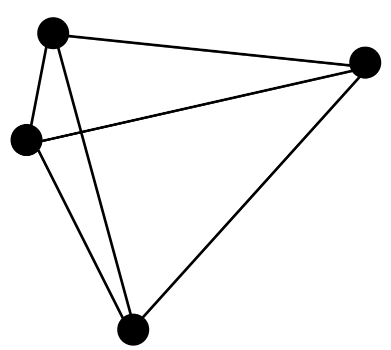
Spectral Clustering

- Consider an undirected graph that connects all data points
- Edge weights w_{ij} are the **similarities** ("closeness")
- We define the weighted **degree** d_i of a node as the sum of all outgoing edges



$$W = \begin{array}{|c|c|c|c|} \hline w_{11} & w_{12} & w_{13} & w_{14} \\ \hline w_{12} & w_{22} & w_{23} & w_{24} \\ \hline w_{13} & w_{23} & w_{33} & w_{34} \\ \hline w_{14} & w_{24} & w_{34} & w_{44} \\ \hline \end{array}$$

$$d_i = \sum_{j=1}^{N} w_{ij}$$

$$D = \begin{array}{|c|c|c|c|} \hline d_1 & 0 & 0 & 0 \\ \hline 0 & d_2 & 0 & 0 \\ \hline 0 & 0 & d_3 & 0 \\ \hline 0 & 0 & 0 & d_4 \\ \hline \end{array}$$

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvalue 0

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvector 0
  - the matrix is symmetric and positive semi-definite

Spectral Clustering

- The Graph Laplacian is defined as:

$$L = D - W$$

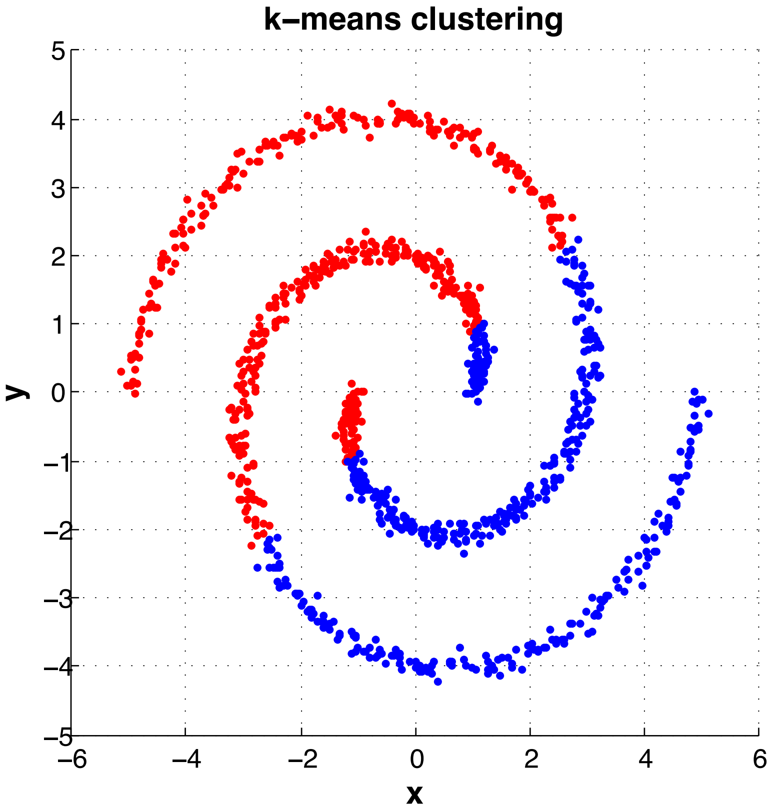
- This matrix has the following properties:
  - the 1 vector is eigenvector with eigenvector 0
  - the matrix is symmetric and positive semi-definite
- With these properties we can show:

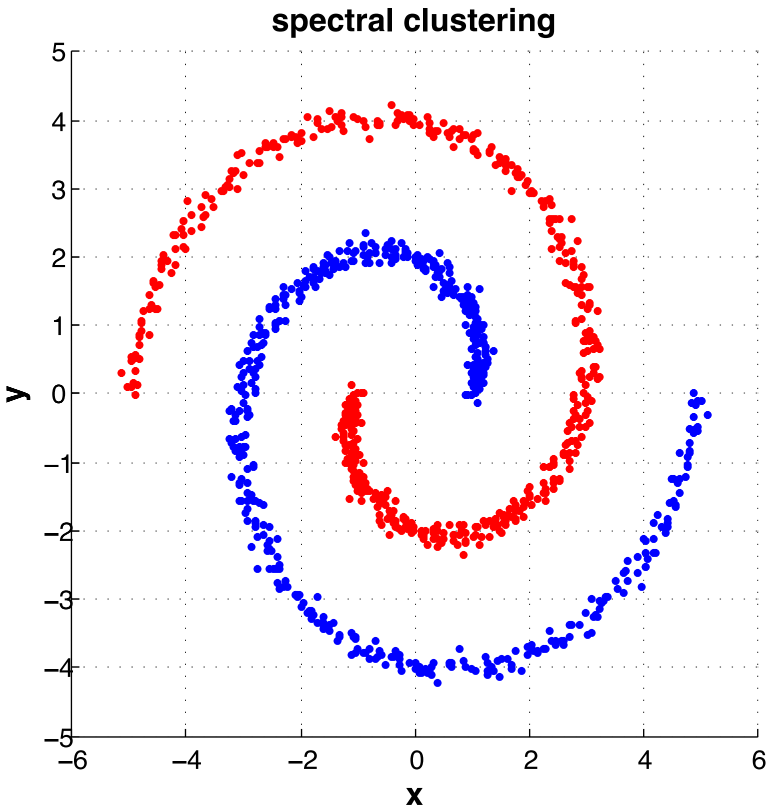
Theorem: The set of eigenvectors of L with eigenvalue 0 is spanned by the indicator vectors $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_K}$, where A_k are the K connected components of the graph.

The Spectral Clustering Algorithm

- Input: Similarity matrix W
- Compute $L = D - W$
- Compute the eigenvectors that correspond to the K smallest eigenvalues
- Stack these vectors as columns in a matrix U
- Treat each row of U as a K-dim data point
- Cluster the N rows with K-means clustering
- The indices of the rows that correspond to the resulting clusters are those of the original data points.

An Example





- Spectral clustering can handle complex problems such as this one
- The complexity of the algorithm is $O(N^3)$, because it has to solve an eigenvector problem
- But there are efficient variants of the algorithm

7. Boosting and Bagging

Repetition: Regression

We start with a set of **basis functions**

$$\phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_{M-1}(\mathbf{x})) \qquad \mathbf{x} \in \mathbb{R}^d$$

The goal is to fit a model into the data

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

To do this, we need to find an error function, e.g.:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \phi(\mathbf{x}_i) - t_i)^2$$

To find the optimal parameters, we derived E with respect to $\mathbf{w}$ and set the derivative to zero.

Some Questions

1. Can we do the same for **classification**? As a special case we consider two classes:

$$t_i \in \{-1, 1\} \quad \forall i = 1, \ldots, N$$

2. Can we use a **different** (better?) error function?
3. Can we learn the basis functions **together** with the model parameters?
4. Can we do the learning **sequentially**, i.e. one basis function after another?

Answer to all questions: Yes, using Boosting!

The Loss Function

Definition: a real-valued function $L(t, y(\mathbf{x}))$, where t is a target value and y is a model, is called a **loss function**.

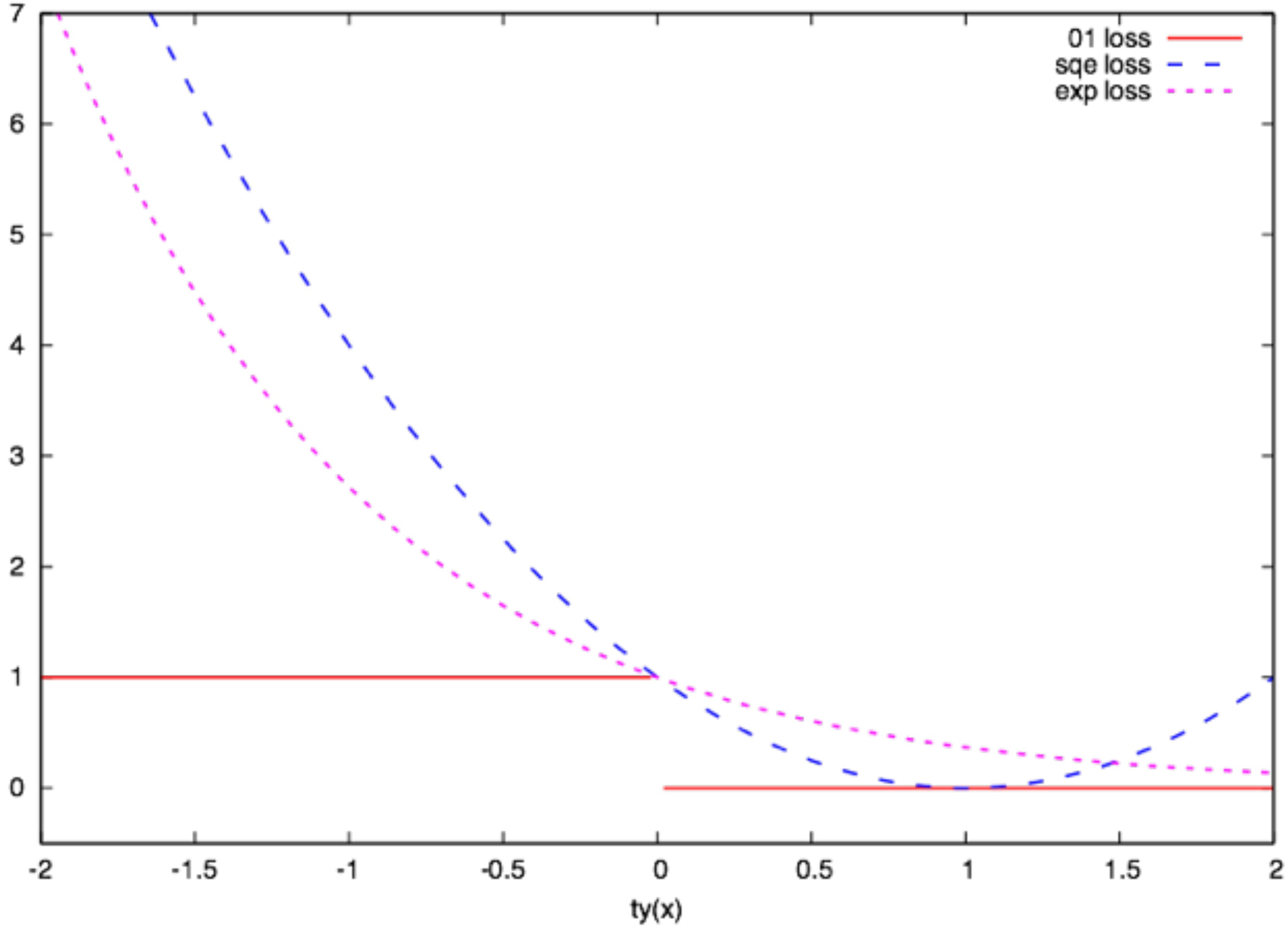
Examples:

01-loss: $$L_{01}(t, y(\mathbf{x})) = \begin{cases} 0 & \text{if } t = y(\mathbf{x}) \\ 1 & \text{else} \end{cases}$$

squared error loss: $$L_{sqe}(t, y(\mathbf{x})) = (t - y(\mathbf{x}))^2$$

exponential loss: $$L_{exp}(t, y(\mathbf{x})) = \exp(-ty(\mathbf{x}))$$

Loss Functions



- 01-loss is not differentiable
- squared error loss has only one optimum

Sequential Fitting of Basis Functions

Idea: We start with a basis function $\phi_0(\mathbf{x})$:

$$y_0(\mathbf{x}, w_0) = w_0 \phi_0(\mathbf{x}) \qquad\qquad w_0 = 1$$

Then, at iteration m, we add a new basis function $\phi_m(\mathbf{x})$ to the model:

$$y_m(\mathbf{x}, w_0, \dots, w_m) = y_{m-1}(\mathbf{x}, w_0, \dots, w_{m-1}) + w_m \phi_m(\mathbf{x})$$

Two questions need to be answered:

1. How do we find a good new basis function?
2. How can we determine a good value for w_m?

Idea: Minimize the **exponential** loss function

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^{N} L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $\quad L(t, y) = \exp(-ty)$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^{N} L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^{N} v_{i,m} \mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^{N} L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^{N} v_{i,m} \mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

$$w_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^{N} L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $$L(t, y) = \exp(-ty)$$

Solution: $$\phi_m = \arg\min_{\phi} \sum_{i=1}^{N} v_{i,m} \mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$$

$$w_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \qquad v_{i,m+1} = v_{i,m} \exp(2 w_m \mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i)))$$

Minimizing the Exponential Loss

Aim: find w_m and ϕ_m so that

$$(w_m, \phi_m) = \arg\min_{w,\phi} \sum_{i=1}^{N} L(t_i, y_{m-1}(\mathbf{x}_i) + w\phi(\mathbf{x}_i))$$

where $L(t, y) = \exp(-ty)$

Condition: Must have training error less than 0.5!

Solution: $\phi_m = \arg\min_{\phi} \sum_{i=1}^{N} v_{i,m} \mathbb{I}(t_i \neq \phi(\mathbf{x}_i))$

$$w_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \qquad v_{i,m+1} = v_{i,m} \exp(2 w_m \mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i)))$$

Factor $\exp(-w_m)$ **would be cancelled out later!**

The AdaBoost Algorithm

1. For $i = 1, \ldots, N$: $\quad v_i \leftarrow 1/N$
2. For $m = 1, \ldots, M$

   Fit a classifier ("basis function") ϕ_m that minimizes

$$\sum_{i=1}^{N} v_i \mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i))$$

$$\alpha_m := 2w_m$$

   Compute $\mathrm{err}_m = \frac{\sum_{i=1}^{N} v_i \mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^{N} v_i}$ and $\alpha_m = \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$

   Update the weights: $\quad v_i \leftarrow v_i \exp(\alpha_m \mathbb{I}(t_i \neq \phi_m(\mathbf{x}_i)))$
3. Use the resulting classifier:

$$y(\mathbf{x}) = \mathrm{sgn} \sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x})$$

The “Basis Functions”

- Can be any classifier that can deal with weighted data
- Most importantly: if these “base classifiers” provide a training error that is at most as bad as a random classifier would give (i.e. it is a **weak** classifier), then AdaBoost can return an arbitrarily small training error (i.e. AdaBoost is a strong classifier)
- Many possibilities for weak classifiers exist, e.g.:
  - Decision stumps
  - Decision trees

Decision Stumps

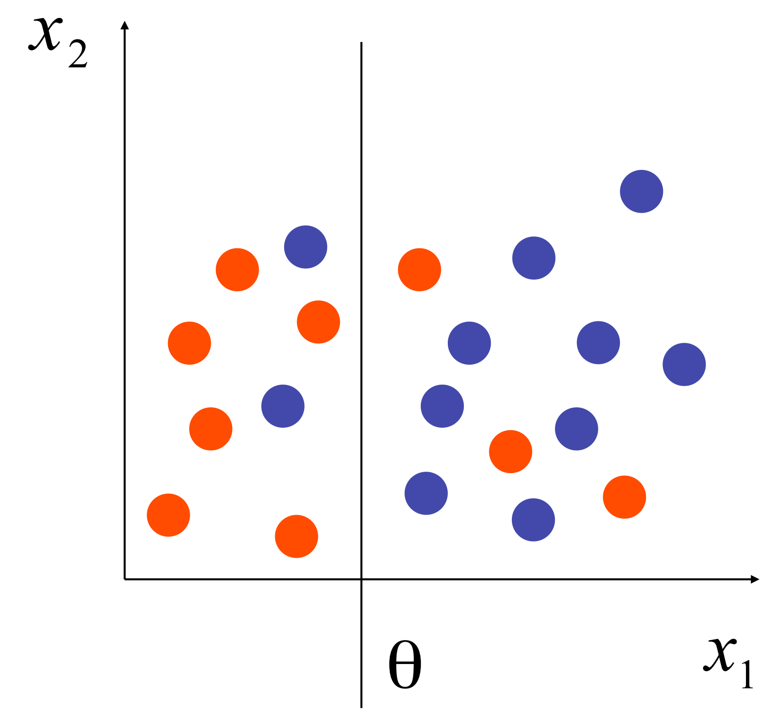
Decision Stumps are a kind of very simple weak classifiers.

Goal: Find an axis-aligned hyperplane that minimizes the class. error

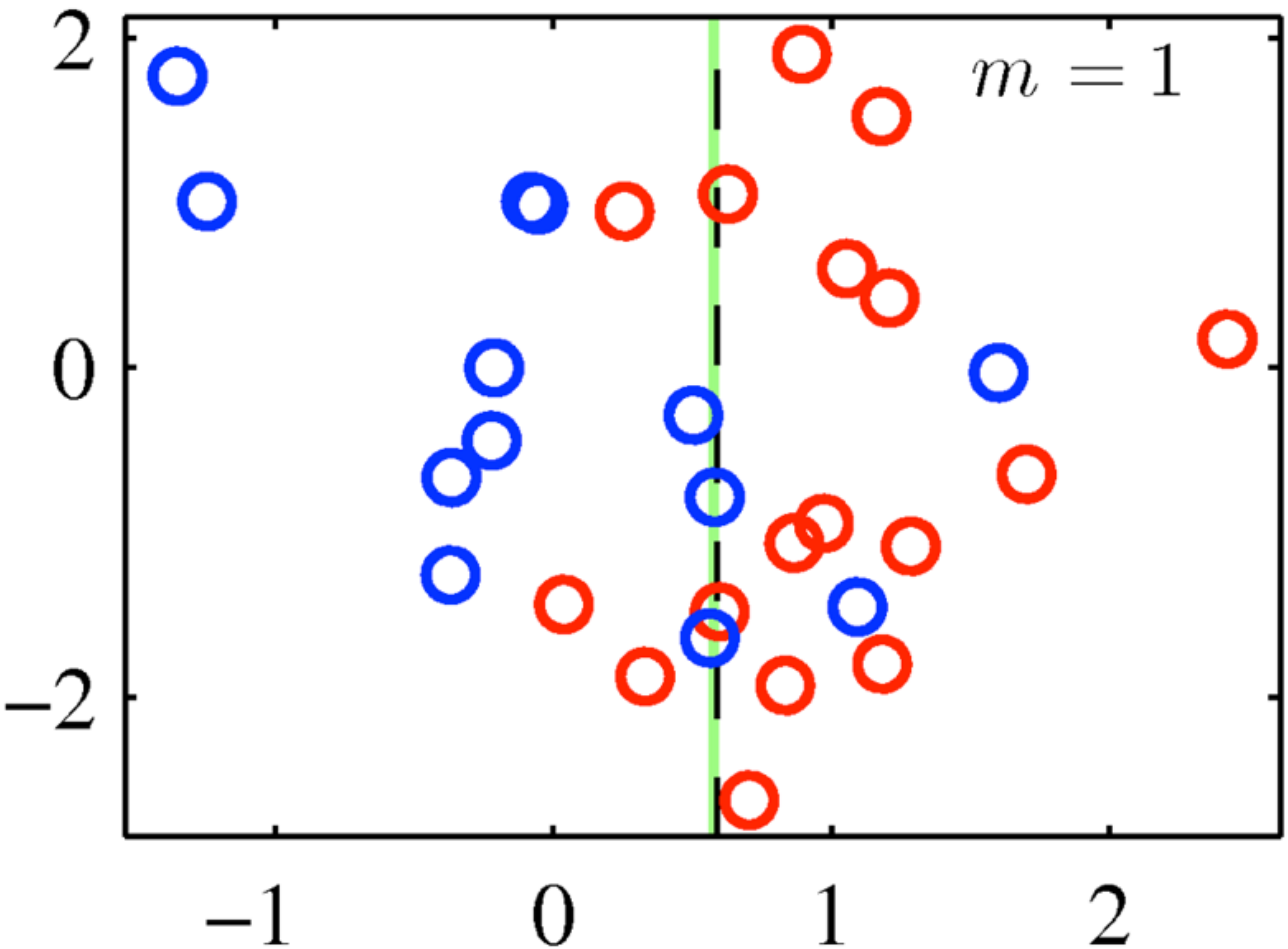
This can be done for each feature (i.e. for each dimension in feature space)

It can be shown that the classif. error is always better than 0.5 (random guessing)

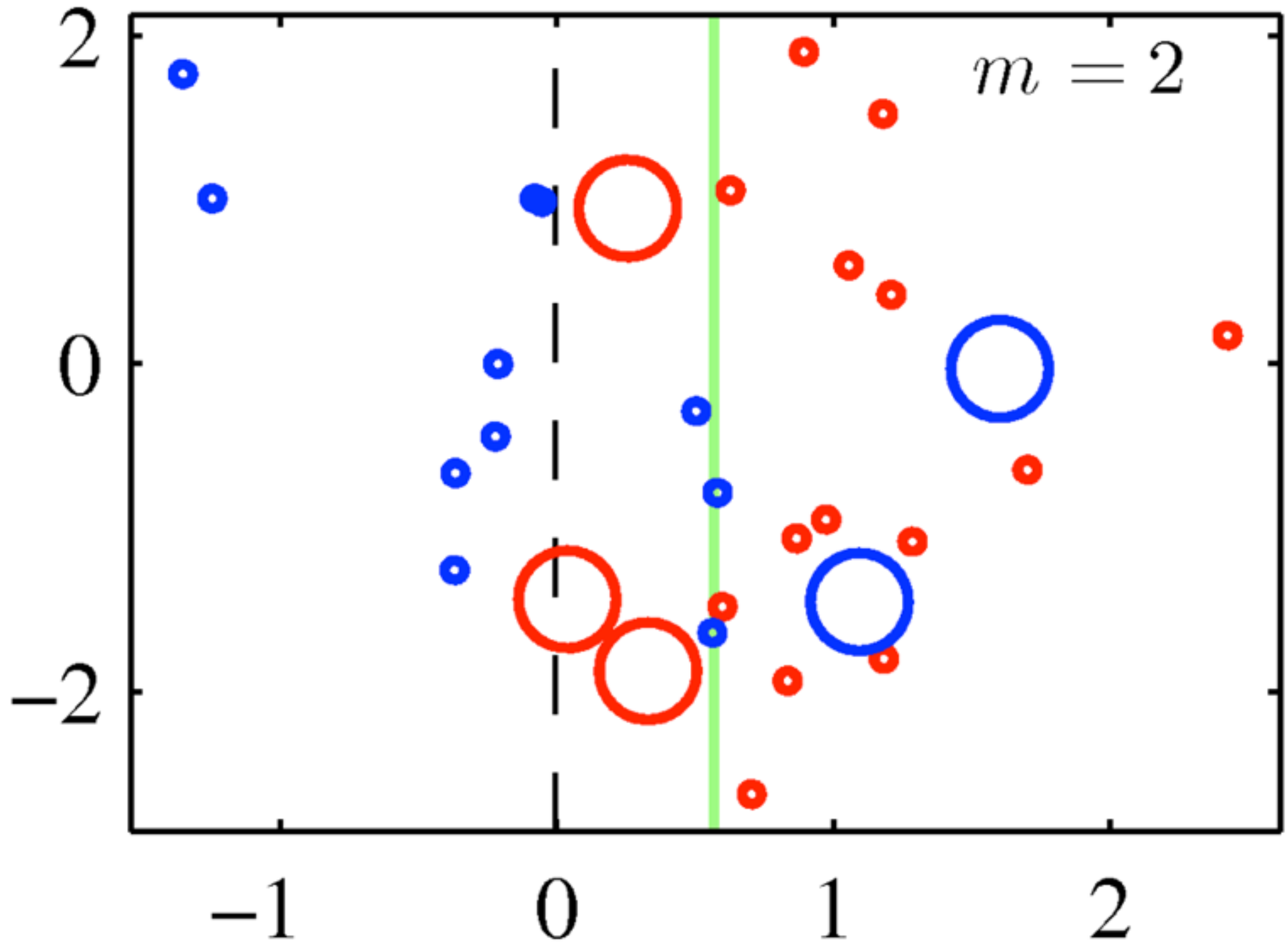
Idea: apply many weak classifiers, where each is trained on the misclassified examples of the previous.



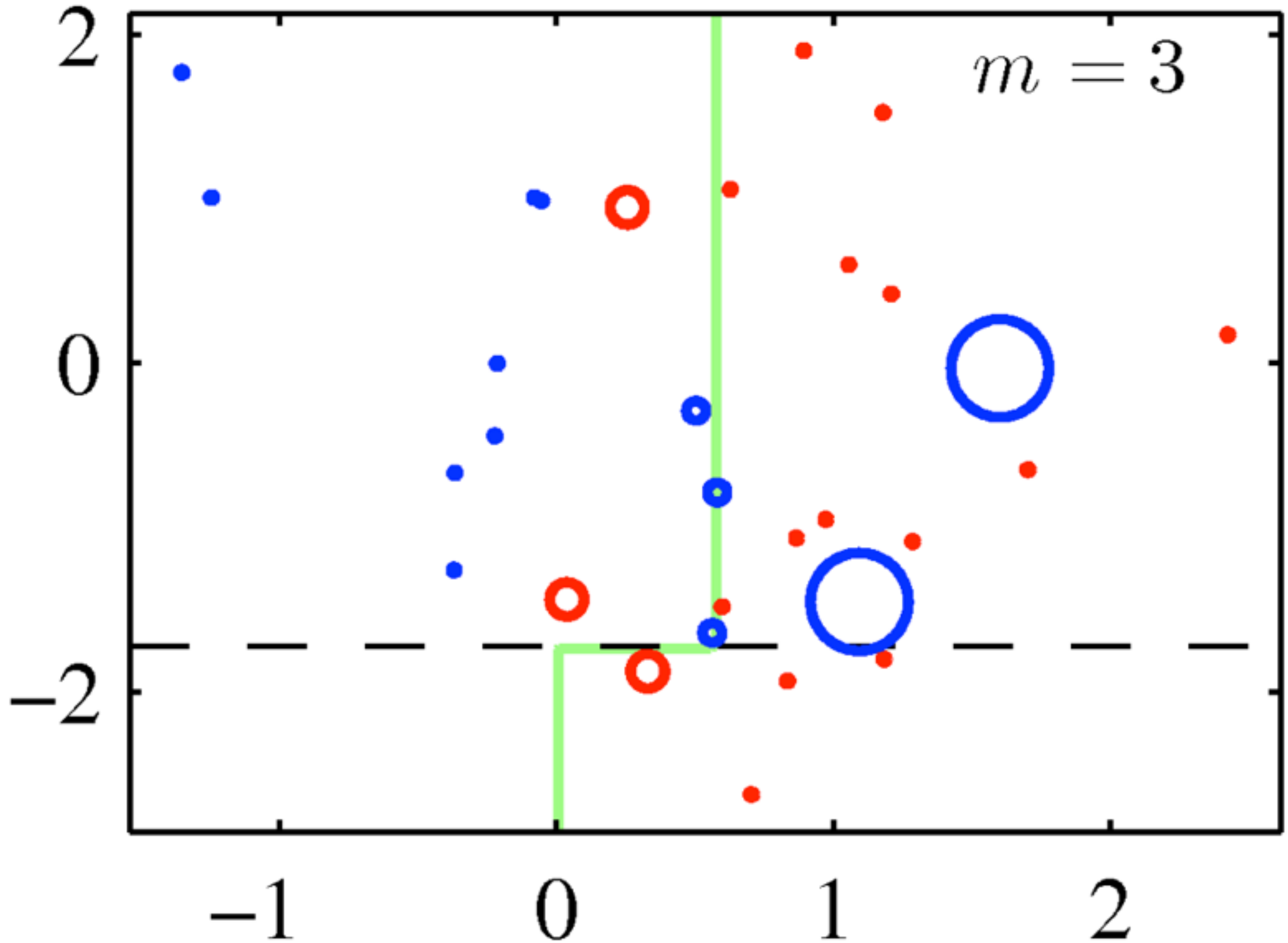
Classification Example



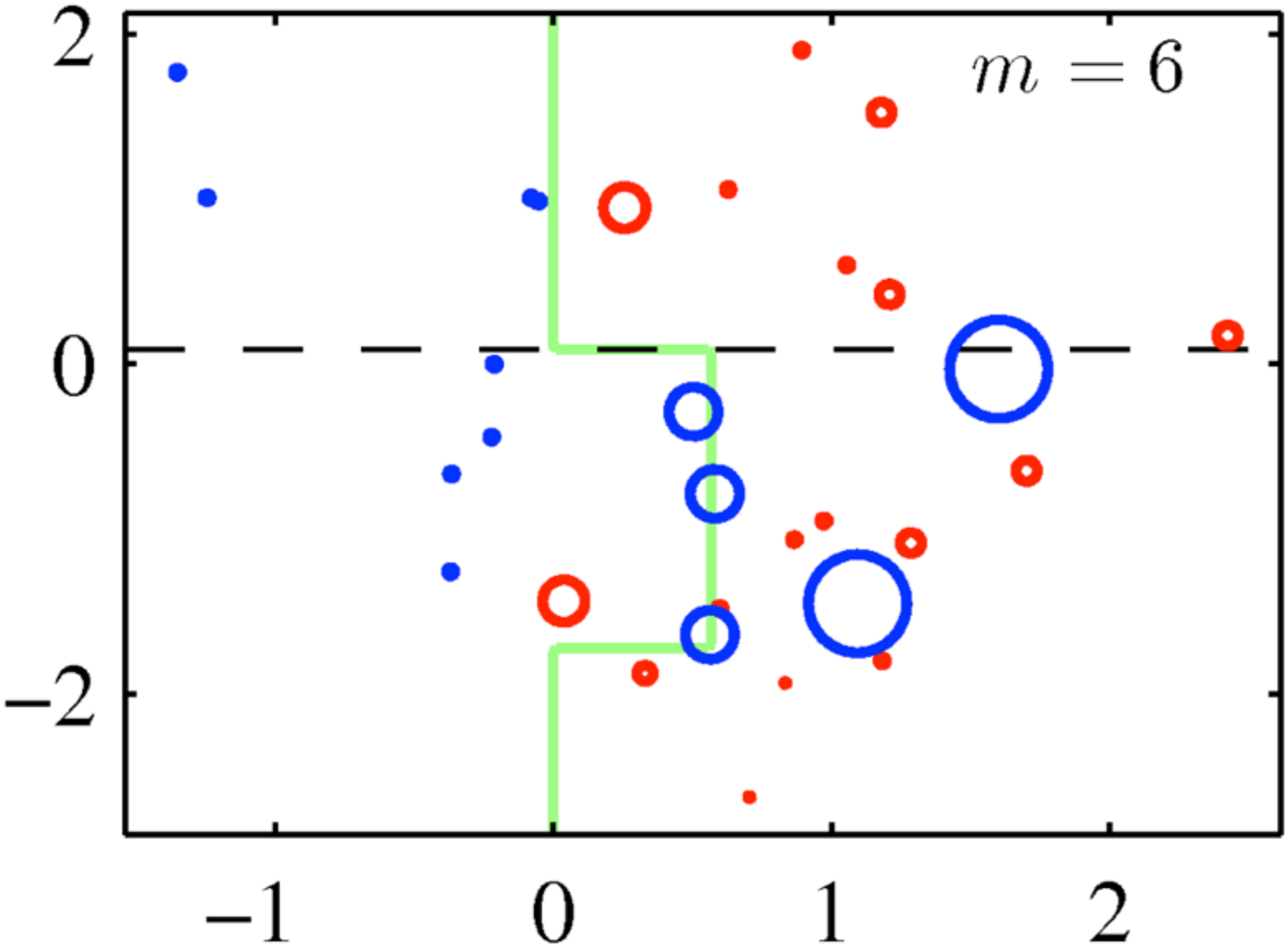
Classification Example



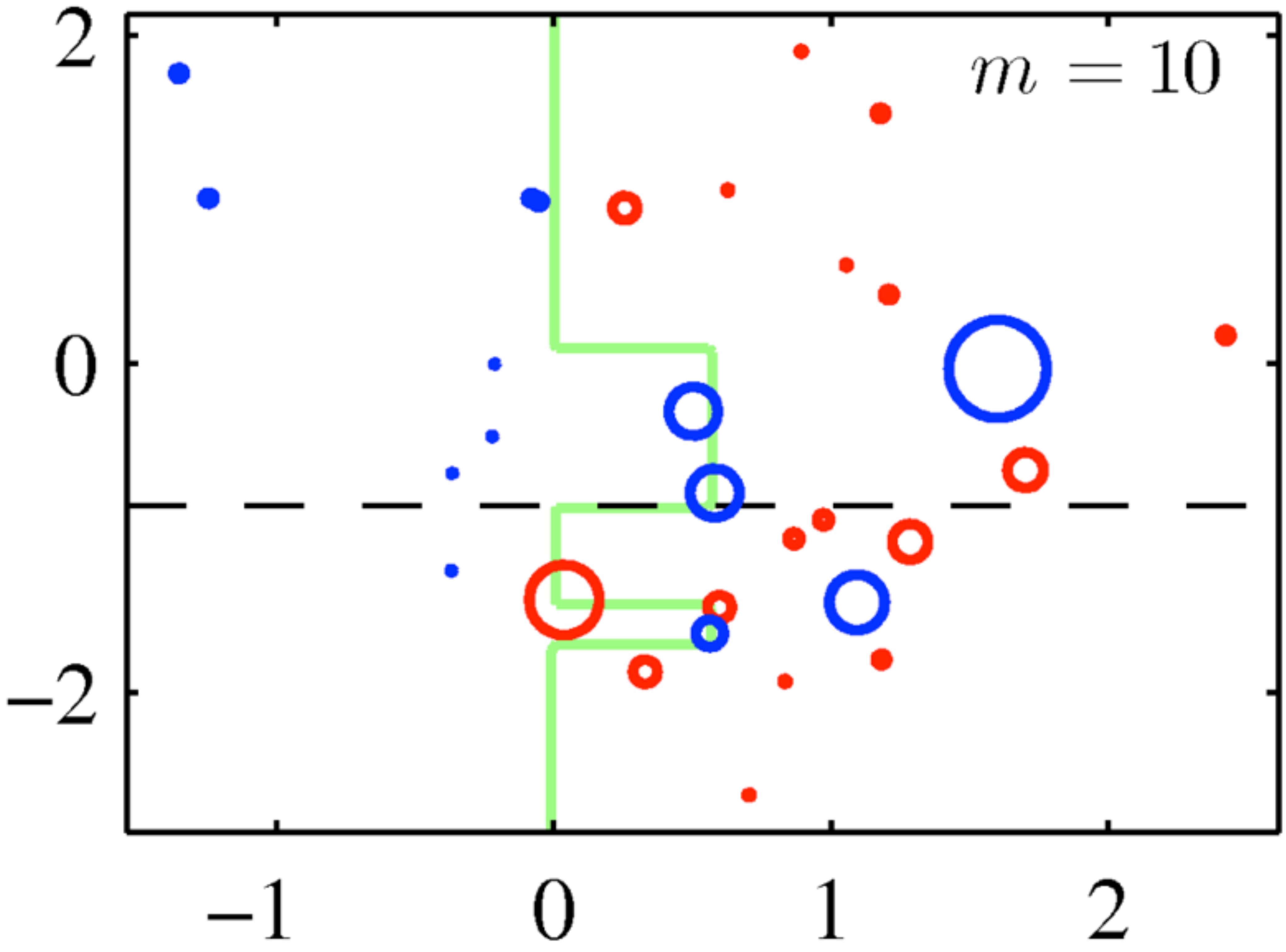
Classification Example



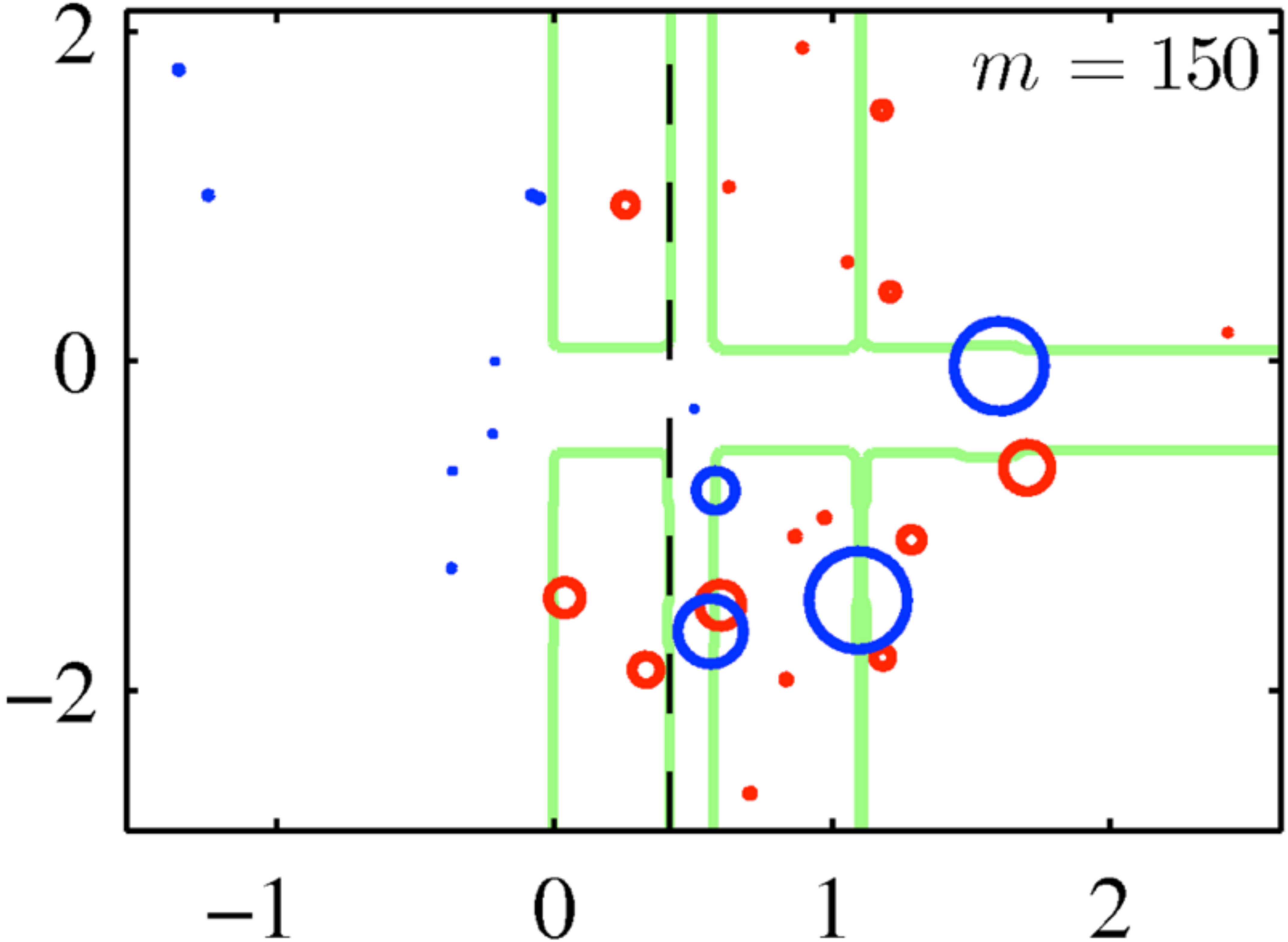
Classification Example



Classification Example

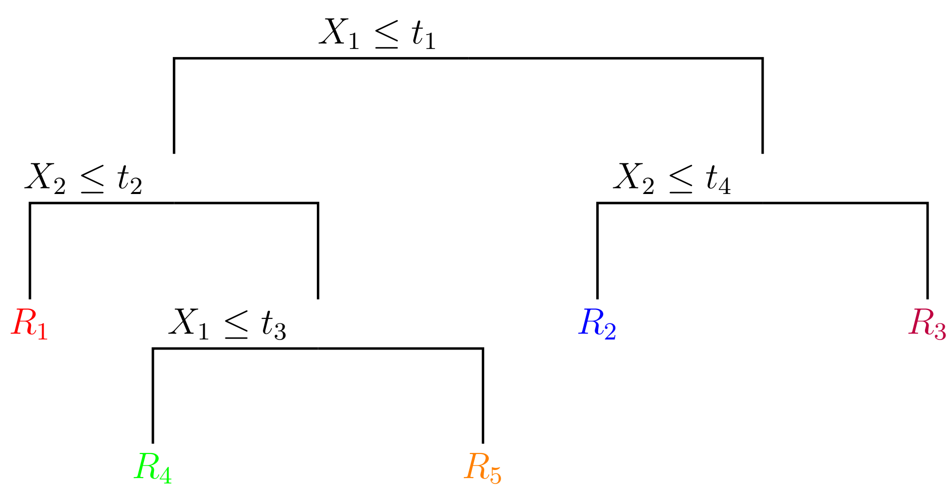


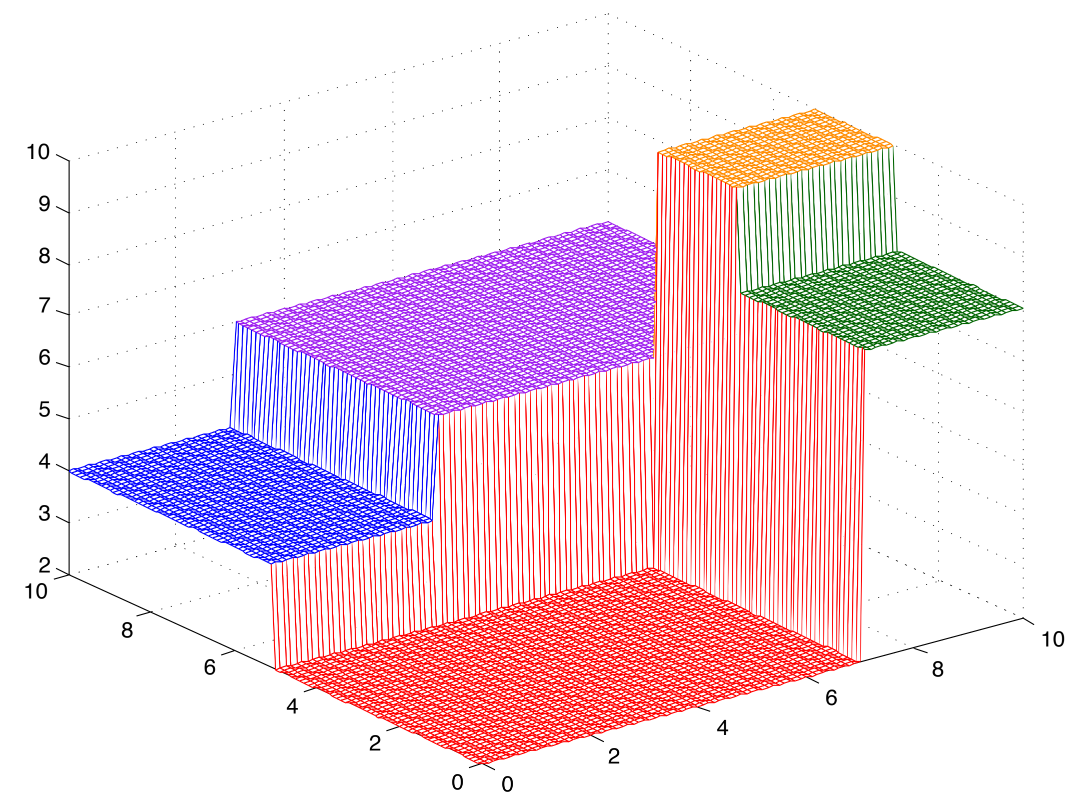
Classification Example



Decision Trees

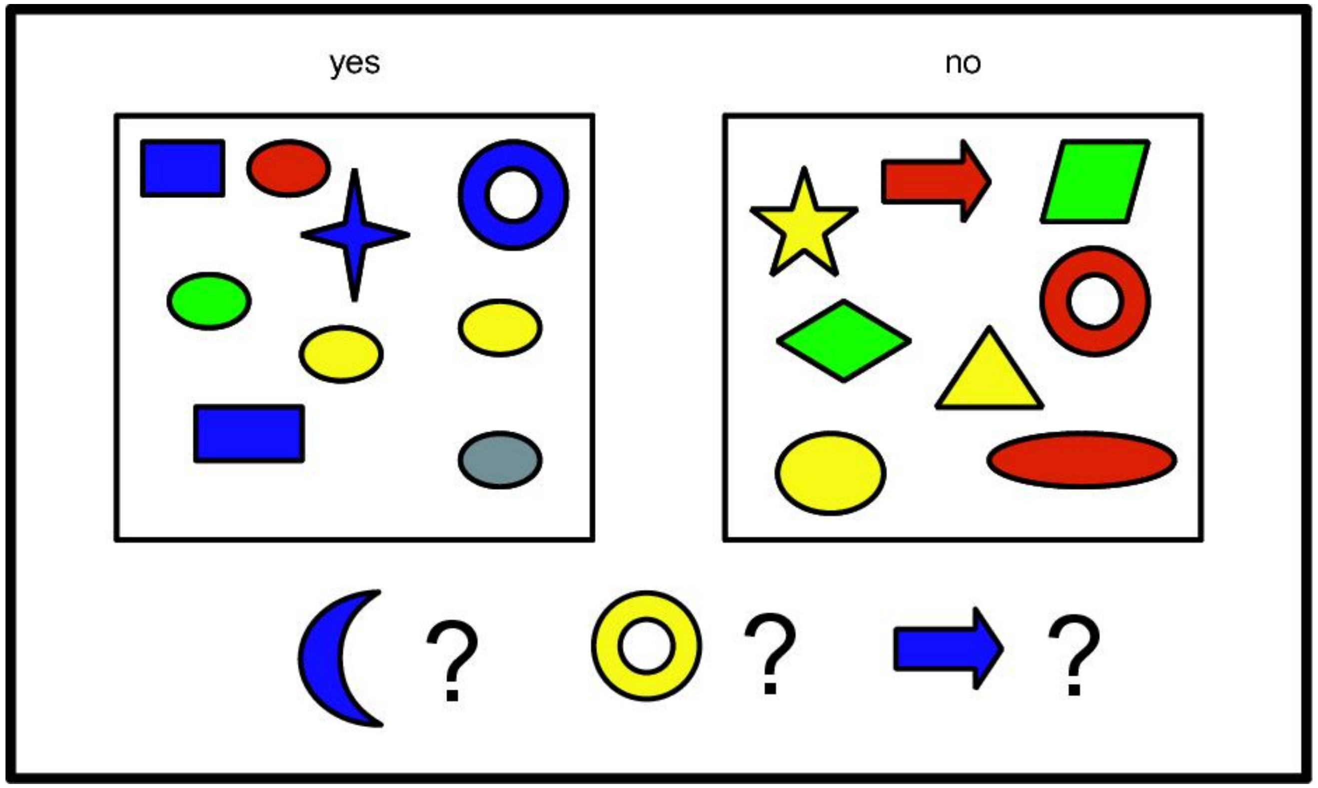
- A more general version of decision stumps are decision **trees:**

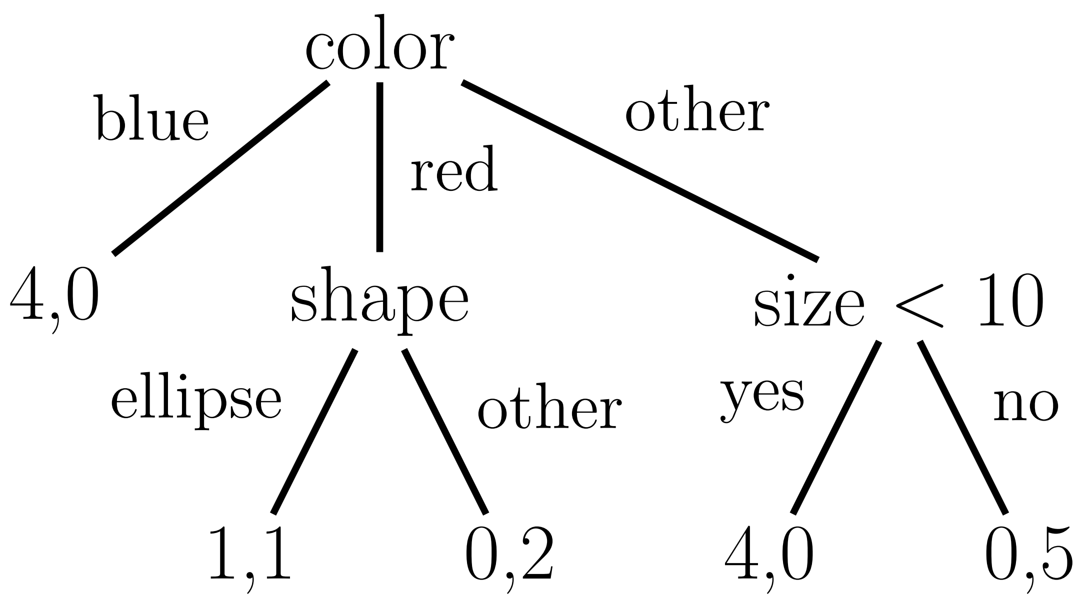




- At every node, a decision is made
- Can be used for classification and for regression (Classification And Regression Trees CART)

Decision Trees for Classification

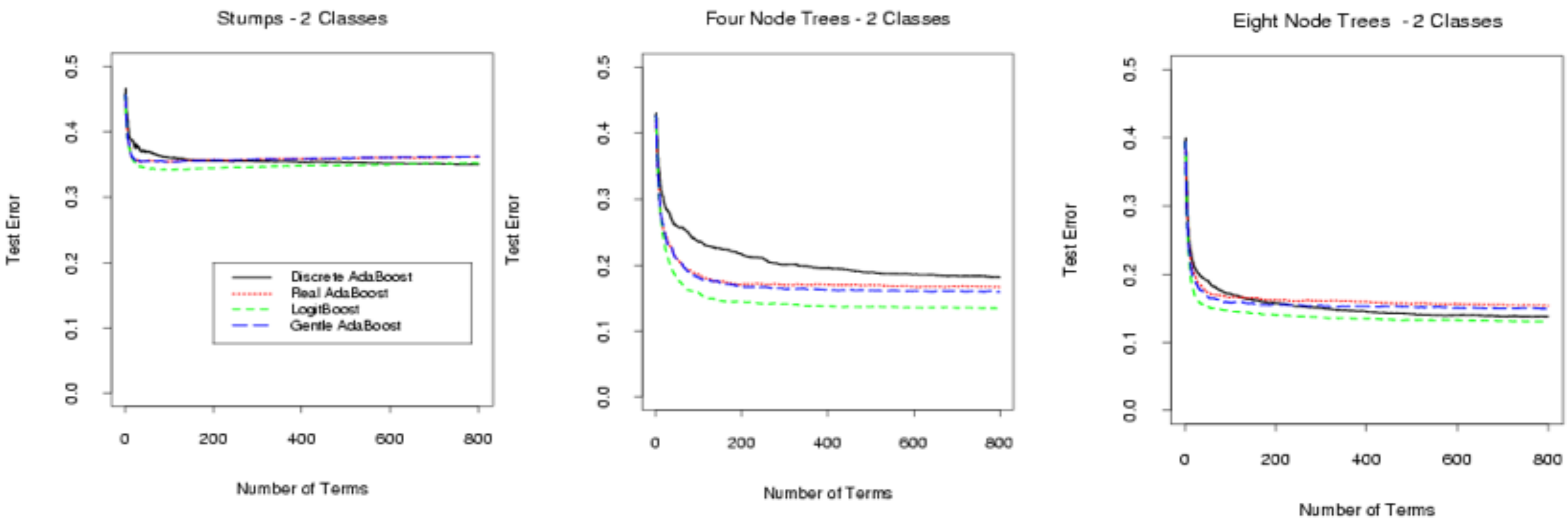




- Stores the distribution over class labels in each leaf (number of positives and negatives)
- With these, we can class label probabilities, e.g. $p(y = 1 \mid \mathbf{x}) = 1/2$ if we have a red ellipse

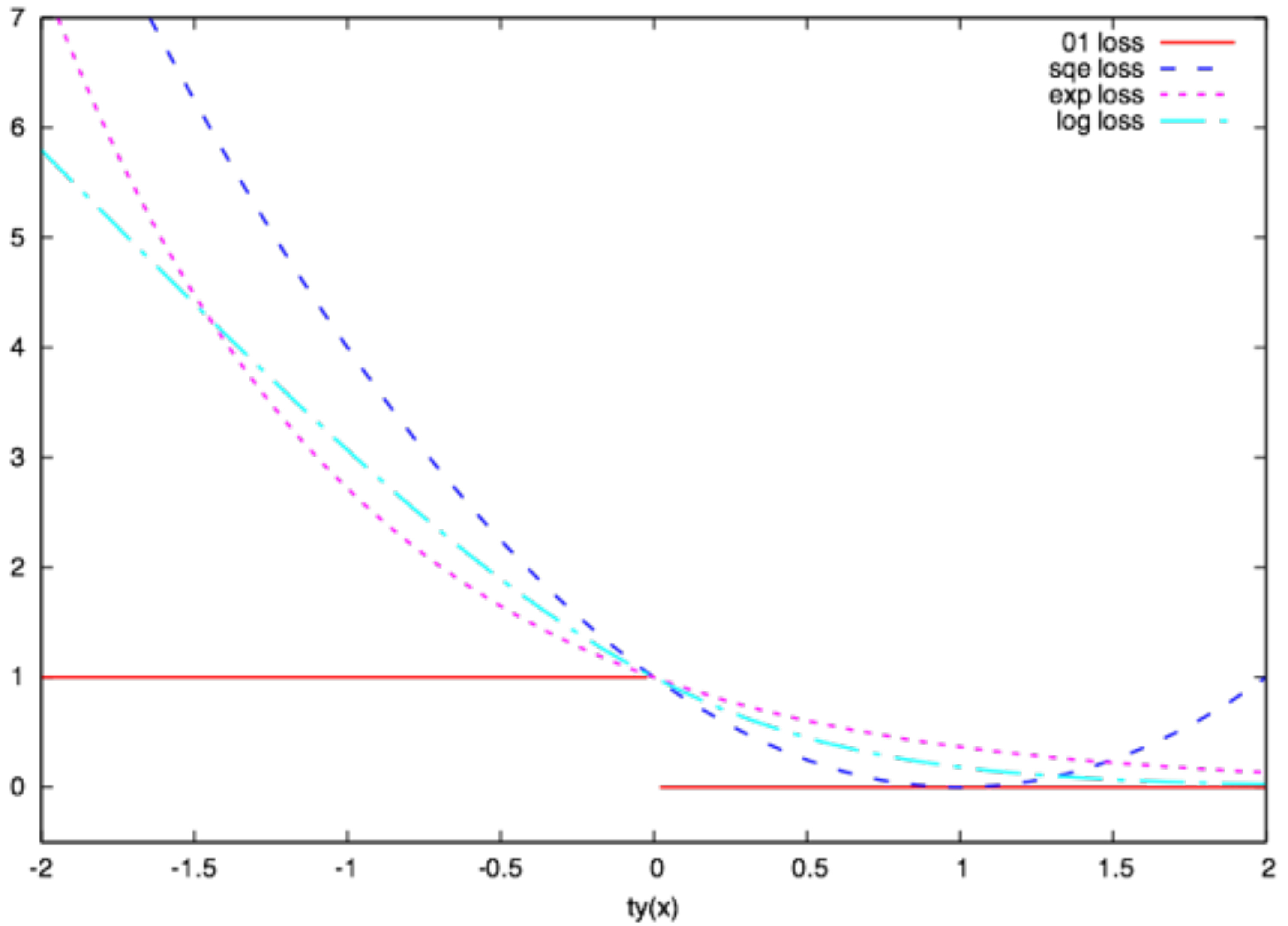
Different Weak Classifiers

- AdaBoost has been shown to perform very well, especially when using decision trees as weak classifiers



- However: the exponential loss weighs misclassified examples very high!

Using the Log-Loss



- The log-loss is defined as:

$$L(t, y(\mathbf{x})) = \log_2(1 + \exp(-2ty(\mathbf{x})))$$

- It penalizes misclassifications only **linearly**

The LogitBoost Algorithm

1. For $i = 1, \ldots, N$: $v_i \leftarrow 1/N \quad \pi_i \leftarrow 1/2$
2. For $m = 1, \ldots, M$

   Compute the working response $z_i = \dfrac{t_i - \pi_i}{\pi_i(1-\pi_i)}$

   Compute the weights $v_i = \pi_i(1-\pi_i)$

   Find ϕ_m that minimizes

$$\sum_{i=1}^{N} v_i (z_i - \phi(\mathbf{x}_i))^2$$

   Update $y(\mathbf{x}) \leftarrow y(\mathbf{x}) + \frac{1}{2}\phi_m(\mathbf{x})$ and $\pi_i \leftarrow \dfrac{1}{1+\exp(-2y(\mathbf{x}_i))}$

3. Use the resulting classifier:

$$y(\mathbf{x}) = \operatorname{sgn} \sum_{m=1}^{M} \phi_m(\mathbf{x})$$

Weighted Least-Squares Regression

- Instead of a weak classifier, LogitBoost uses "weighted least-squares regression"
- This is very similar to standard least-squares regression:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} v_i(\mathbf{w}^T \quad \mathbf{x}_i \quad - t_i)^2$$

- This results in a matrix $\hat{\Phi} = V^{1/2}\Phi$ where

$$V^{1/2} = \mathrm{diag}(\sqrt{v_1}, \ldots, \sqrt{v_N})$$

- The solution is

$$\mathbf{w} = (\hat{\Phi}^T\hat{\Phi})^{-1}\hat{\Phi}^T\mathbf{t}$$

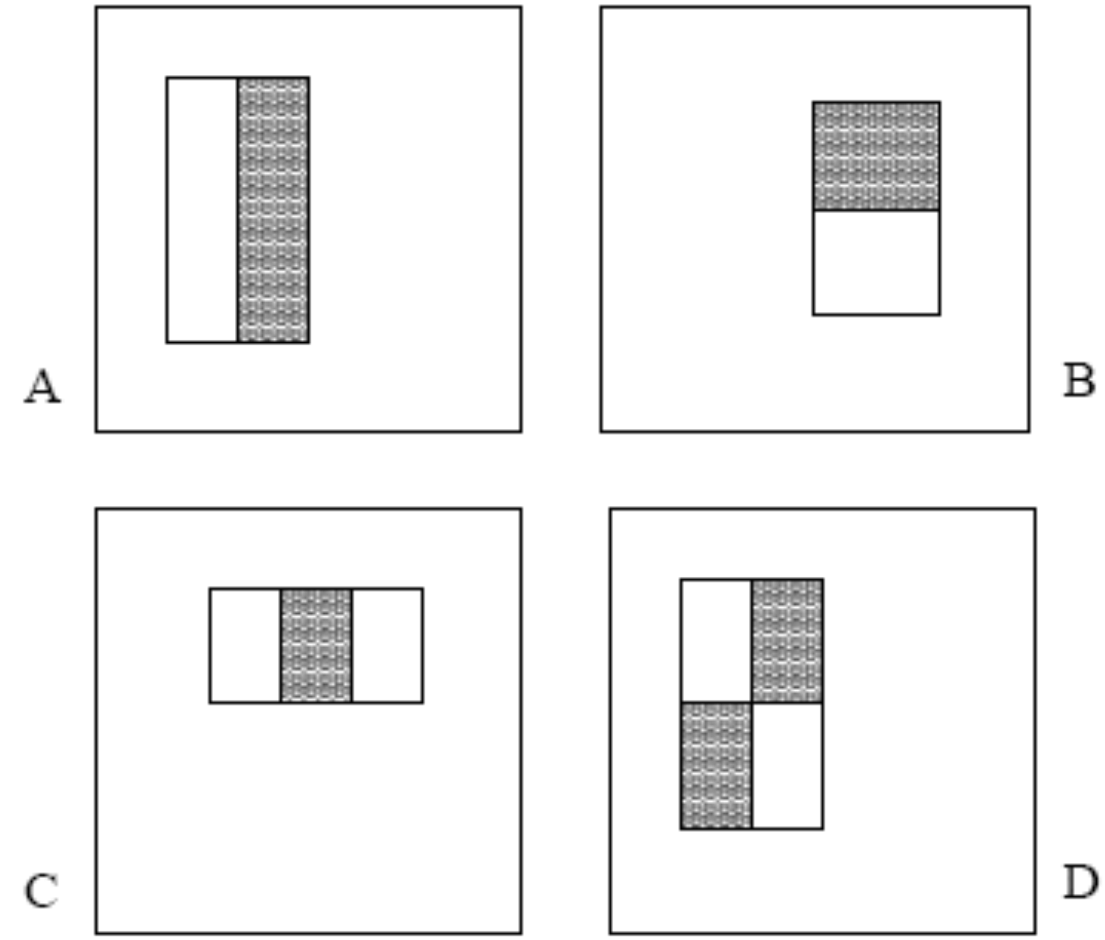
Application of AdaBoost: Face Detection

- The biggest impact of AdaBoost was made in face detection
- Idea: extract features ("Haar-like features") and train AdaBoost, use a cascade of classifiers
- Features can be computed very efficiently
- Weak classifiers can be decision stumps or decision trees
- As inference in AdaBoost is fast, the face detector can run in **real-time**!

Haar-like Features

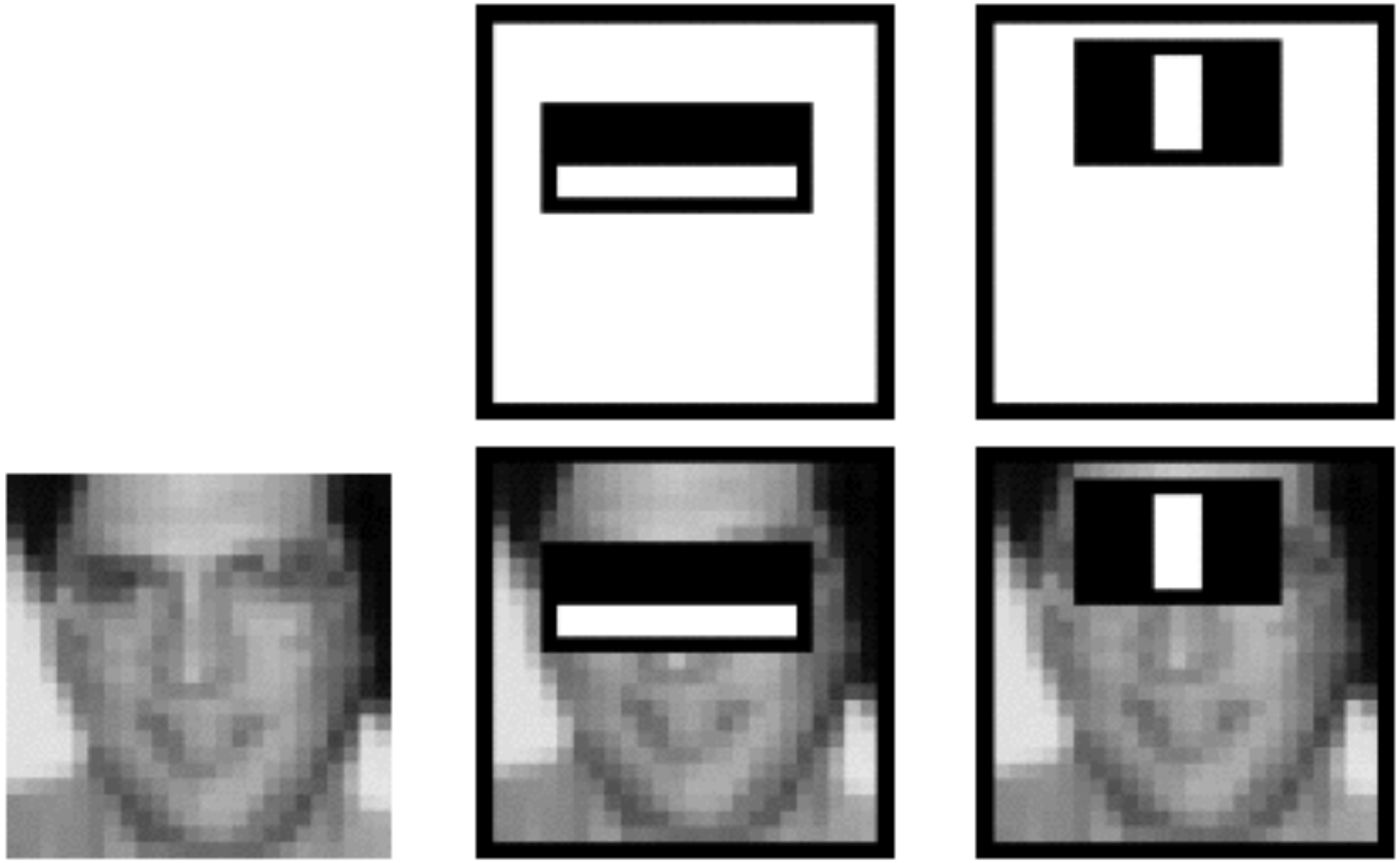
- Defined as difference of rectangular integral area:
  - The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles.

$$\left(\iint_{White} I(x,y)\,dxdy\right) - \left(\iint_{Grey} I(x,y)\,dxdy\right)$$



- One feature defined as:
  - Feature type: A,B,C or D
  - Feature position and size

Two First Classifiers Selected by AdaBoost



A classifier with only this two features can be trained to recognise 100% of the faces, with 40% of false positives

Results