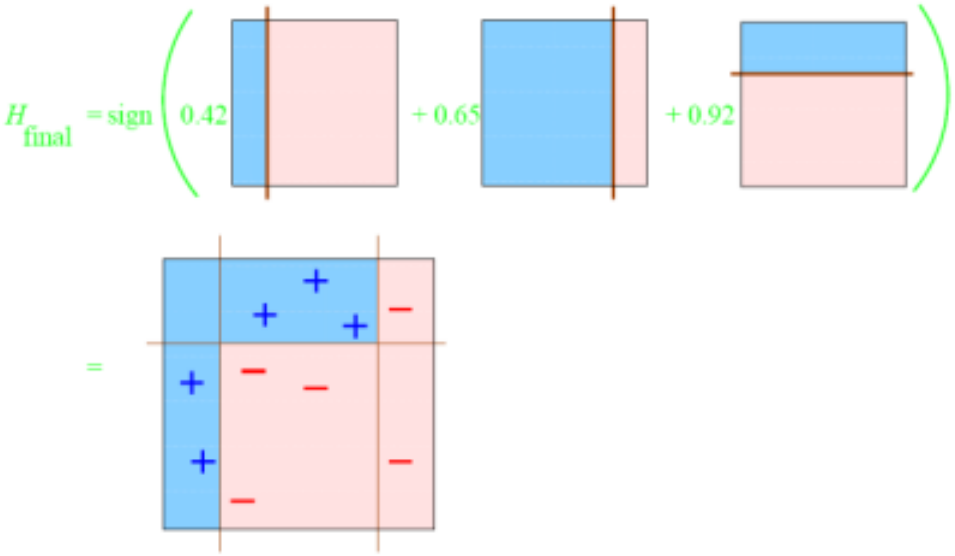
Introduction to Machine Learning



Week 4: Ensembles: Adaboost & Random Forests

Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Lecture outline

Ensemble Methods

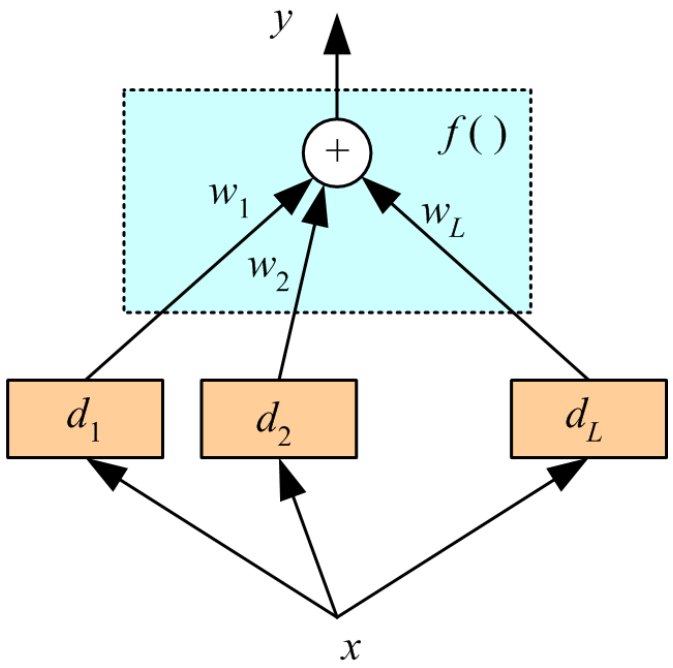
Adaboost

Decision Trees

Random Forests

Ensemble Methods

- Give up idea of building `the' classifier
- Generate a group of base-learners which has higher accuracy when combined
- Main tasks
  - Generating the learners
  - Combining them



Why should this work?

- Committee of M predictors for target output $$y_{COM}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$
- Output: true value + error $$y(\mathbf{x}) = h(\mathbf{x}) + \epsilon(\mathbf{x})$$
- Expected sum of squares error for m-th expert:

$$\mathbb{E}_{\mathbf{x}}[(y_m(\mathbf{x}) - h(\mathbf{x}))^2] = \mathbb{E}_{\mathbf{x}}[e_m(\mathbf{x})^2]$$

- Average error of individual members:

$$\mathbb{E}_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

- Average error of committee:

$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

Why should this work? (continued)

- Average error of committee:

$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

- **If** committee members have uncorrelated errors: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$

  then:

$$\mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\frac{1}{M^2}\sum_{m=1}^{M}\left\{\epsilon_m^2(\mathbf{x}) + \sum_{k\neq m}\epsilon_m(\mathbf{x})\epsilon_k(\mathbf{x})\right\}\right]$$

$$= \frac{1}{M}\left[\frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}\left[\epsilon_m^2(\mathbf{x})\right]\right]$$

In conclusion: $\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$

Consider having a jury in an exam: each examiner has own criteria, but they can't all be wrong by accident.

Ensemble methods

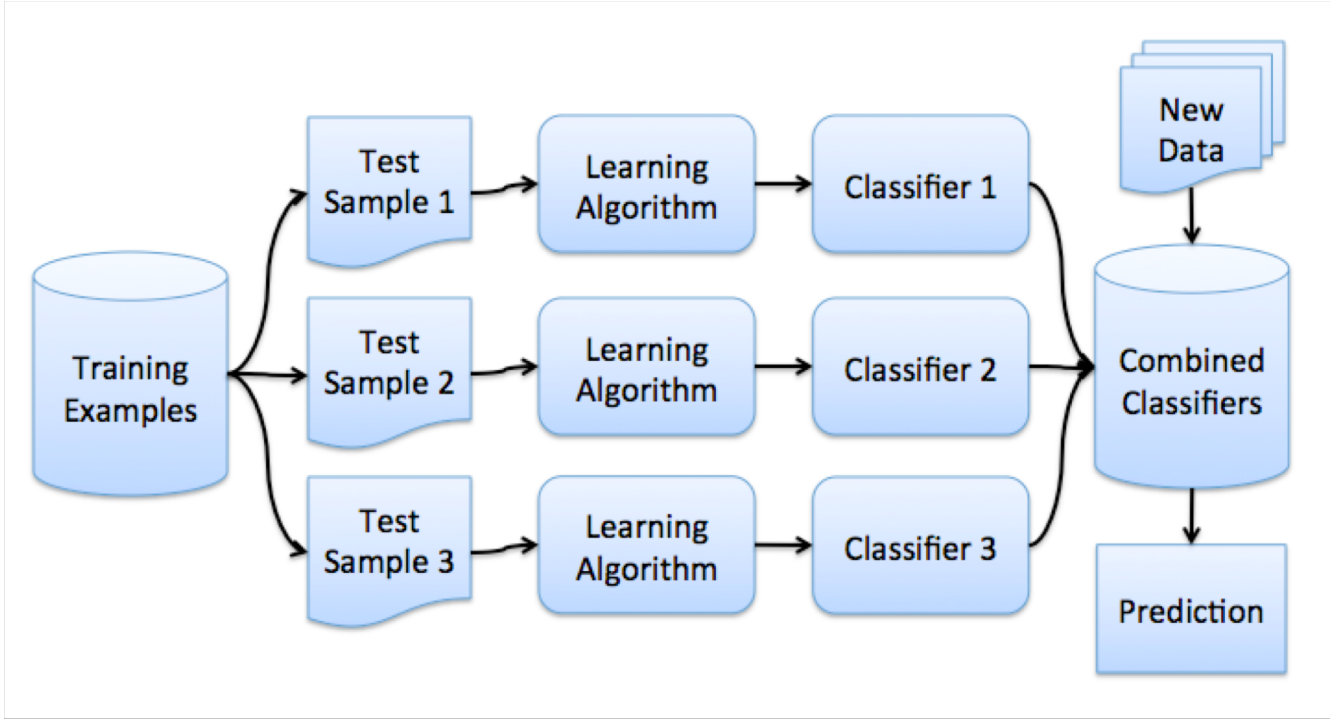
1) Train a set of simple classifiers (‘weak learners’) so that they are complementary to each other

2) Obtain an aggregate decision by combining their results

Two main methods: bagging & boosting

Bagging Idea

- General algorithm
  - Iterate
    - Pick subset of training data
    - Obtain weak learner
      - Easy to train and evaluate
  - Output final classifier by majority voting of the weak learners



We have done this before already…

- Cross-Validation
  - Split the available data into N disjunct subsets.
  - In each run, train on N-1 subsets for training a classifier.
  - Estimate the generalization error on the held-out validation set

- E.g. 5-fold cross-validation

train	train	train	train	test
train	train	train	test	train
train	train	test	train	train
train	test	train	train	train
test	train	train	train	train

Boosting Idea

- General algorithm
  - Iterate
    - Pick subset of training data using a sampling distribution
    - Obtain weak learner
      - Use weak learner to update sampling distribution

Adaboost

- Adaptive Boosting
  - `A decision theoretic generalization of on-line learning and an application to boosting', Freund and Schapire '95
  - Versatile (Discrete data, arbitrary distributions, multi-class, regression)
  - Very easy to program
  - Excellent results
- Defines a classifier using an additive model (weighted voting):

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

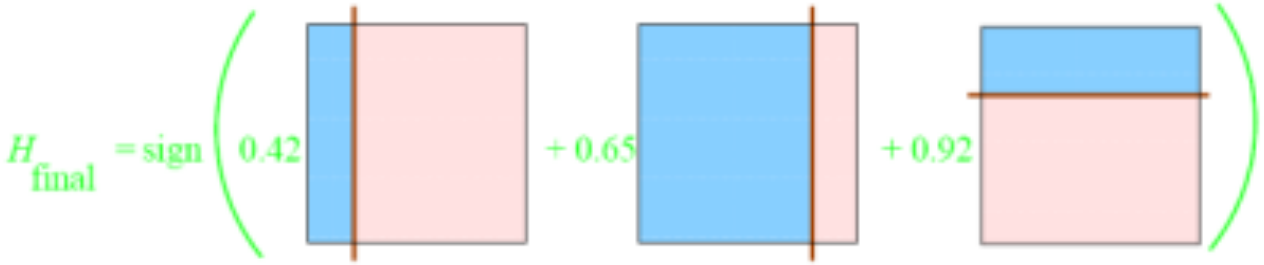
Strong classifier

Weak classifier

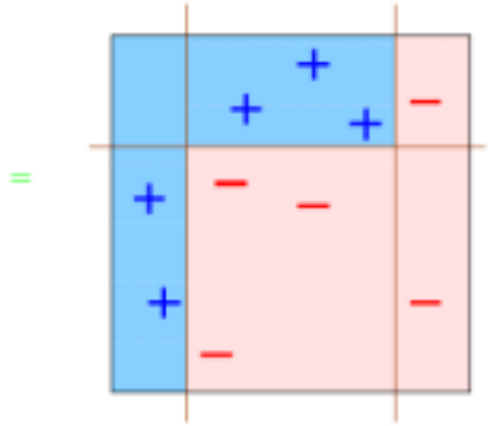
Weight

Feature vector

Adaboost, in pictures

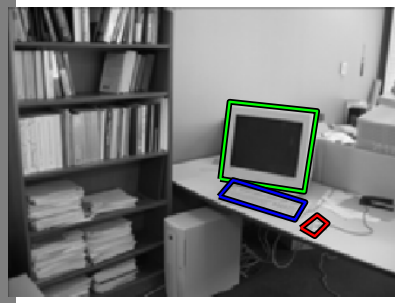


elementary functions

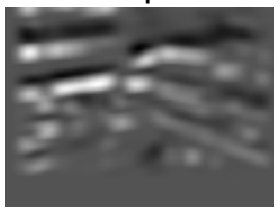


nonlinear decision boundary!

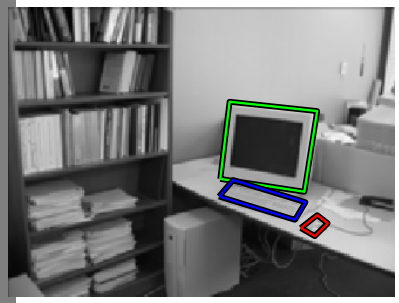
Example: screen detection



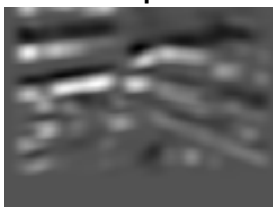
Feature output



Example: screen detection

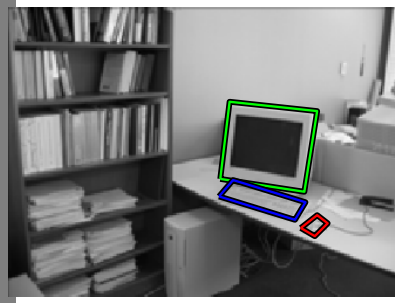


Feature output

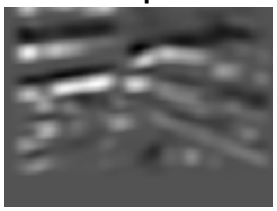


Thresholded output

Example: screen detection



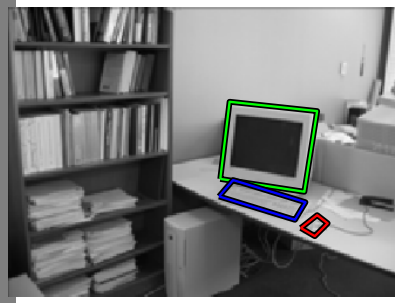
Feature output



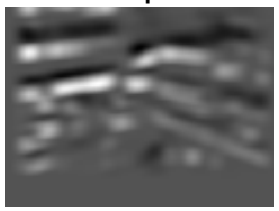
Thresholded output

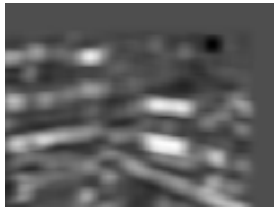
Strong classifier at iteration 1

Example: screen detection



Feature output

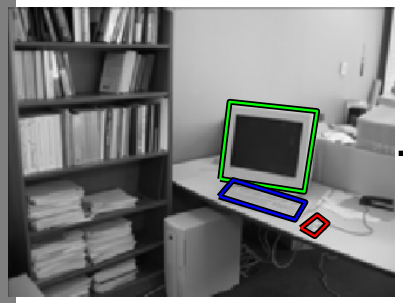


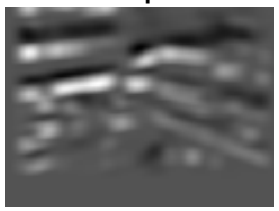


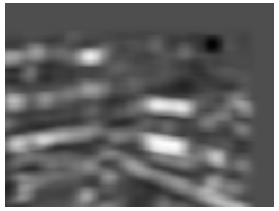
Thresholded output

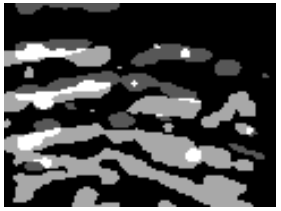
Strong classifier

Example: screen detection







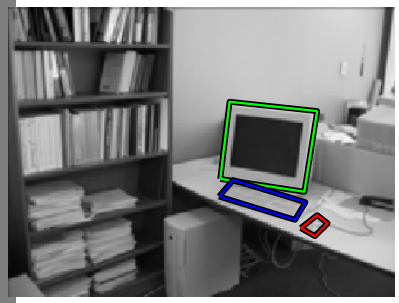


Feature output

Thresholded output

Strong classifier

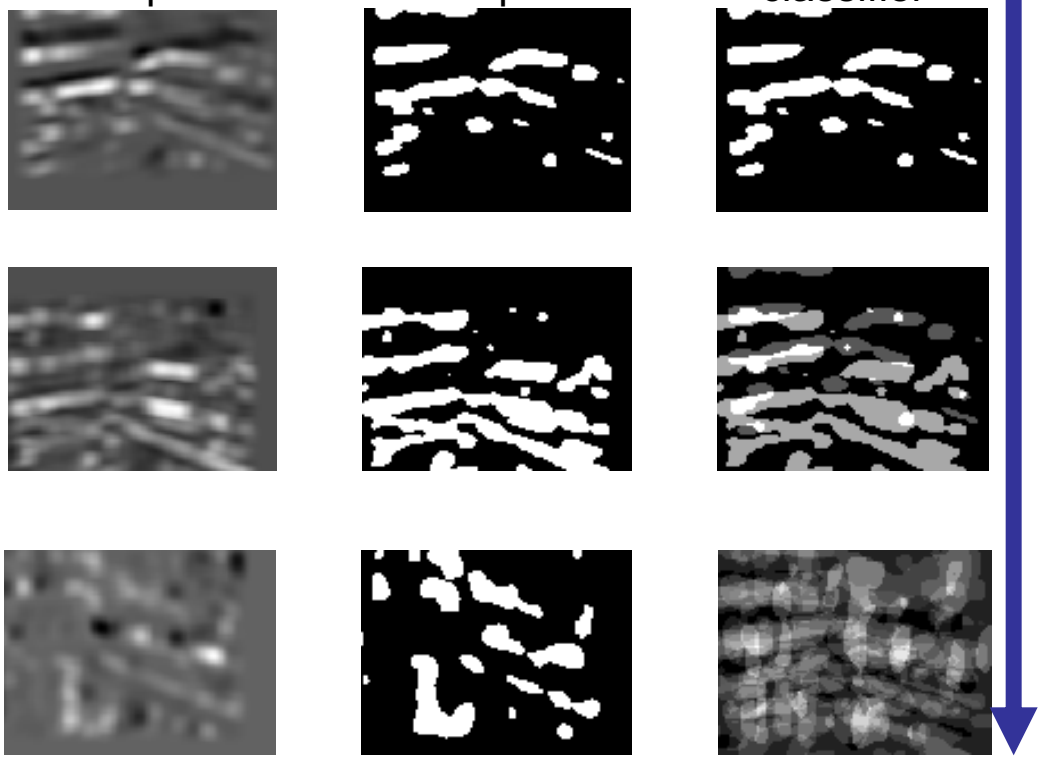
Example: screen detection



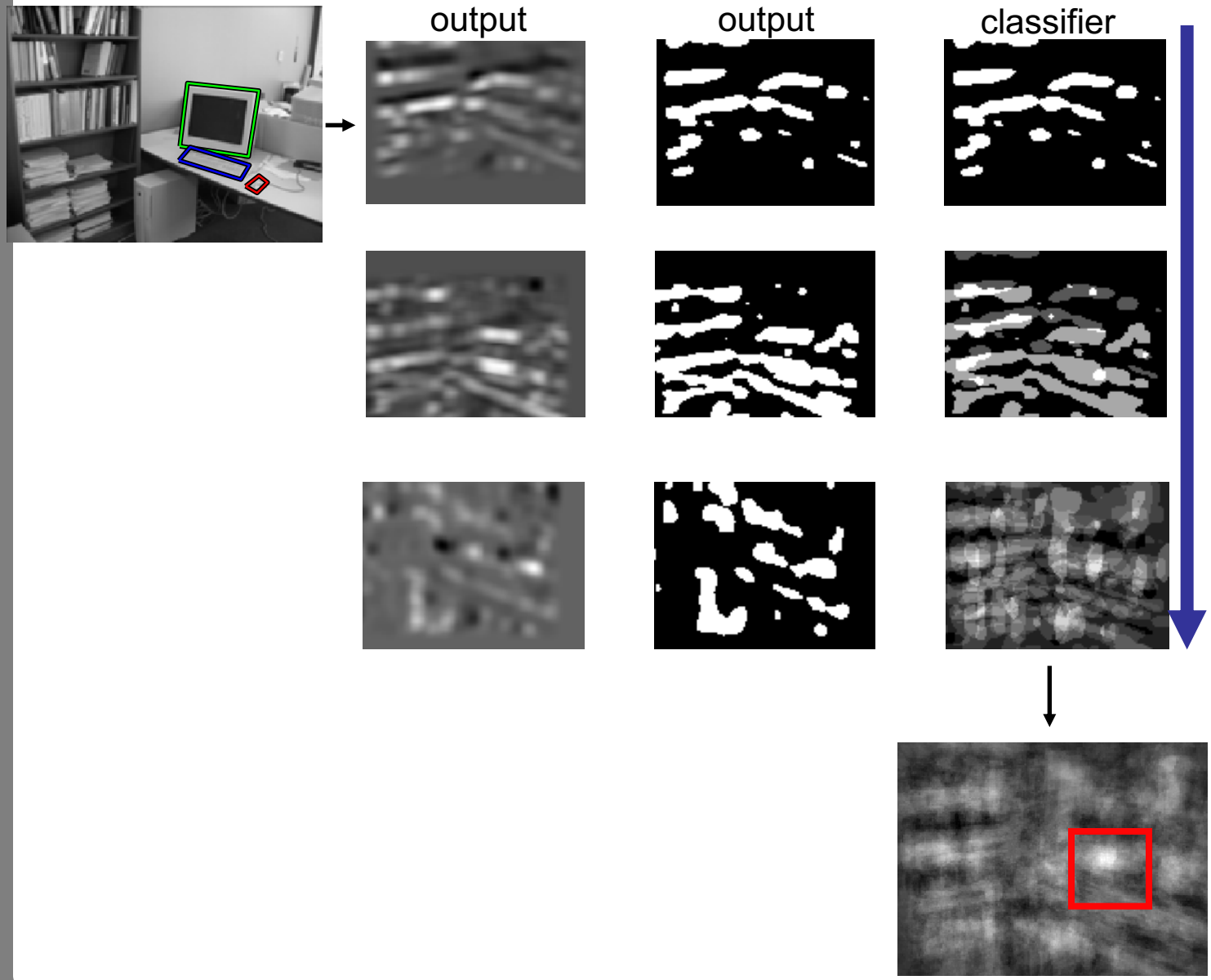
Feature output

Thresholded output

Strong classifier

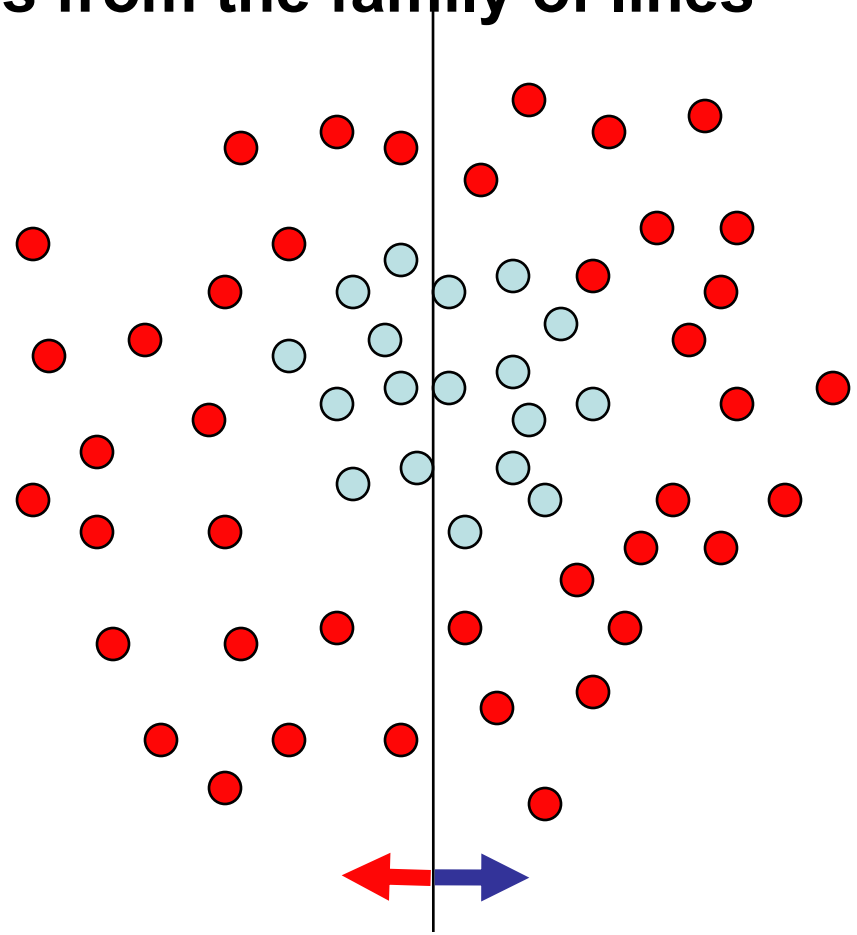


Example: screen detection



Toy example

Weak learners from the family of lines



Each data point has a class label:

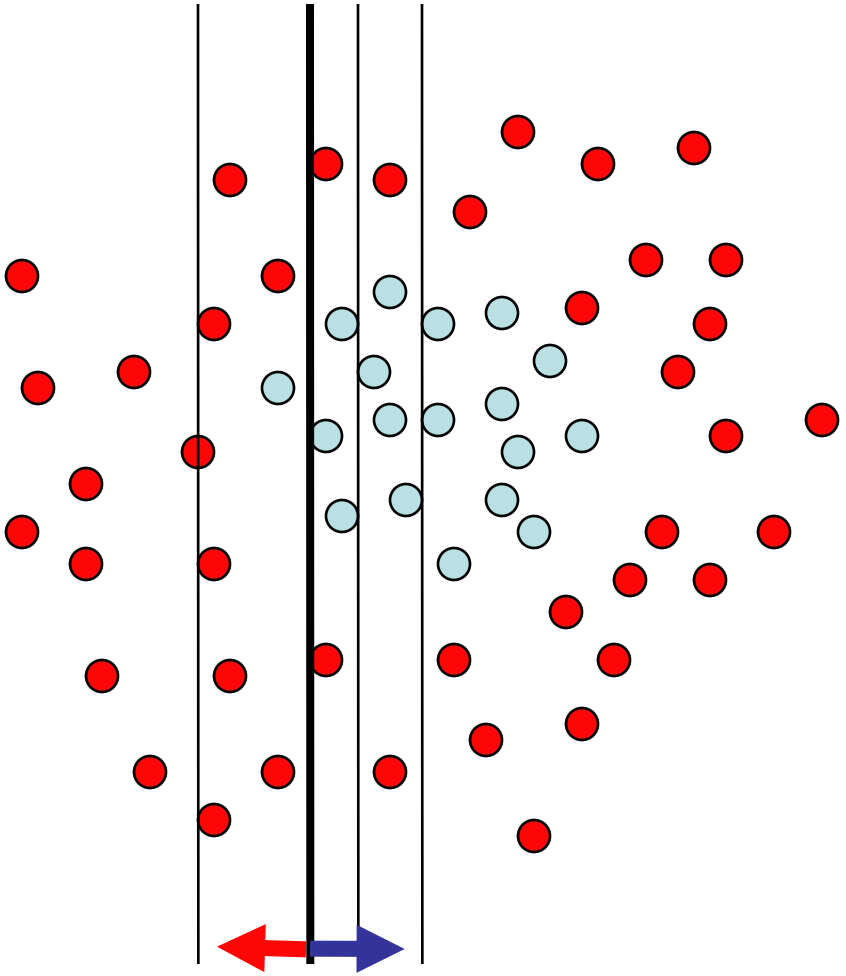
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

and a weight:

$$w_t = 1$$

h => p(error) = 0.5 it is at chance

Toy example



Each data point has a class label:

$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

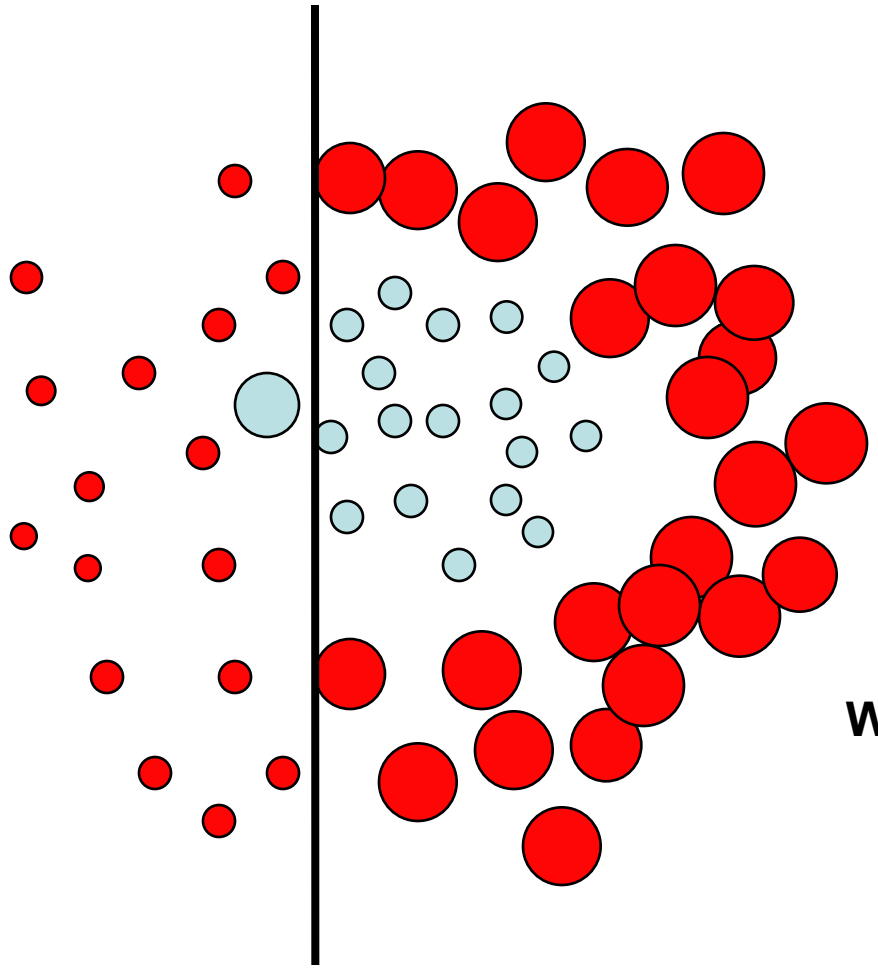
and a weight:

$$w_t = 1$$

This one seems to be the best

This is a ‘weak classifier’: It performs slightly better than chance.

Toy example



Each data point has a class label:

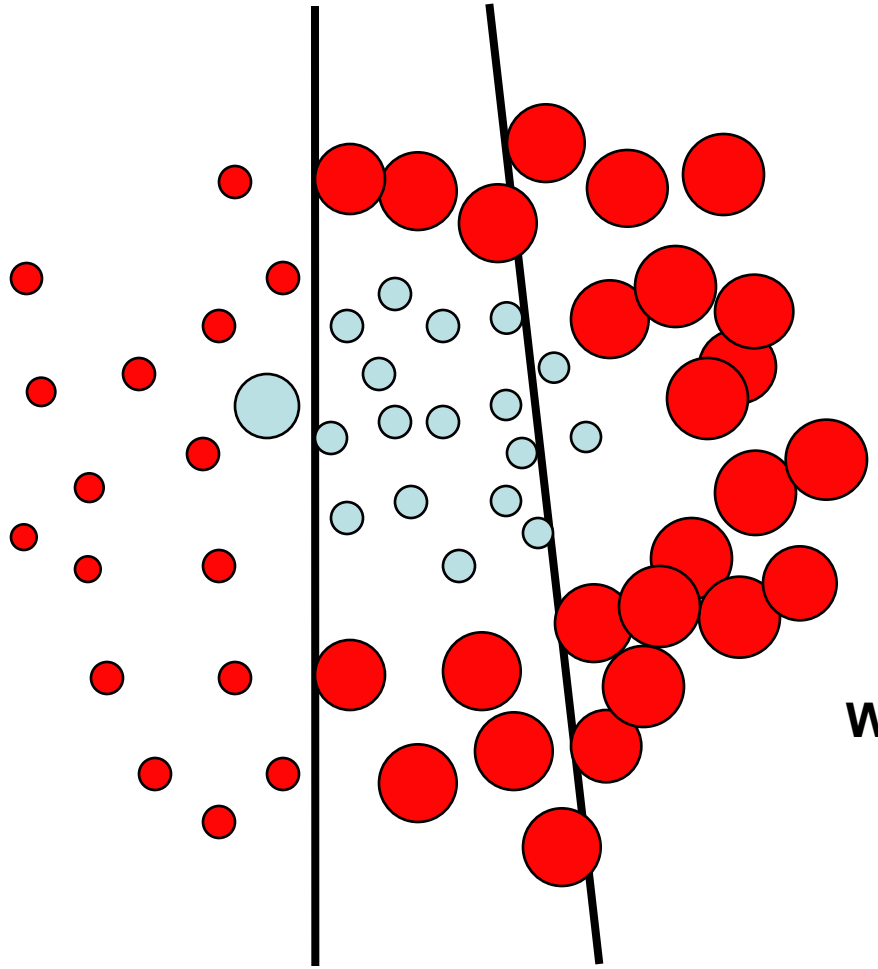
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has a class label:

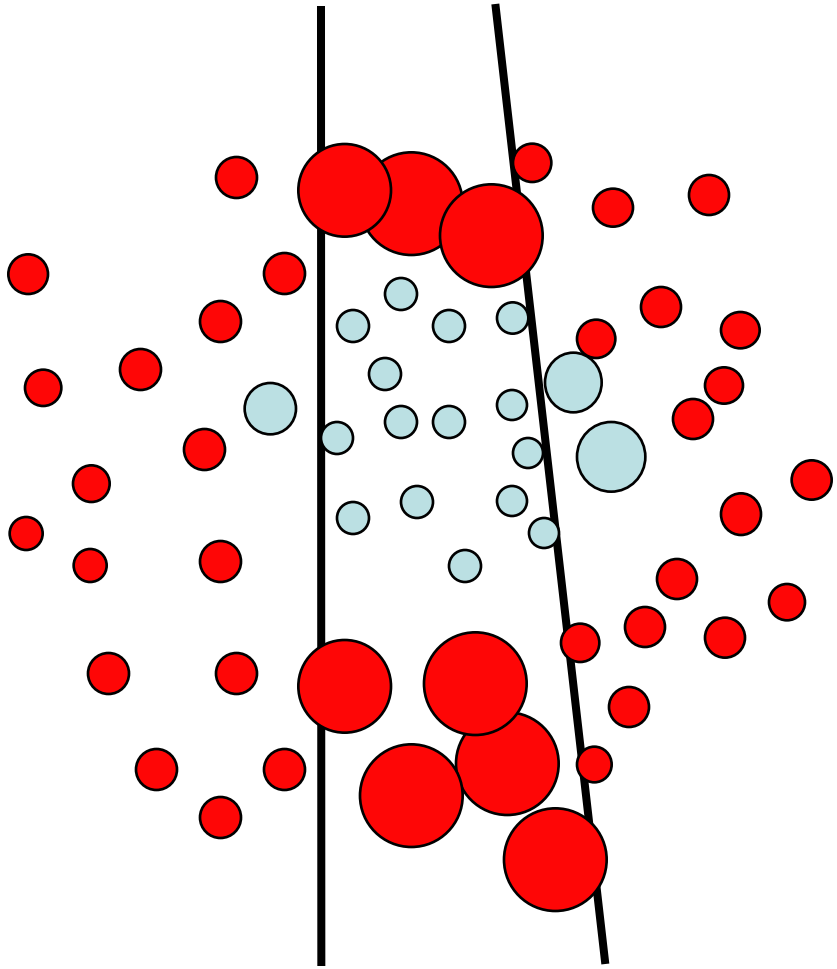
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has a class label:

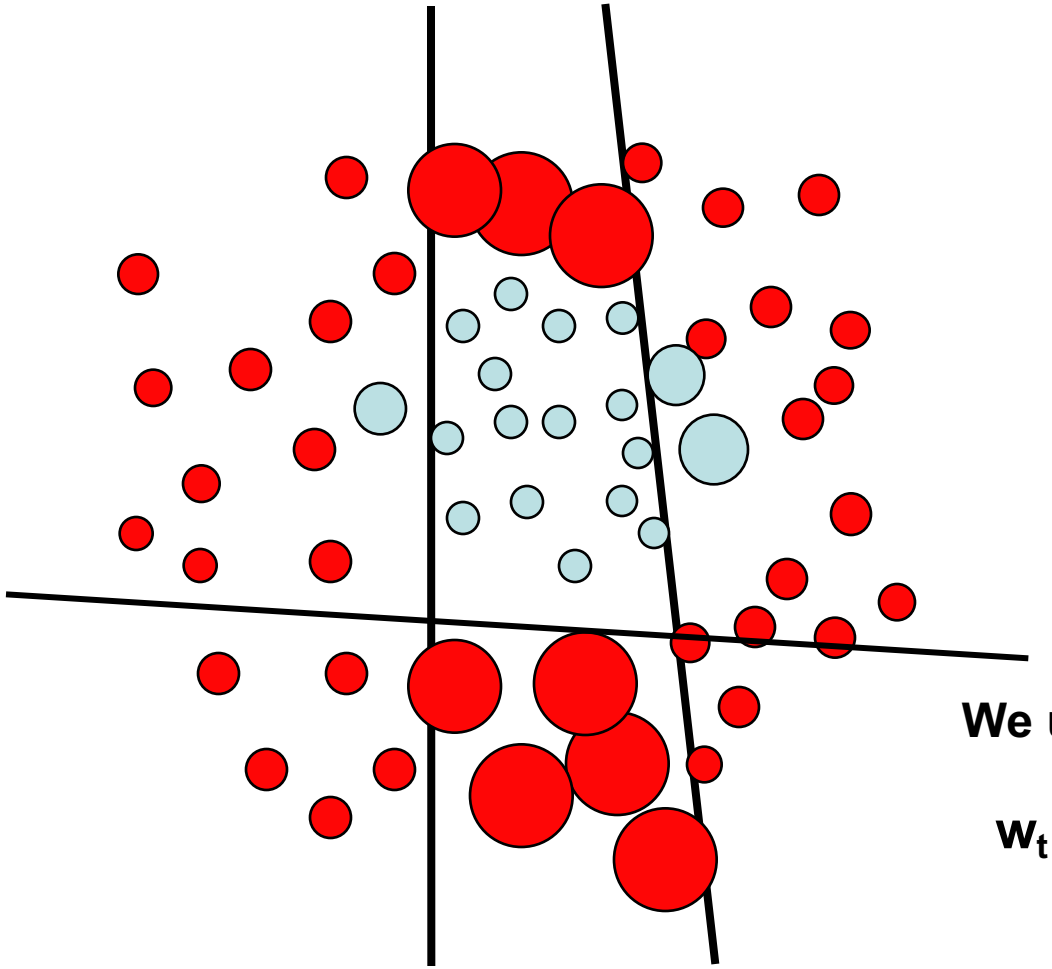
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has a class label:

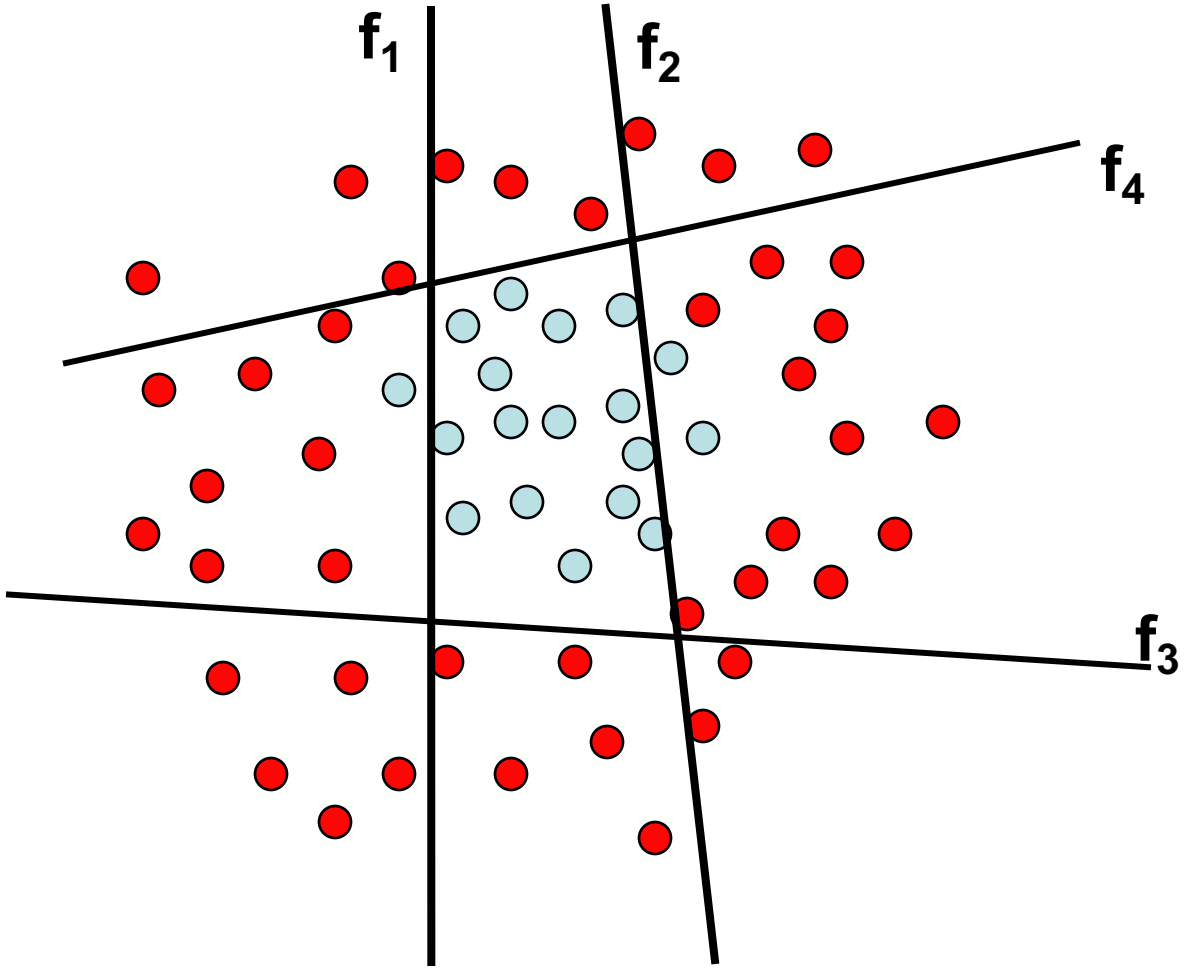
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

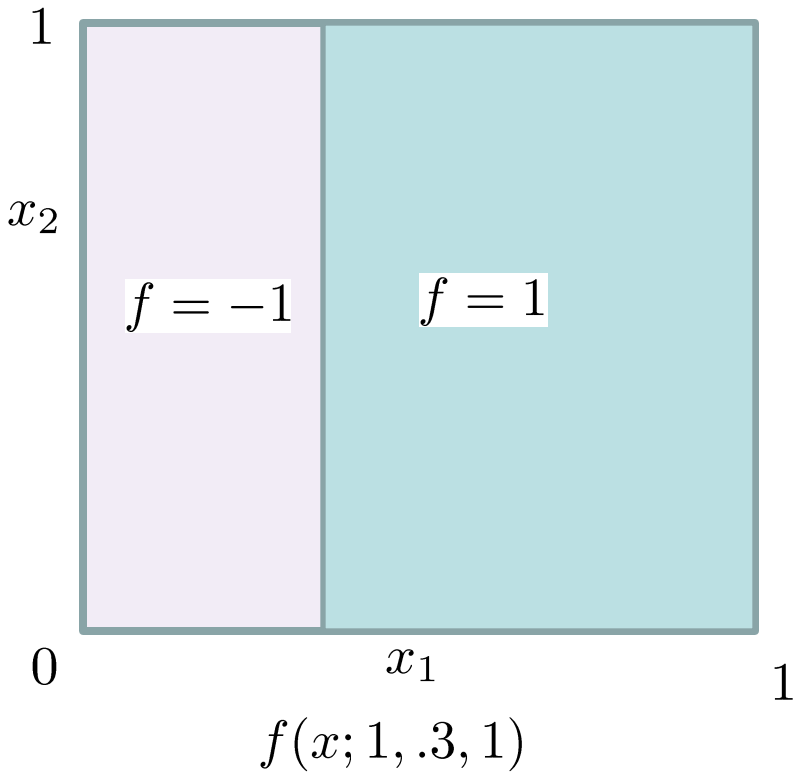
Toy example



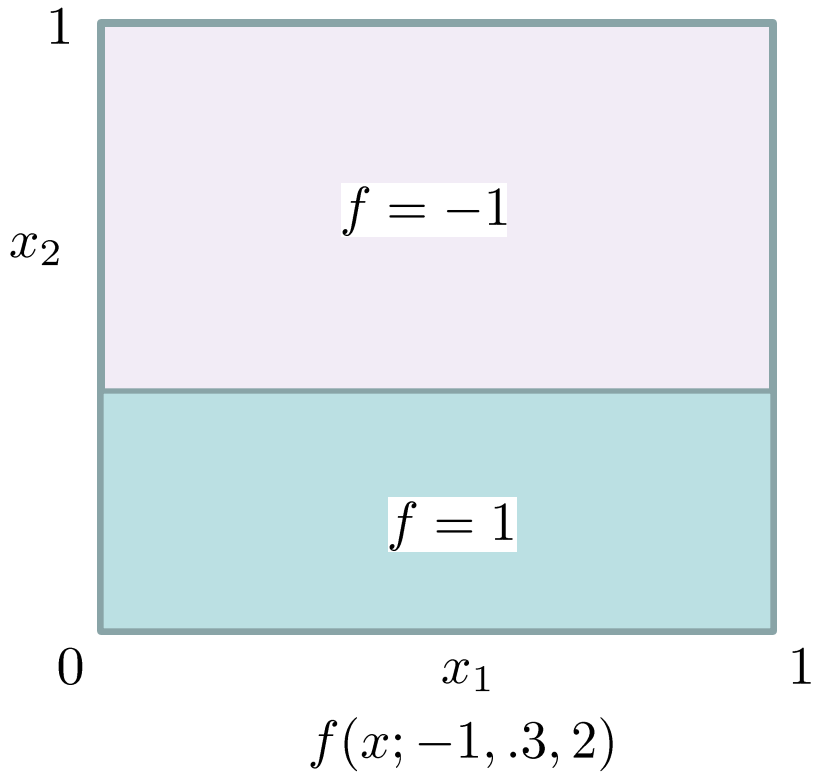
The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

Weak Learner Example

- Decision stump $f(x; s, \theta, i) = 2s[x_i > \theta] - s, \quad s \in \{-1, 1\}$



$f(x; 1, .3, 1)$



$f(x; -1, .3, 2)$

  - Extremely easy to train and evaluate

Adaboost Algorithm

- Given: $(x^i, y^i), x^i \in \mathcal{X}, y^i \in \{-1, 1\}, i = 1, \ldots, N$
- Initialize: $D_1(i) = \frac{1}{N}$ **D: distribution on samples**
- For $t = 1 \ldots T$
  - Find classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ with smallest weighted error

$$\epsilon_t = \frac{\sum_{i=1}^{N} D_t^i [y^i \neq h_t(x^i)]}{\sum_i D_t^i}$$

  - Update distribution

$$D_{t+1}^i = \frac{D_t^i}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if} y^i = h_t(x^i) \\ \exp(\alpha_t), & \text{if} y^i \neq h_t(x^i) \end{cases}$$

$$= \frac{D_t^i}{Z_t} \exp(-\alpha_t y^i h_t(x^i))$$

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

$$Z_t = \sum_i D_t^i \exp(-\alpha_t y^i h_t(x^i))$$

- Final classifier $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Optimization perspective of Adaboost

Claim: each round minimizes: $\sum_{i=1}^{N} \exp(-y^i f(x^i))$

by adding a new term to current f: $f'(x) = f(x) + \alpha h(x)$

Proof:

$$\sum_{i=1}^{N} \exp(-y^i f'(x^i)) \quad = \quad \sum_{i=1}^{M} \exp(-y^i [f(x^i) + \alpha h(x^i)])$$

Optimization perspective of Adaboost

Claim: each round minimizes: $\sum_{i=1}^{N} \exp(-y^i f(x^i))$

by adding a new term to current f: $f'(x) = f(x) + \alpha h(x)$

Proof:

$$\sum_{i=1}^{N} \exp(-y^i f'(x^i)) = \sum_{i=1}^{M} \exp(-y^i [f(x^i) + \alpha h(x^i)])$$

2nd order Taylor expansion

$$\simeq \sum_{i=1}^{N} \exp(-y^i f(x^i))[1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}(y^i)^2 h^2(x^i)]$$

Optimization perspective of Adaboost

Claim: each round minimizes: $\sum_{i=1}^{N} \exp(-y^i f(x^i))$

by adding a new term to current f: $f'(x) = f(x) + \alpha h(x)$

Proof:

$$\sum_{i=1}^{N} \exp(-y^i f'(x^i)) = \sum_{i=1}^{M} \exp(-y^i [f(x^i) + \alpha h(x^i)])$$

2nd order Taylor expansion

$$\simeq \sum_{i=1}^{N} \exp(-y^i f(x^i))[1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}(y^i)^2 h^2(x^i)]$$

$h(x^i), y^i$: +1 or -1

$$= \sum_{i=1}^{N} \exp(-y^i f(x^i))[1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}]$$

Optimization perspective of Adaboost

Claim: each round minimizes: $\sum_{i=1}^{N} \exp(-y^i f(x^i))$

by adding a new term to current f: $f'(x) = f(x) + \alpha h(x)$

Proof:

$$\sum_{i=1}^{N} \exp(-y^i f'(x^i)) = \sum_{i=1}^{M} \exp(-y^i [f(x^i) + \alpha h(x^i)])$$

2nd order Taylor expansion

$$\simeq \sum_{i=1}^{N} \exp(-y^i f(x^i))[1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}(y^i)^2 h^2(x^i)]$$

$h(x^i), y^i$: +1 or -1

$$= \sum_{i=1}^{N} \exp(-y^i f(x^i))[1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}]$$

$$= Z \sum_{i=1}^{N} \underbrace{\frac{\exp(-y^i f(x^i))}{Z}}_{D^i} [1 - \alpha y^i h(x^i) + \frac{\alpha^2}{2}]$$

since $D_t^i = \frac{D_{t-1}}{Z_t} \exp(-y^i \alpha_t h_t(x^i) = \frac{D_{t-2}}{Z_t Z_{t-1}} \exp\left(-y^i [\alpha_t h_t(x^I) + \alpha_{t-1} h_{t-1}(x^i)]\right) = \ldots \frac{1}{Z} \exp(-y^i f(x^i)$

Optimization perspective of Adaboost

Claim: each round minimizes: $\sum_{i=1}^{N} \exp(-y^i f(x^i))$

by adding a new term to current f: $f'(x) = f(x) + \alpha h(x)$

Last slide: $\sum_{i=1}^{N} \exp(-y^i f'(x^i)) = \sum_{i=1}^{M} \exp(-y^i [f(x^i) + \alpha h(x^i)])$

$$\simeq \sum_{i=1}^{N} D^i \left[1 - \alpha y^i h(x^i) + \frac{1}{2}\alpha^2\right]$$

Chose h to minimize the approximation's value for a given α:

$$h^* = \text{argmin}_h \sum_i D^i \left(-\alpha y^i h(x^i)\right) \overset{\alpha > 0}{=} \text{argmax}_h \sum_i D^i y^i h(x^i)$$

(Adaboost's Step 1)

$$= \text{argmin}_h \sum_i D^i [y^i \neq h(x^i)]$$

Fix h, mimimize loss $\sum_{i=1}^{N} D^i \exp(-\alpha y^i h(x^i))$ w.r.t. α: $\alpha = \frac{1}{2} \log \frac{1-\epsilon}{\epsilon}$

When does Adaboost work?

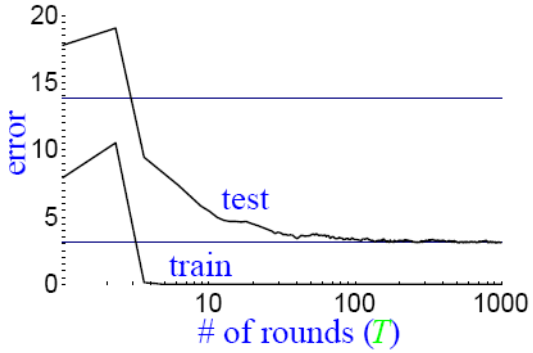
- Introduce `edge': $\epsilon_t = \frac{1}{2} - \underbrace{\gamma_t}_{\text{Edge}}$
- Theorem: if $\gamma_t \geq \gamma > 0, \quad \forall t$

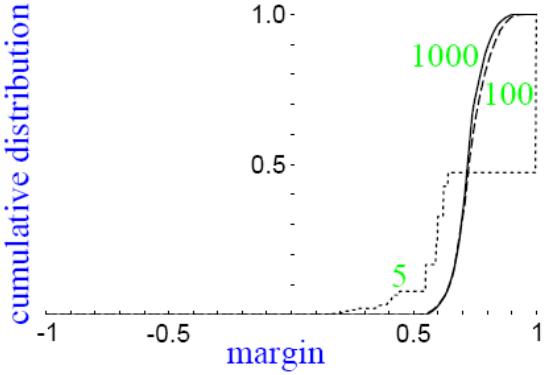
$$\sum_{i=1}^{N} \frac{1}{N} [y^i \neq H(x^i)] \leq \exp(-2T\gamma^2)$$

- Proof: 20-30 minutes, in Appendix

Generalization Error for Adaboost

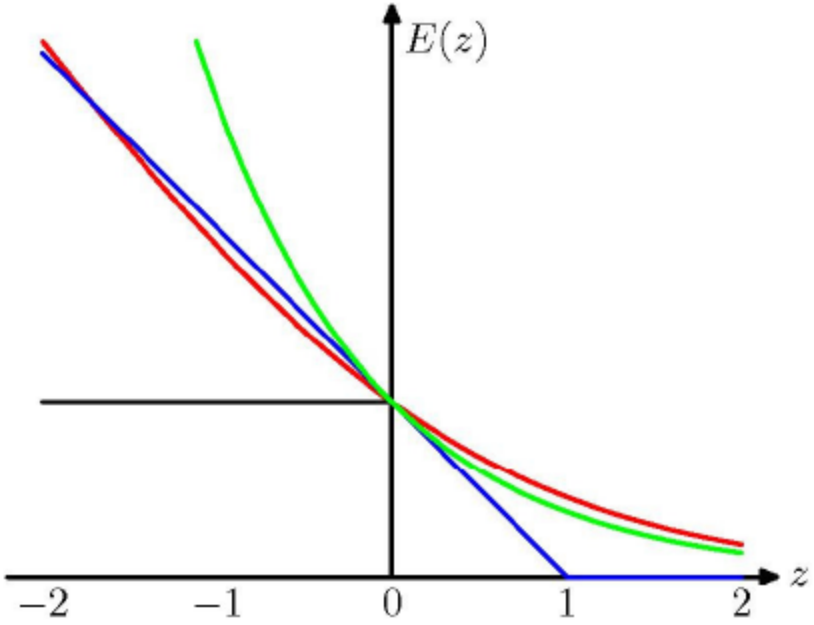
- **Empirical Evidence**





- **Theoretical Justification:**
  - **Margin of example i:** $y_i f(x_i)$
  - **Adaboost is increasing the margins of the training set**
  - **Large margin classifiers lead to good generalization**

Loss functions



- z: y*f(x)
- Ideal misclassification cost H(-z) (# training errors)
- Exponential Error exp(-z) (Adaboost)
- Cross Entropy error ln(1 + exp(-z)) (Logistic regression)
- Hinge loss max(0,1-z) (SVMs)

Using Adaboost to compute posteriors

- Consider `Empirical distribution' of data

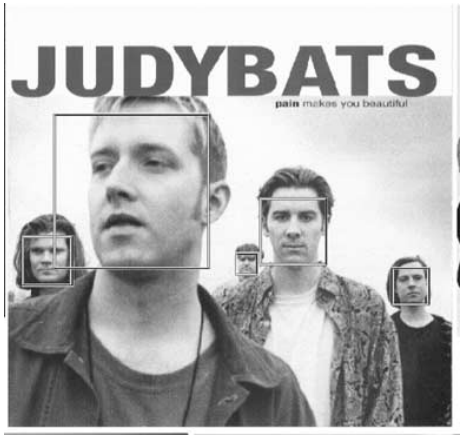
$$
\begin{aligned}
E_{X,Y}\exp(-yf(x)) &= \int_x \sum_y \exp(-yf(x))P(x,y)dxdy \\
&= \int_x \sum_y \exp(-yf(x))P(y|x)dyP(x)dx \\
&= \int_x [P(y=1|x)\exp(-f(x)) + P(y=-1|x)\exp(f(x))]dy]\,dx
\end{aligned}
$$

- Optimize w.r.t $f(x)$: $\quad f(x) = \log \dfrac{P(y=1|x)}{P(y=-1|x)}$

- We can therefore use $f(x)$ for a probabilistic classifier

Application to Vision

- Viola & Jones, `Rapid Object Detection using a Boosted Cascade of Simple Features', CVPR 01
  - First reliable, real-time face detection system
  - Used in commercial products, e.g. digital cameras
- Sample detections (back in 2001)

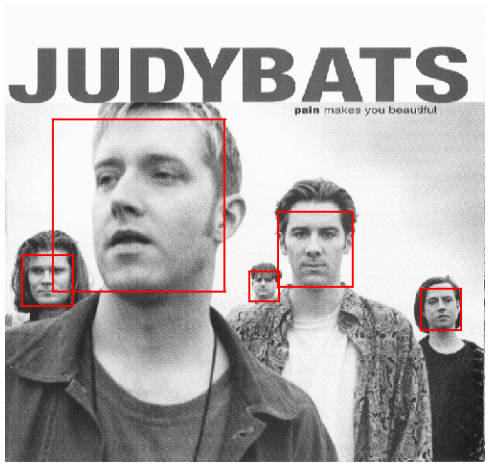


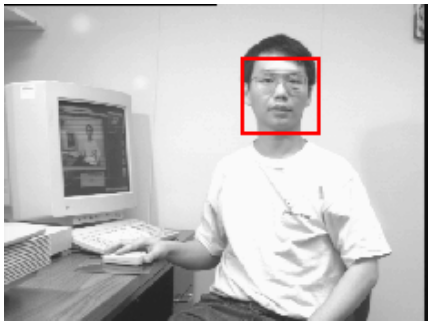
System performance

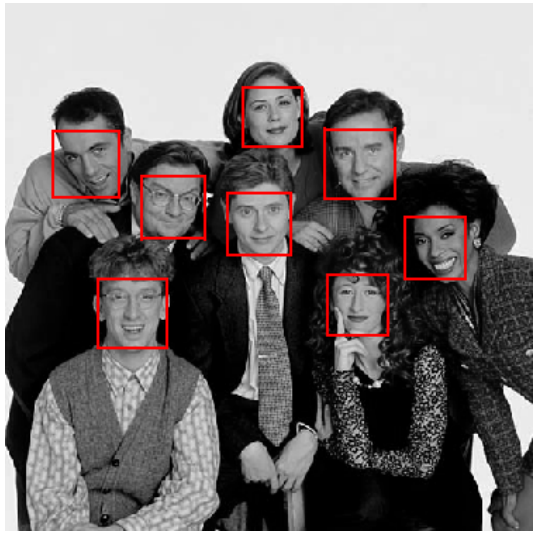
- Training time: “weeks” on 466 MHz Sun workstation
- 38 layers, total of 6061 features
- Average of 10 features evaluated per window on test set
- “On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds”
  - 15 Hz
  - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)

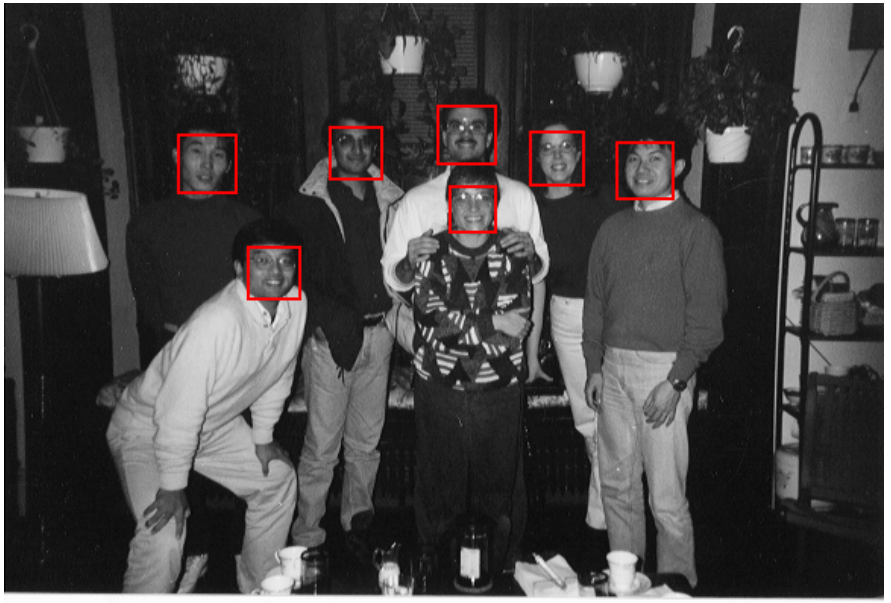
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. CVPR 2001.

Detection results (2001)










MPEGMOVIE VX
W
T
MENU
DISP
HOME
SONY

Lecture outline

- Recap
- Adaboost
- Decision Trees
- Random Forests & Ferns

When does Adaboost work?

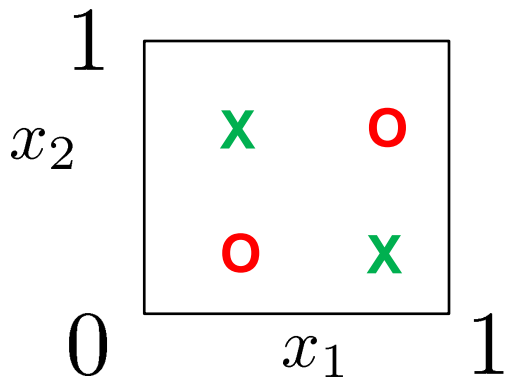
- Introduce `edge': $\epsilon_t = \frac{1}{2} - \underbrace{\gamma_t}_{\text{Edge}}$
- Theorem: if $\gamma_t \geq \gamma > 0, \quad \forall t$

$$\sum_{i=1}^{N} \frac{1}{N} [y^i \neq H(x^i)] \leq \exp(-2T\gamma^2)$$

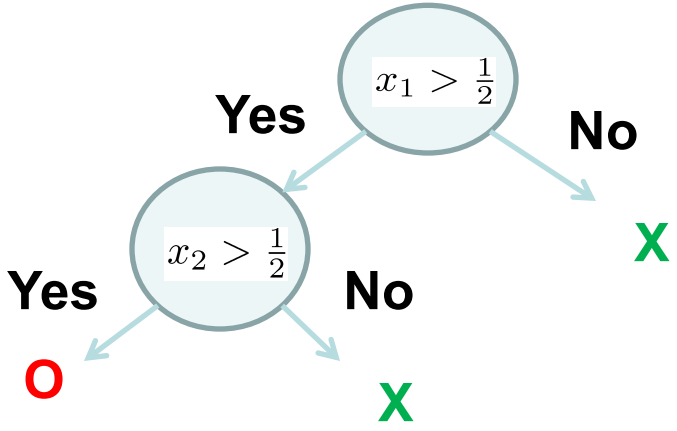
- Can the assumption of the theorem be violated?

Exclusive-or problem

- No decision stump can separate these data with error less than 1/2

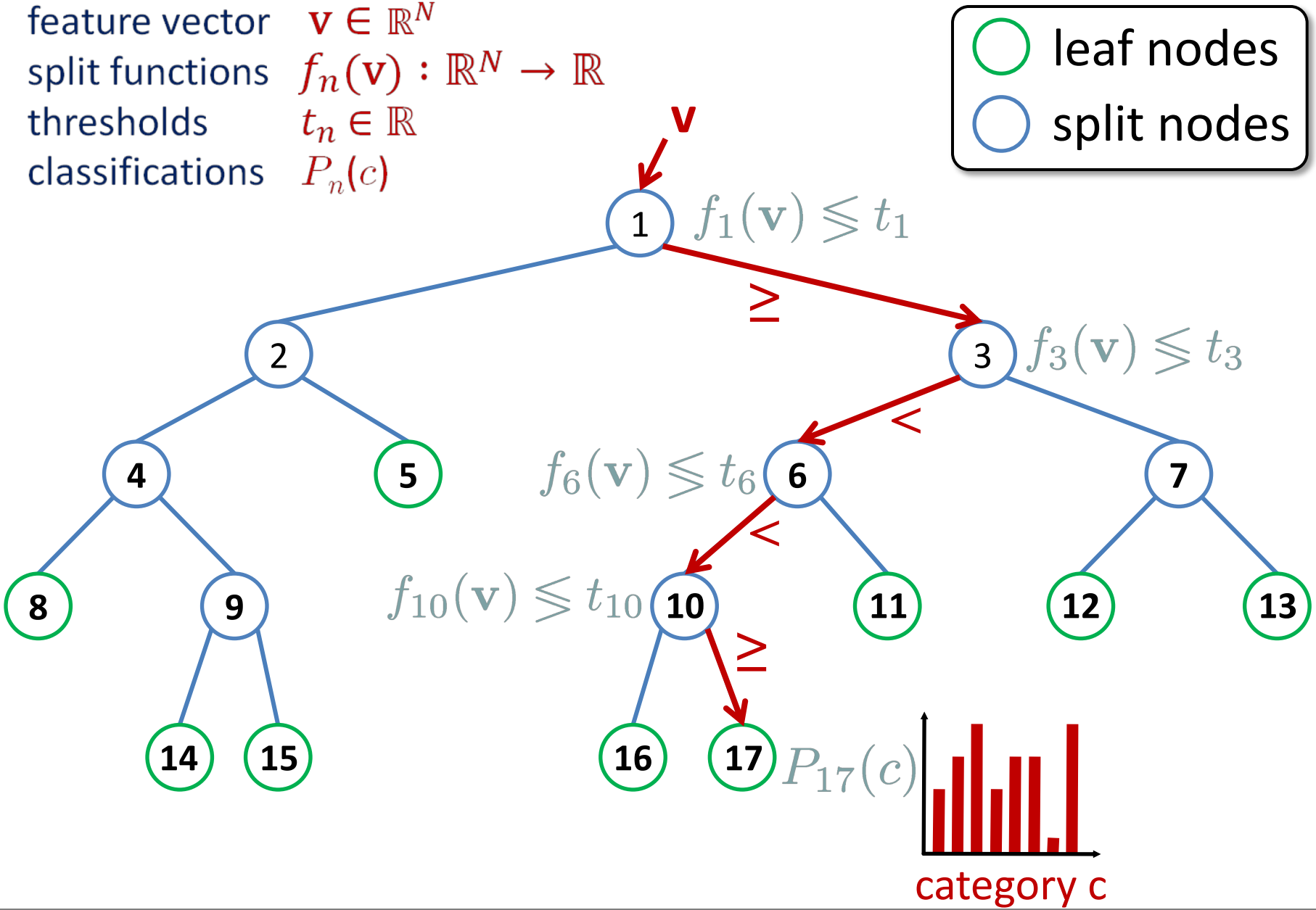


- two-step alternative: error 1/4

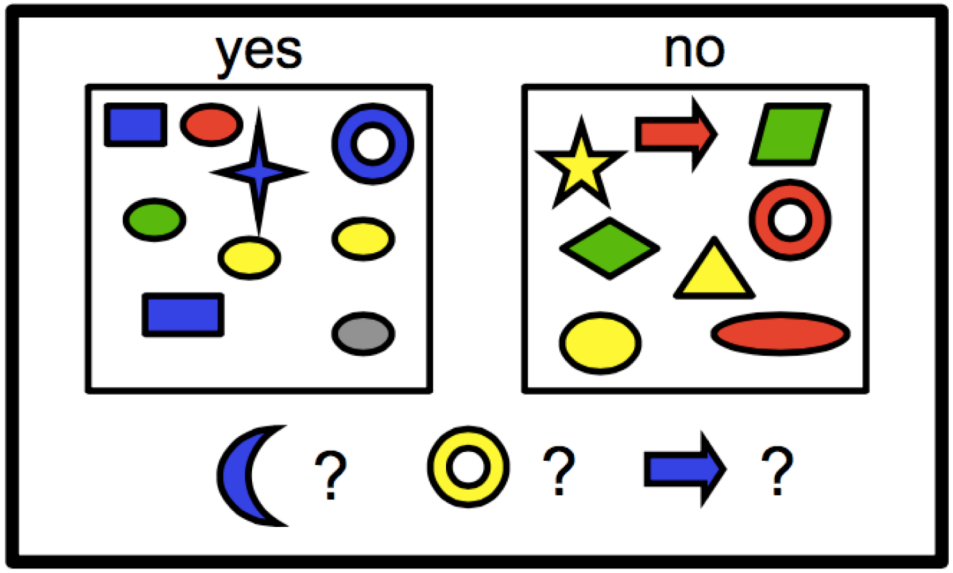


Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$

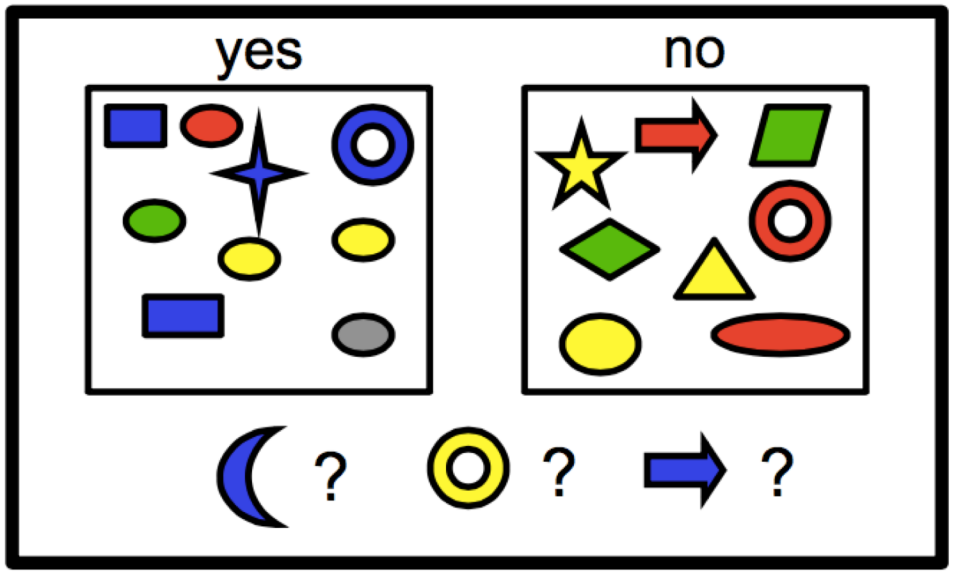


Example: Decision Trees for Classification

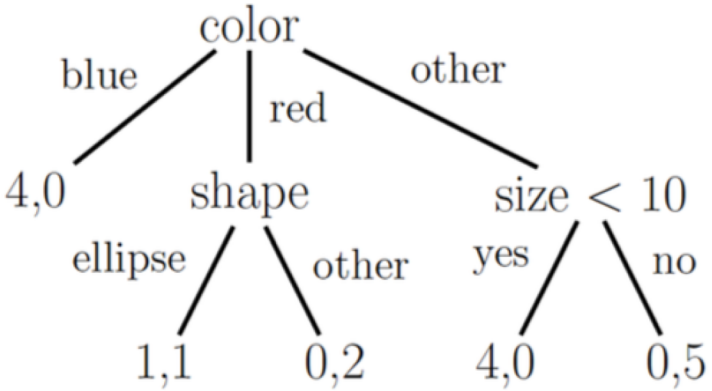


Features: color, shape, size

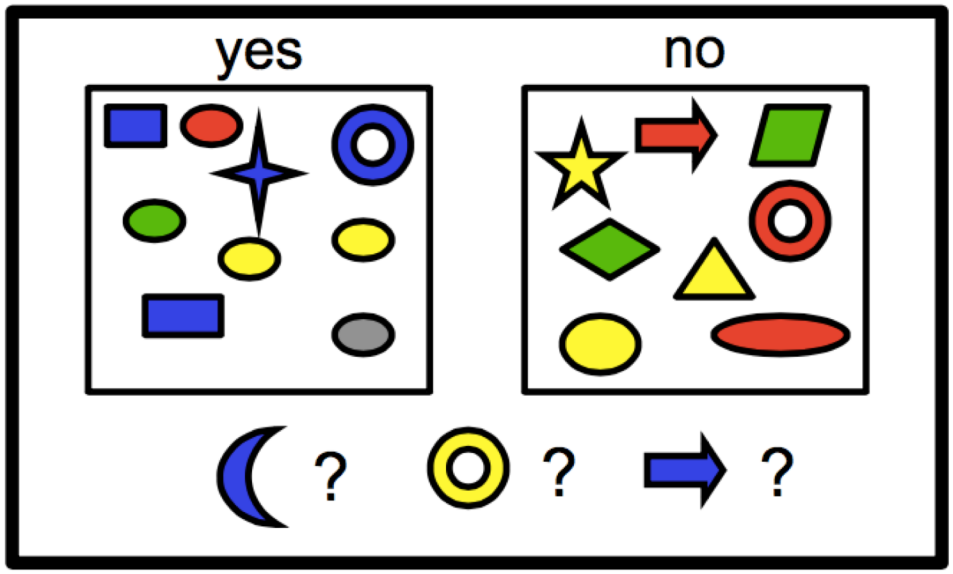
Example: Decision Trees for Classification

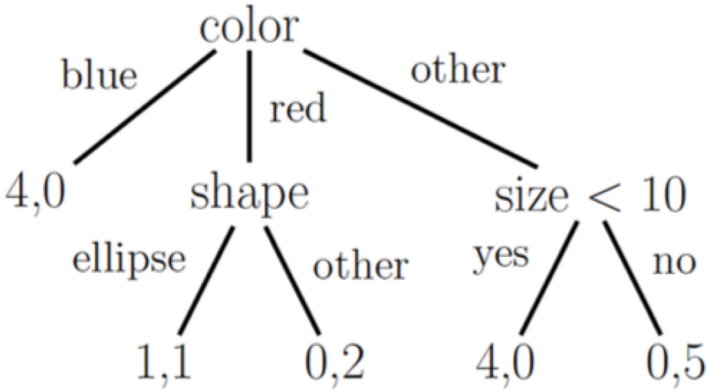


Features: color, shape, size



Example: Decision Trees for Classification

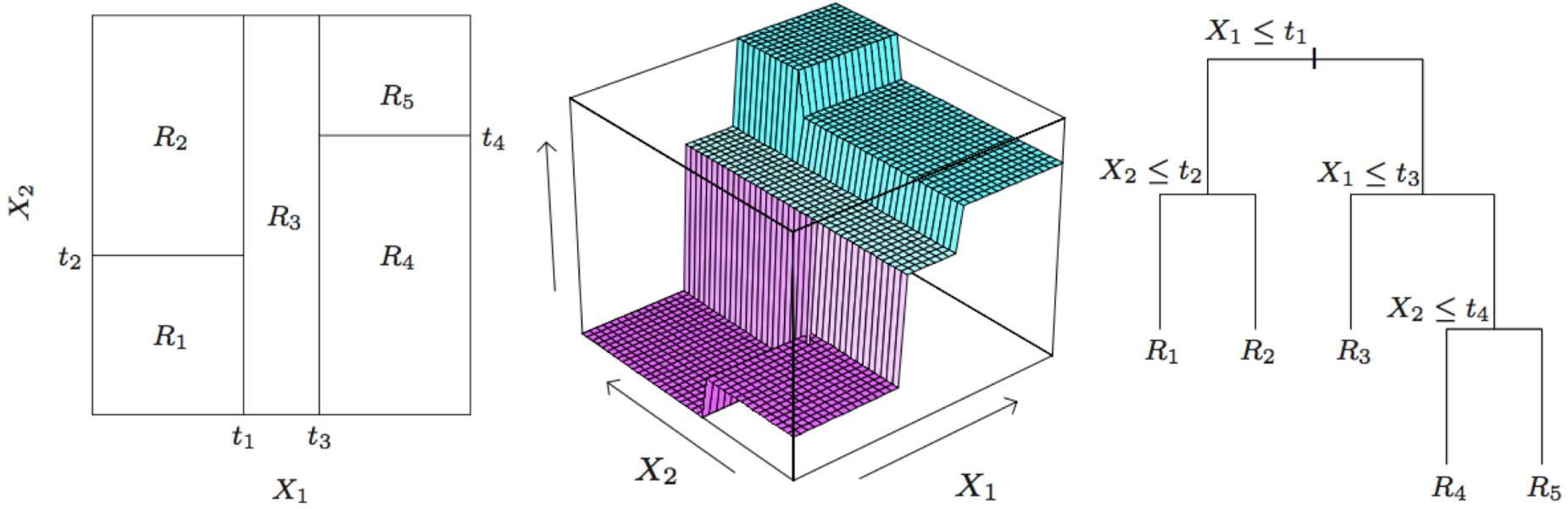




Features: color, shape, size

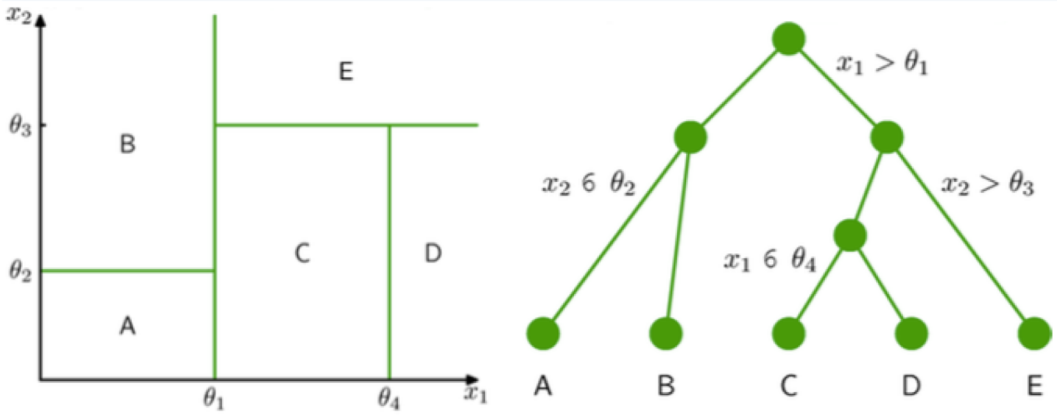
- Each node in tree splits examples according to a single feature
- Leaves have categorical distribution learned from labels of all training data whose path through tree ends there

Example: Decision Trees for Regression



- Each node in tree splits examples according to a single feature
- Leaves have Gaussian distribution (mean and variance) learned from values of all training data whose path through tree ends there

Decision Tree Learning



Leaves of tree: $R_1, R_2, \ldots, R_J$.

Regression:

$$p(t_n \mid x_n \in R_j) = \mathcal{N}(t_n \mid \mu_j, \sigma_j^2)$$

Binary Classification:

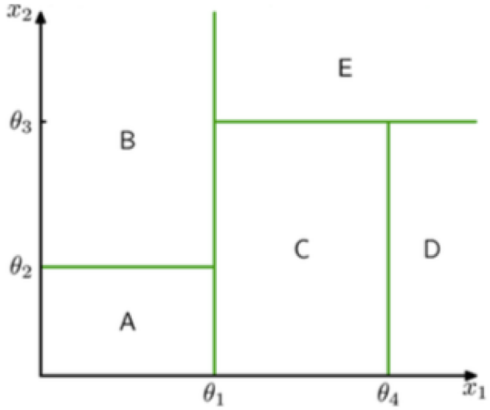
$$p(t_n \mid x_n \in R_j) = \text{Ber}(t_n \mid \mu_j)$$

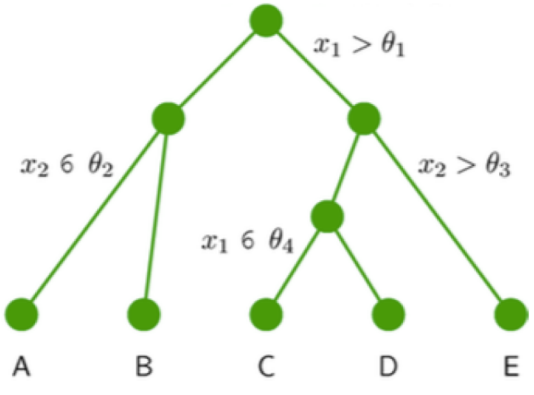
- Let *w* denote decision tree parameters: structure, splits, leaf distributions
- Conceptually, we would like to minimize the following objective:

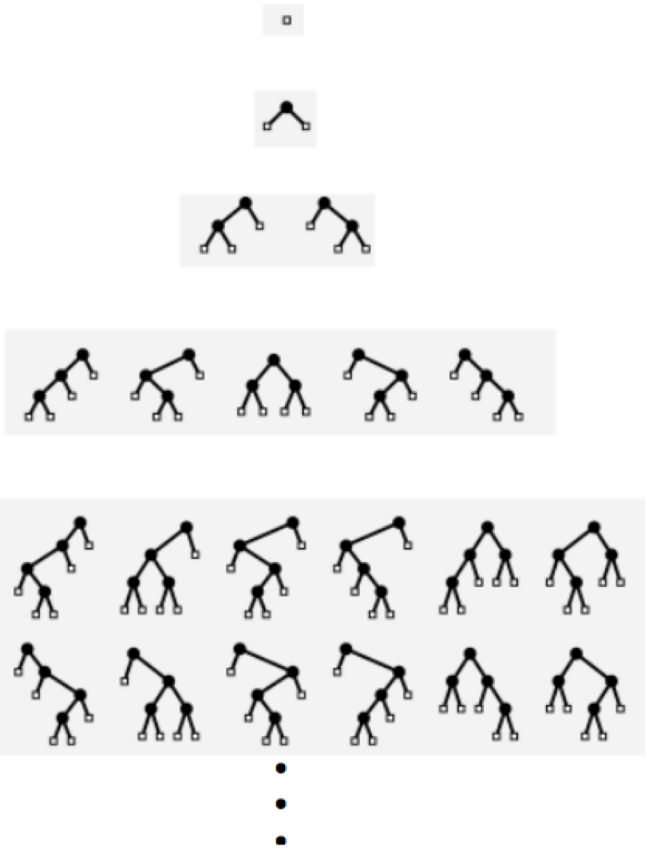
$$f(w) = \lambda L(w) - \sum_{n=1}^{N} \log p(t_n \mid x_n, w)$$

- Here, L(w) is some measure of tree complexity:
  count of number of nodes, negative log of prior on trees, etc.

Decision Tree Learning







- **PROBLEM:** There are an enormous number of tree structures & split thresholds
- Optimizing leaf distributions given structure is easy, but searching structures is hard
- Classic approach: Build tree greedily, using various heuristics to control complexity, and check with validation data

CART: Classification & Regression Trees, C4.5, ID3, …

Greedy: grow a tree by optimizing a local objective – stick to the choices made in earlier iterations

Q: what is the objective for growing a tree?

Information Content and Entropy

Information content of an outcome x

$$h(x) = \log_2 \frac{1}{P(x)}$$

measure of surprise: predictable events do not provide information

The French live to eat and the British eat to **live (low IC, cliché)**
die (high IC, funny)

Entropy of a distribution: expected information content of a random variable

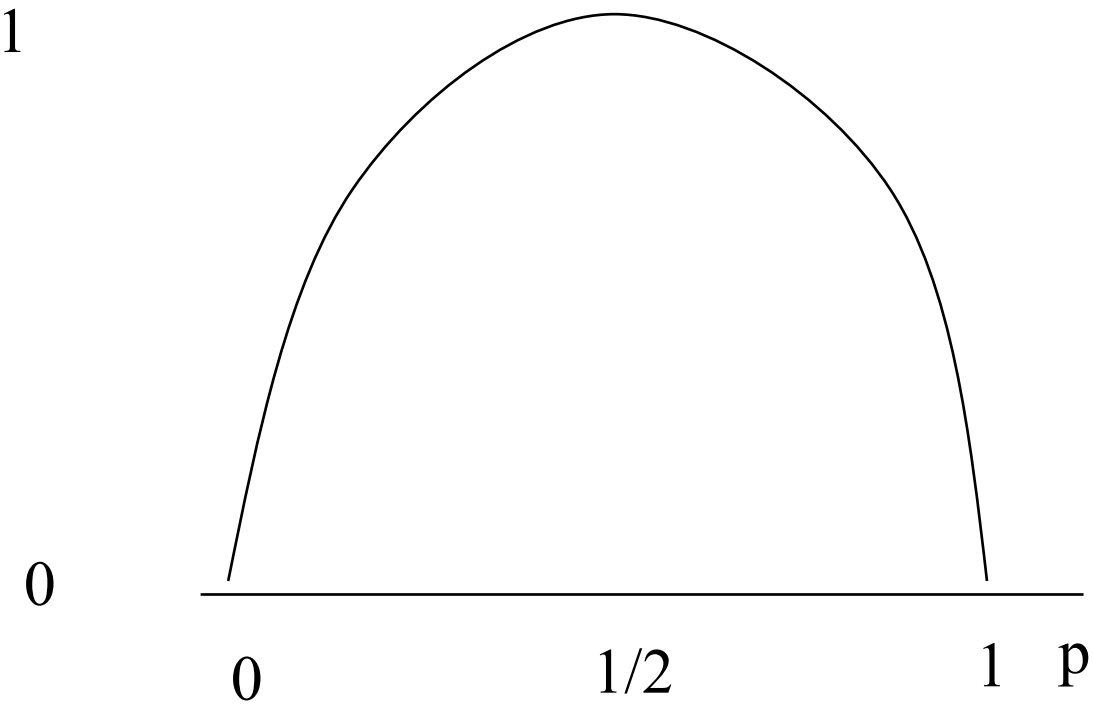
$$H(X) = \sum_{x \in \mathcal{A}_X} P(x) \log \frac{1}{P(x)}$$

measure of uncertainty: should we expect to be surprised?

Entropy of Bernoulli-distributed binary variable

P(x=1) = p
P(x=0) = 1-p

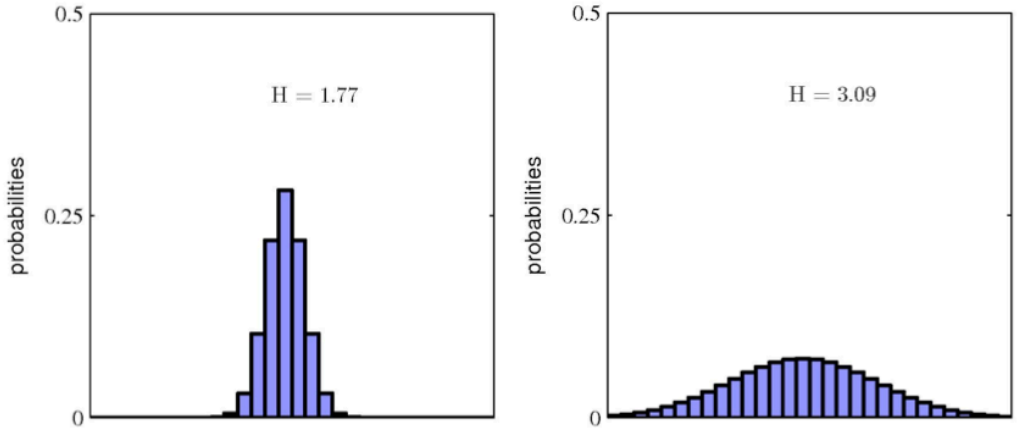
E = p log(1/p) + (1-p) log(1/(1-p))



p = 0.5: as unpredictable as it gets

p = 0, or p=1: no information gained by observing

Entropy: measure of uncertainty



- Histograms of two probability distributions over 30 bins.
- The largest entropy will arise from a uniform distribution
  H = -ln(1/30) = 3.40.

Entropy =0: Deterministic

Entropy =ln(N): Maximally unpredictable (uniform)

Discrete: Entropy

$$H(\mathbf{p}) = -\sum_i p_i log(p_i)$$

Continuous: Differential Entropy

$$H(\mathbf{p}) = -\int_x p(x) \log\left(p(x)\right) dx$$

x can be random vector!

Entropy Example

The name $\mathcal{A}$ is mnemonic for 'alphabet'. One example of an ensemble is a letter that is randomly selected from an English document. This ensemble is shown in figure 2.1. There are twenty-seven possible letters: a–z, and a space character '–'.

Example 2.12. The entropy of a randomly selected letter in an English doc ment is about 4.11 bits, assuming its probability is as given in table 2 We obtain this number by averaging $\log 1/p_i$ (shown in the fourth c umn) under the probability distribution p_i (shown in the third colum

i	a_i	p_i	$h(p_i)$
1	a	.0575	4.1
2	b	.0128	6.3
3	c	.0263	5.2
4	d	.0285	5.1
5	e	.0913	3.5
6	f	.0173	5.9
7	g	.0133	6.2
8	h	.0313	5.0
9	i	.0599	4.1
10	j	.0006	10.7
11	k	.0084	6.9
12	l	.0335	4.9
13	m	.0235	5.4
14	n	.0596	4.1
15	o	.0689	3.9
16	p	.0192	5.7
17	q	.0008	10.3
18	r	.0508	4.3
19	s	.0567	4.1
20	t	.0706	3.8
21	u	.0334	4.9
22	v	.0069	7.2
23	w	.0119	6.4
24	x	.0073	7.1
25	y	.0164	5.9
26	z	.0007	10.4
27	–	.1928	2.4
		$\sum_i p_i \log_2 \frac{1}{p_i}$	4.1

Table 2.9. Shannon information contents of the outcomes a–z.

Evaluating a split in terms of Entropy

$$\begin{aligned} \text{left split } I_{\mathrm{l}} &= \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \text{right split } I_{\mathrm{r}} &= I_n \setminus I_{\mathrm{l}} \end{aligned}$$

- Features f(v) chosen from feature pool
- Thresholds t in range $t \in (\min_i f(\mathbf{v}_i), \max_i f(\mathbf{v}_i))$
- Choose f and t to maximize gain in information

$$\Delta E = -\frac{|I_{\mathrm{l}}|}{|I_n|} E(I_{\mathrm{l}}) - \frac{|I_{\mathrm{r}}|}{|I_n|} E(I_{\mathrm{r}})$$

E: Entropy calculated from histogram of labels in I

Objective: both of the resulting trees should be "pure", i.e. have low label uncertainty

Binary Decision Trees Summary

- Fast greedy training algorithms
  - Work with heterogeneous pool of features (e.g. shape, size, color,…)
- Fast testing algorithm
- Needs careful choice of hyper-parameters
  - maximum depth
  - number of features and thresholds to try
- Prone to over-fitting

Lecture outline

Recap

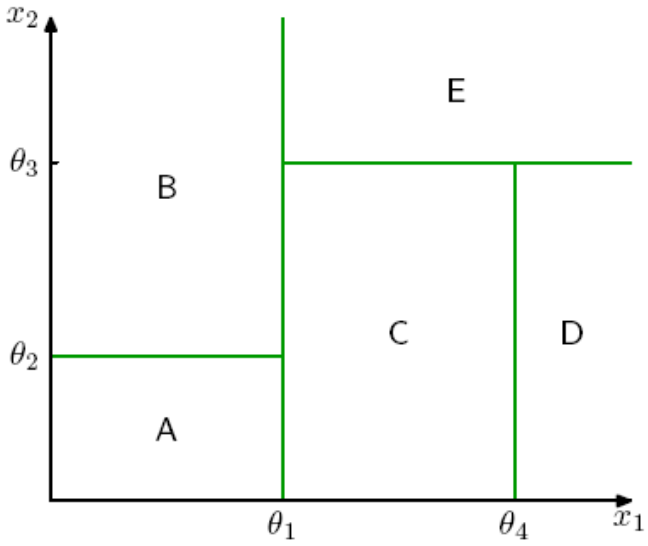
Adaboost

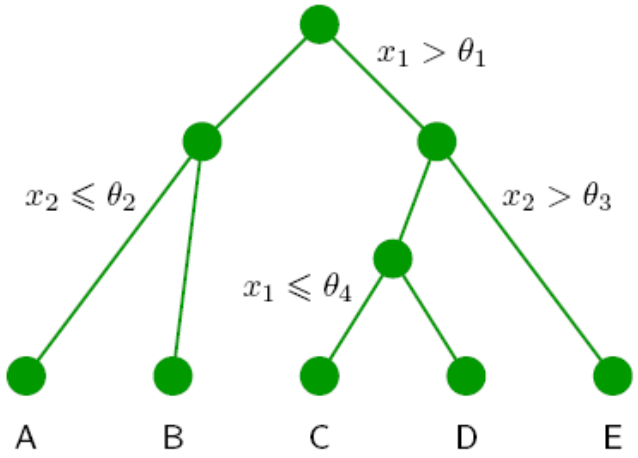
Decision Trees

Random Forests & Ferns

Which level of classification tree?

- Deeper trees result in more complex classifiers



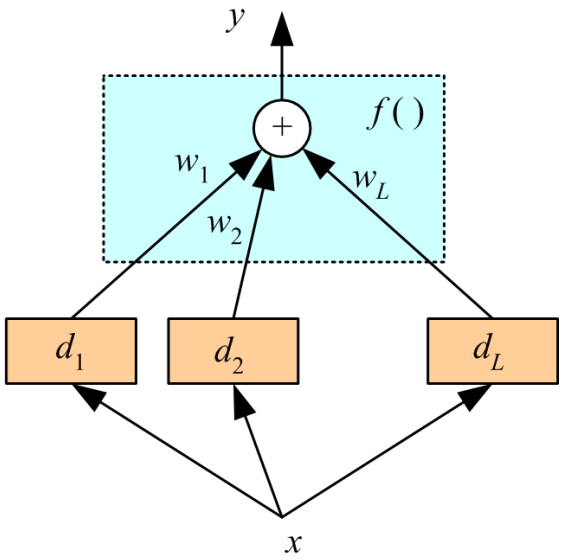


- After some point overfitting occurs

Voting Methods

- Give up idea of building `the' classifier
- Generate a group of base-learners which has higher accuracy when combined
- Main tasks
  - Generating the learners
  - Combining them

$$y_{COM}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$



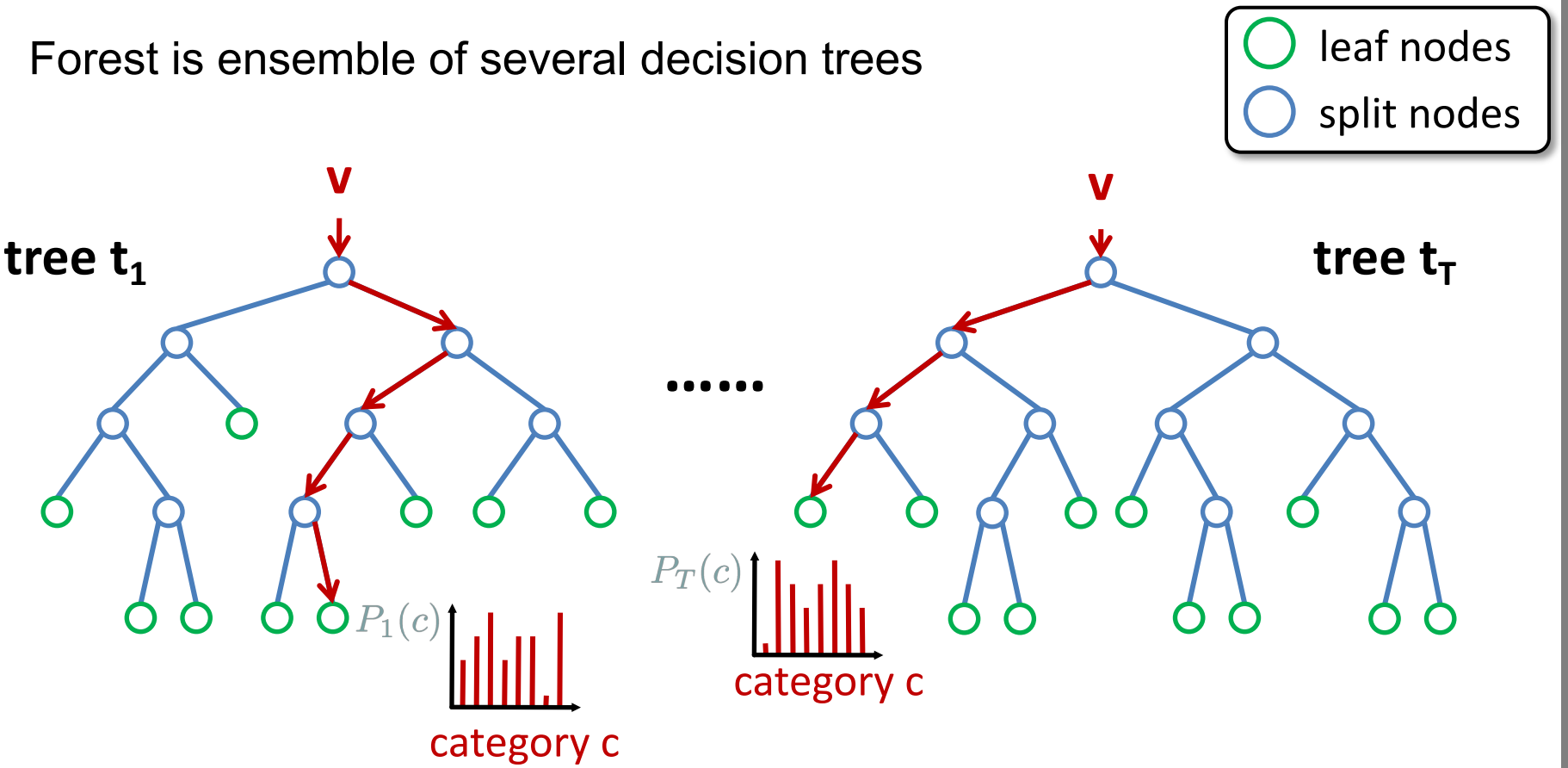
If errors have zero mean and are uncorrelated: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] = 0$

$$\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$$

then $\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$

A Forest of Trees

- Forest is ensemble of several decision trees



– classification is $$P(c|\mathbf{v}) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|\mathbf{v})$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

Learning a Forest

- Divide training examples into T subsets I_t
  - **improves generalization**
  - reduces **memory requirements** & **training time**
- Train each decision tree t on subset I_t
  - same decision tree learning as before

Randomized Learning

- Recursively split examples at node n
  - set I_n indexes labeled training examples ($\mathbf{v}_i$, l_i):

left split

$$I_\mathrm{l} = \{i \in I_n \mid f(\mathbf{v}_i) < t\}$$

right split

$$I_\mathrm{r} = I_n \setminus I_\mathrm{l}$$

threshold

function of example i's feature vector

- At node n, $P_n(c)$ is histogram of example labels l_i

More Randomized Learning

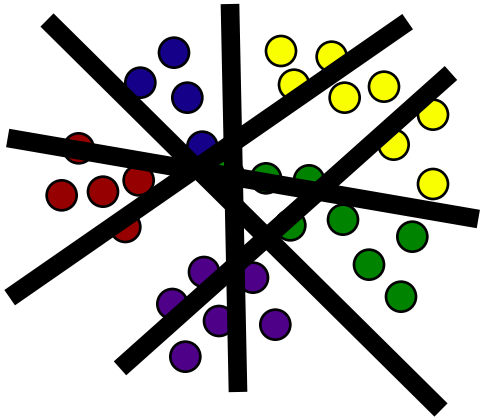
$$\begin{aligned} \textbf{left split } I_{\mathrm{l}} &= \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \textbf{right split } I_{\mathrm{r}} &= I_n \setminus I_{\mathrm{l}} \end{aligned}$$

- Features f(v) chosen at random from feature pool f 2 F
- Thresholds t chosen in range $t \in (\min_i f(\mathbf{v}_i), \max_i f(\mathbf{v}_i))$
- Choose f and t to maximize quality of lower trees

Why do random tests work?

For a small number of classes

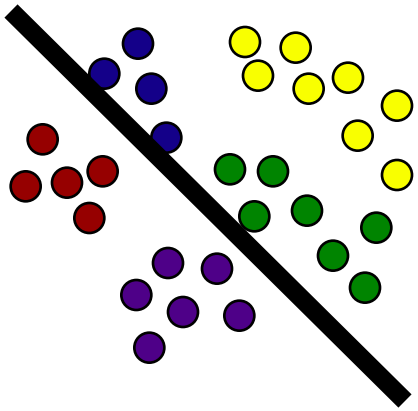
- we can try several tests, and
- retain the best one according to some criterion.



Why do random tests work?

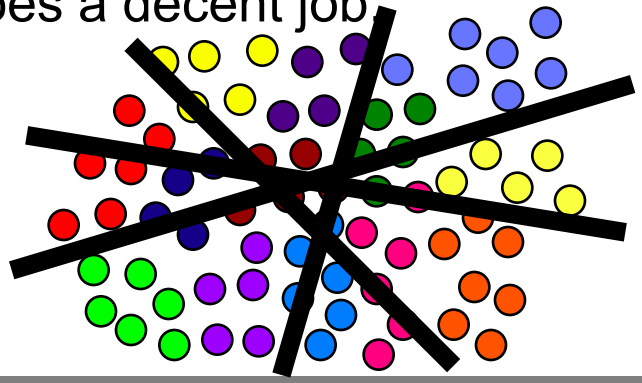
For a small number of classes

- we can try several tests, and
- retain the best one according to some criterion.



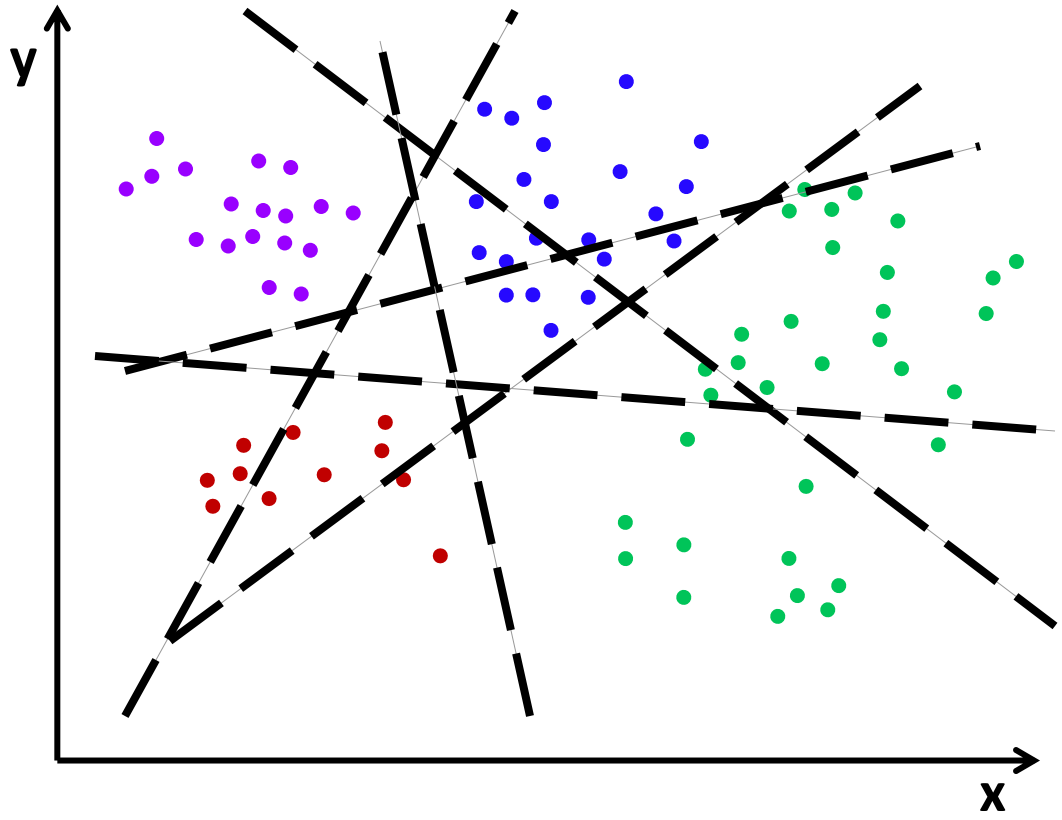
When the number of classes is large

- any test does a decent job:



Toy Learning Example

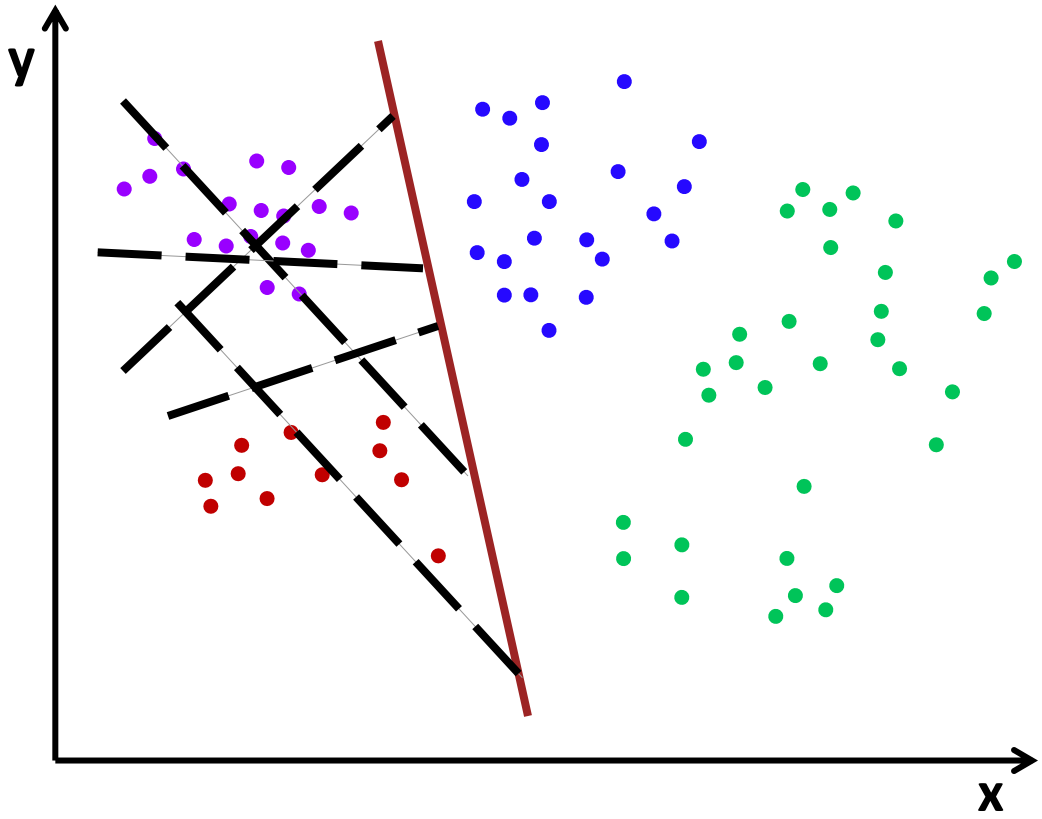
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

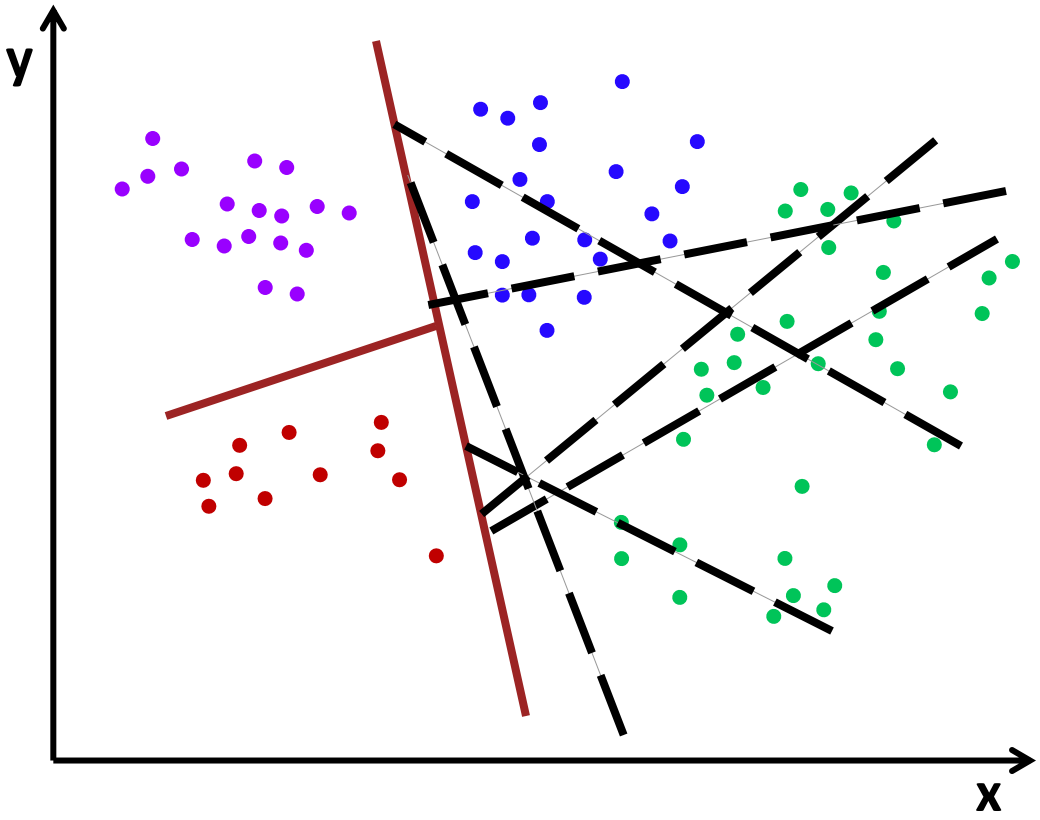
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

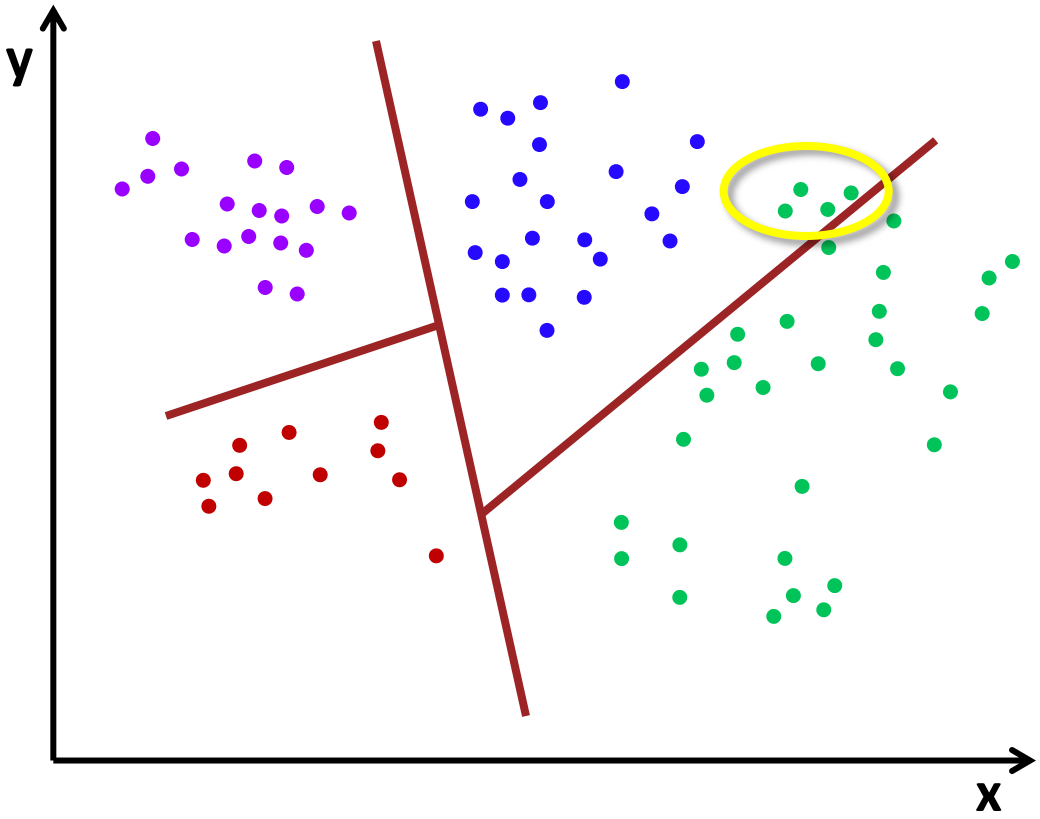
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

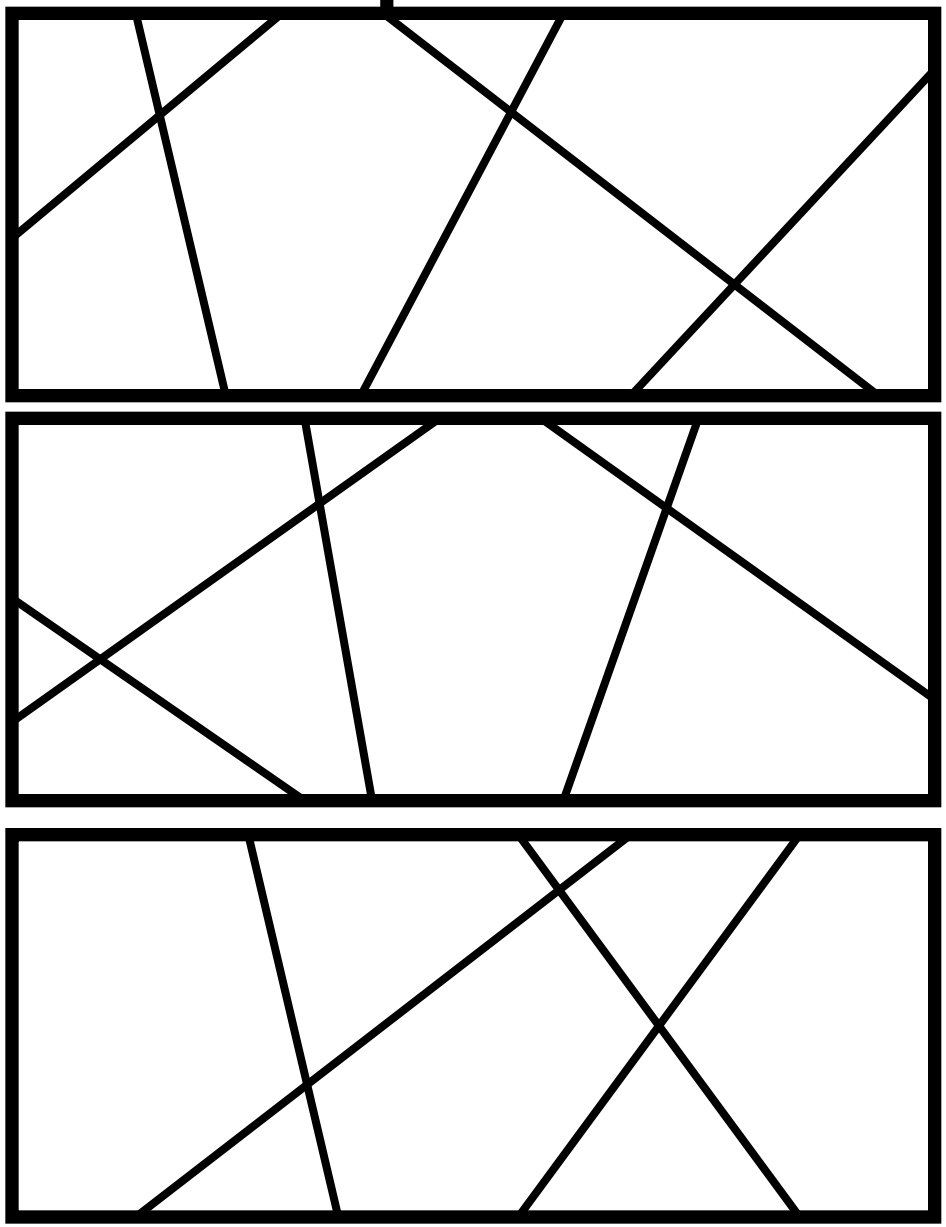
Toy Learning Example

- Try several lines, chosen at random
- Keep line that best separates data
  - information gain
- Recurse

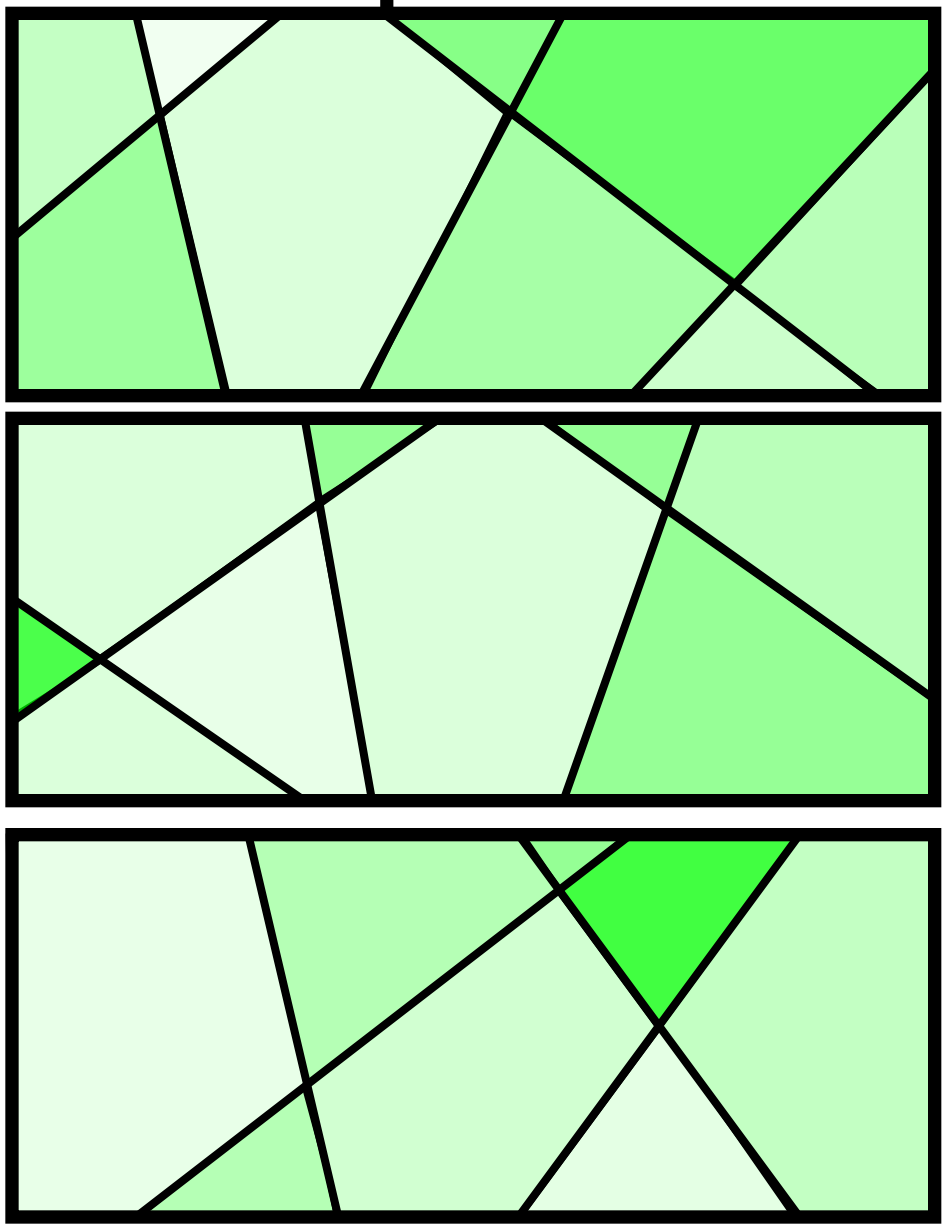


- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

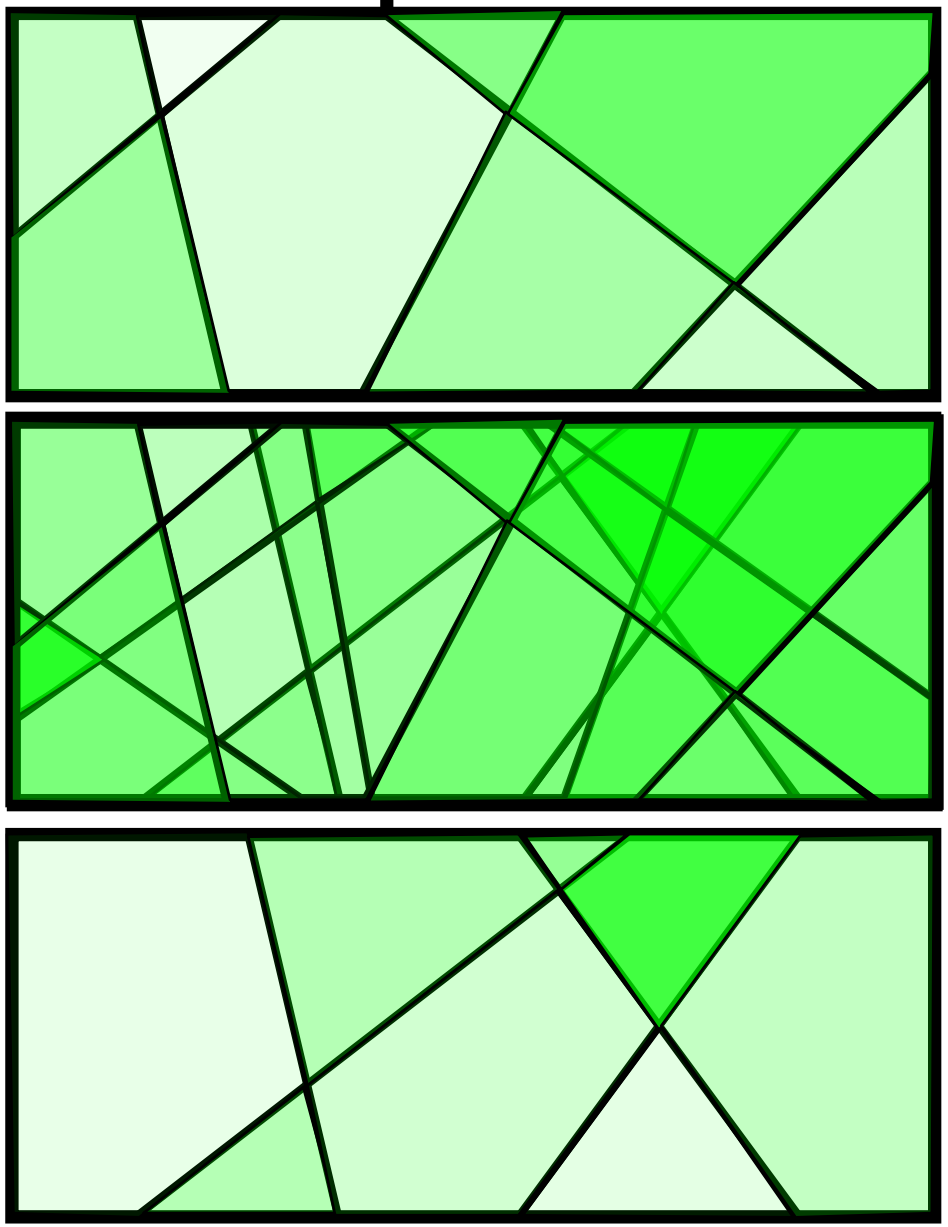
A Graphical Interpretation



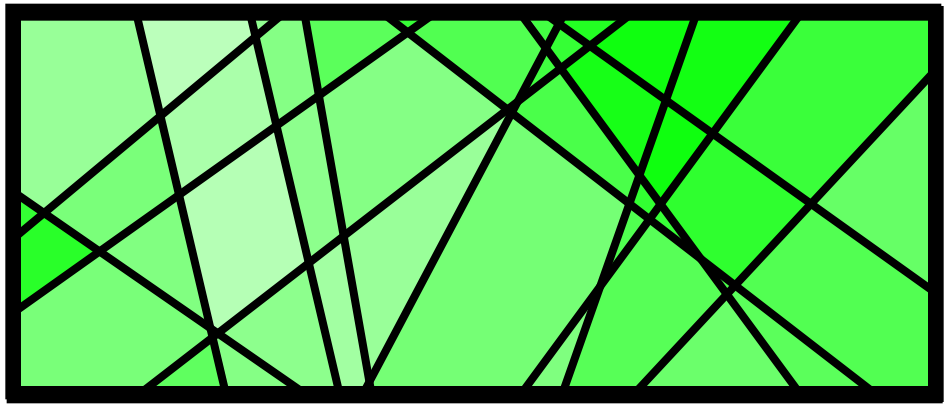
A Graphical Interpretation



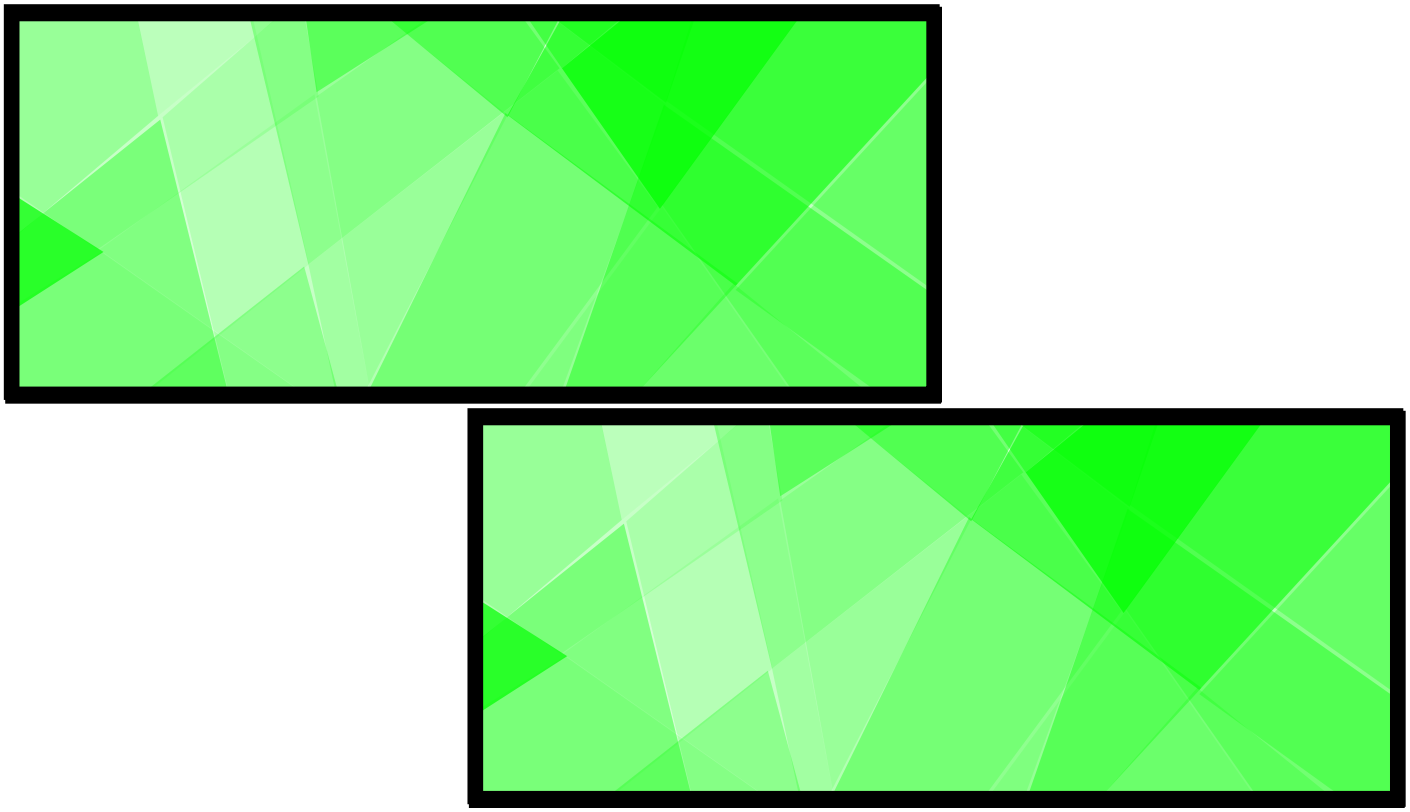
A Graphical Interpretation



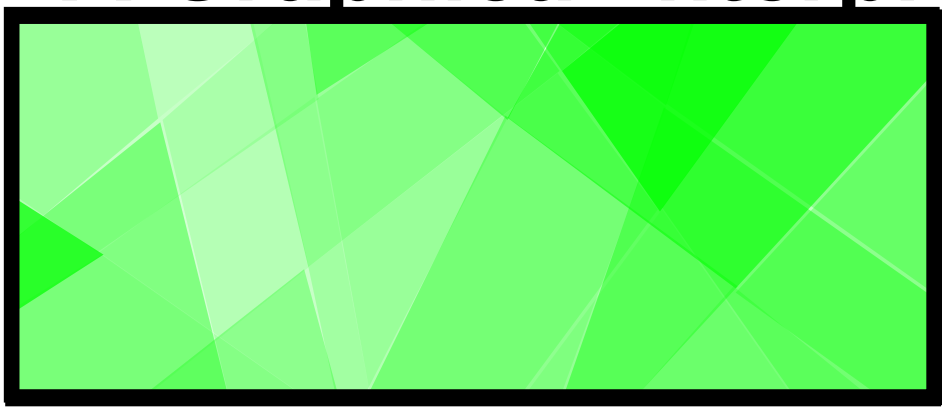
A Graphical Interpretation

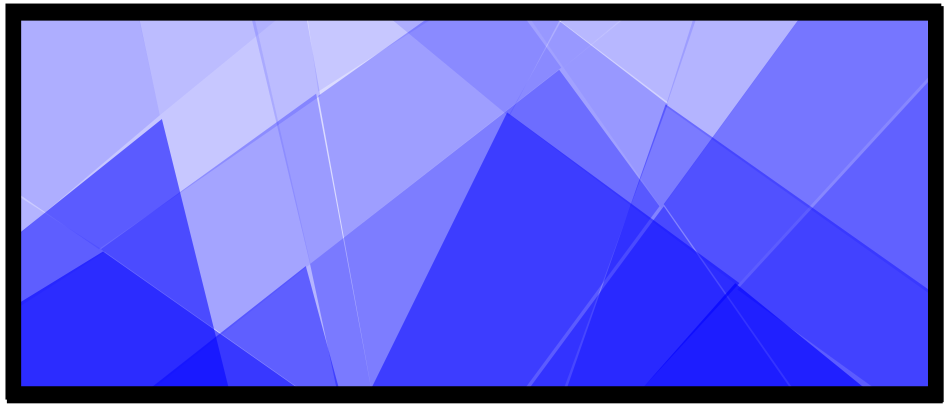


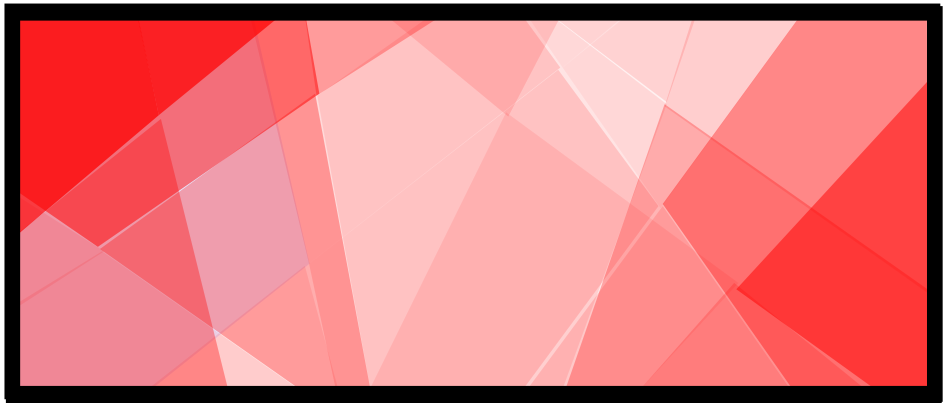
A Graphical Interpretation



A Graphical Interpretation







Relation to Cascades [Viola & Jones 04]

- Boosted Cascades
  - very unbalanced tree
  - good for unbalanced binary problems
    e.g. sliding window object detection

- Randomized forests
  - less deep, fairly balanced
  - ensemble of trees gives robustness
  - good for multi-class problems