Review

- Neural Networks
- Perceptrons
- Multilayer Perceptrons
- Project 4-b (NNets) out next week

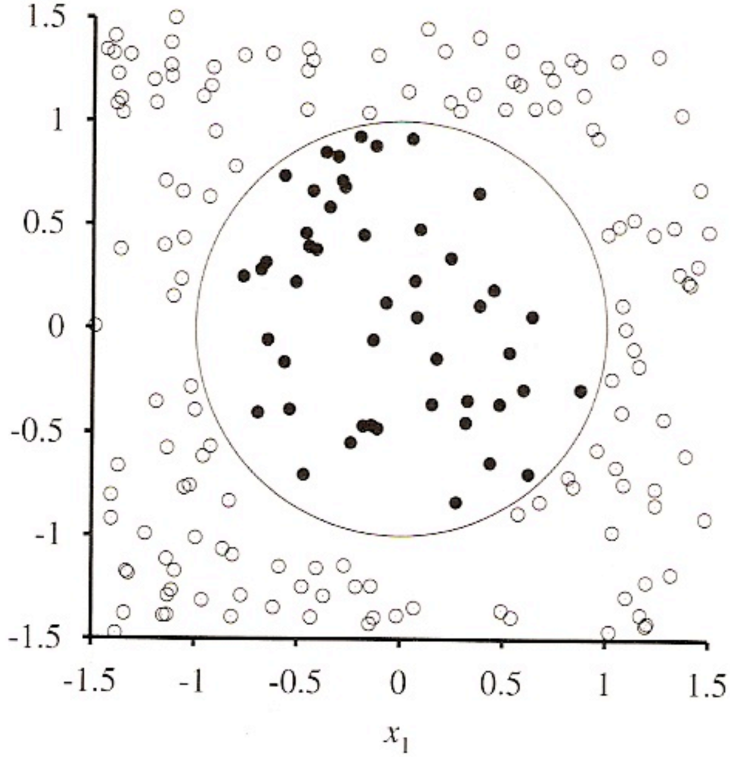
Supervised Learning, and there's more....

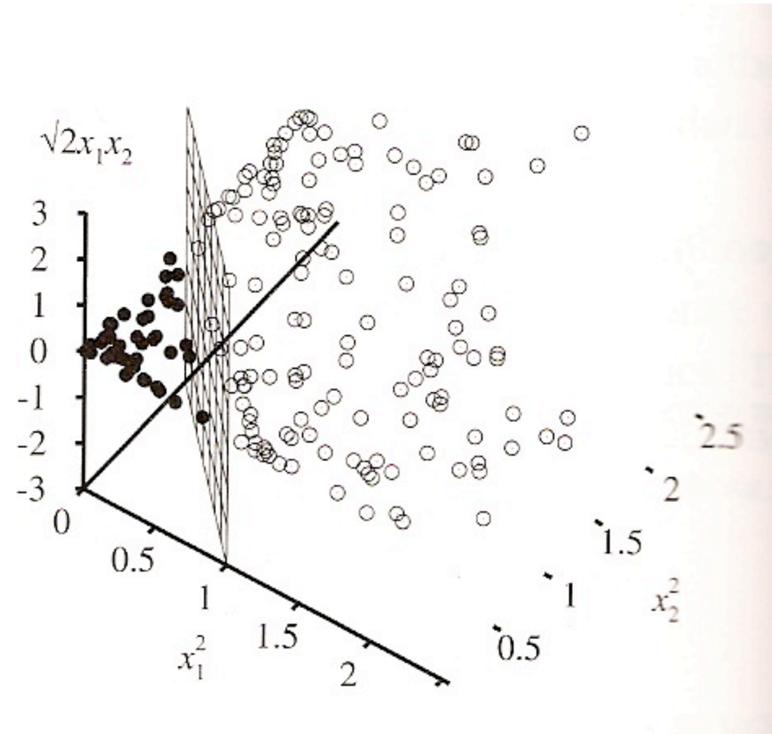
- Support Vector Machines (SVMs)
- k-Nearest Neighbors (KNN)
- Ensemble methods

Support Vector Machines

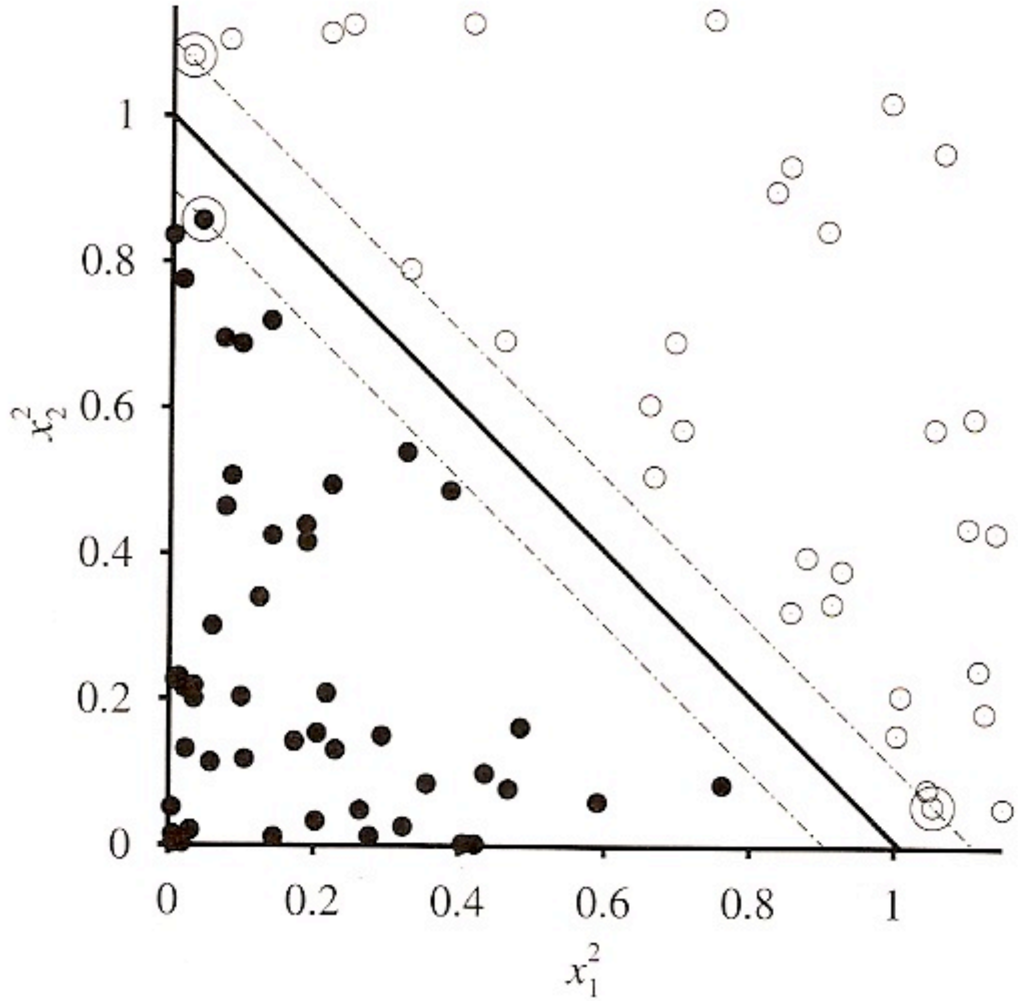
- One of the most popular "off the shelf" supervised learning techniques
- Two key properties of its success
  - 1) Make non-linear problems linear, in a higher dimensional space
  - 2) Find linear separator that maximizes the margin, represent by saving all examples on the margin

The Kernel Trick





- Find a higher dimension in which the d-dimensional data is linearly separable!
- Optimal = the features that maximize the margin between the classes



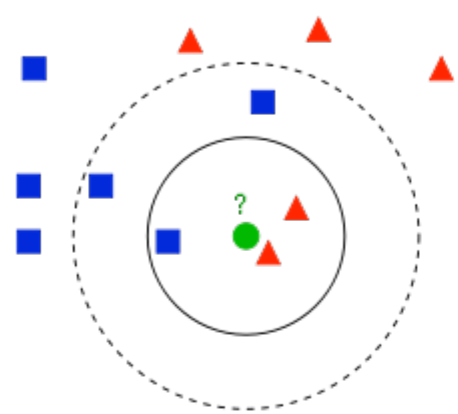
SVM model: support vectors as defined by the training examples closest to the margin

Instance-based Methods

- SVM is one of a class of algorithms called instance-based
- The model is the data (as opposed to the weights in a NNet or to a DTree)

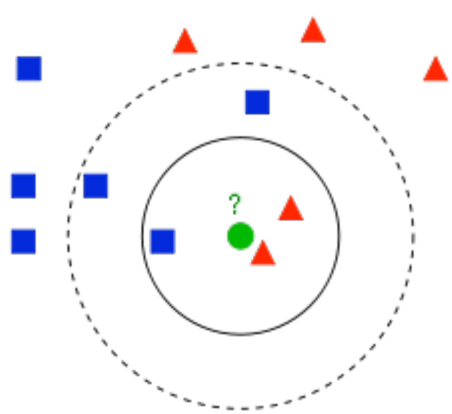
Nearest Neighbors

- Basis: a point in the data space is likely to be similarly classified as its neighbors
  - Given a new instance to classify
  - Find the points closest to it in your labeled training data, this is it's "neighborhood"
  - Give it the same classification as the neighborhood



Nearest Neighbors

- Problem: how to define "neighborhood"
  - k-nearest neighbors: pick an arbitrary number of points to consider for the neighborhood
  - distance metrics: need to define what "near" means for a feature (location on a map? hair color?)



Ensemble Learning

- Why use just one hypothesis when you could use many
- EX: instead of one Decision Tree for classifying, generate several
- Save an ensemble of hypothesis and combine their predictions

Why Ensemble

- Example -- 5 classifiers instead of 1
  - classification = majority vote
  - 3 of 5 have to fail in order to misclassify
- If each *h* has an error $p=.1$ then the ensemble reduces this error to $p < .001$
- Works only if hypotheses are somewhat different (independent)

Boosting

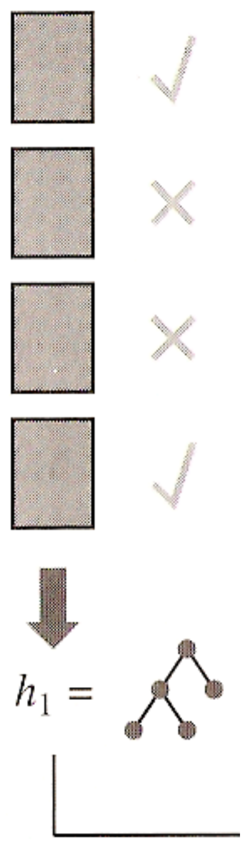
- Weighted Training Set: each example has an associated weight, signifying its importance in the learning process.

Example 1	weight 1
...	...
...	...
...	...
Example i	weight i

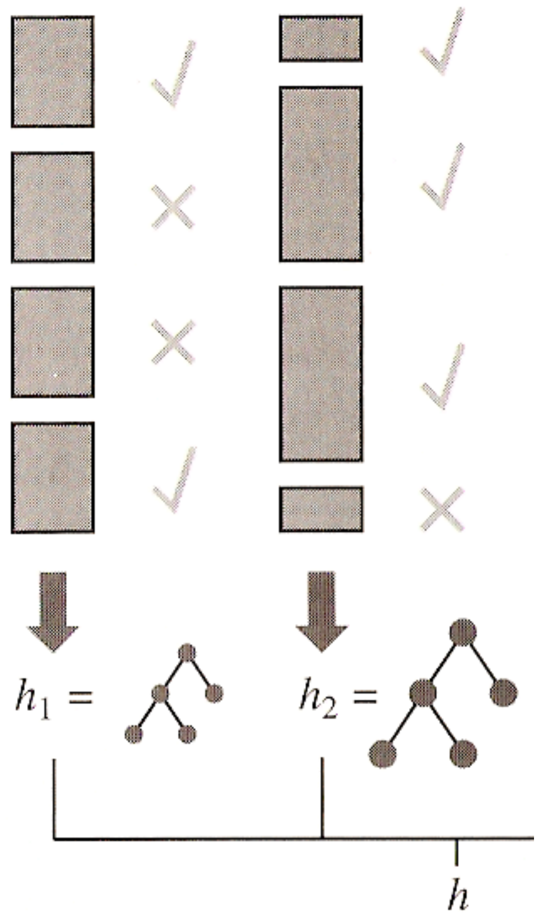
Boosting

- Start with all $w=1$; generate h_1
- Increase weight on all examples misclassified by h_1; generate h_2
- Repeat to generate M hypotheses for the ensemble
- Use the ensemble by taking weighted majority classification

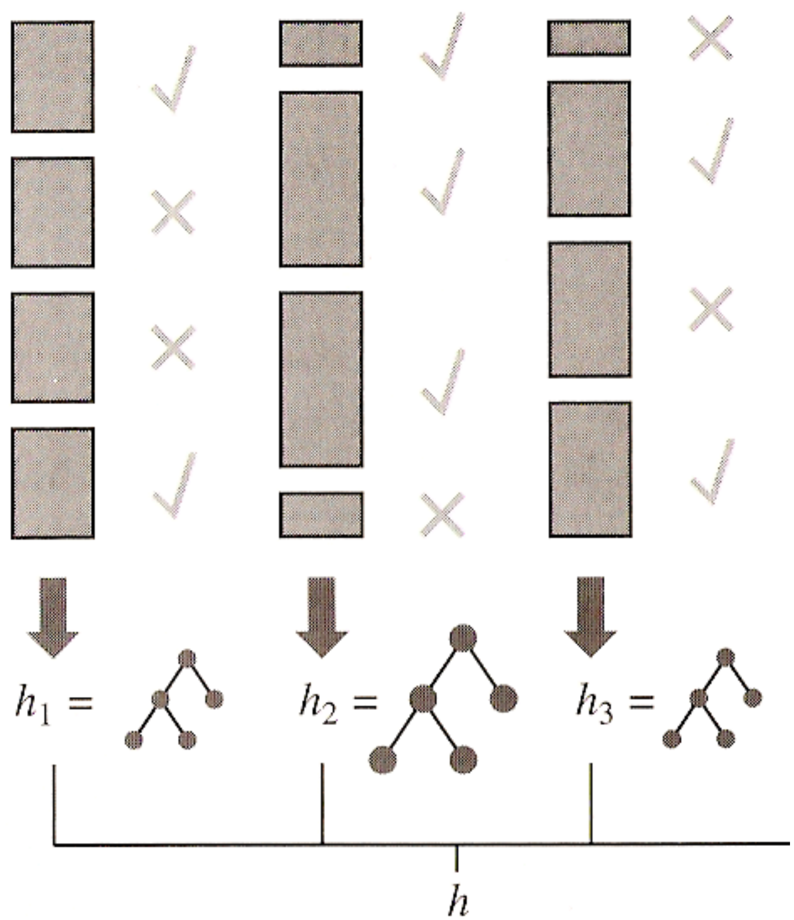
Boosting



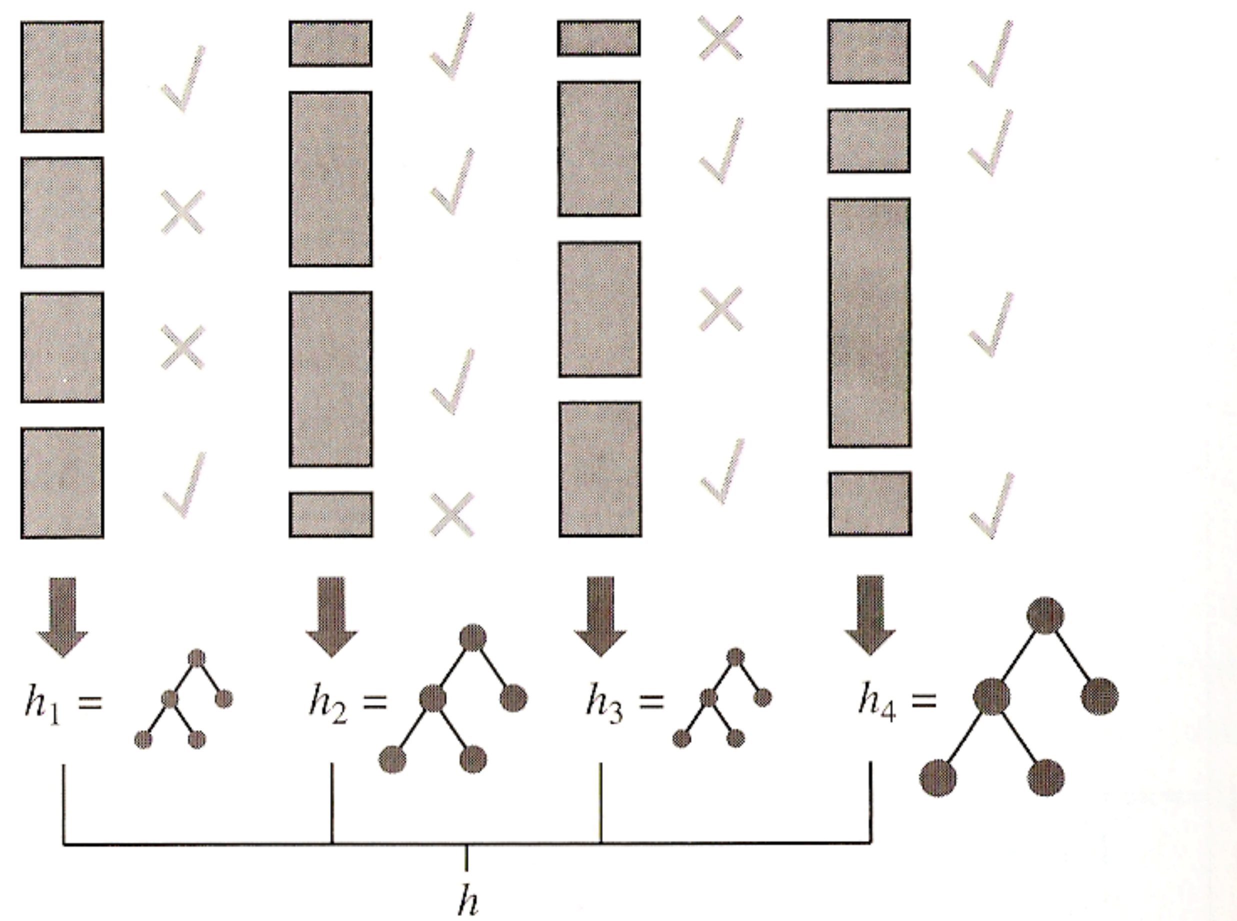
Boosting



Boosting



Boosting



function ADABOOST($examples$, L, M) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labelled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    M, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of M hypotheses
    z, a vector of M hypothesis weights

  for m = 1 **to** M **do**
    $\mathbf{h}[m] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for j = 1 **to** N **do**
      if $\mathbf{h}[m](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for j = 1 **to** N **do**
      if $\mathbf{h}[m](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE($\mathbf{w}$)
    $\mathbf{z}[m] \leftarrow \log\,(1 - error)/error$
  return WEIGHTED-MAJORITY($\mathbf{h}, \mathbf{z}$)

Computational Learning Theory

- How do we know if *h* is close to *f* ?
- PAC Learning (Probably Approx. Correct)
  - if *h* has been correct for a large number of examples it is very unlikely that it is an incorrect approximation of *f*

Stationary

- PAC learning assumes training and test drawn from the same distribution
- Future is like the past: therefore learned model is relevant
- Ex Computer Vision: training in different lighting than testing is non-stationary

How many examples?

- PAC learning assumes a large number of examples -- how many is needed?
    - X = all possible examples
    - D = distribution they come from
    - H = all possible hypotheses
    - N = number of training examples
- Calculate the probability that a hypothesis is bad but gets the first N samples right.
    - Depends mostly on size of H (i.e., number of features in your state)

Learning from Examples Summary

- Supervised learning: find simple hypothesis approximately consistent with training examples
- Number examples needed is related to size of hypothesis space
- Decision Tree learning using information gain
- Neural Nets: single layer easy but limited, multilayer does non-linear, SVMs typically better
- K-nearest neighbors, training data is your model
- AdaBoost: build multiple hypotheses and vote