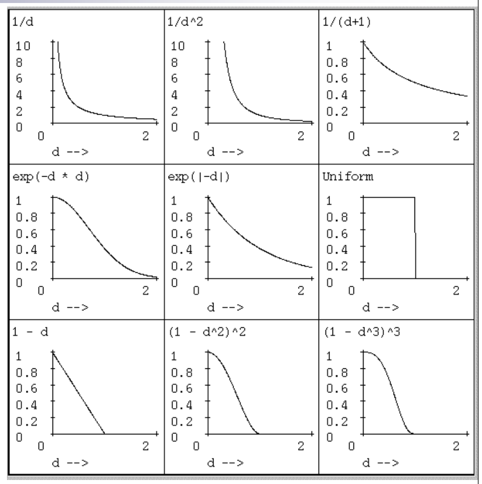
Kernel regression

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)*
   $w_i = \exp(-D(x_i, query)^2 / K_w^2)$
   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 = $\Sigma w_i y_i / \Sigma w_i$

Weighting functions

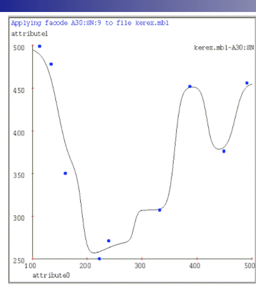
$w_i = exp(-D(x_i, query)^2 / K_w^2)$



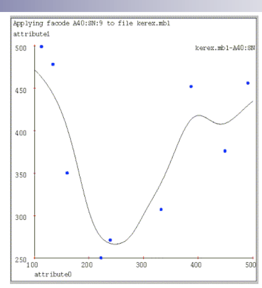
Typically optimize K_w using gradient descent

(Our examples use Gaussian)

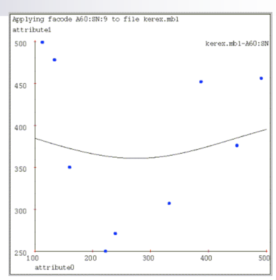
Kernel regression predictions



$K_W=10$



$K_W=20$

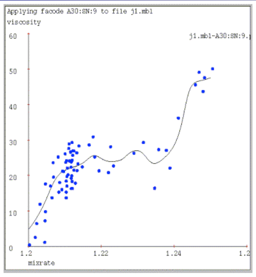


$K_W=80$

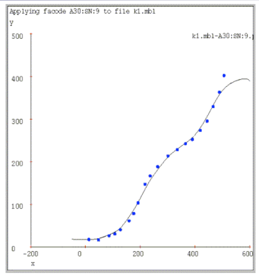
Increasing the kernel width K_w means further away points get an opportunity to influence you.

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

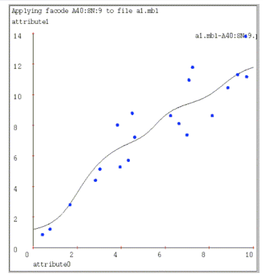
Kernel regression on our test cases



KW=1/32 of x-axis width.



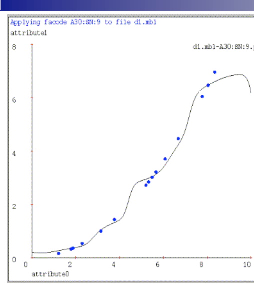
KW=1/32 of x-axis width.



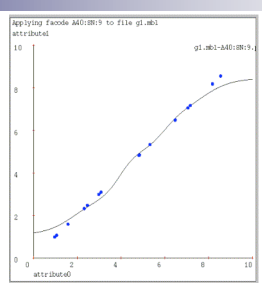
KW=1/16 axis width.

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

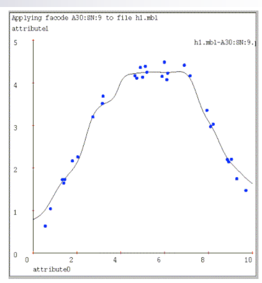
Kernel regression can look bad



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:

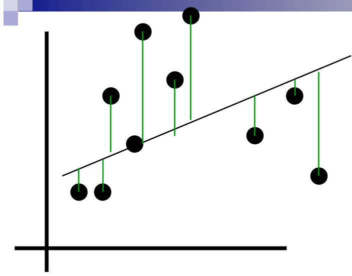
Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

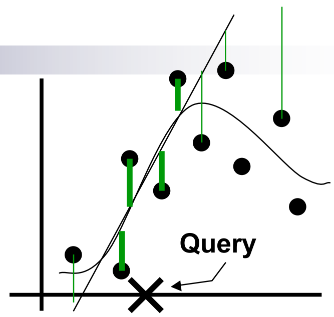
Locally weighted regression

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

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

How LWR works





Linear regression

- Same parameters for all queries

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

Locally weighted regression

- Solve weighted linear regression for each query

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

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

Another view of LWR

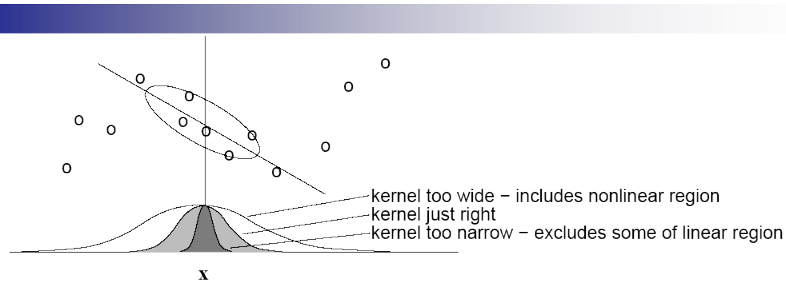
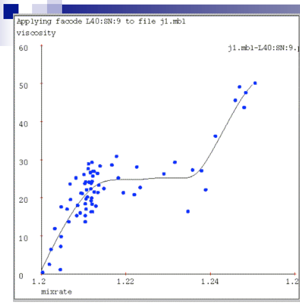
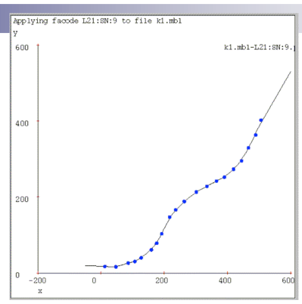


Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996) "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

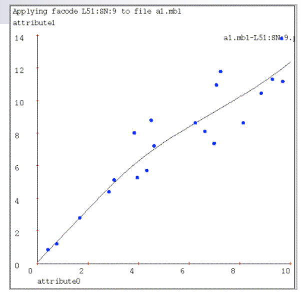
LWR on our test cases



KW = 1/16 of x-axis width.

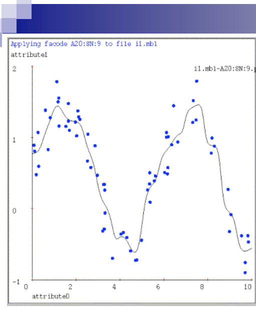


KW = 1/32 of x-axis width.



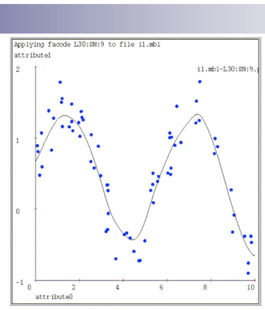
KW = 1/8 of x-axis width.

Locally weighted polynomial regression



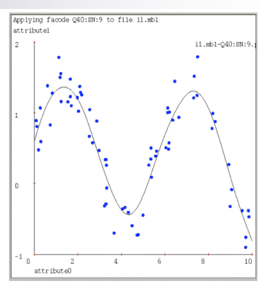
Kernel Regression
Kernel width K_W at optimal level.

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!
  - 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

Announcements

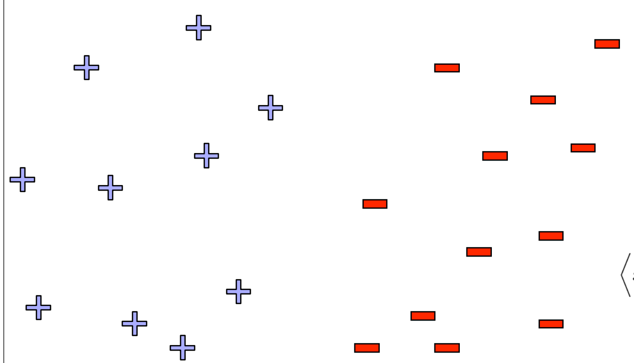
- Recitation this week: Neural networks
- Project proposals due next Wednesday
  - Exciting data:
    - Swivel.com - user generated graphs
    - Recognizing Captchas
    - Election contributions
    - Activity recognition
    - …

Support Vector Machines

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

October 15^{th}, 2007

Linear classifiers – Which line is better?



Data:

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

Example i:

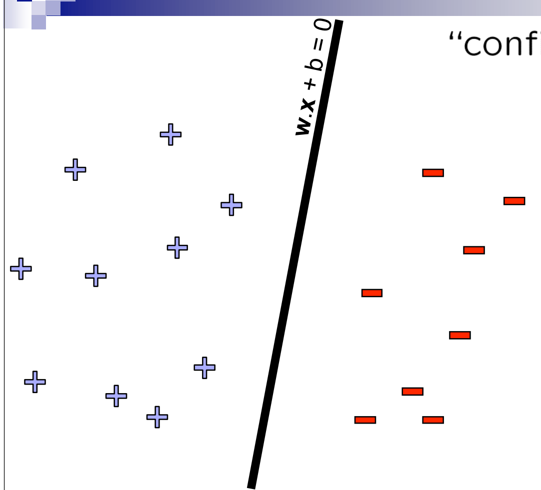
$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

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

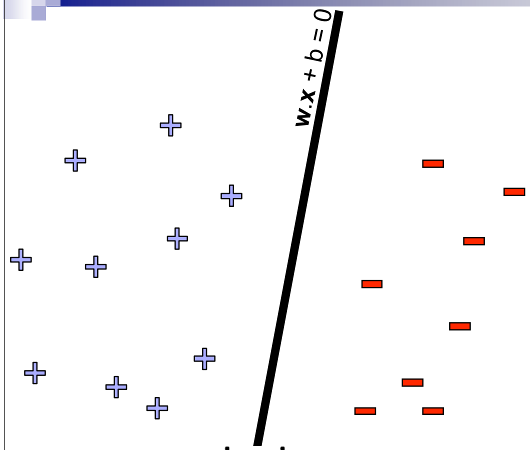
Pick the one with the largest margin!

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



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

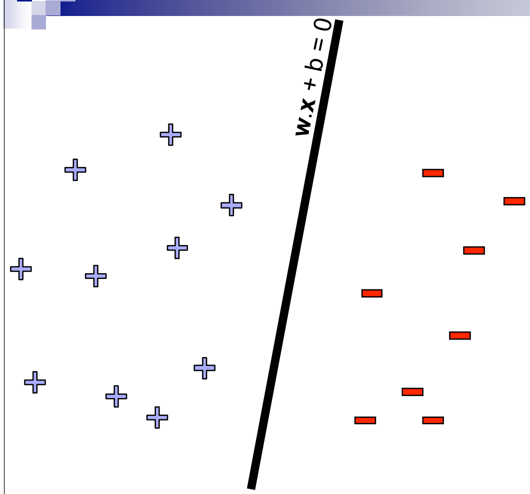
Maximize the margin



$$\text{maximize}_{\gamma,\mathbf{w},b} \quad \gamma$$

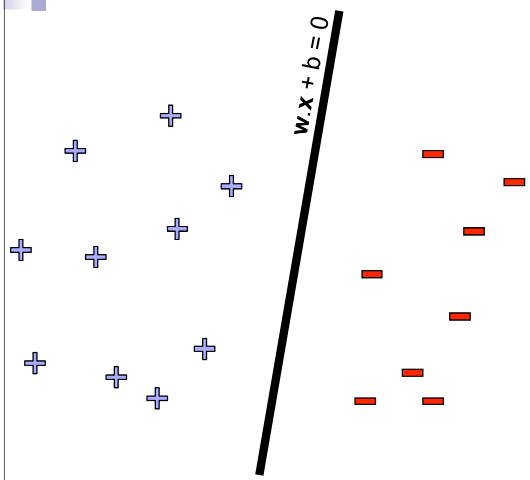
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq \gamma, \;\; \forall j \in \text{Dataset}$$

But there are a many planes…



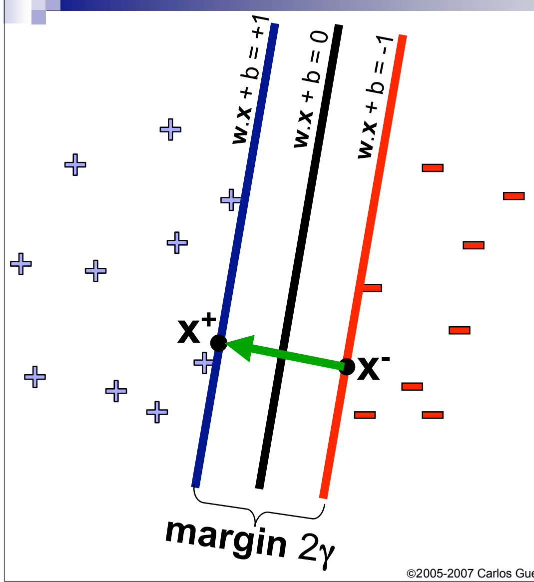
Review: Normal to a plane

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$

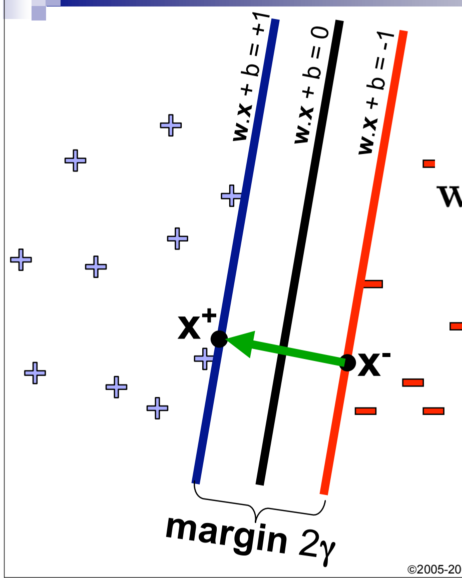


Normalized margin – Canonical hyperplanes

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$



Normalized margin – Canonical hyperplanes



$$\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$$

$$\mathbf{w}.\mathbf{x}^+ + b = 1$$

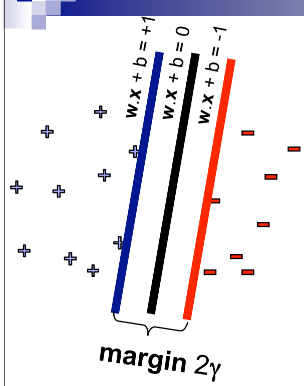
$$\mathbf{w}.(\mathbf{x}^- + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}) + b = 1$$

$$\lambda = \frac{2}{||\mathbf{w}||}$$

$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Margin maximization using canonical hyperplanes

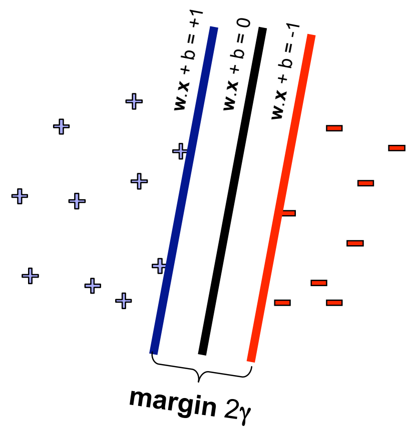
$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$



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

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

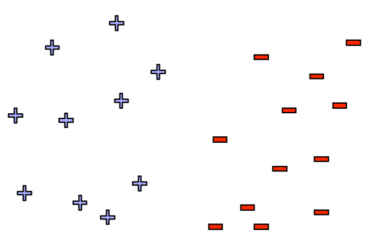
Support vector machines (SVMs)



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

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

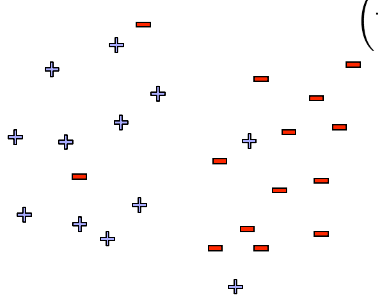
What if the data is not linearly separable?



Use features of features of features of features….

What if the data is still not linearly separable?

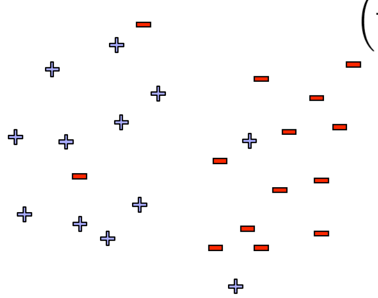
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad ,\forall j$$



- 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

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



- 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:

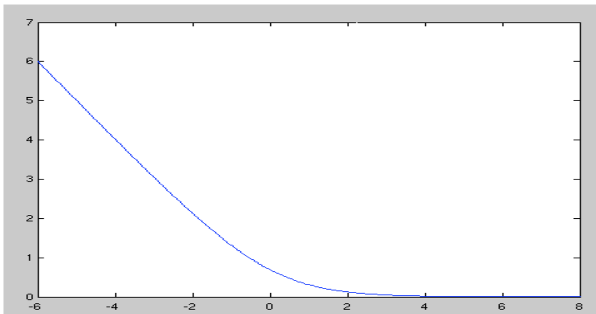
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C \sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

Logistic regression:

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

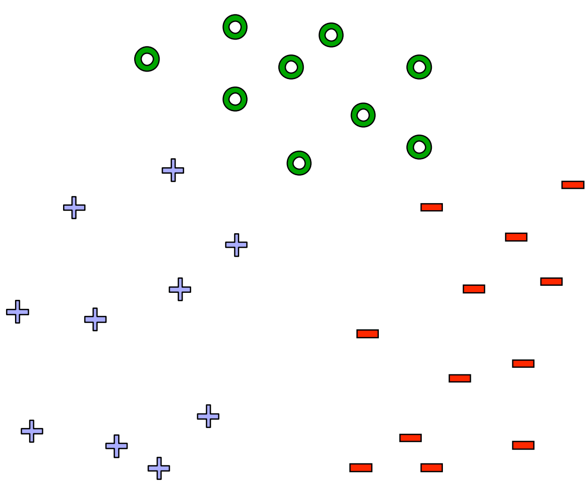
Log loss:

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



 37

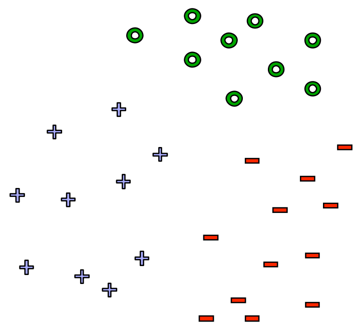
What about multiple classes?



 38

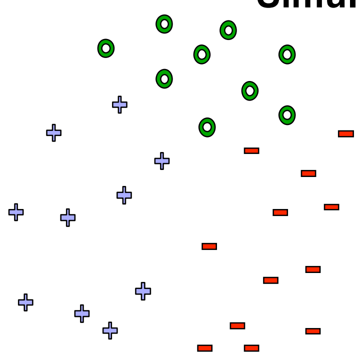
One against All

Learn 3 classifiers:



Learn 1 classifier: Multiclass SVM

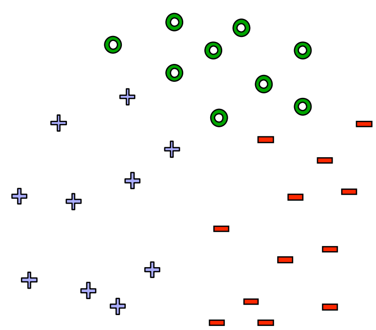
Simultaneously learn 3 sets of weights



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

Learn 1 classifier: Multiclass SVM

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



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