CSE 537 Fall 2015

LEARNING FROM EXAMPLES
AIMA CHAPTER 18.9-11

Instructor: Sael Lee

Slides are mostly made from AIMA resources and slides from U of Texas Austin ML Group Ray Mooney

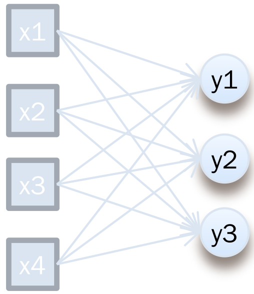
SUPPORT VECTOR MACHINES (18.9)

REVIEW

linear regression

$$lh_{\mathrm{w}}(\mathrm{x}) = (\mathrm{w}^T\mathrm{x})$$

* Perceptron



$$h_{\mathrm{w1}}(\mathrm{x}) = Logistic(\mathrm{w}^T\mathrm{x}) = \frac{1}{1+e^{-(\mathrm{w}^T\mathrm{x})}}$$

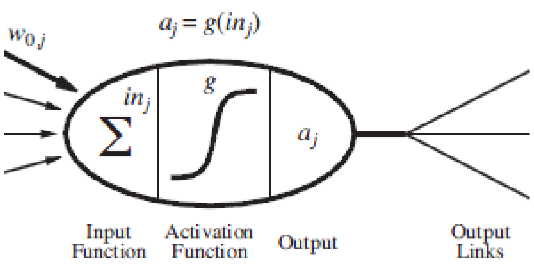
$$h_{\mathrm{w2}}(\mathrm{x}) = Logistic(\mathrm{w}^T\mathrm{x}) = \frac{1}{1+e^{-(\mathrm{w}^T\mathrm{x})}}$$

$$h_{\mathrm{w3}}(\mathrm{x}) = Logistic(\mathrm{w}^T\mathrm{x}) = \frac{1}{1+e^{-(\mathrm{w}^T\mathrm{x})}}$$

Logistic regression

Estimated prob. that y=1
on input x

$$h_{\mathrm{w}}(\mathrm{x}) = Logistic(\mathrm{w}^T\mathrm{x}) = \frac{1}{1+e^{-(\mathrm{w}^T\mathrm{x})}}$$



LEARNING THE WEIGHTS

$$w^* = argmin_w Loss(h_w)$$

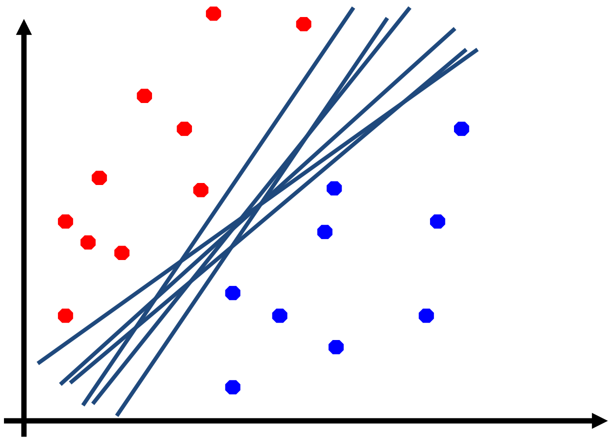
linear regression:

$$Loss(h_w) = \sum_{j=1}^{N} (y_i - (\mathrm{w}^T \mathrm{x}))^2$$

logistic regression:

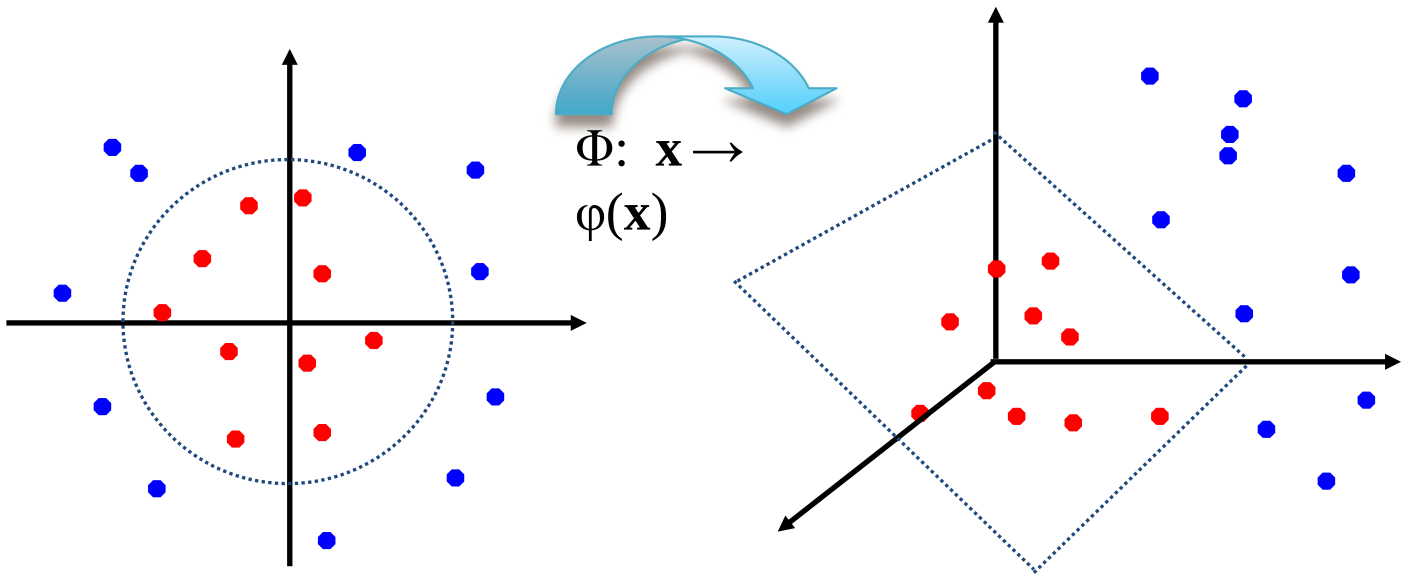
$$Loss(h_w) = \sum_{j=1}^{N} -y_i(\log(h_w(x))) - (y_i - 1)(\log(1 - h_w(x)))$$

- y is classification label in logistics regression (0 or 1)
- y is scalar values in linear regression



KERNELS

- The original feature space can always be mapped to some higher-dimensional feature space (even infinite) where the training set is separable



KERNEL TRICK

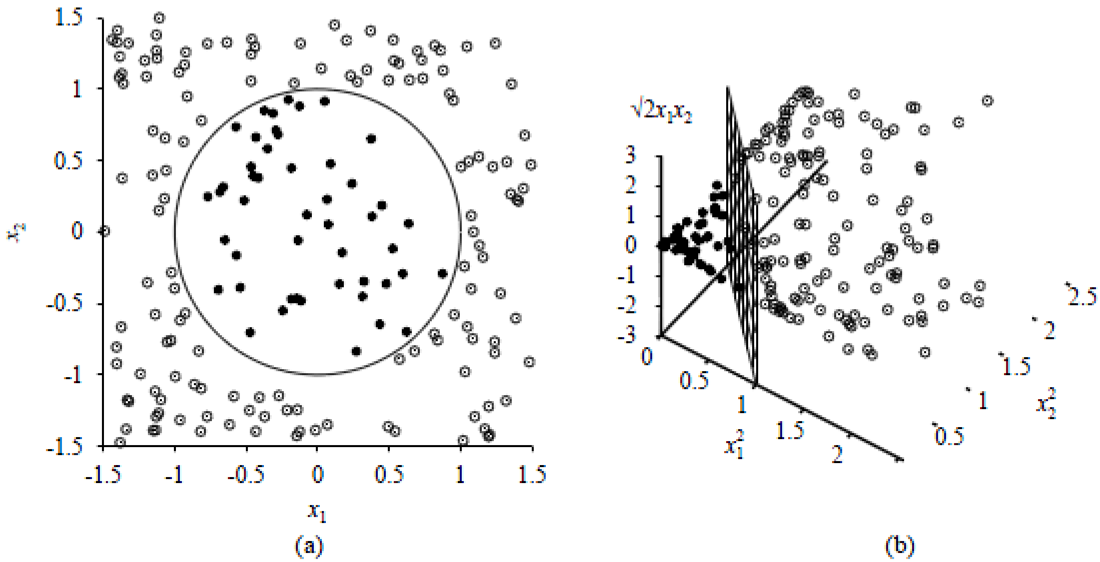


Figure 18.31 **FILES: .** (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions. Figure 18.29(b) gives a closeup of the separator in (b).

KERNELS

- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:
$$K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$$
- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

- Kernel function should measure some similarity between data
- kernel must be positive semi-definite

- You should scale the features to have same scale!!

- Most widely used is **linear kernels** and **Gaussian kernels**

GAUSSIAN KERNELS

$$k(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2}{2\sigma^2}\right)$$

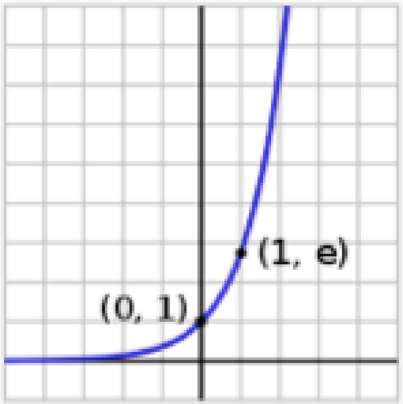
If x_i *and* x_j is similar:

$$k(x_i, x_j) \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x_i *and* x_j is different:

$$k(x_i, x_j) \approx \exp\left(-\frac{(large\ number)^2}{2\sigma^2}\right) \approx 0$$

If you use Gaussian kernel,
You will need to pick σ



The natural exponential function
$y = e^x$

SUPPORT VECTOR MACHINES

- SVMs constructs a maximum margin separator
- SVMs create a linear separating hyperplane
  - But have ability to embed that in to higher-dimensional space (via **Kernel trick**)
- SVM are a nonparametric method
  - Retain training examples an potentially need to store all or part of the data
  - Some example are more important then others (support vectors)

SVM TERMS

- Distance from example $\mathbf{x}_i$ to the separator is

$$r = \frac{(w^T x + b)}{||w||}$$

- Examples closest to the hyperplane are *support vectors.*
- ***Margin*** *ρ* of the separator is the distance between support vectors

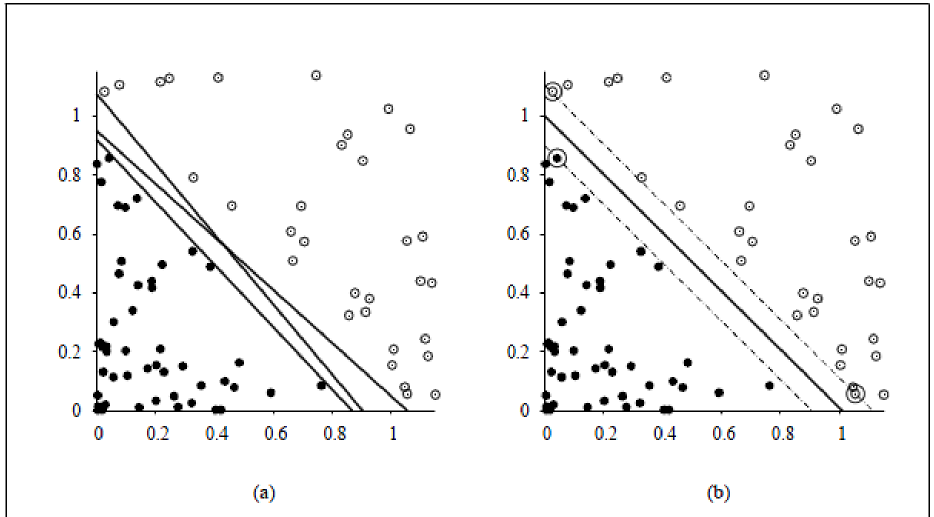
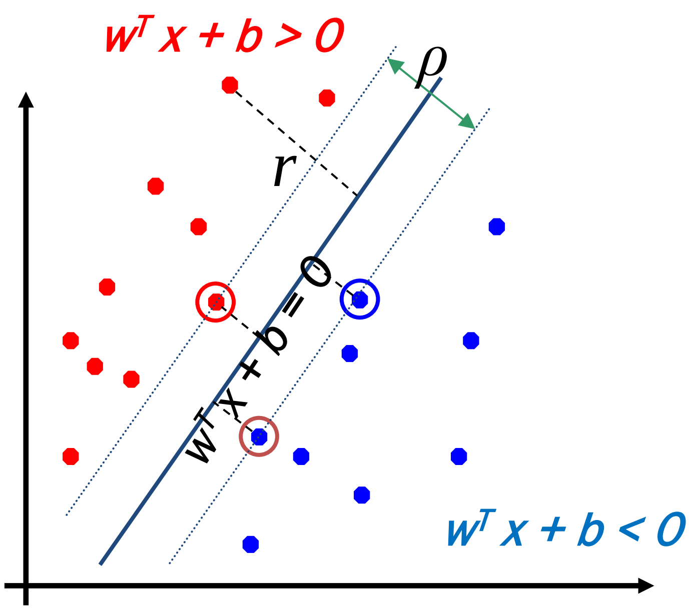


Figure 18.30 FILES: . Support vector machine classification: (a) Two classes of points (black and white circles) and three candidate linear separators. (b) The maximum margin separator (heavy line), is at the midpoint of the **margin** (area between dashed lines). The **support vectors** (points with large circles) are the examples closest to the separator.



MARGINS

Instead of minimizing expected empirical loss in the training data, SVM attempts to minimize expected generalization loss.

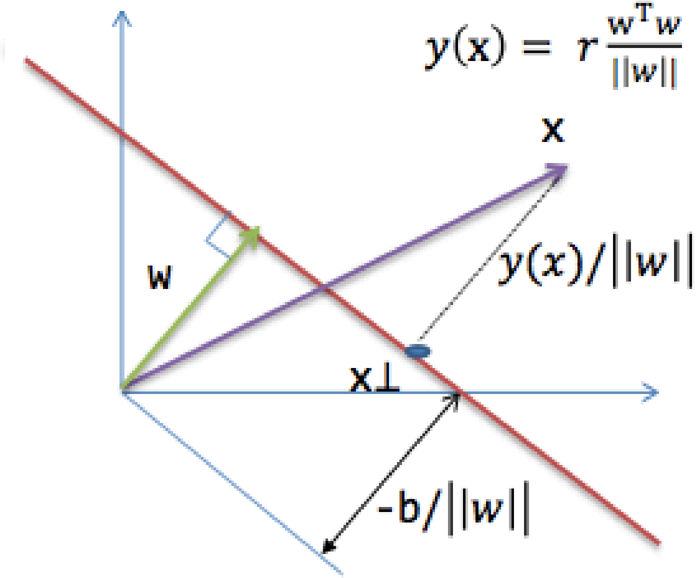
$y(x) = \mathrm{w}^{T}\mathrm{x} + b$ where w is weight vector and b is bias

$\mathrm{x} = \mathrm{x}\perp + r\frac{w}{||w||}$ (multiply w^{T} and add b)

$\mathrm{w}^{T}\mathrm{x} + b = \mathrm{w}^{T}(\mathrm{x}\perp + r\frac{w}{||w||}) + b$ $(y(x) = \mathrm{w}^{T}\mathrm{x} + b\ )$

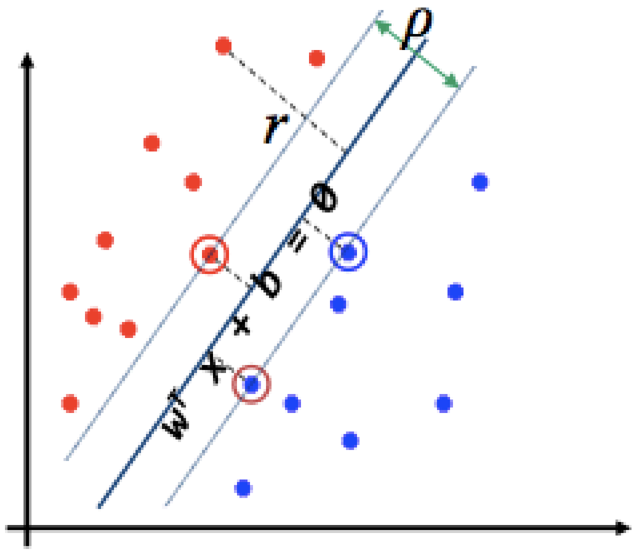
$y(x) = \mathrm{w}^{T}\mathrm{x}\perp + r\frac{\mathrm{w}^{T}w}{||w||} + b$ $(y(\mathrm{x}\perp) = \mathrm{w}^{T}\mathrm{x}\perp + b = 0)$

$y(x) = r\frac{\mathrm{w}^{T}w}{||w||} = \mathrm{I}$



or $r = {}^{(\mathrm{w}^{T}\mathrm{x}+\mathrm{b})}/_{||\mathrm{w}||}$

MAXIMUM MARGINS



Solving this is non-trivial and
will not be discussed in class

$\phi(x_n)$ in the feature space

$$r = {(w^T x + b)}/{||w||}$$

$$argmax_{w,b} \{\frac{1}{||w||} min_n [t_n (w^T x_n + b)]\}$$

$$argmin_{w,b} \frac{1}{2} ||w||^2$$

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

SOFT MARGINS

Idea: Allow data point to be in the wrong side of the margin boundary, but with a penalty that increases with the distance from that boundary.

Penalty for each data poin : slack variable ξ
$\xi_n = 0$ if point is on the right side
$\xi_n = |t_n - y(\mathrm{x}_n)|$ if point is on the wrong side
Such that
$t_n y(\mathrm{x}_n) \geq 1 - \xi_n$ for n = 1, …, N and $\xi_n \geq 0$

- $0 < \xi_n \leq 1$ for points inside the margin
- $\xi_n = 1$ for points on the margin
- $\xi_n > 1$ for points that are on the wrong side

Goal now is to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary

$$argmin_{w,b}\; C \sum_{n}^{N} \xi_n \;+ \frac{1}{2}||w||^2$$

OPTIMIZATION ON SOFT MARGINS

$$argmin_{w,b}\ C\sum_n^N \xi_n \quad + \frac{1}{2}||w||^2$$

subjected to $t_n y(\mathrm{x}_n) \geq 1 - \xi_n$ for n = 1, …, N and $\xi_n \geq 0$

ξ_n: slack variable
for training data x_n

Complex calculations
Lagrangian
Etc.

$$\mathrm{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathrm{x}_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

a_n is Lagrangian multiplier related to w_n

$$a_n = C - \mu_n$$

μ_n is Lagrangian multiplier related to ξ_n

$$\mathrm{b} = \frac{1}{N_M}\sum_{n\in M}(t_n - \sum_{n\in S}(a_m t_m k(x_n x_m))$$

PREDICTION USING KERNELS

$$y(x) = \mathrm{w}^T \phi(\mathrm{x_n}) + b$$

$$\mathrm{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathrm{x_n})$$

a_n is a Lagrangian multiplier

$$y(x) = \sum_{n=1}^{N} a_n t_n k(\mathrm{x}, \mathrm{x_n}) + b$$

New Data

Training data

Training data target (-1,1)

Use SVM packages: ex> libSVM
- there are numerical optimization steps you don’t want to code

Need to come up with
- Choice of Kernel
    - Choosing Kernels are critical
    - however you could choose not to use the kernels esp when the training set is small (linear kernel)
- Choice of parameter related to the kernel
    - Ex> Gaussian: σ
- Choice parameter C

When is SVM good
- Have medium size feature (1~1000) and have medium size training set (10~10,000)
- If you have many feature and little training set use logistic regression of linear kernels
- Little features and many training set use logistic regression of linear kernels because SVM is still slow.

ENSEMBLE LEARNING (18.10)

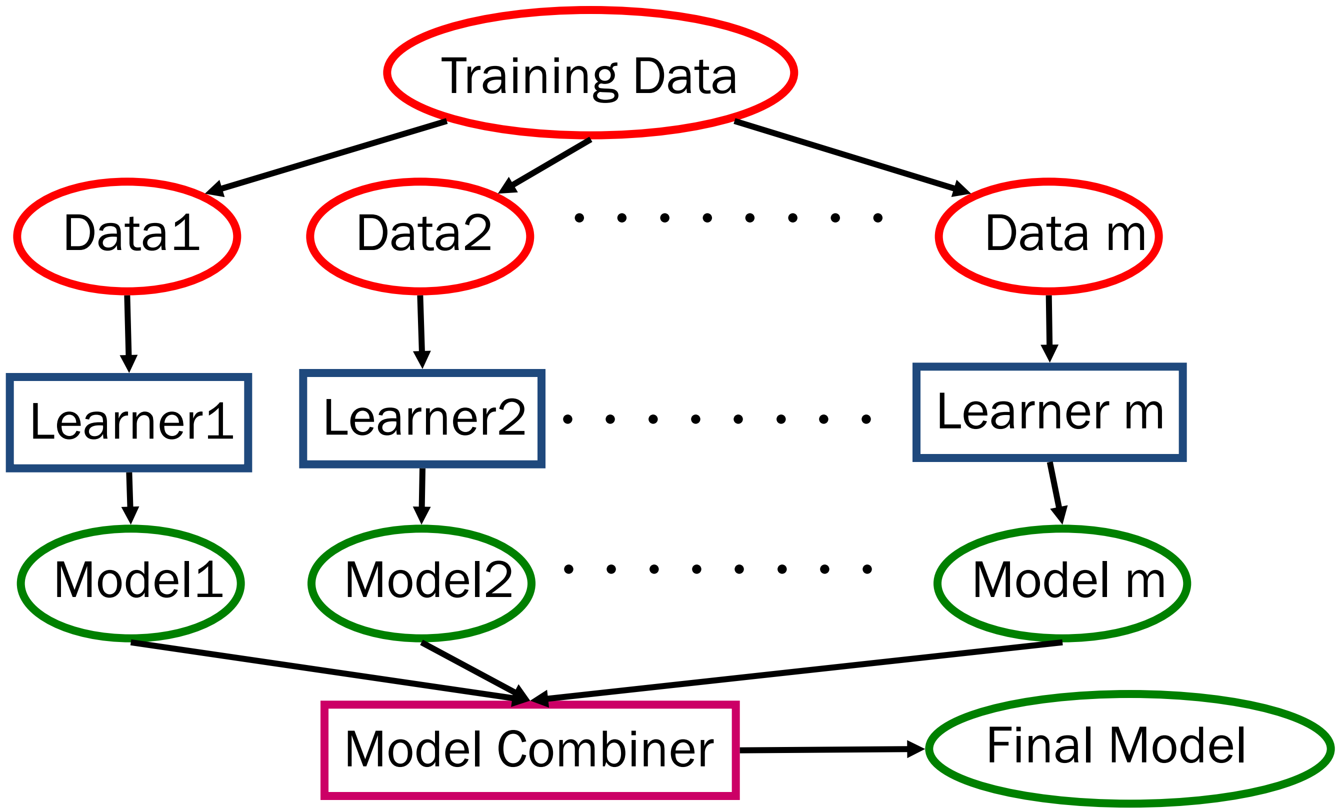
Adapted from CS4700 slides by Prof. Carla P. Gomes
gomes@cs.cornell.edu

ENSEMBLE LEARNING

- Idea: select a collection, or **ensemble**, of hypotheses from the hypothesis space and combine their predictions
  - Ex> During cross-validation we might generate twenty different decision trees, and have them vote on the best classification for a new example.
- Key motivation: reduce the error rate. Hope is that it will become much more unlikely that the ensemble of will misclassify an example.

LEARNING ENSEMBLES

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.
- Combine decisions of multiple definitions, e.g. using weighted voting.



Source: Ray Mooney

VALUE OF ENSEMBLES

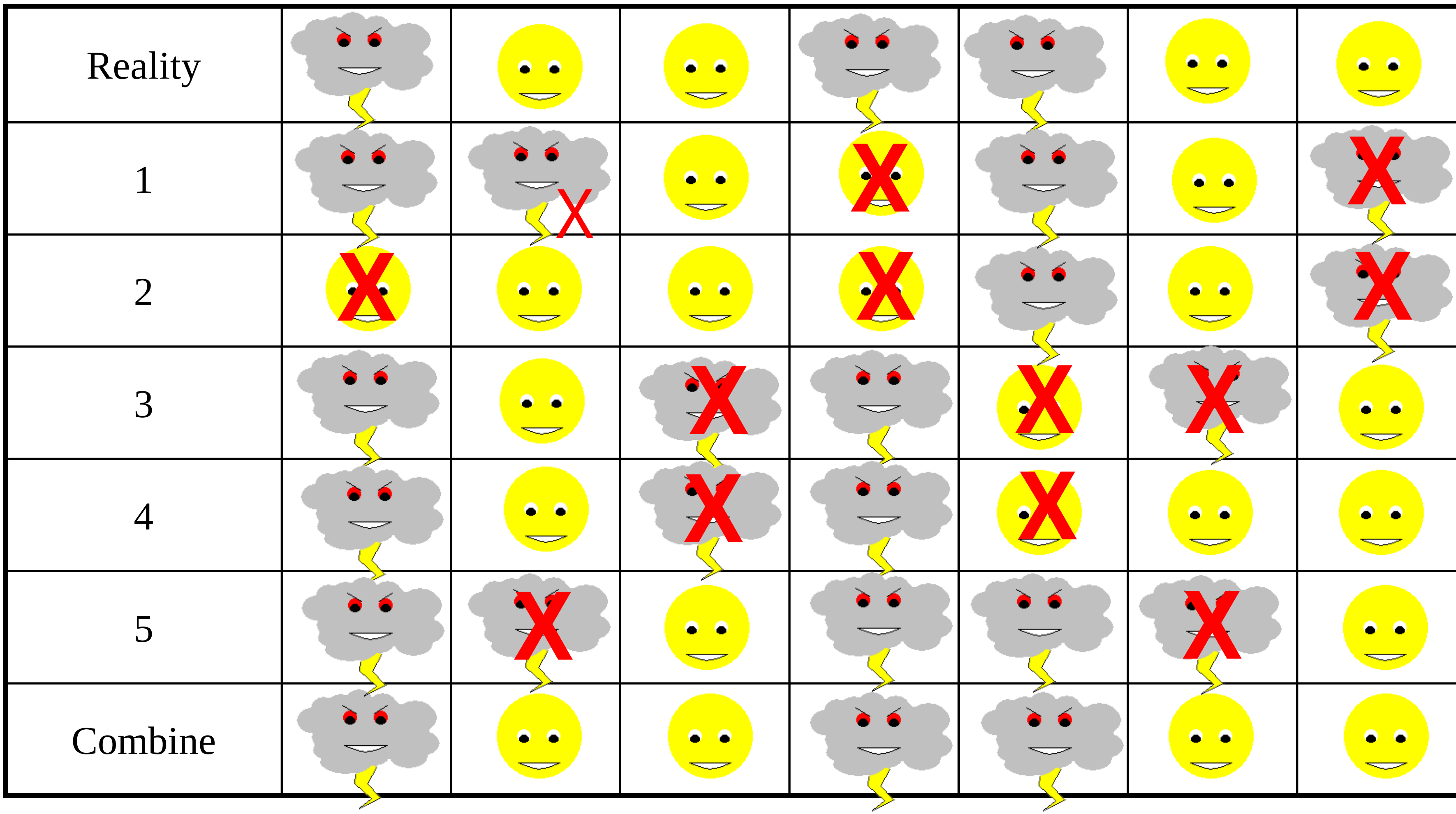
- "No Free Lunch" Theorem
  - No single algorithm wins all the time!

- When combing multiple independent and diverse decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.

- Examples: Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.

Source: Ray Mooney

EXAMPLE: WEATHER FORECAST



Source: Carla P. Gomes

Intuitions

- Majority vote
- Suppose we have 5 completely independent classif iers…
  - If accuracy is 70% for each
    - $(.7^5)+5(.7^4)(.3)+10(.7^3)(.3^2)$
    - **83.7% majority vote accuracy**
  - 101 such classifiers
    - **99.9% majority vote accuracy**

Note: Binomial Distribution: The probability of observing *x* heads in a sample of *n* independent coin tosses where in each toss the probability of heads is *p*, is

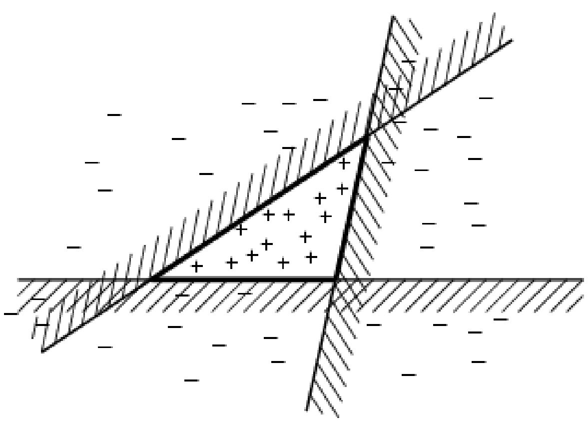
$$P(X = x|p, n) = \frac{n!}{r!(n-x)!}p^x(1-p)^{n-x}$$

Source: Carla P. Gomes

ENSEMBLE LEARNING

- Another way of thinking about ensemble learning:
- → way of enlarging the hypothesis space, i.e., the ensemble itself is a hypothesis and the new hypothesis space is the set of all possible ensembles constructible form hypotheses of the original space.

Increasing power of ensemble learning:



Three linear threshold hypothesis
(positive examples on the non-shaded side);
Ensemble classifies as positive any example classified positively be all three. The resulting triangular region hypothesis is not expressible in the original hypothesis space.

Source: Carla P. Gomes

DIFFERENT LEARNERS

- Different learning algorithms
- An algorithm with different choice for parameters
- Data set with different features
- Data set = different subsets

Source: Carla P. Gomes

HOMOGENOUS ENSEMBLES

- Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.
  - Data1 $\neq$ Data2 $\neq$ … $\neq$ Data m
  - Learner1 = Learner2 = … = Learner m
- Different methods for changing training data:
  - Bagging: Resample training data
  - Boosting: Reweight training data
- In WEKA, these are called ***meta-learners***, they take a learning algorithm as an argument (***base learner***) and create a new learning algorithm.

Source: Carla P. Gomes

BAGGING

- Create ensembles by *"bootstrap aggregation"*, i.e., repeatedly randomly resampling the training data (Brieman, 1996).

- Bootstrap: draw N items from X with replacement

- Bagging
  - Train *M* learners on *M* bootstrap samples
  - Combine outputs by voting (e.g., majority vote)

- Decreases error by decreasing the variance in the results due to ***unstable learners***, algorithms (like decision trees and neural networks) whose output can change dramatically when the training data is slightly changed.

Source: Carla P. Gomes

BAGGING - AGGREGATE BOOTSTRAPPING

- Given a standard training set *D* of size *n*
- For i = 1 .. M
  - Draw a sample of size $n^* < n$ from *D* uniformly and with replacement
  - Learn classifier C_i
- Final classifier is a vote of $C_1 .. C_M$
- Increases classifier stability/reduces variance

Source: Carla P. Gomes

STRONG AND WEAK LEARNERS

- Strong Learner (Objective of machine learning)
  - Take labeled data for training
  - Produce a classifier which can be *arbitrarily well-correlated with the true classification*

- Weak Learner
  - Take labeled data for training
  - Produce a classifier which is more accurate than random guessing

Source: Carla P. Gomes

BOOSTING

"**Boosting** is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance[1] in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones.[2] " (wikipedia)

- Weak Learner: only needs to generate a hypothesis with a training accuracy greater than 0.5, i.e., < 50% error over any distribution
- Learners
  - Strong learners are very difficult to construct
  - Constructing weaker Learners is relatively easy
- Questions: Can a set of **weak learners** create a single **strong learner** ?
- YES ☺

Boost weak classifiers to a strong learner

Source: Carla P. Gomes

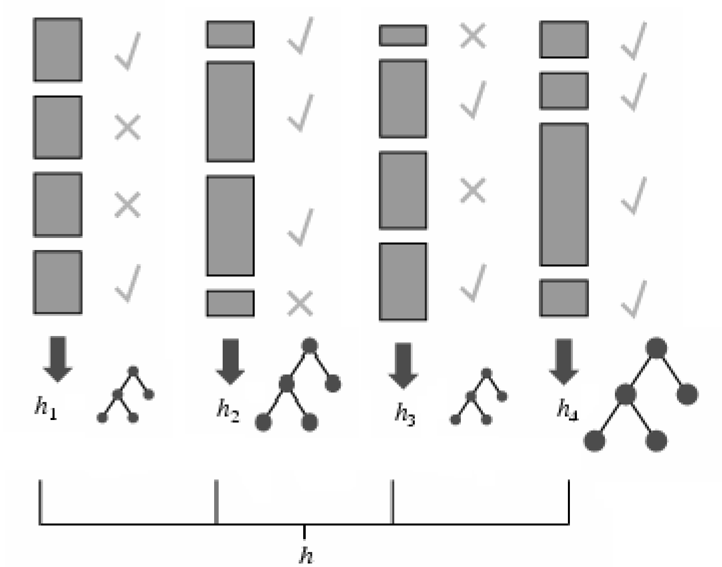
BOOSTING

- Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a ***weak learner*** that only needs to generate a hypothesis with a training accuracy greater than 0.5 (Schapire, 1990).
- Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).
- Key Insights
  - Instead of sampling (as in bagging) re-weigh examples!
  - Examples are given weights. At each iteration, a new hypothesis is learned (weak learner) and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.
  - Final classification based on weighted vote of weak classifiers

Source: Carla P. Gomes

ADAPTIVE BOOSTING

- Each rectangle corresponds to an example,
- with weight proportional to its height.
- Crosses correspond to misclassified examples.
- Size of decision tree indicates the weight of that hypothesis in the final ensemble.



Source: Carla P. Gomes

CONSTRUCT WEAK CLASSIFIERS

- Using Different Data Distribution
  - Start with uniform weighting
  - During each step of learning
    - Increase weights of the examples which are not correctly lea rned by the weak learner
    - Decrease weights of the examples which are correctly learn ed by the weak learner
- Idea
  - Focus on difficult examples which are not correctly cl assified in the previous steps

Source: Carla P. Gomes

COMBINE WEAK CLASSIFIERS

- Weighted Voting
  - Construct strong classifier by weighted voting of the weak classifiers
- Idea
  - Better weak classifier gets a larger weight
  - Iteratively add weak classifiers
    - Increase accuracy of the combined classifier through minimi zation of a cost function

Source: Carla P. Gomes

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

   for k = 1 to K do
       h[k] ← L(examples, w)
       error ← 0
       for j = 1 to N do
           if h[k](x_j) ≠ y_j then error ← error + w[j]
       for j = 1 to N do
           if h[k](x_j) = y_j then w[j] ← w[j] · error/(1 − error)
       w ← NORMALIZE(w)
       z[k] ← log (1 − error)/error
   return WEIGHTED-MAJORITY(h, z)
```

Figure 18.34 The ADABOOST variant of the boosting method for ensemble learning. The algorithm generates hypotheses by successively reweighting the training examples. The function WEIGHTED-MAJORITY generates a hypothesis that returns the output value with the highest vote from the hypotheses in **h**, with votes weighted by **z**.

ADAPTIVE BOOSTING: HIGH LEVEL DESCRIPTION

- C =0; /* counter*/
- M = m; /* number of hypotheses to generate*/

- 1 Set same weight for all the examples (typically each example has weight = 1);

- 2 While (C < M)
- 2.1 Increase counter C by 1.
- 2.2 Generate hypothesis h_C .
- 2.3 Increase the weight of the misclassified examples in hypothesis h_C
- 3 Weighted majority combination of all M hypotheses (weights according to how well it performed on the training set).

- Many variants depending on how to set the weights and how to combine the hypotheses. ADABOOST → quite popular!!!!

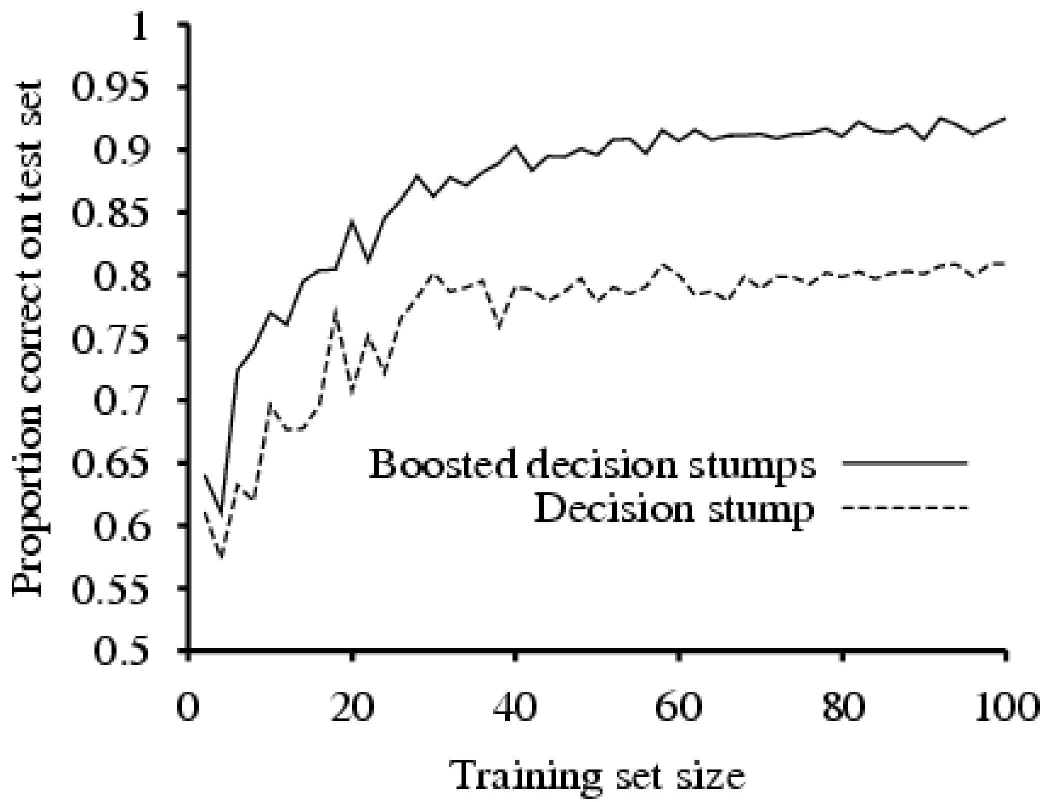
Source: Carla P. Gomes

PERFORMANCE OF ADABOOST

- Learner = Hypothesis = Classifier
- Weak Learner: < 50% error over any distribution
- M number of hypothesis in the ensemble.
- If the input learning is a Weak Learner, then ADABOOST will ret urn a
- hypothesis that classifies the training data perfectly for a large enough M,
- boosting the accuracy of the original learning algorithm on the t raining
- data.
- Strong Classifier: thresholded linear combination of weak learn er outputs.

Source: Carla P. Gomes

RESTAURANT DATA



Decision stump: decision trees with just one test at the root.

Source: Carla P. Gomes