# Nonlinear models for classification

Devika Subramanian

Rice University

*devika@rice.edu*

# Outline

# Recap of models for classification

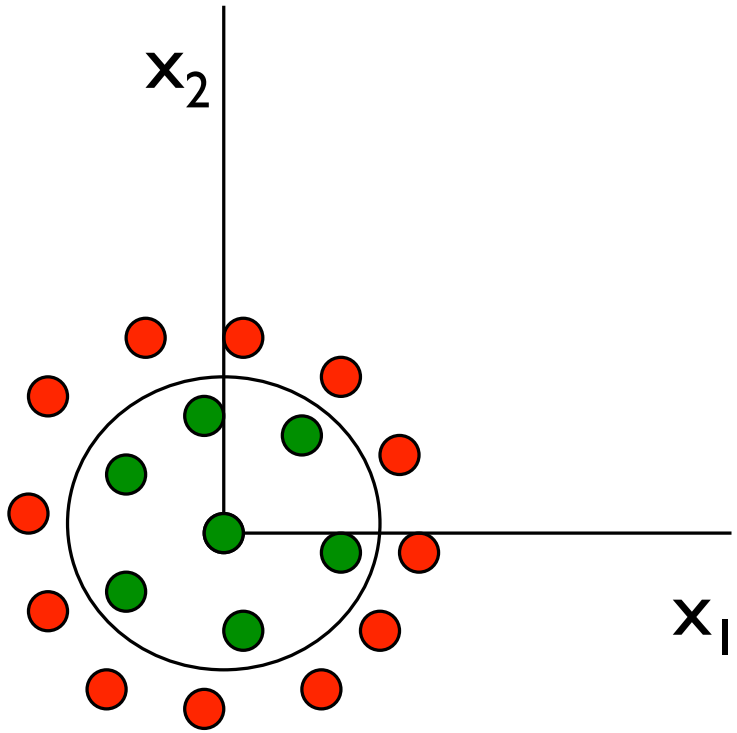
Given $\mathcal{D} = \{(x^{(i)}, y^{(i)}) | 1 \leq i \leq m, x^{(i)} \in \Re^d, y^{(i)} \in \{0, 1\}\}$, we have identified two parametric classes of models for classification

- discriminative models: e.g., logistic regression where $P(y = 1|x) = g(\theta^T x)$ and parameter $\theta$ is estimated by minimizing the cross-entropy $J$ function.
- generative models: e.g., GDA and Naive Bayes where the joint distribution $P(xy)$ is estimated in terms of components $P(y)$ and $P(x|y)$ with appropriate parametric forms.

Both models yield linear separating hyperplanes of the form $\theta^T x = 0$ for a parameter vector $\theta$.

# Can these two classes be separated?

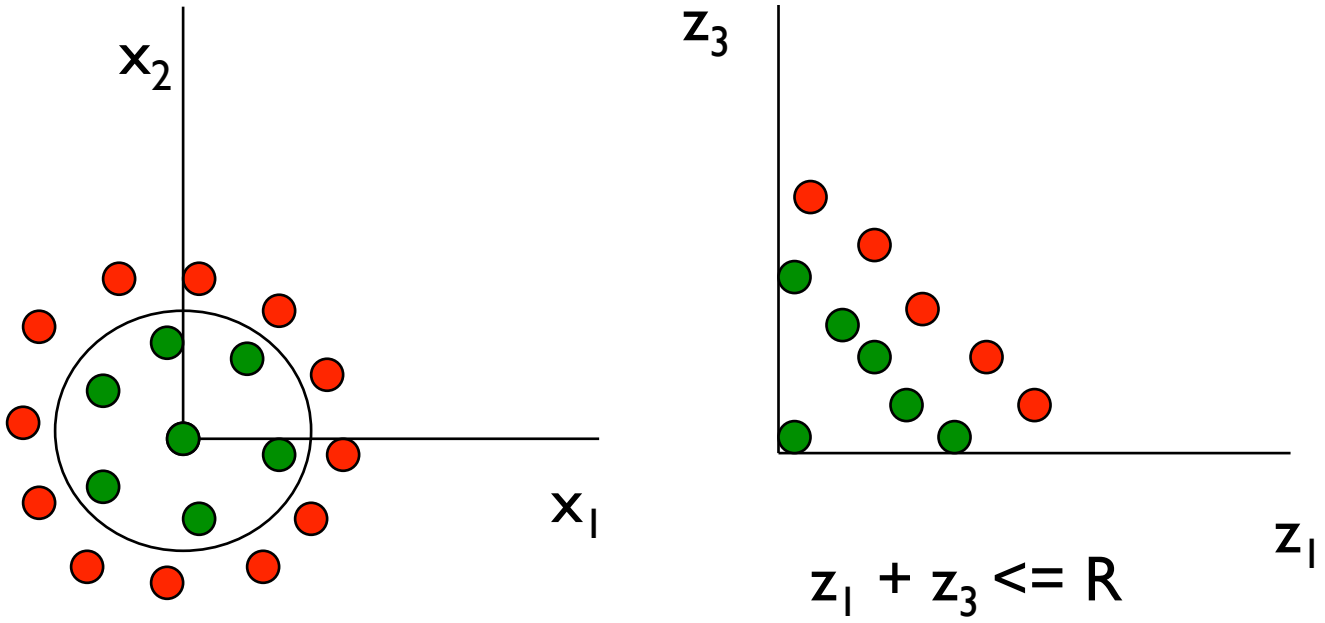
yes, basis function expansion



# Yes!

We use expanded basis functions and map each $(x_1, x_2)$ pair to $(x_1^2, x_2^2)$.

$$\phi((x_1, x_2)) = (x_1^2, x_2^2) = (z_1, z_3)$$



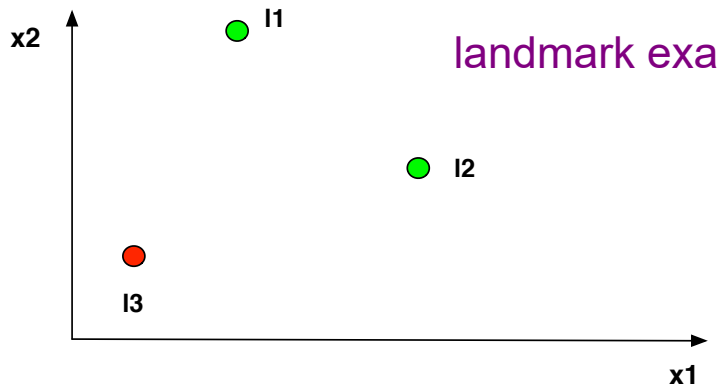

Now, the linear separating hyperplane can be characterized by $\theta^T \phi(x) = 0$.

# Approaches to constructing nonlinear classifiers

Nonlinear separating boundaries can be learned by linear models, but it places the burden of defining appropriate $\phi$ functions on the human.
There are two approaches to constructing nonlinear classifiers without explicit construction of basis functions.

- Compute classifications based on similarity between examples: kernel methods.
- Construct models by chaining together or layering simpler learning models: decision trees, neural networks, ensemble models.

# Kernel methods: the intuition



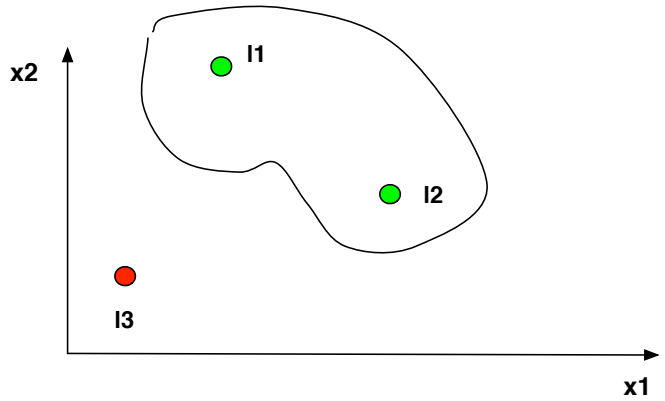

Let the input space be $\Re^2$ and let $l^{(1)}$, $l^{(2)}$, and $l^{(3)}$ be three labeled landmark points in that space. We define a classifier $h_\theta(x)$ parameterized by $\theta$ as follows:

$$h_\theta(x) = \theta_0 + \theta_1 similarity(x, l^{(1)}) + \theta_2 similarity(x, l^{(2)}) + \theta_3 similarity(x, l^{(3)})$$

Note that the new "features" are defined not just on $x$ alone, but on the relationship between $x$ and the three landmarks.

The decision rule for classification is: if $h_\theta(x) \geq 0$ then $y = 1$ else $y = 0$.

# Kernel methods: an example




$$h_\theta(x) = -0.5 + 1 * similarity(x, l^{(1)}) + 1 * similarity(x, l^{(2)}) + 0 * similarity(x, l^{(3)})$$

Eucl. dist

$$similarity(x, l) = exp\left(-\frac{||x - l||^2}{2\sigma^2}\right) \quad \text{(Gaussian kernel)}$$

values between 0 and 1

Key idea: we can form complex boundaries with features that are based on similarities between x and landmarks.

# How to use kernels for classification

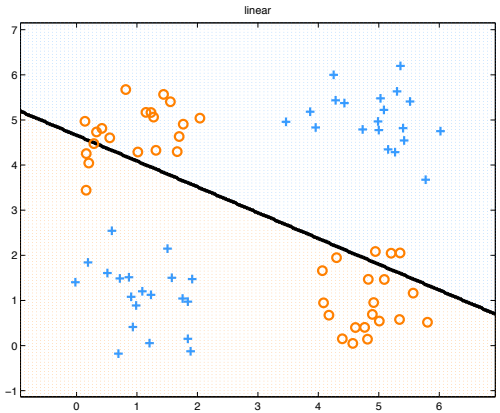
Given $\mathcal{D} = \{(x^{(i)}, y^{(i)}) | 1 \leq i \leq m, x^{(i)} \in \Re^d, y^{(i)} \in \{0, 1\}\}$, choose all the points in $\mathcal{D}$ as landmarks. Then, represent each $x^{(i)}$ by a feature vector of length $m$ as follows:

re-representing every point ~ similarity val.

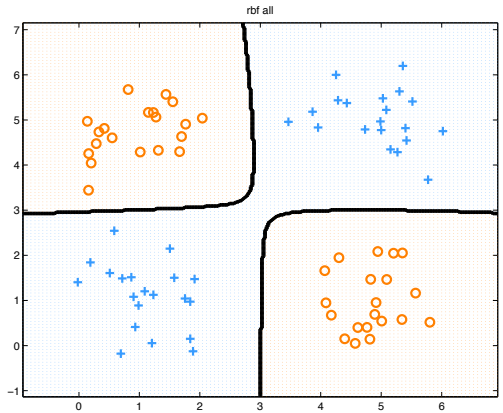
$$x^{(i)} \rightarrow \begin{bmatrix} sim(x^{(i)}, x^{(1)}) \\ \dots \\ sim(x^{(i)}, x^{(m)}) \end{bmatrix} = fx^{(i)}, fX = \begin{bmatrix} 1 & \text{———} fx^{(1)\,T} \text{———} \\ 1 & \text{———} fx^{(2)\,T} \text{———} \\ \dots & \\ 1 & \text{———} fx^{(m)\,T} \text{———} \end{bmatrix}$$

Now we can use regularized logistic regression in the usual way to estimate $\theta \in \Re^{m+1}$. Given a new example $x$, we can compute $g(\theta^T fx)$ to predict the class associated with $x$.

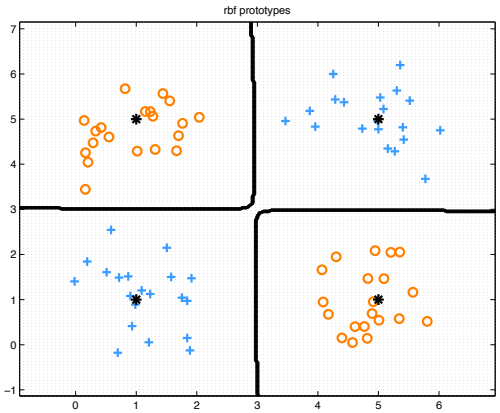
# Example: kernelized logistic regression




**logistic**




**all points are landmarks**




m + 1
more param than
number of features
~ don't want to use
all points as
landmarks

# How to pick landmarks

- Use all examples in $\mathcal{D}$ as landmarks and use aggressive regularization, especially if $m$ is large.
- Cluster the examples in $\mathcal{D}$ and choose cluster centers as landmarks.
- if the input space is $\Re^d$ where $d$ is small, choose landmarks that tile $\Re^d$ uniformly.

# Similarity/kernel functions

A similarity or kernel function $k$ measures how similar two $x$'s from $\Re^d$ are.

$$k : \Re^d \times \Re^d \rightarrow \Re$$

Examples of kernel functions

- Gaussian kernel

scaled measure bet. 0 and 1

$$k(x, x') = exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

- Second-order polynomial kernel

euclidean distance

$$k(x, x') = (x^T x')^2$$

# Mercer's theorem

build a kernel from scratch

choosing a kernel effect. is choosing a basis function

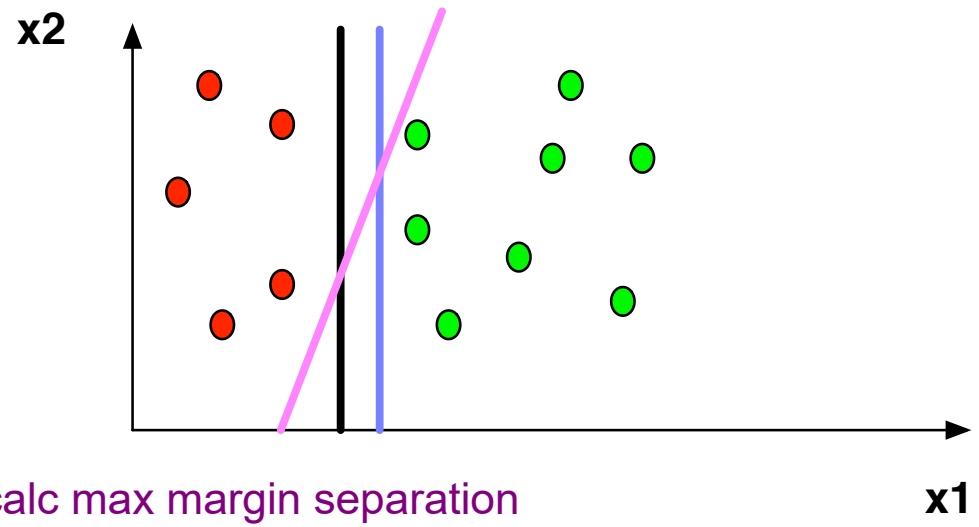
Mercer's theorem provides a necessary and sufficient condition for a function $k(x, x')$ to be a valid kernel. The $m \times m$ Gram matrix $K$ whose elements are $k(x^{(i)}, x^{(j)})$, $1 \leq i, j \leq m$ should be positive definite for all choices of the set $\{x^{(i)} : 1 \leq i \leq m\}$.

If $K$ is positive definite, then there exists a basis function $\phi$ such that every entry $k(x^{(i)}, x^{(j)})$ in $K$ can be written as

$$k(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$$

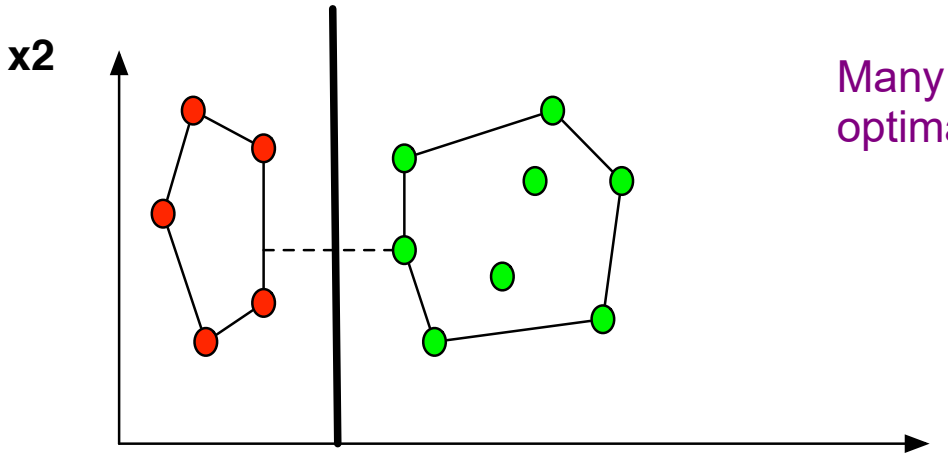
By using the kernel function $k$ we bypass the construction of $\phi$. For example, if $d = 16 \times 16$ and we consider all fifth-order polynomial terms on the original feature space, we would have to construct a feature space of size $10^{10}$ if we explicitly constructed the basis function $\phi$. Instead, we can use a 5th order polynomial kernel and compute $(x^T x')^5$ in $O(d)$ time.

# Separating hyperplanes




- When the training examples are linearly separable there are many choices for separating hyperplanes. We can use the notion of margin (distance of a point from a separating boundary) and formulate an optimization problem in which we seek the separating hyperplane with the maximum margin.

# Maximum margin separating hyperplanes



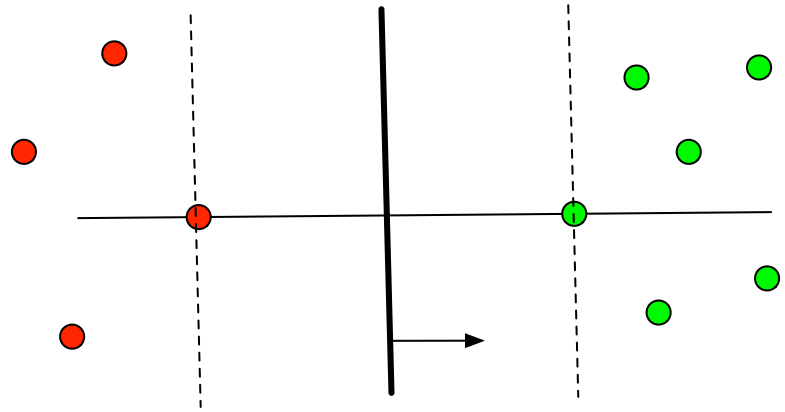

Many line options but seek optimal answer

convex hull -- construct smallest polygon which encloses all points

- The maximum margin separating hyperplane is the perpendicular bisector of the line joining the closest points on the convex hulls of the two (linearly separable) classes. The idea is to separate the two classes as much as possible – framed as an optimization problem, it yields a unique answer for the separating hyperplane.

Additional reading http://svms.org/tutorials/Burges1998.pdf

# Formulating the optimization problem for SVM



negative distance <--decision boundary--> positive distance

- The decision boundary is characterized by $\theta_0 + \theta^T x = 0$. The margin of a point $(x, y)$ from this decision boundary is $\frac{\theta^T x + \theta_0}{||\theta||}$ and is either positive or negative based on which side of the boundary it is on.
- The location of the decision boundary is dependent only on the points that are on the margin. Points far from the boundary do not affect $\theta$. Points on the margin are called support vectors.

# The SVM optimization problem (separable case)

- For separable data, the problem of finding the maximum margin separating hyperplane can be stated as

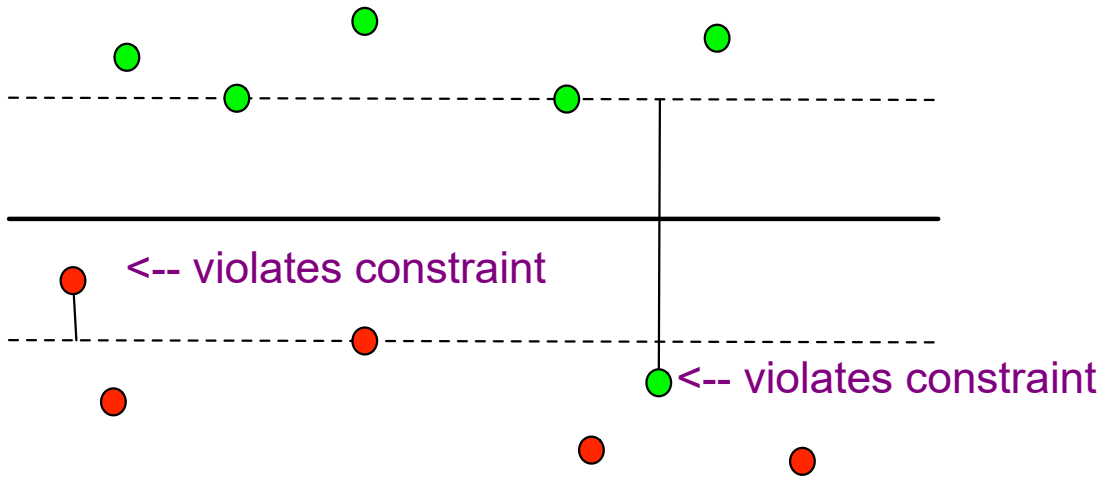
$$min \frac{1}{2}||\theta||^2$$

labeled +1 and -1 (not 1 and 0)

subject to the constraints $y^{(i)}(\theta^T x^{(i)} + \theta_0) \geq 1$ for $1 \leq i \leq m$

- Can be solved by a quadratic program solver.

# The SVM optimization problem (non-separable case)




- For non-separable data, the problem can be formulated as

$$min \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{m} \xi^{(i)}$$

points violating constraint get a positive Ci ~ penalty

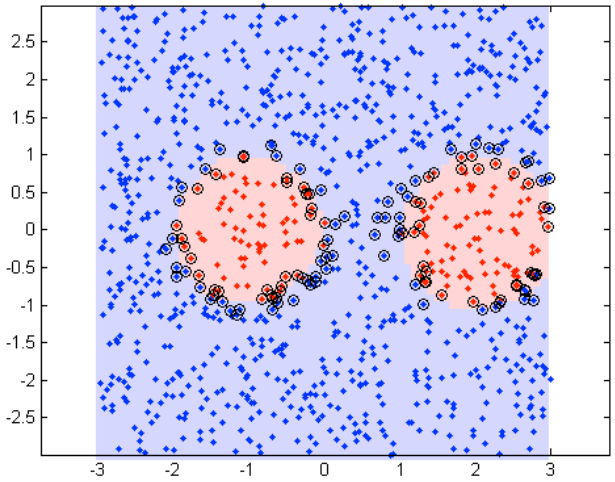
subject to the constraints $y^{(i)}(\theta^T x^{(i)} + \theta_0) \geq 1 - \xi^{(i)}$ for $1 \leq i \leq m$ where $C$ is a regularization parameter and $\xi$ is a slack variable denoting distance of a point from its margin.

# SVMs and kernels

The SVM decision function has the form

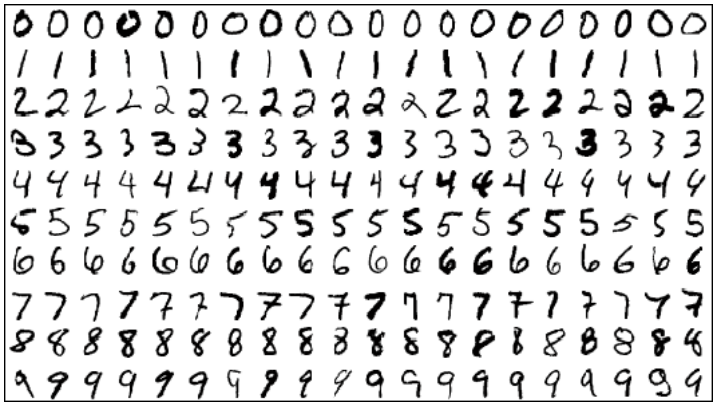
$$sign(\theta^T x + \theta_0) = sign(\sum_{i=1}^{m} y^{(i)}(\alpha^{(i)} {x^{(i)}}^T x + \theta_0))$$

where $\alpha^{(i)}$ is the importance weight calculated for all examples by the QP solver. Only support vectors have non-zero $\alpha^{(i)}$. We can replace the dot product between $x^{(i)}$ and $x$ by a kernel function $k(x^{(i)}, x)$ and have the SVM learn non-linear decision boundaries.



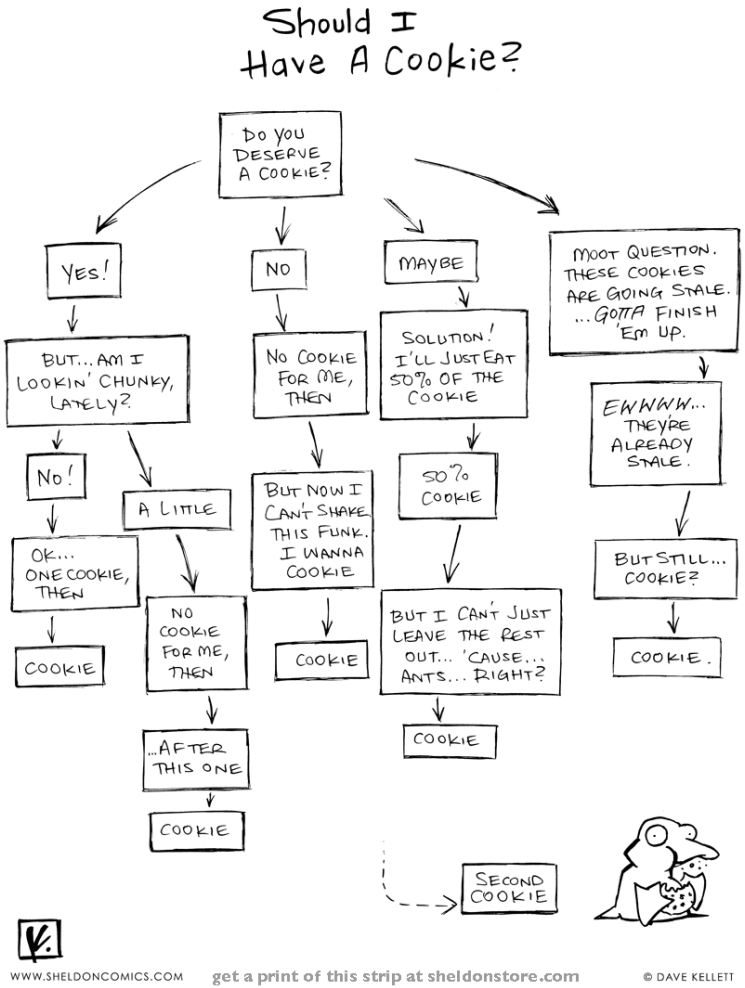
# SVM performance on the MNIST problem

usps

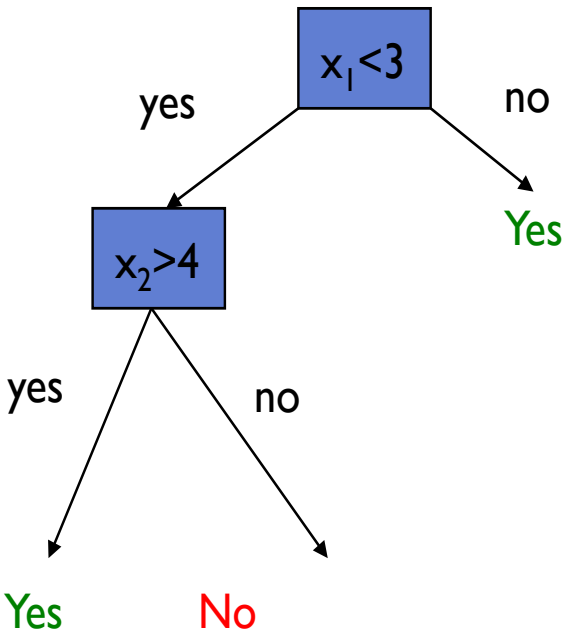


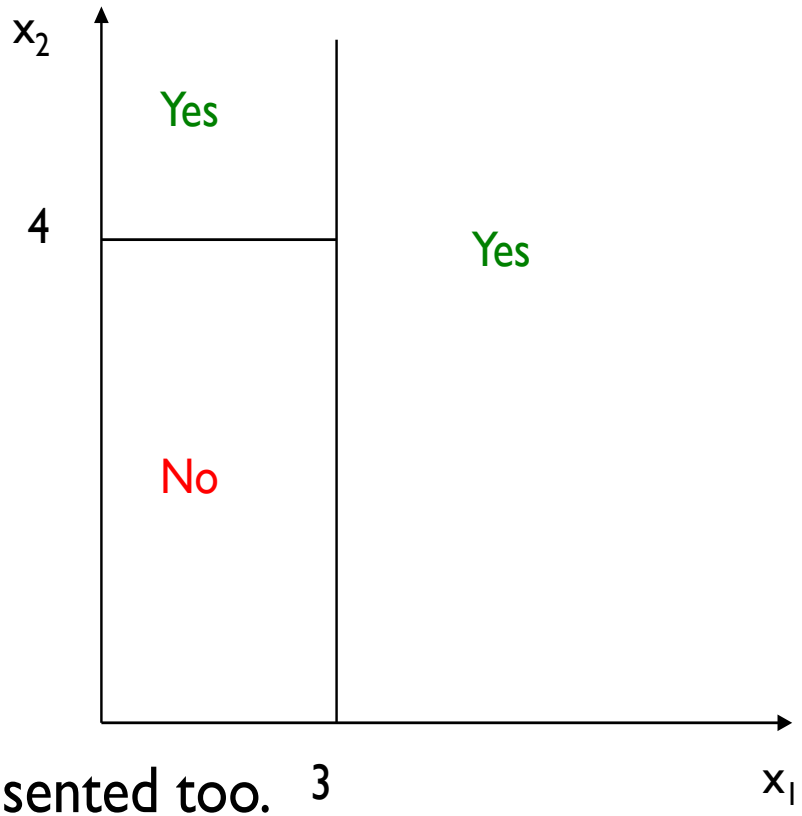
| Classifier | Test error |
|---|---|
| Linear SVM | 8.4% |
| Gaussian kernel SVM | 1.4% |
| 3-NN | 2.4% |
| LeNet | 1.1% |
| Boosted LeNet | 0.7% |

# A decision tree



# Decision trees




Continuous attributes can be represented too.
Decision boundaries are axis-parallel rectangles.

# Decision tree representation

- Each internal node tests a feature.
- Each branch corresponds to a feature value. When features are continuous decision boundaries are axis-parallel rectangles.
- Each leaf node assigns a classification.
- Any Boolean formula can be represented as a decision tree (by converting it to DNF).
- Try representing the majority function (e.g., 2-out-of-3), and the xor function on n boolean variables.

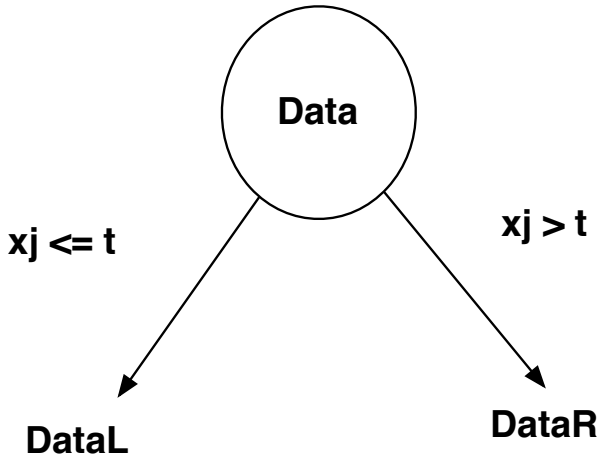
# Decision trees and linear classifiers

- Define three features corresponding to each path in the example decision tree.
  - $\phi_1(x) = [x_1 < 3 \text{ and } x_2 > 4]$
  - $\phi_2(x) = [x_1 < 3 \text{ and } x_2 \leq 4]$
  - $\phi_3(x) = [x_1 \geq 3]$
- Assign weights to the features based on the majority label at the leaf at the end of the path.
  - $\theta_1 = 1, \theta_2 = -1, \theta_3 = 1$
- Final decision function is $\theta_1\phi_1(x) + \theta_2\phi_2(x) + \theta_3\phi_3(x)$

this linear function is an exact replica of the decision tree

look ma no hands but optimal decision tree, being greedy, is a hard problem -- growing ~ just approximations (no guarantee on quality, no bounds on how bad); cannot specify loss function

# Growing decision trees




- A greedy recursive procedure to repeatedly consider splitting the data on some feature $x_j$ and a threshold $t$.
- Three split criteria: misclassification rate, entropy, Gini index.
- Post pruning with a validation set to control overfitting.
- Excellent base classifier for ensemble methods.

not recomm. recomm. recomm.

tweak parameter to try for example try 10 times, etc.

# The CART algorithm

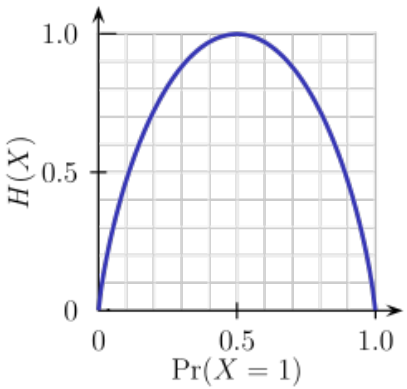
- Inputs:
  - $\mathcal{D} = \{(x^{(i)}, y^{(i)}) | x^{(i)} \in \Re^d, y^{(i)} \in \{0, 1\}; 1 \leq i \leq m\}$
  - The features are the column names of the data matrix $X$ $(m \times d)$ and the target is the vector $y$ $(m \times 1)$.
- Output: a decision tree on the features (root and internal nodes are tests on features), and leaves are labeled by the values of the target.

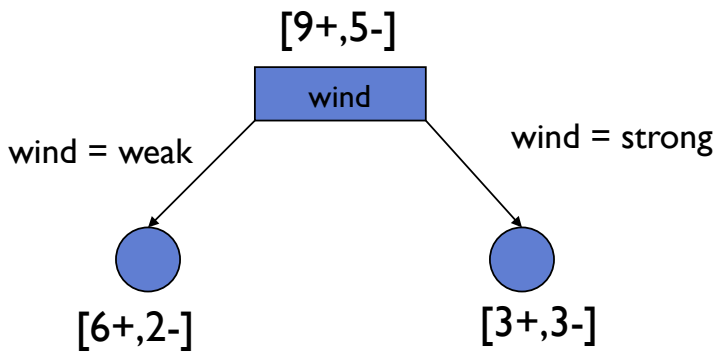
greedy - will get stuck on local minima
DS never goes beyond 10 deep

# The CART Algorithm (X,y,features)

- Init: Create a root node for the tree.
- Base Case 1: If all examples in $X$ are labeled 1 (resp. 0) return the single node tree root with label 1 (resp. label 0).
- Base case 2: If features is empty, return the single node tree root with label = most common value of target in $y$.
- Recursive case:
  - Let A be the feature from features that best classifies $X$. Make A the root of the decision tree.
  - For each possible value v of A
    - Let X[X.A==v] be the subset of X for which feature A equals v
    - Add a new tree branch below root corresponding to the test A = v
    - If X[X.A==v] is empty, then add a new leaf with label = most common value of target in y[X.A==v]
    - Else add the subtree CART(X[X.A==v],y[X.A==v],features-{A})

# How to split a node in a decision tree: entropy




$$H(p) = -plogp - (1-p)log(1-p)$$

There are 9 positive and 5 negative examples at the root.

$$H(9/14) = 9/14log(9/14) - 5/14log(5/14) = 0.94$$

# How to split a node in a decision tree: conditional entropy

- Let $Y$ and $C$ be two random variables with joint distribution $P(Y, C)$. The conditional entropy of $Y$ with respect to $C$, $H(Y|C)$ is defined as

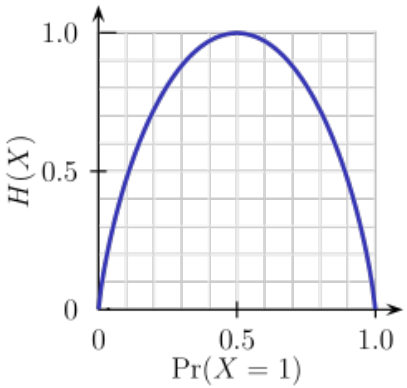
$$H(Y|C) = -\sum_{(y,c)} p(y,c) log p(y|c) = -\sum_{(y,c)} p(c)p(y|c) log p(y|c)$$

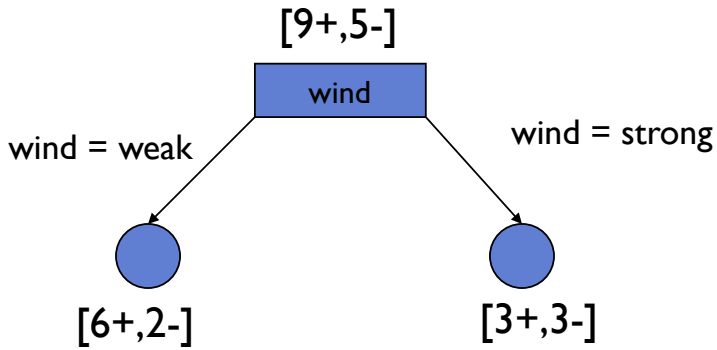
- The information gain of $Y$ with respect to $C$ is

$$IG(Y|C) = H(Y) - H(Y|C)$$

- How much is the entropy of $Y$ reduced by observing $C$?

# How to split a node in a decision tree: information gain




$$
\begin{aligned}
Gain(PlayTennis|wind) &= H(PlayTennis) - H(PlayTennis|wind) \\
&= 0.94 - [-p(wind = weak) \\
&\quad [-6/8 log 6/8 - 2/8 log 2/8] \\
&\quad -p(wind = strong) \\
&\quad [-3/6 log 3/6 - 3/6 log 3/6]] \\
&= 0.94 - (8/14)0.811 - (6/14)1.00 = 0.048
\end{aligned}
$$

# Properties of CART

- It is a greedy algorithm that tries to make all the leaves pure (belong to one class).
- It can get stuck in local minima.
- Splits are made on information-theoretic grounds: it is therefore robust to noisy data.
- CART can overfit by introducing splits that improve accuracy for the training data, but degrade test set accuracy.

# Approaches to controlling overfitting in CART

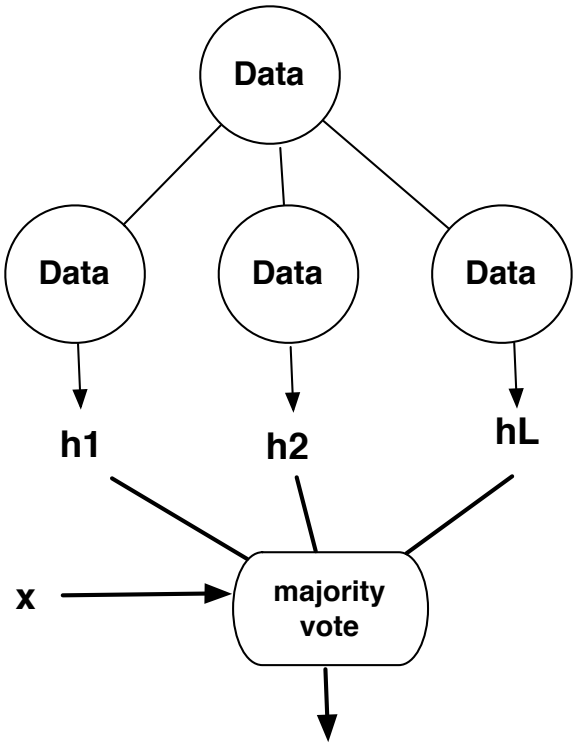
- Early stopping: stop growing tree when data splits not statistically significant. Apply the chi-squared test to determine whether splitting a node improves validation set accuracy.
- Post pruning: Grow the full tree, then systematically prune nodes using a validation set.
- Minimum Description Length-based stopping criteria: use an explicit measure of the complexity of encoding the training examples and decision tree, halting growth when this encoding size is minimized.

# Pros/Cons of decision trees

- Pros:
  - easy to interpret
  - handle continuous and discrete features
  - insensitive to data scaling
  - perform automatic feature selection
  - robust to outliers
- Cons:
  - axis-aligned cuts limit accuracy
  - high variance estimators (unstable)

    hence run estimators 10 times, etc.

# Ensembles: bagging – the idea




uniform random sampling with replacement = bootstrap

# Ensembles: bagging – when it works

An ensemble of classifiers is more accurate than its component classifiers if the errors made by the component classifiers are uncorrelated, AND each has an error rate $< 0.5$. better than a coin; max error should be > 50%

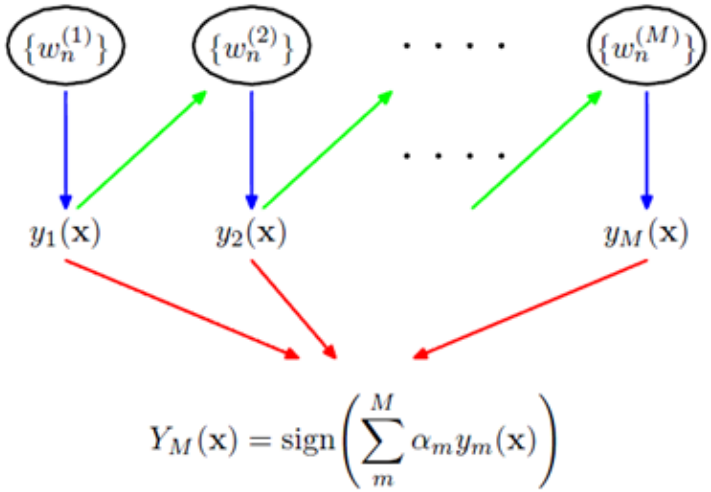
Let there be L hypotheses, each with an individual error rate of $p$. If we use majority vote to determine the class, then the probability that the majority vote is wrong is

$$\sum_{k=L/2+1}^{L} p^k (1-p)^{L-k} \binom{L}{k}$$

If $L = 21$ and $p = 0.3$, then the probability that the ensemble is wrong is 0.026.

Random forest: bagged ensemble of decision trees where each tree is learned on randomly selected subset of features of size $\sqrt{d}$.

# Ensembles: boosting – intuition




- Members of the ensembles are learned sequentially, with each member focusing on the mistakes made by the previous members in the ensemble.
- Weights are associated with each example to focus each classifier on these mistakes. Each classifier also has a weight associated with its decision.
- Final decision is a weighted majority vote.

# Ensembles: the Adaboost algorithm

Most popular

- Initialize $w_1^{(i)} = 1$ for $i = 1, \ldots, m$.
- for $l = 1 \ldots L$ do
  - Fit classifier $h_l$ to the training data to minimize the weighted loss function $\sum_{i=1}^{m} w_l^{(i)} L(y^{(i)}, h_l(x^{(i)}))$
  - Evaluate error $\epsilon_l$ and use it to calculate the vote weight $\alpha_l$ of classifier $h_l$

.3 = .42

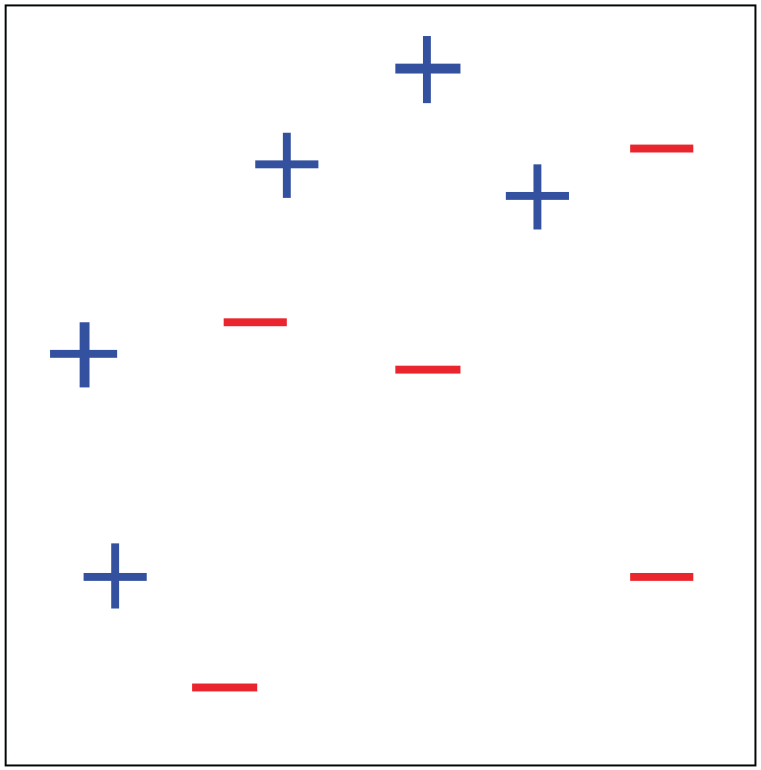
$$\epsilon_l = \frac{\sum_{i=1}^{m} w_l^{(i)} I(h_l(x^{(i)}) \neq y^{(i)})}{\sum_{i=1}^{m} w_l^{(i)}} \quad (1)$$

$$\alpha_l = \frac{1}{2} ln \frac{1 - \epsilon_l}{\epsilon_l} \quad (2)$$

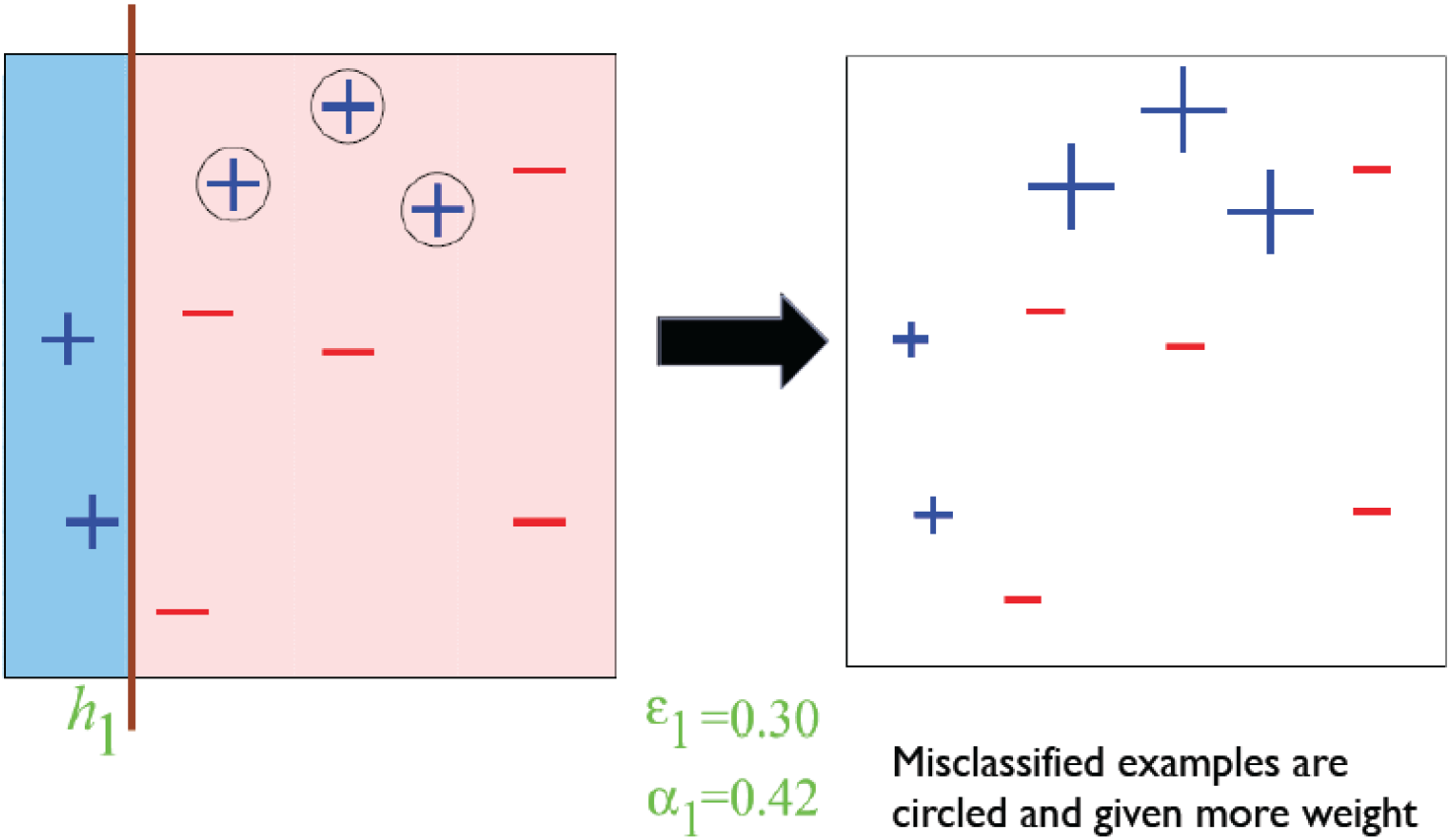
  - Update $w$: $w_{l+1}^{(i)} = w_l^{(i)} exp(\alpha_l I(h_l(x^{(i)}) \neq y^{(i)}))$
- Final decision: $\sum_{l=1}^{L} \alpha_l h_l(x)$

# Ensembles: boosting

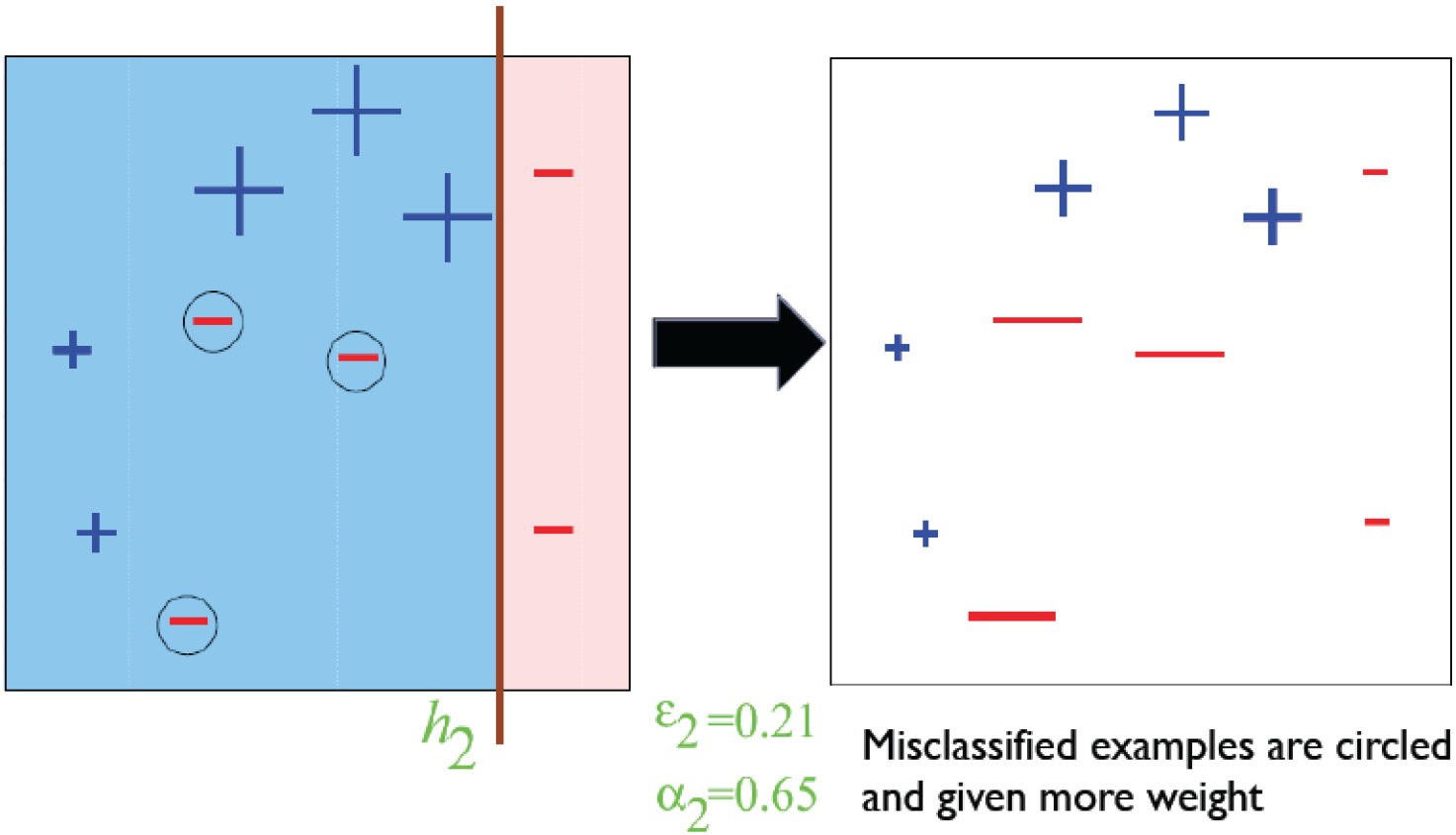
- Linear separators are the base classifiers (h(x))
- 2-dimensional plane
- 10 examples
- 3 boosting iterations



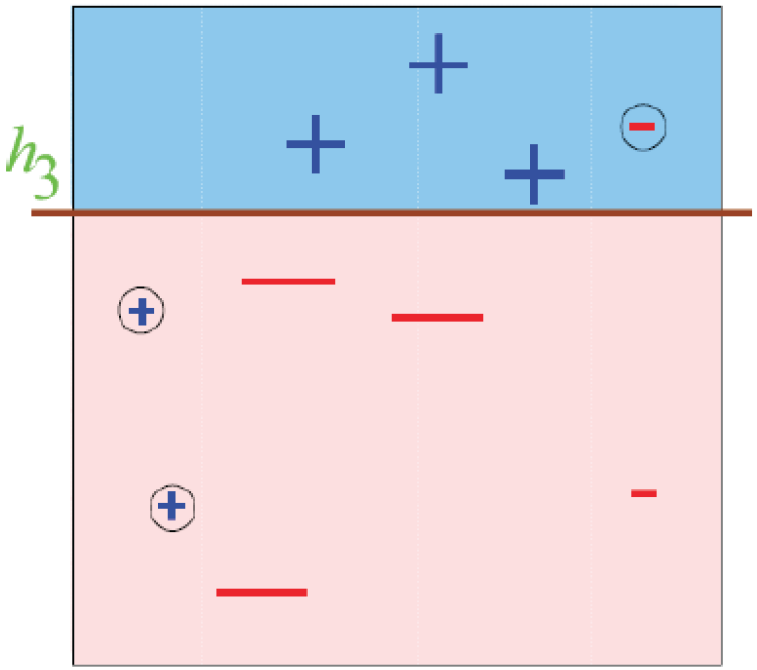
# Ensembles: boosting



# Ensembles: boosting



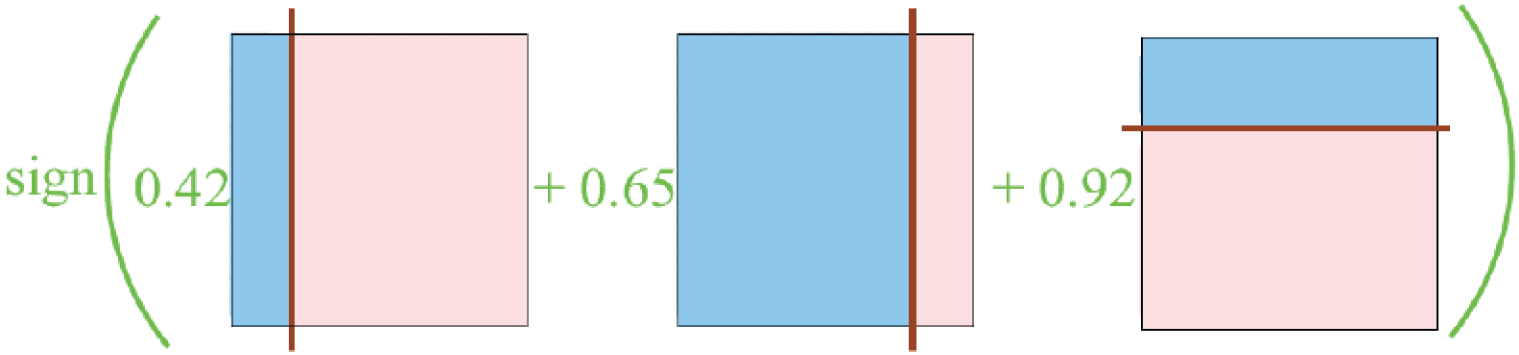
# Ensembles: boosting

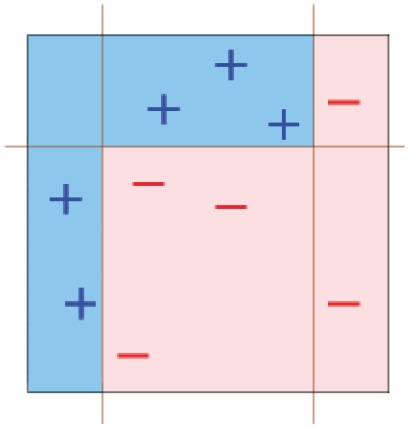



$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

# Ensembles: boosting



# Summary of nonlinear classifiers

- Similarity-based approaches: kernel regression and sparse kernel methods (support vector machines)
  - best for problems in which it is easier to specify similarity measures between examples, rather than discrete features for each example.
- Compositional models: decision trees, ensembles (random forests and Adaboost)
  - decision trees are best for problems that need interpretable models greedy, unstable
  - ensembles are opaque but are best for problems that need strong performance guarantees.

    how to explain? feature importance -- assign importance scores and sum up to come up with a feature importance plot