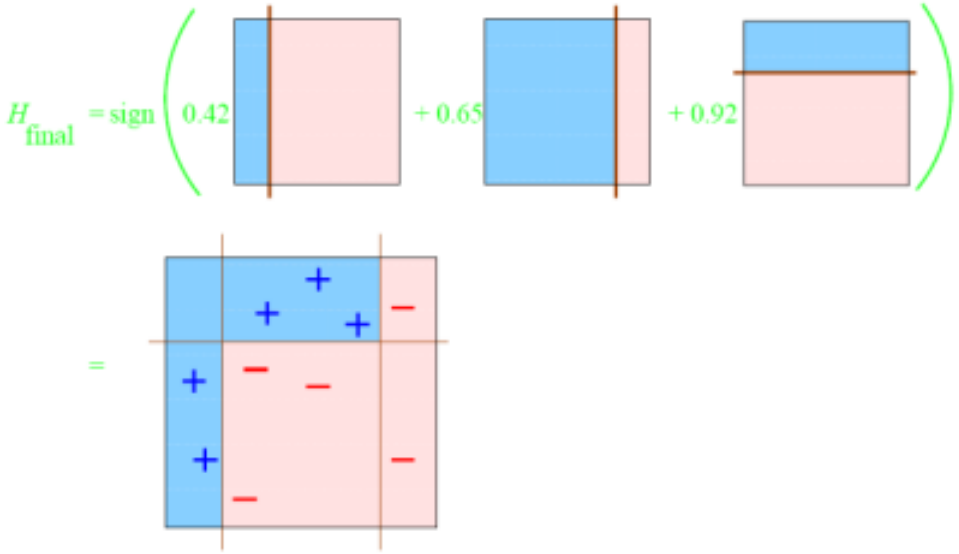
Introduction to Supervised Learning



Week 6:
Ensemble Methods
Adaboost, Random Forests

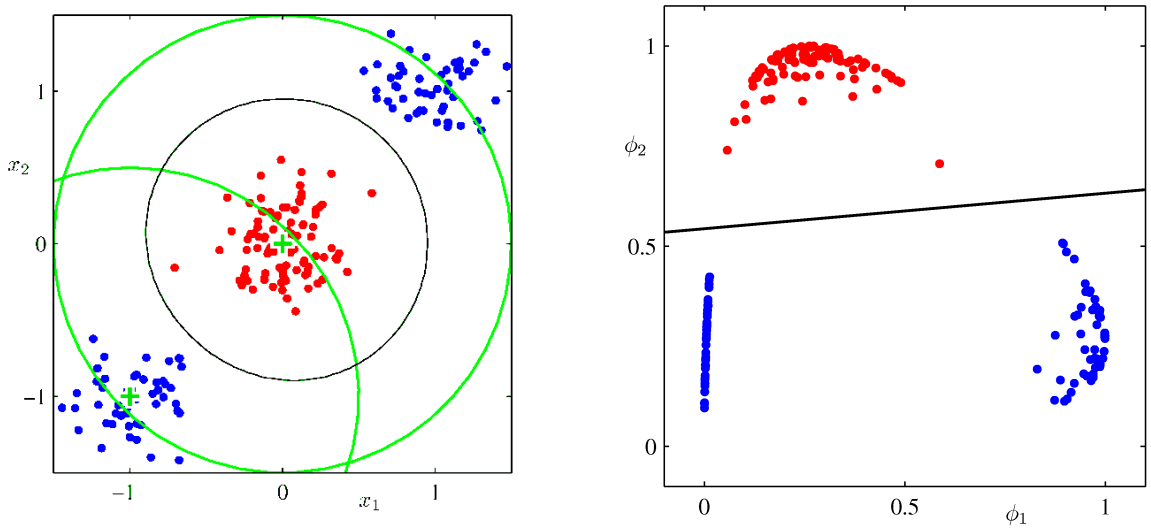
Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Beyond linear boundaries

- Straighftorward extension: More features

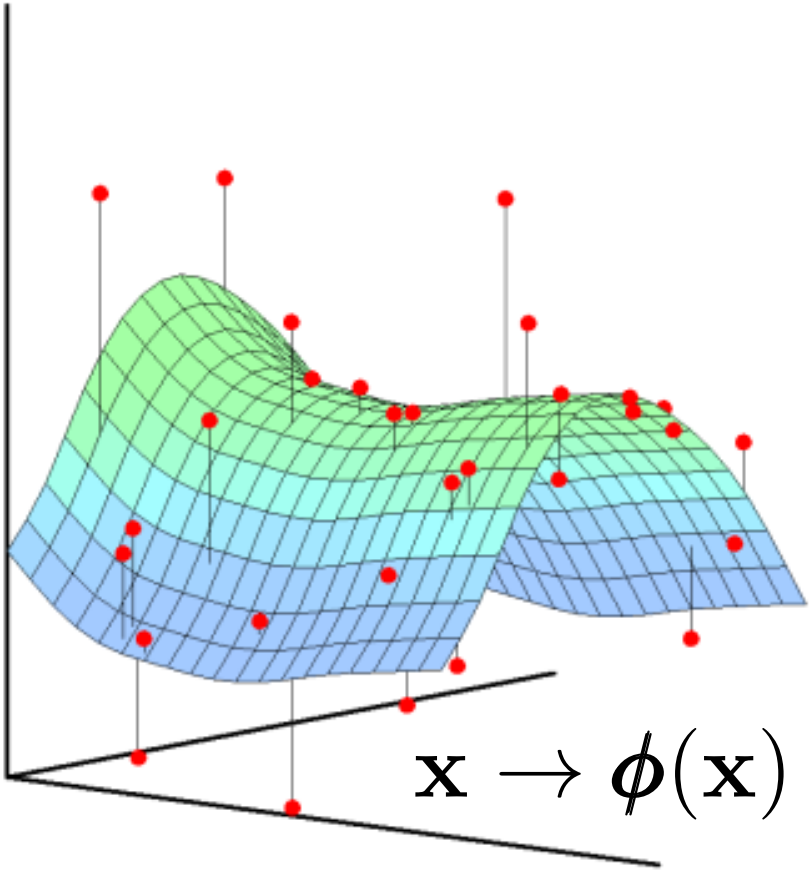


- Which features?
- How can we control complexity?

Small announcement

- Theory exercises: a few typos have been corrected (they appear in red)
- Due date: extended by 6 days – 2pm, 22 November
- Today's meeting hours: cancelled (more hours tomorrow/next week)

Weeks 2-3: nonlinear features



$$\mathbf{x} \to \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}$$

$$\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$

Weeks 4-5: SVMs & Kernel trick

Define Kernel: $K(\mathbf{x}, \mathbf{y}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle$

Classifier: $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha^i y^i K(\mathbf{x}^i, \mathbf{x}) + b$

Complexity of evaluation: O(#Support Vectors)

Lecture outline

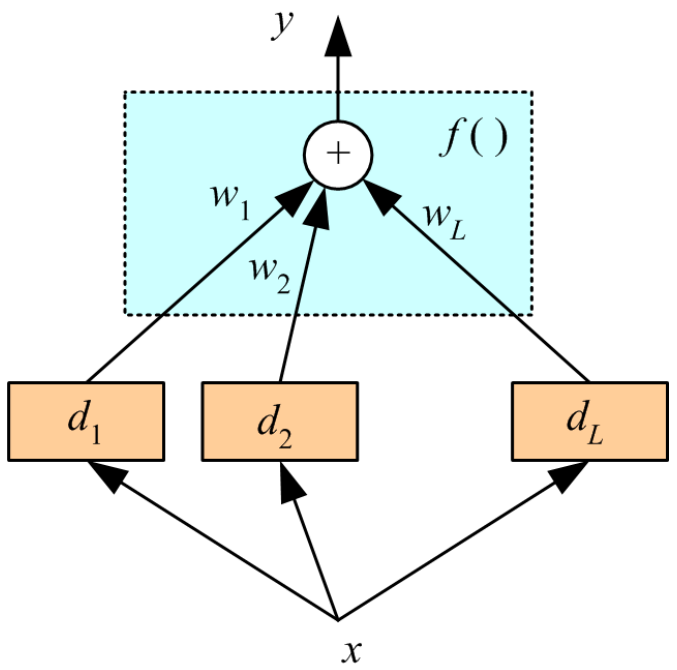
Adaboost

Decision Trees

Random Forests

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



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:**

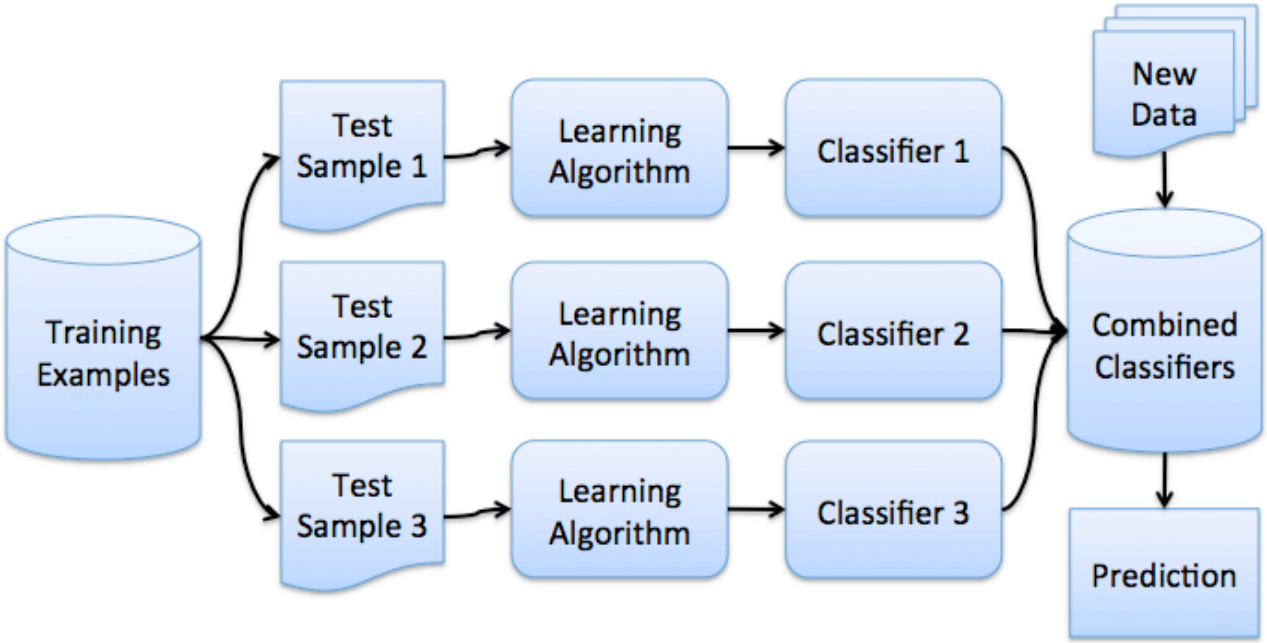
$$\mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\frac{1}{M^2}\sum_{m=1}^{M}\left\{\epsilon_m^2(\mathbf{x}) + \sum_{k\neq m}\epsilon_m(\mathbf{x})\epsilon_k(\mathbf{x})\right\}\right]$$

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

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

Boosting 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

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):

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

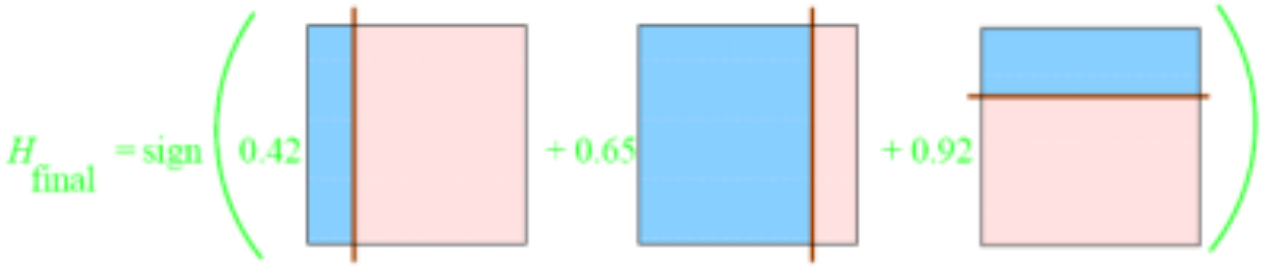
Strong classifier

Weak classifier

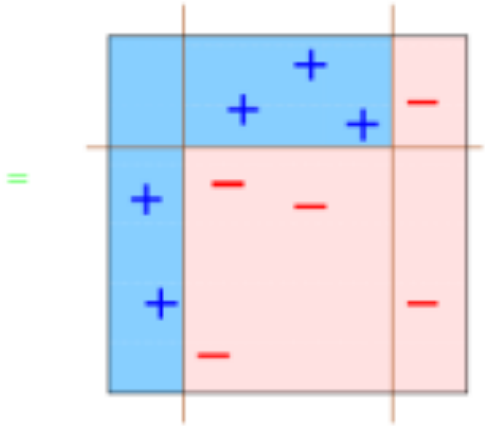
Weight

Feature vector

Adaboost, in pictures

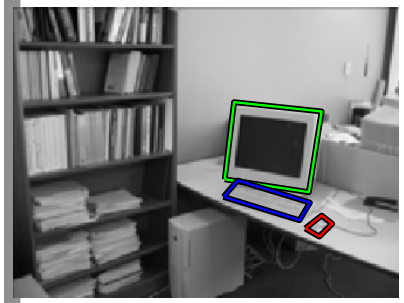


elementary functions

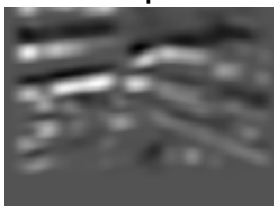


nonlinear decision boundary!

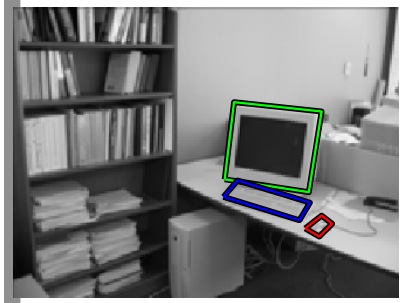
Example: screen detection



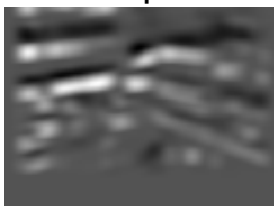
Feature output



Example: screen detection

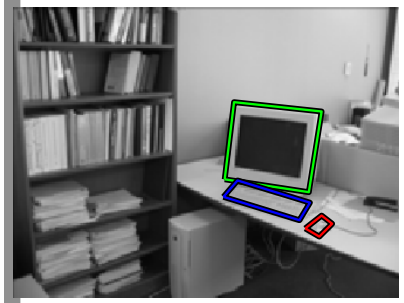


Feature output

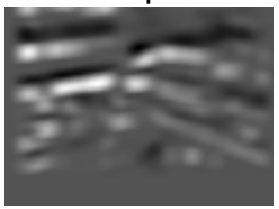


Thresholded output

Example: screen detection



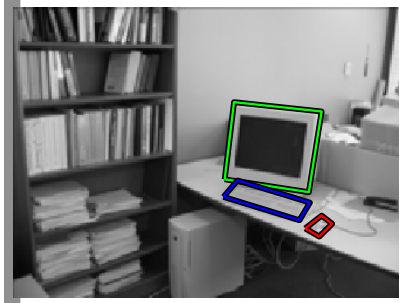
Feature output



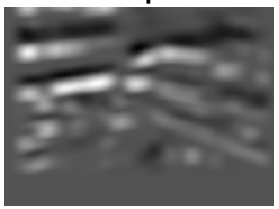
Thresholded output

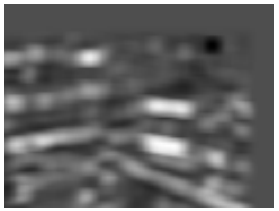
Strong classifier at iteration 1

Example: screen detection



Feature output

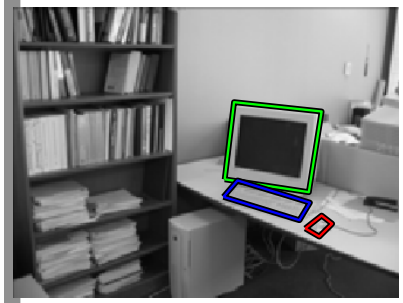




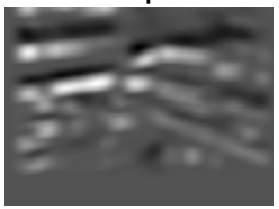
Thresholded output

Strong classifier

Example: screen detection

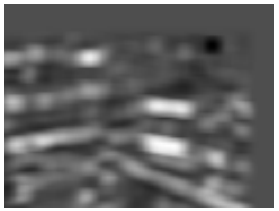


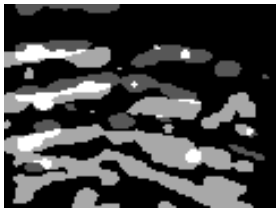
Feature output



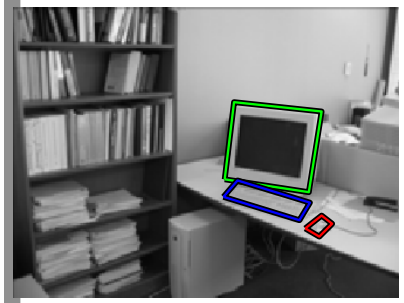
Thresholded output

Strong classifier

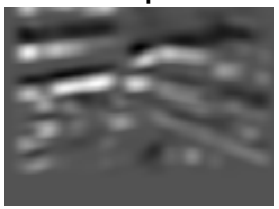


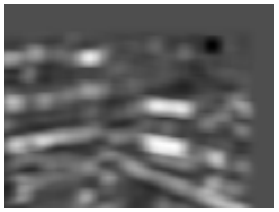


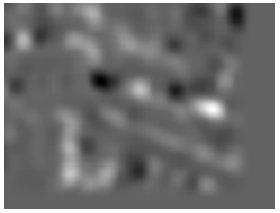
Example: screen detection



Feature output

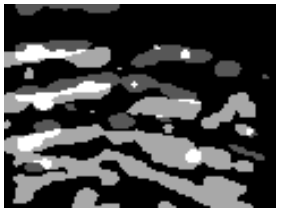


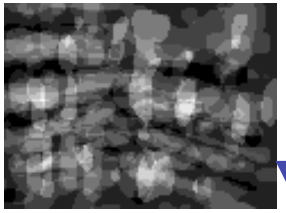




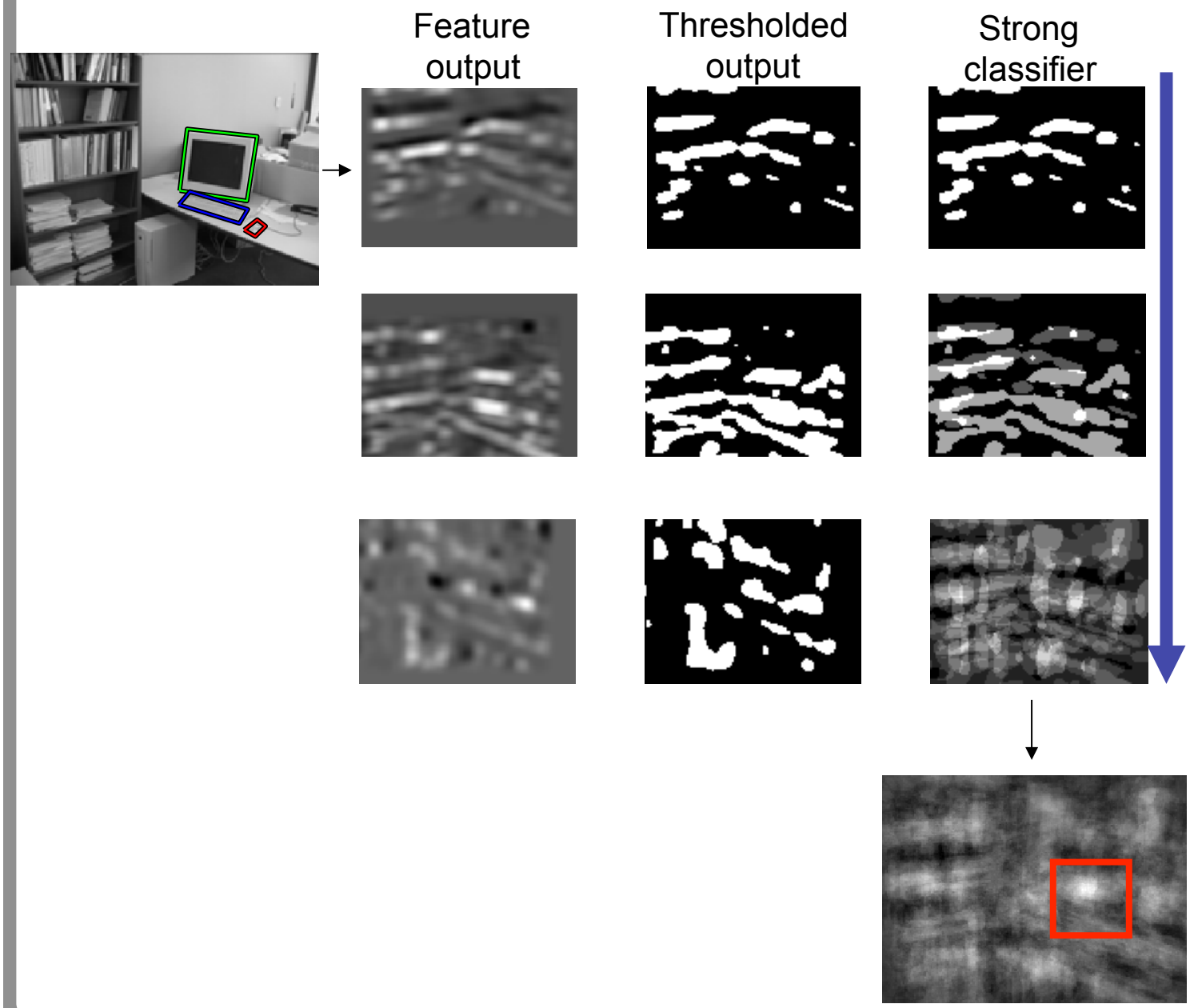
Thresholded output

Strong classifier



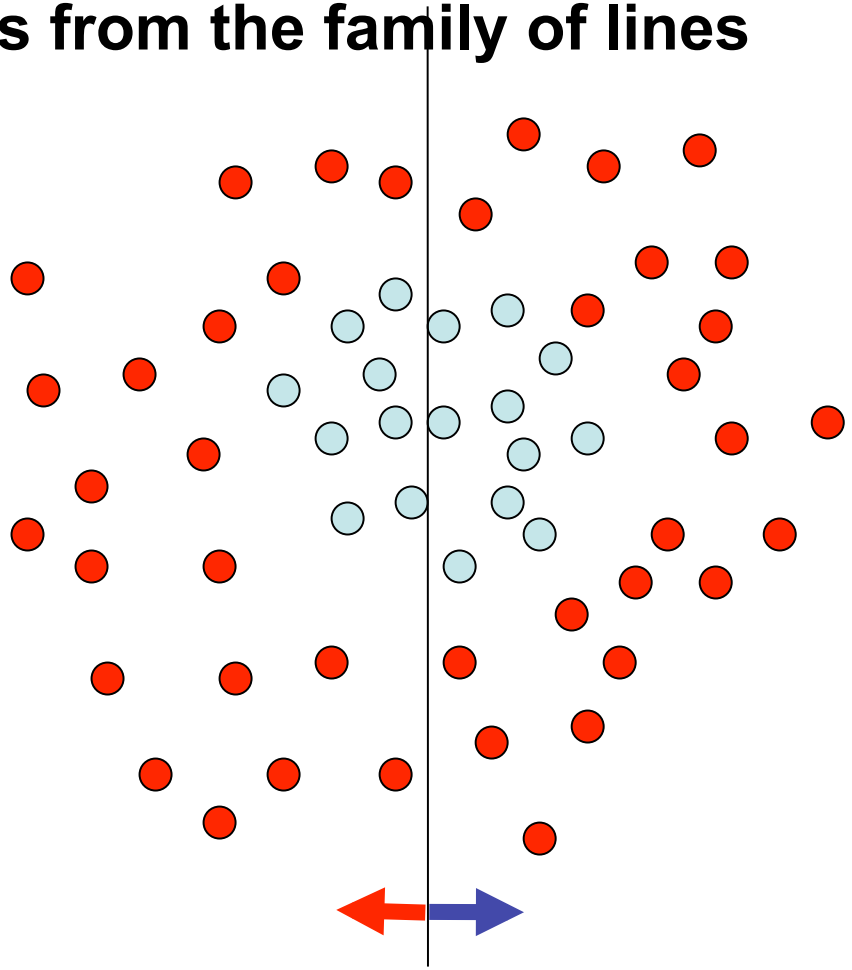


Example: screen detection



Toy example

Weak learners from the family of lines



Each data point has a class label:

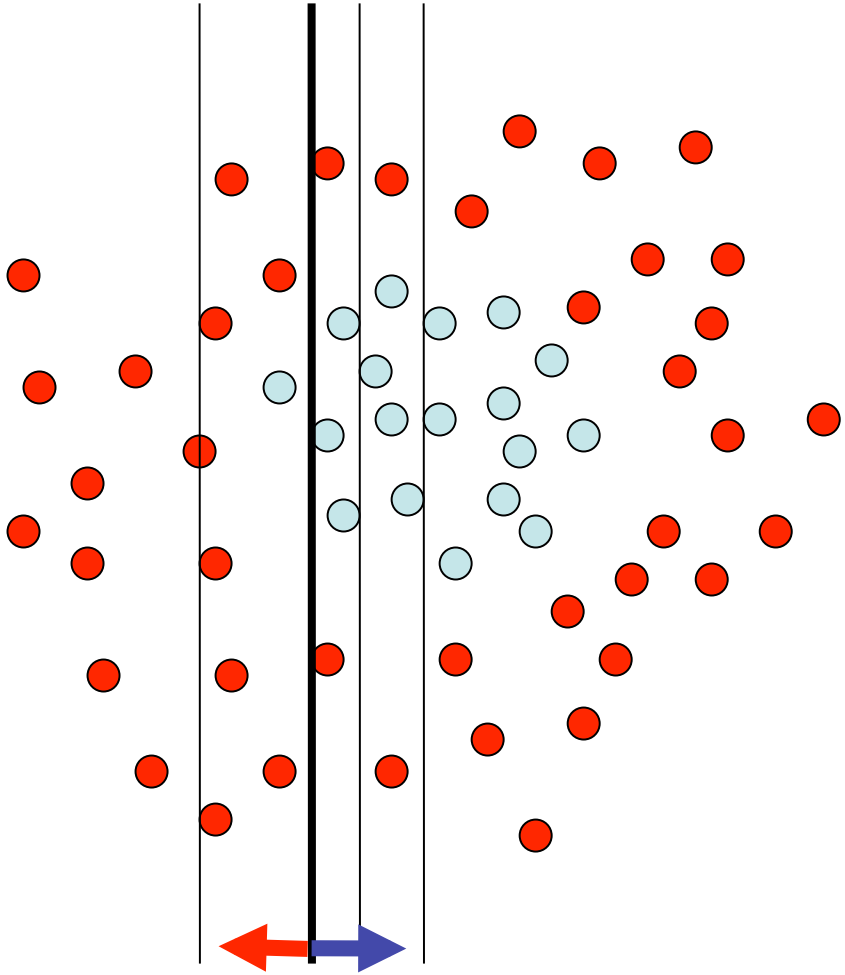
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

and a weight:

$$w_t = 1$$

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

Toy example



Each data point has a class label:

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

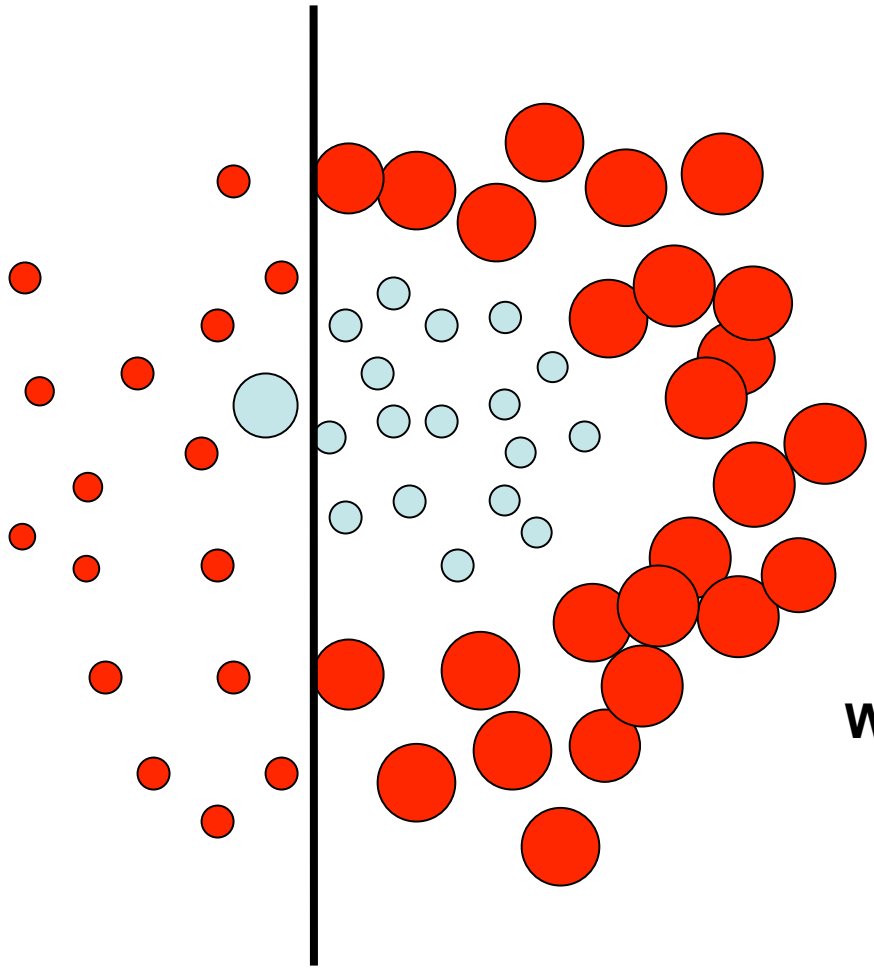
and a weight:

$$w_t = 1$$

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:

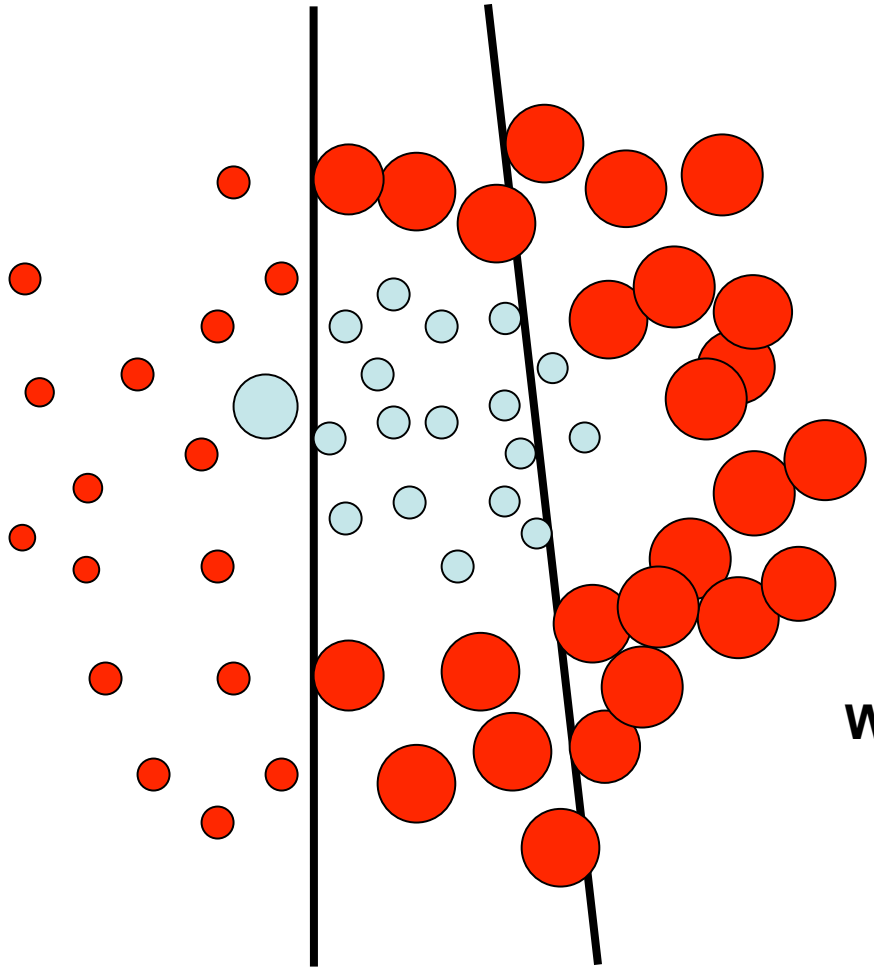
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

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:

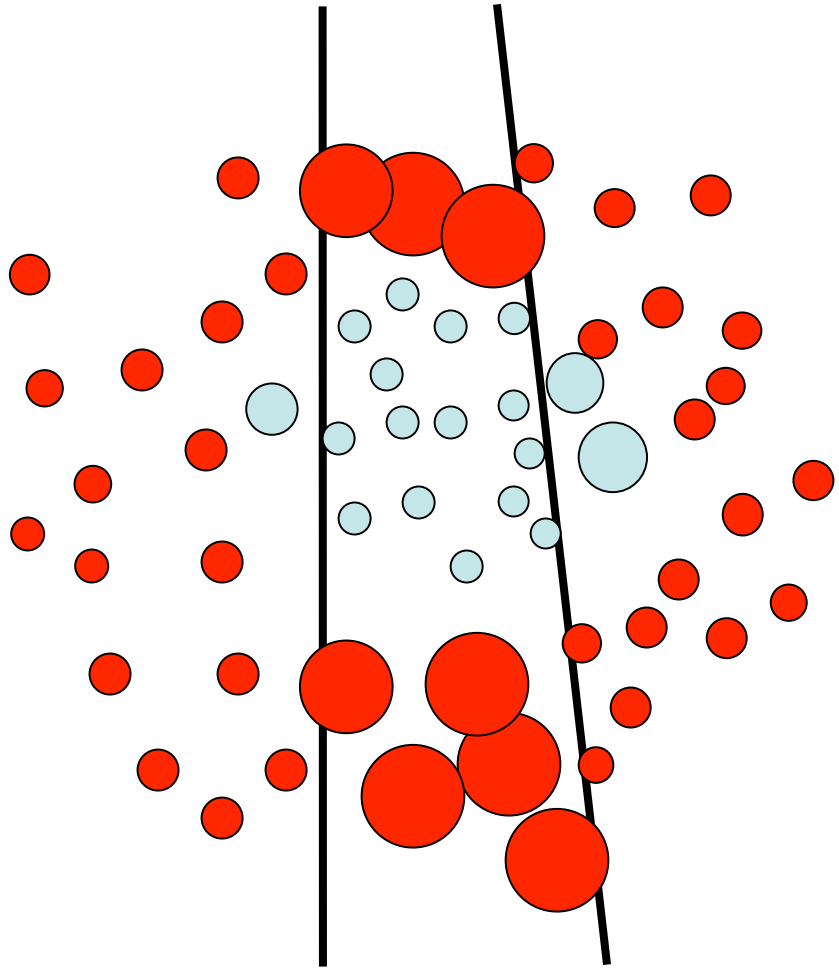
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

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:

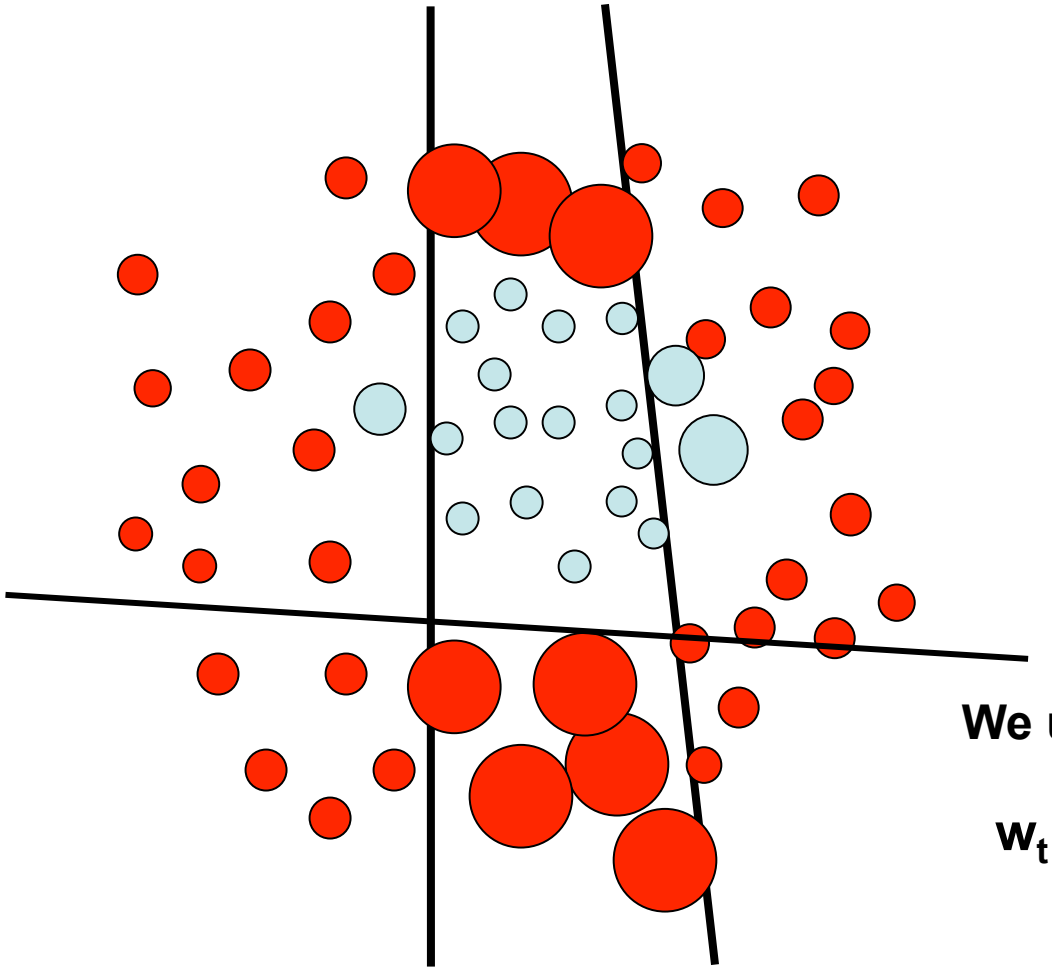
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

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:

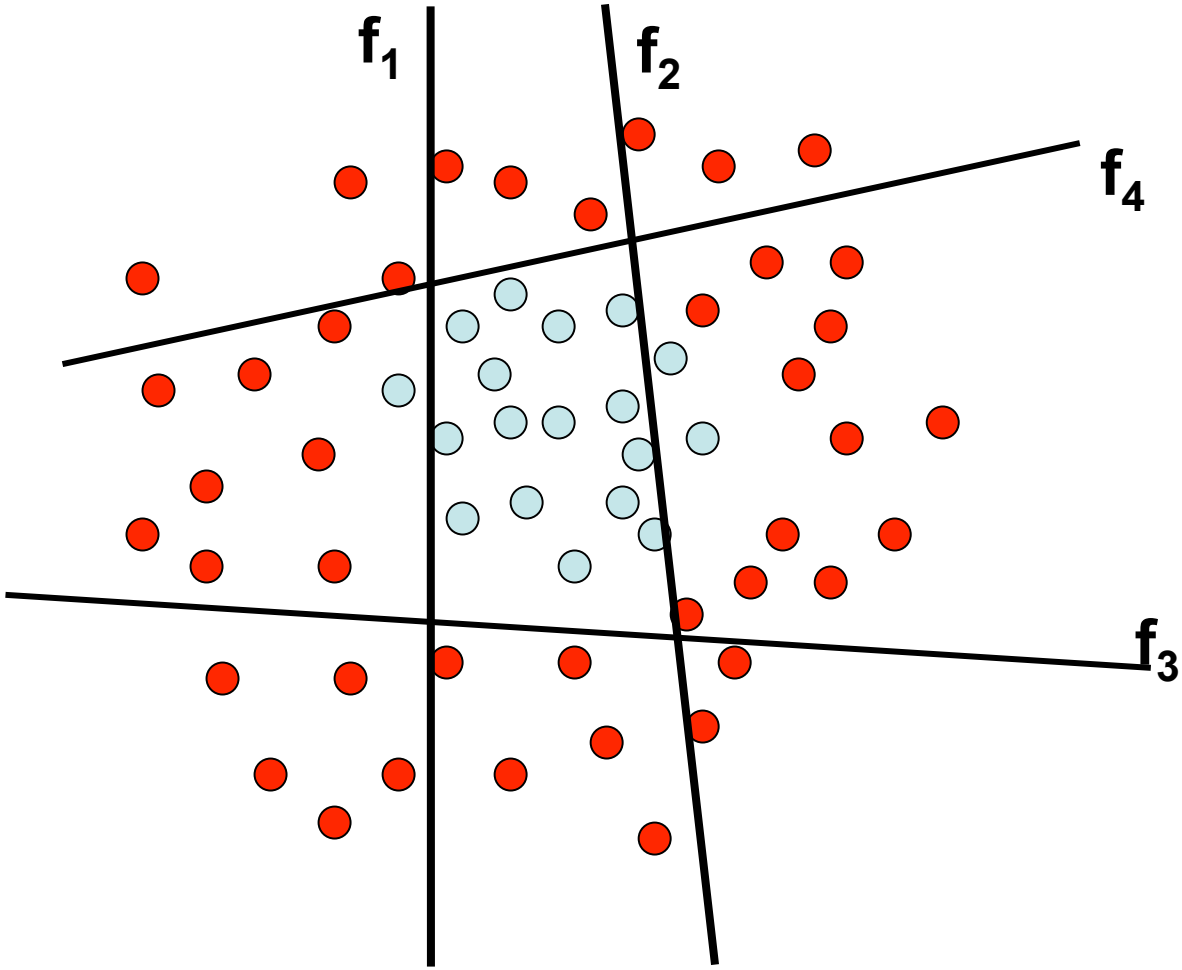
$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\circ) \end{cases}$$

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

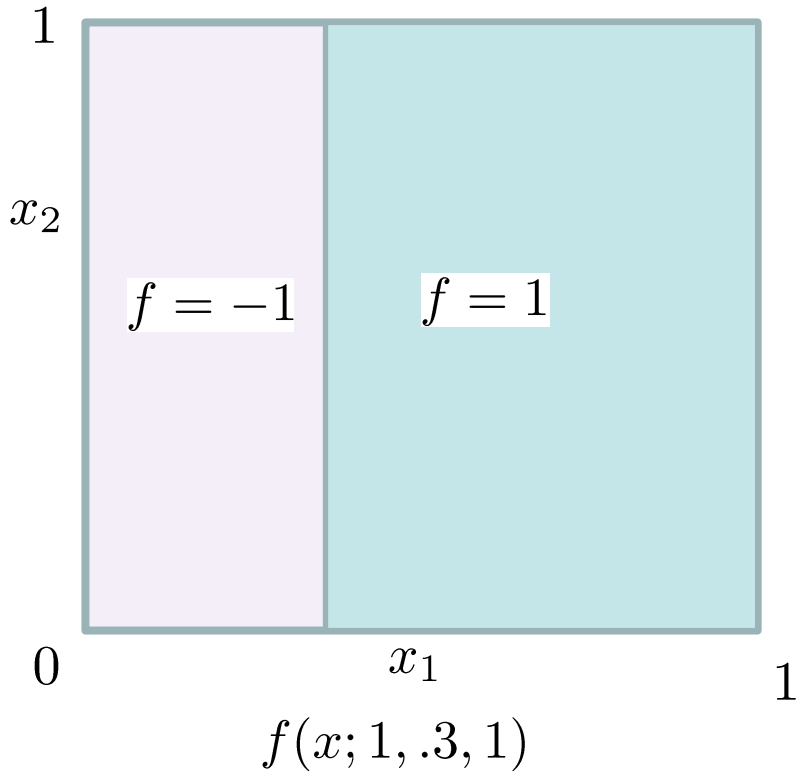
Toy example

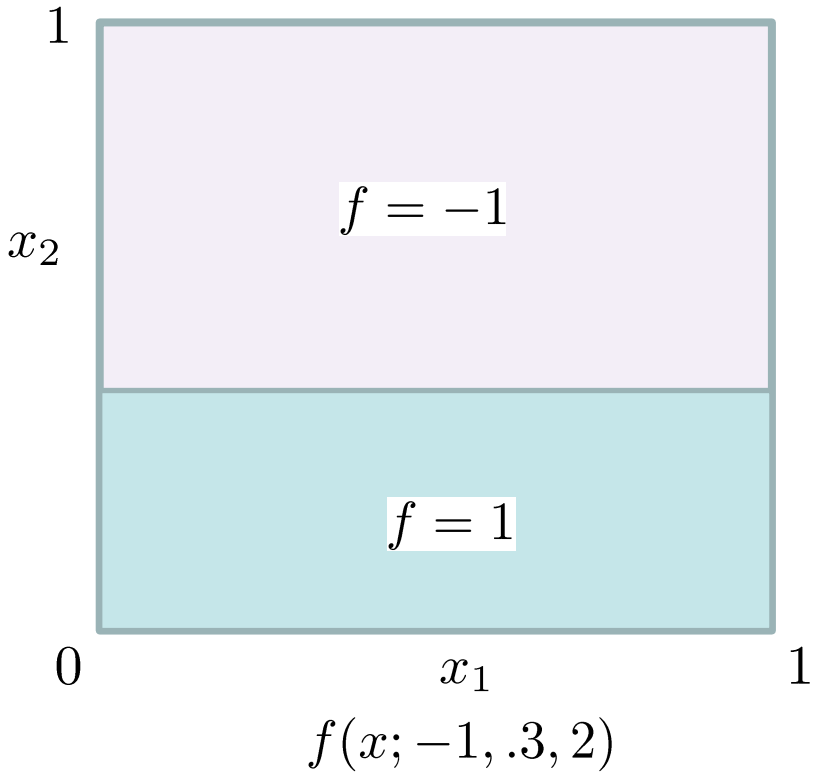


The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

Weak Learner Example

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





  - 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

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

$$a_t = \frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t} \qquad = \frac{D_t^i}{Z_t} \exp(-\alpha_t y^i h_t(x^i))$$

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

- Final classifier $H(x) = \text{sign}\left(\sum_t a_t h_t(x)\right)$

When and why does this work?

- Introduce `edge': $\epsilon_t = \frac{1}{2} - \underbrace{\gamma_t}_{\text{Edge}}$
- Theorem:

$$\underbrace{\sum_{i=1}^{N} \frac{1}{N}[y^i \neq H(x^i)]}_{\text{Final Training Error}} \leq \prod_{t=1}^{T} \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right]$$

$$= \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2}$$

$$\leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

- Assumption $\gamma_t \geq \gamma > 0, \quad \forall t$

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

Dissecting Adaboost – step I

- Given: $(x^i, y^i), x^i \in \mathcal{X}, y^i \in \{-1, 1\}, i = 1, \ldots, N$
- Initialize: $D_1(i) = \frac{1}{N}$
- 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

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

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

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

$$a_t = \frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t}$$

- Final classifier $H(x) = \text{sign}\left(\sum_t a_t h_t(x)\right)$

Step I: Relate weights with vote

- Recursion for D $D^i_{t+1} = \frac{D^i_t}{Z_t}\exp(-\alpha_t y^i h_t(x^i))$

$$= \frac{\frac{D^i_{t-1}}{Z_{t-1}}\exp(-\alpha_{t-1} y^i h_{t-1}(x^i))}{Z_t}\exp(-\alpha_t y^i h_T(x^i))$$

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

$$= \frac{1}{N}\frac{\exp\left(-y^i \sum_{k=1}^t \alpha_k h_k(x^i)\right)}{\prod_{k=1}^t Z_k}$$

- So $D^i_{T+1} = \frac{\exp(-y^i f^T(x^i))}{N\prod_{k=1}^T Z_k}$

$$f^T(x) \doteq \sum_{t=1}^T a_t h_t(x)$$

Step II: Bound training error in terms of Z

- Lower bound for error

$$\sum_{i=1}^{N} \frac{1}{N}[y^i \neq H(x^i)] \overset{H(x)=\text{sign}f(x)}{=} \frac{1}{N}\sum_i \begin{cases} 1 & y^i f(x^i) \leq 0 \\ 0 & y_i f(x_i) > 0 \end{cases}$$

$$\leq \sum_{i=1}^{N} \frac{1}{N} \exp(-y^i f(x^i))$$

$$= \sum_{i=1}^{N} D_{T+1}^i \prod_{k=1}^{T} Z_k$$

$$= 1 \prod_{k=1}^{T} Z_k$$

Dissecting Adaboost – step II

- Given: $(x^i, y^i), x^i \in \mathcal{X}, y^i \in \{-1, 1\}, i = 1, \ldots, N$
- Initialize: $D_1(i) = \frac{1}{N}$
- 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

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

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

$$a_t = \frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t}$$

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

- Final classifier $H(x) = \text{sign}\left(\sum_t a_t h_t(x)\right)$

Step III: Relate Z with ε

- We set out to show

$$\sum_{i=1}^{N} \frac{1}{N}[y^i \neq H(x^i)] \leq \prod_{t=1}^{T} \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right]$$

- So far we have shown

$$\sum_{i=1}^{N} \frac{1}{N}[y^i \neq H(x^i)] \leq \prod_{k=1}^{T} Z_k$$

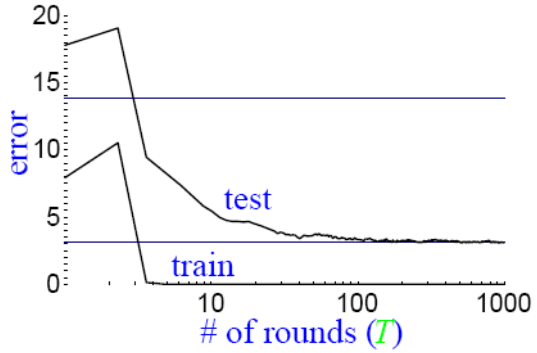
- But

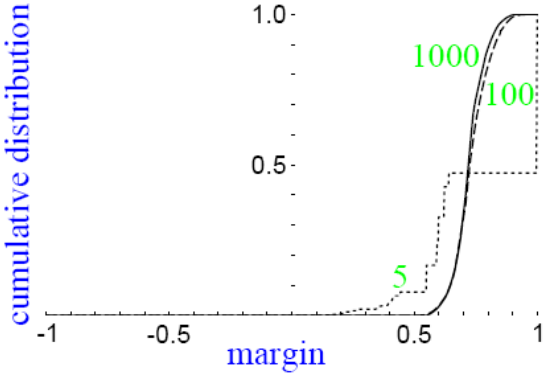
$$\begin{aligned}
Z_t &= \sum_{i=1}^{N} D_t^i \exp(-\alpha_t y^i h_t(x^i)) \\
&= \sum_{i:y^i \neq h_t(x^i)} D_t^i \exp(-\alpha_t y^i h_t(x^i)) + \sum_{i:y^i = h_t(x^i)} D_t^i \exp(-\alpha_t y^i h_t(x^i)) \\
&= \sum_{i:y^i \neq h_t(x^i)} D_t^i \exp(\alpha_t) + \sum_{i:y^i = h_t(x^i)} D_t^i \exp(-\alpha_t) \\
&= (\epsilon_t)\exp(\alpha_t) + (1-\epsilon_t)\exp(-\alpha_t) \\
\overset{\alpha=\frac{1}{2}\log(\frac{1-\epsilon}{\epsilon})}{=} & \epsilon_t \frac{\sqrt{1-\epsilon_t}}{\sqrt{\epsilon_t}} + (1-\epsilon_t)\frac{\sqrt{\epsilon_t}}{\sqrt{1-\epsilon_t}} \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}$$

- Q.E.D.: Quite Easily Done

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 (next lecture)**

Optimization perspective of Adaboost

- Each round minimizes $\sum_{i=1}^{N} \exp(-y^i f(x^i))$ by fitting an additive model

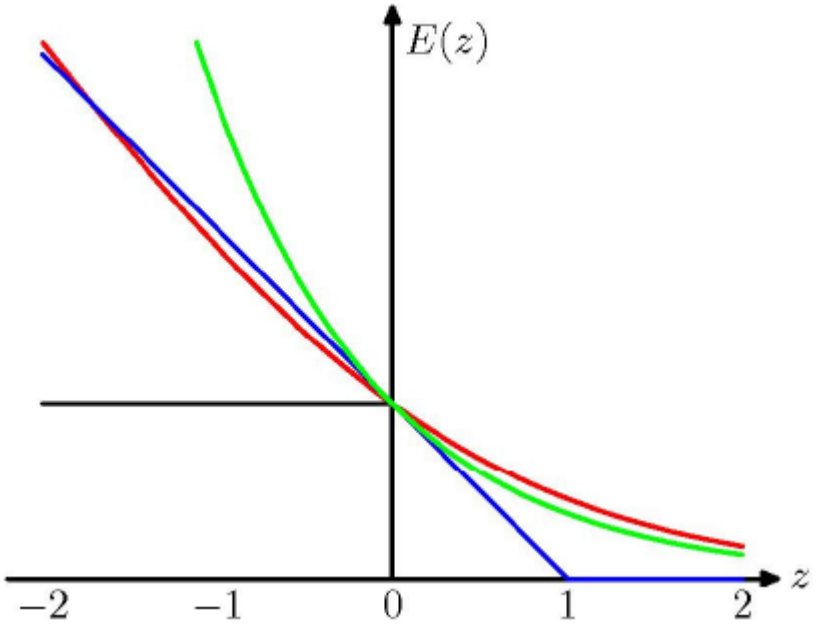
$$f'(x) = f(x) + \alpha h(x)$$

- Proof:

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

- Choice of h: minimize $\epsilon = \sum D^i y^i h(x^i)$
- Choice of a : minimize $\sum_{i=1}^{N} D^i \exp(-\alpha y^i h(x^i))$

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

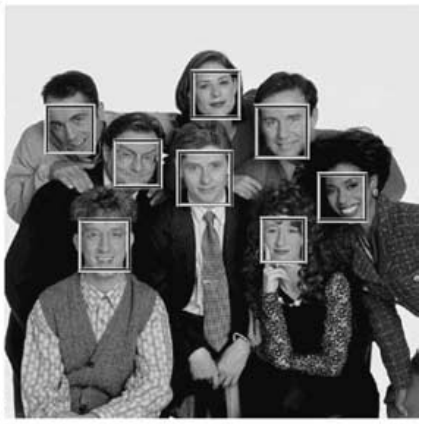
$$
\begin{aligned}
E_{X,Y}\exp(-yf(x)) &= \int_x \sum_y \exp(-yf(x))P(x,y)dxdy \\
&= \int_x \sum_y \exp(-yf(x))P(y|x)dyP(x)dx \\
&= \int_x \left[P(y=1|x)\exp(-f(x)) + P(y=-1|x)\exp(f(x))\right]dy\,dx
\end{aligned}
$$

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

- 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)



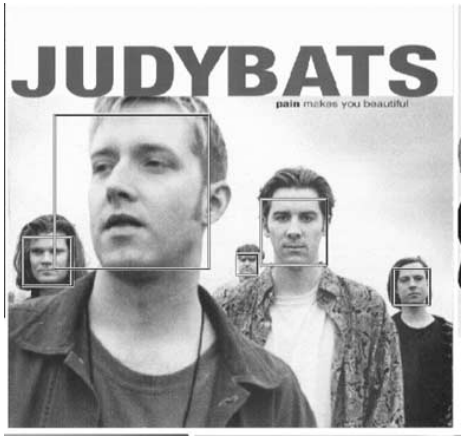
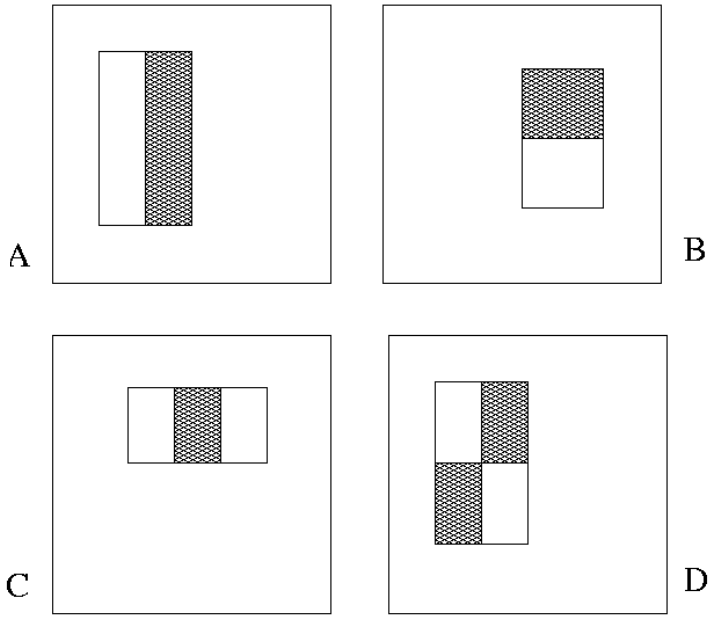


Image Features (Weak learners)

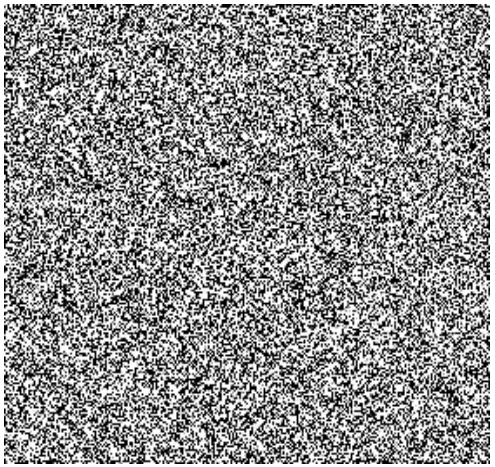
“Rectangle filters”



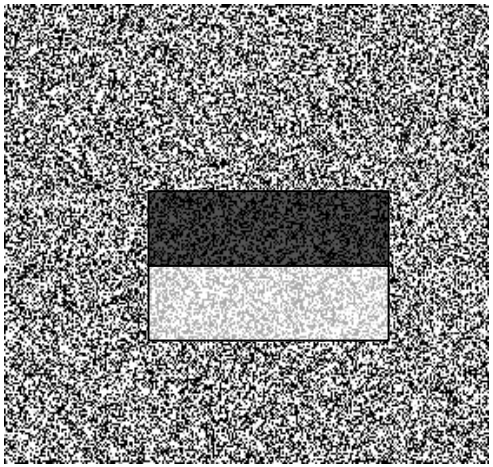
Value* = ∑ *(pixels in white area)* – ∑ *(pixels in black area)

Decision stump: Threshold difference
Why these features?
Extremely fast to compute (4 pixel operations per box)

Example



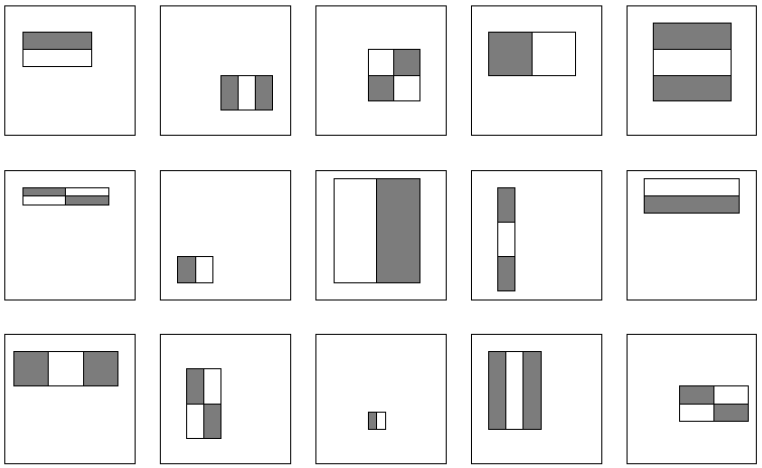
Source



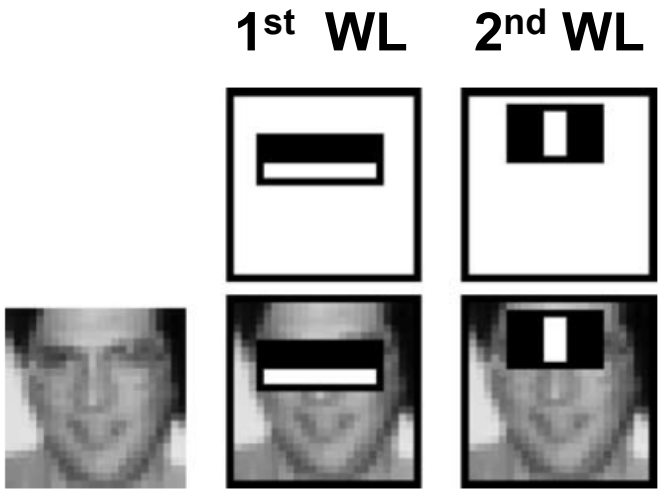
Result

Feature selection

- For a 24x24 detection region, the number of possible rectangle features is ~180,000!

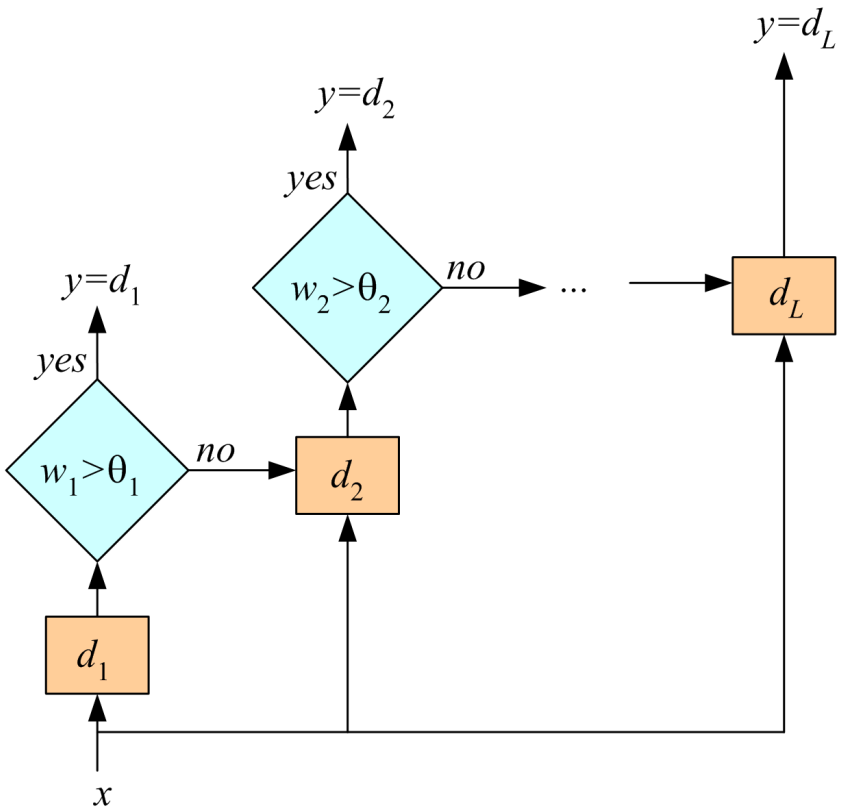


- Chosen automatically by Adaboost



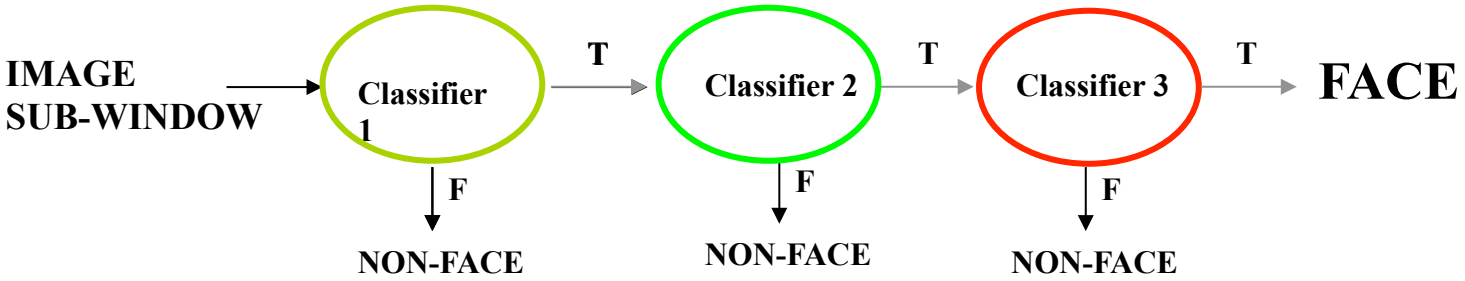
Viola & Jones: Cascade

- Reject large portions of the image based on thresholding first few weak learners



Speedup tricks

- Attentional cascade



- Integral images (4 pixel operations per filter)
- C++ implementation available in OpenCV [Lienhart, 2002]
  - http://sourceforge.net/projects/opencvlibrary/
- Matlab wrappers for OpenCV code available, e.g. here
  - http://www.mathworks.com/matlabcentral/fileexchange/19912

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

The implemented system

- Training Data
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 300 million non-faces
    - 9500 non-face images
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose

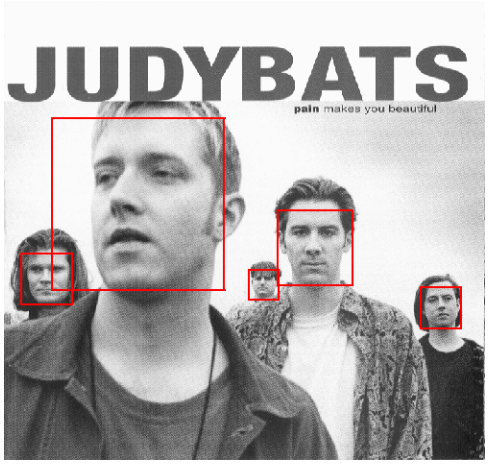
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. CVPR 2001.

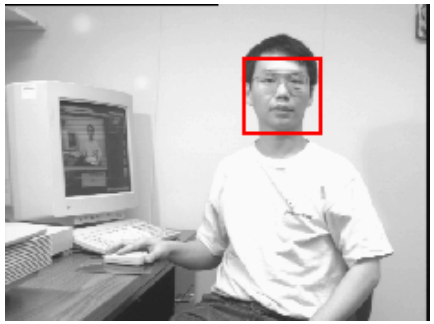
System performance

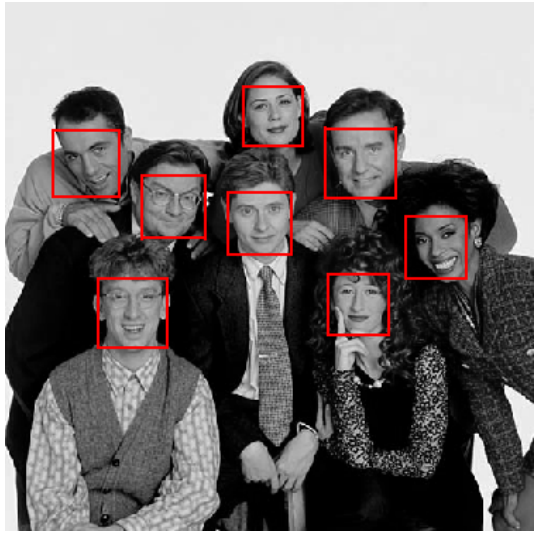
- Training time: “weeks” on 466 MHz Sun workstation
- 38 layers, total of 6061 features
- Average of 10 features evaluated per window on test set
- “On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds”
  - 15 Hz
  - 15 times faster than previous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

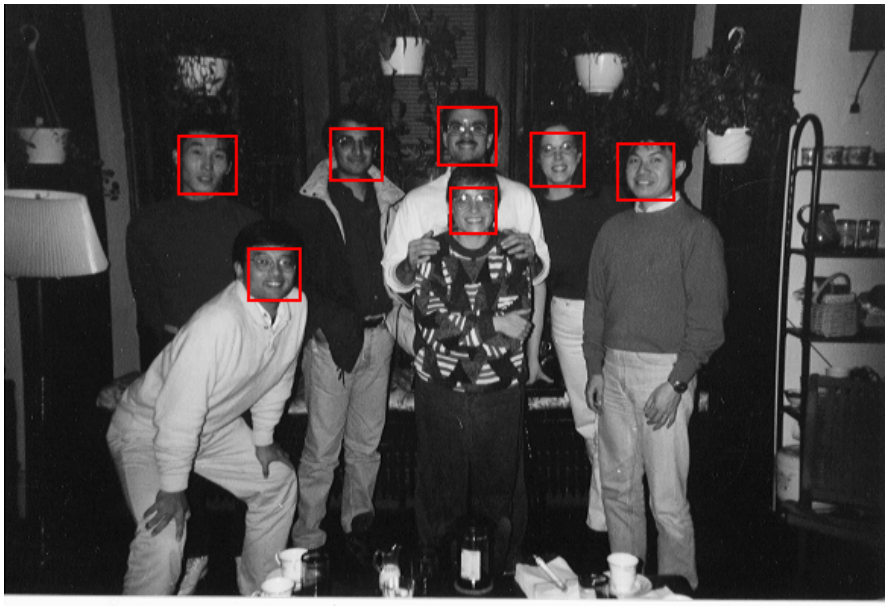
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. CVPR 2001.

Detection results (2001)










MPEGMOVIE VX
W
T
MENU
DISP
HOME
SONY

Lecture outline

Recap

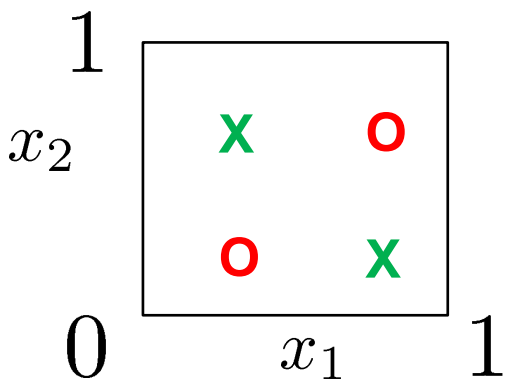
Adaboost

Decision Trees

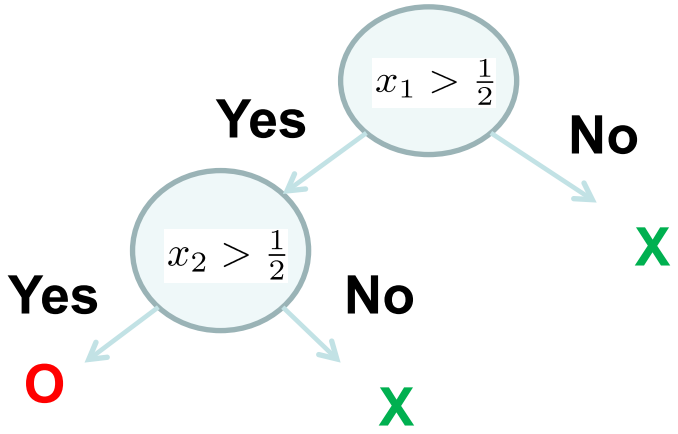
Random Forests & Ferns

Will AdaBoost always work?

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

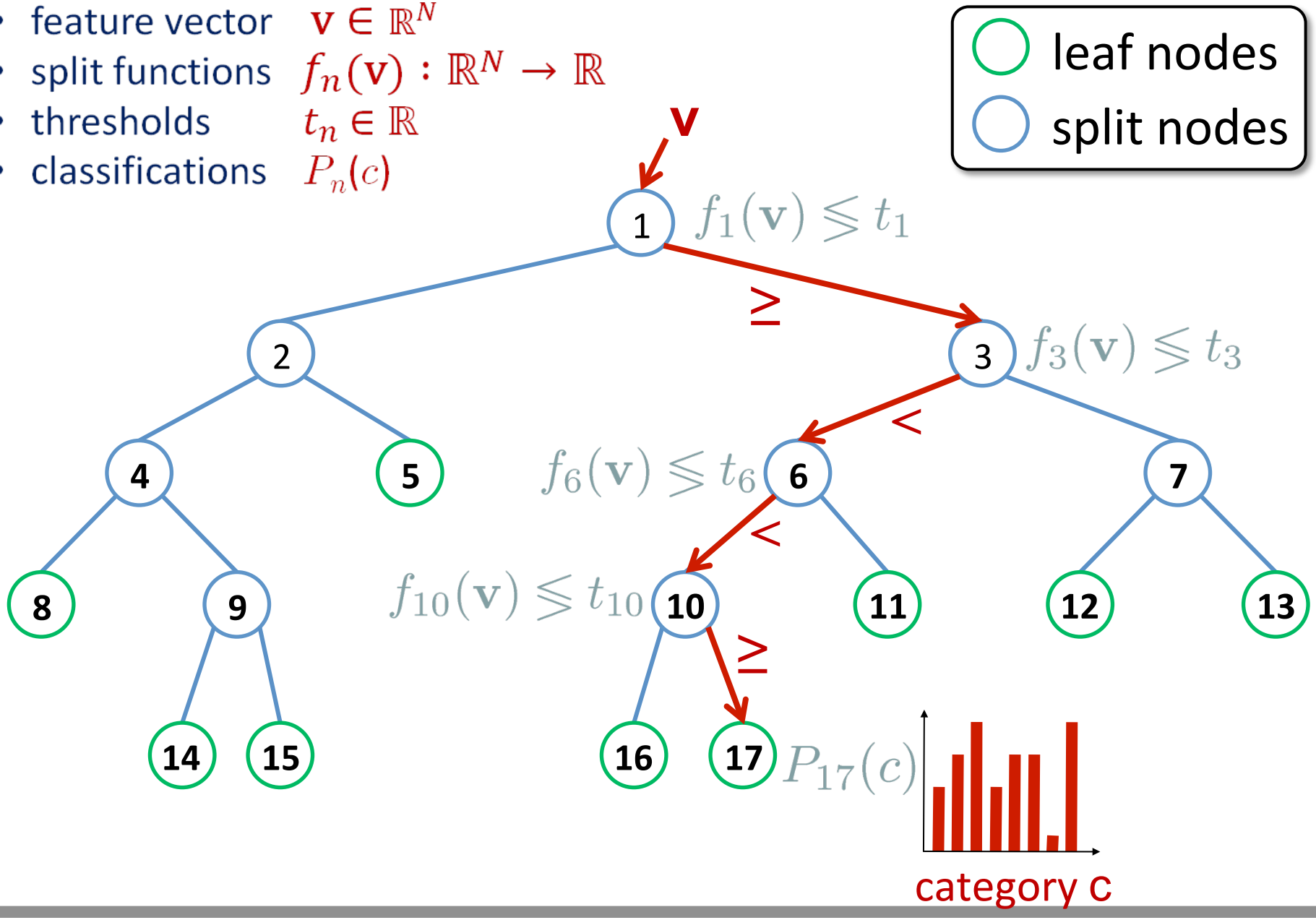


- In two steps: error 1/4

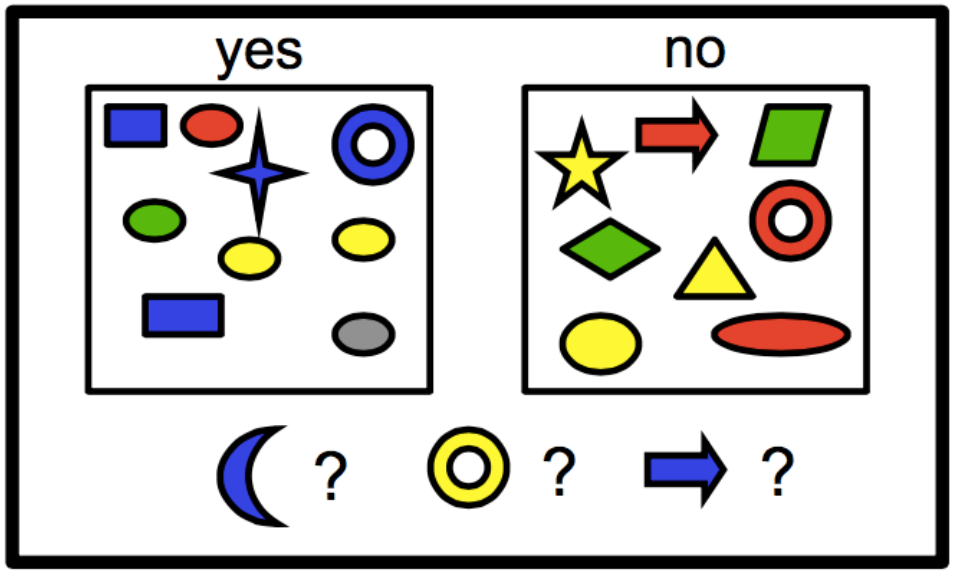


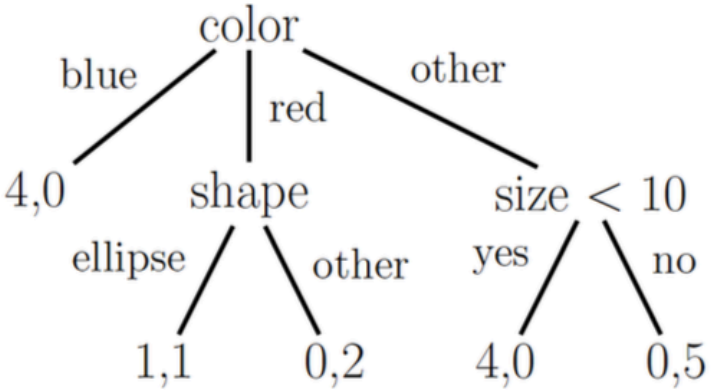
Binary Decision Trees

- 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

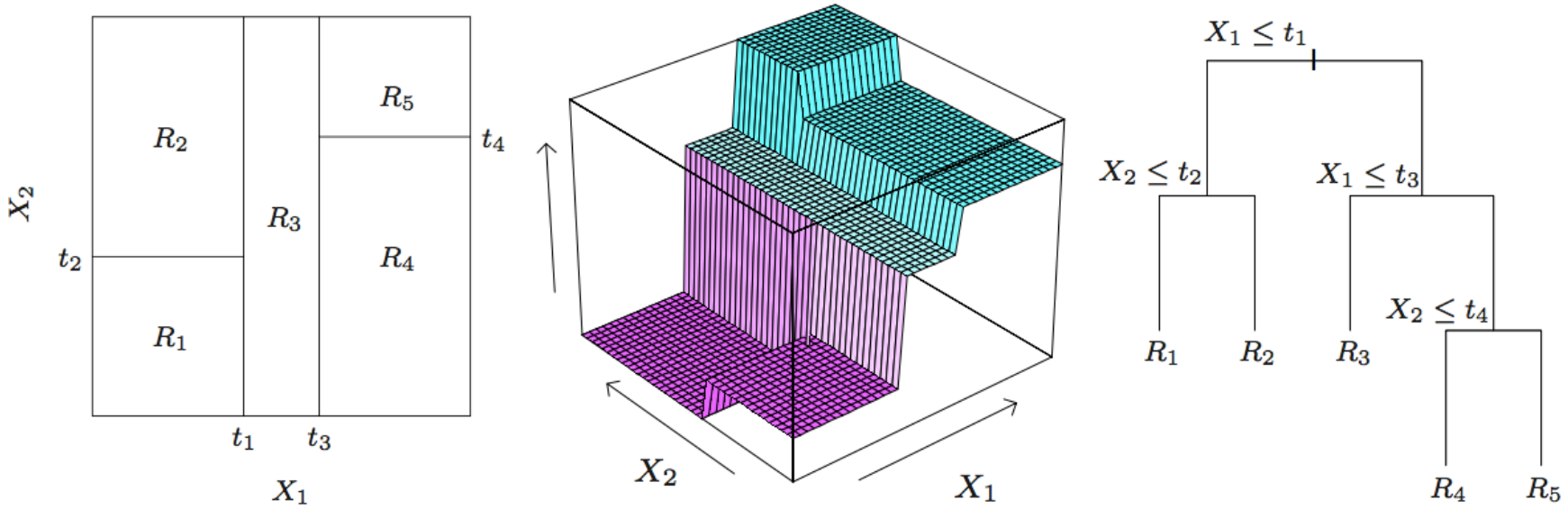




Features: color, shape, size

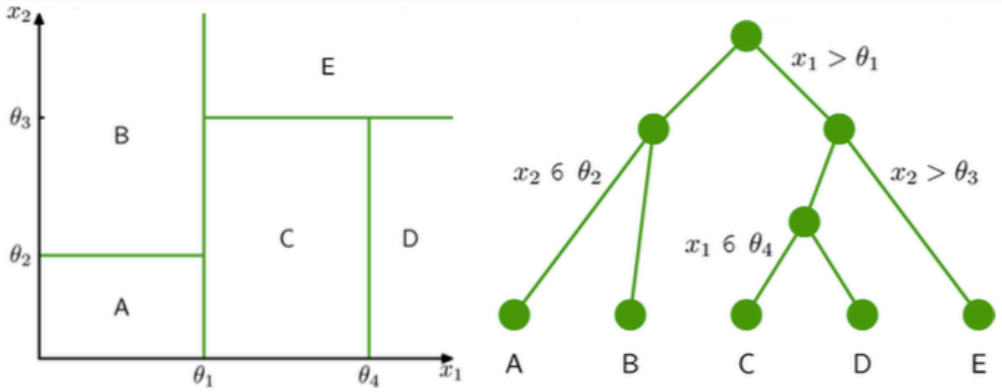
- Each node in tree splits examples according to a single feature
- Leaves have categorical distribution learned from labels of all training data whose path through tree ends there

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:

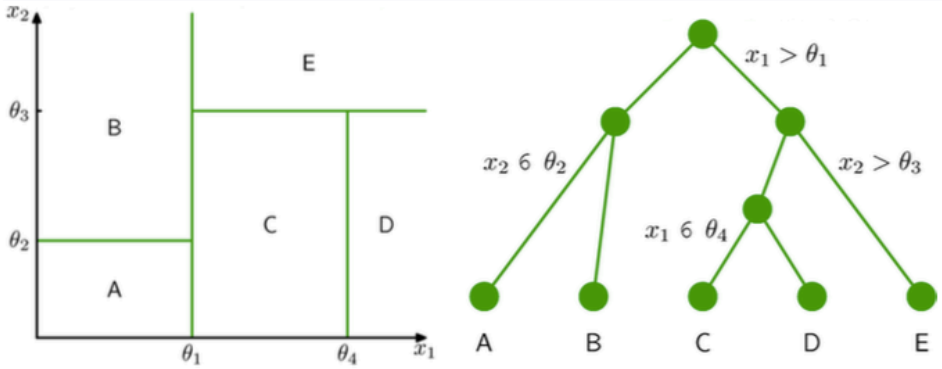
$$p(t_n \mid x_n \in R_j) = \mathrm{Ber}(t_n \mid \mu_j)$$

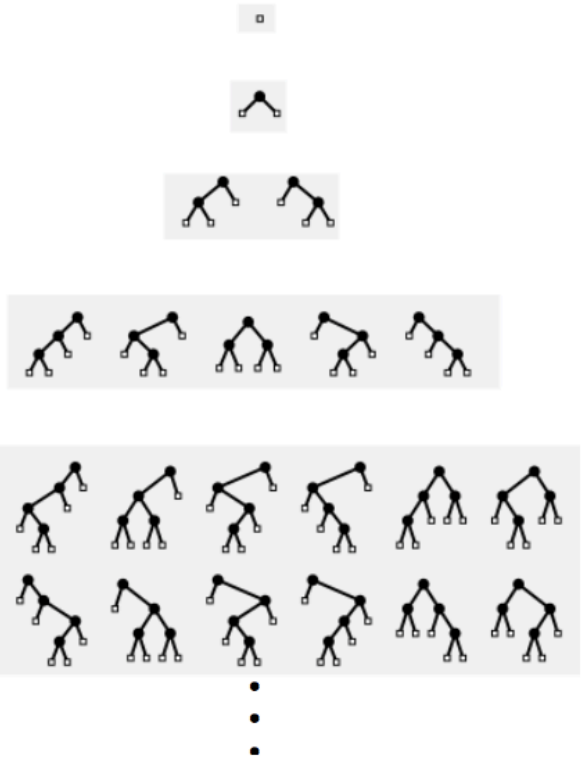
- Let *w* denote decision tree parameters: structure, splits, leaf distributions
- Conceptually, we would like to minimize the following objective:

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

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

Decision Tree Learning

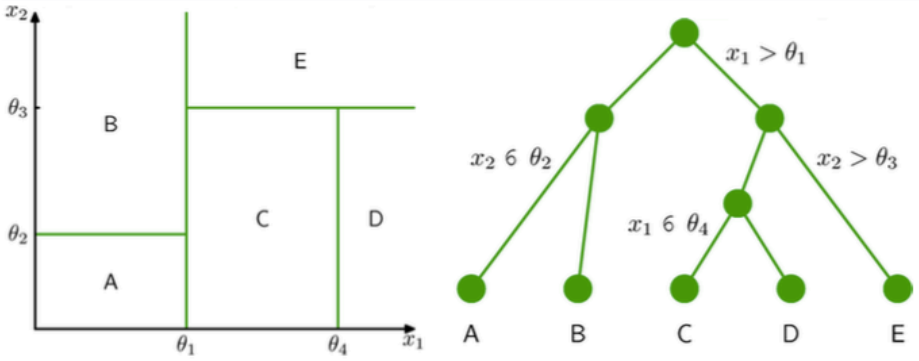




- **PROBLEM:** There are an enormous number of tree structures & split thresholds
- Optimizing leaf distributions given structure is easy, but searching structures is hard
- Classic approach: Build tree greedily, using various heuristics to control complexity, and check with validation data

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

Greedy 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:

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

N_j = number of training data for leaf j

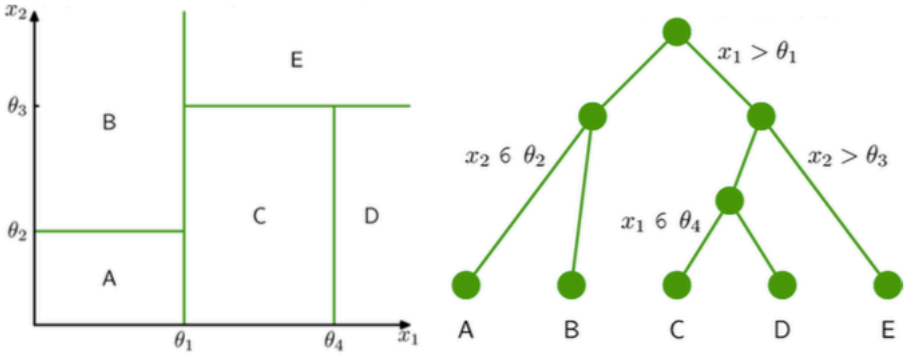
- Given a candidate tree, estimate leaf parameters via maximum likelihood:

Regression: $\mu_j = \frac{1}{N_j} \sum_{n \mid x_n \in R_j} t_n \qquad \sigma_j^2 = \frac{1}{N_j} \sum_{n \mid x_n \in R_j} (t_n - \mu_j)^2$

Classification: $\mu_j = \frac{1}{N_j} \sum_{n \mid x_n \in R_j} t_n \qquad \text{Var}(t_n \mid x_n \in R_j) = \mu_j(1 - \mu_j)$

- Greedily, pick the leaf with largest variance, choose new split that minimizes sum of child variances (solvable via exhaustive search)
- Recurse to refine tree. Initialize with all data in single root node.

Controlling Decision Tree Complexity



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:

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

- **PROBLEM:** Once number of leaves in tree equals N, can set variance in each leaf to zero. This perfectly predicts each training data, but overfits!
- **Pruning:** Recursively prune leaves from tree, use validation data to check whether increases or decreases prediction accuracy
- **Random forests:** Randomly subsample data and features. Greedily fit a decision tree to each, then average their predictions. This is a form of the bootstrap that can lead to enormous gains in prediction accuracy.

Binary Decision Trees Summary

- Fast greedy training algorithms
  - can search infinite pool of features
  - heterogeneous pool of features
- 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

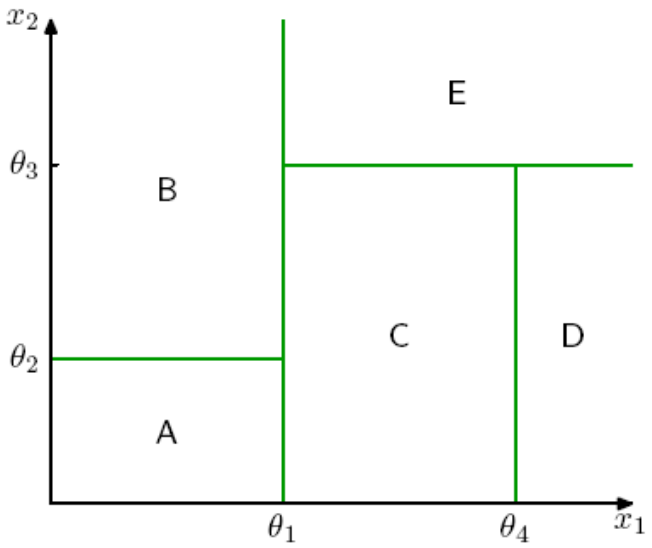
Adaboost

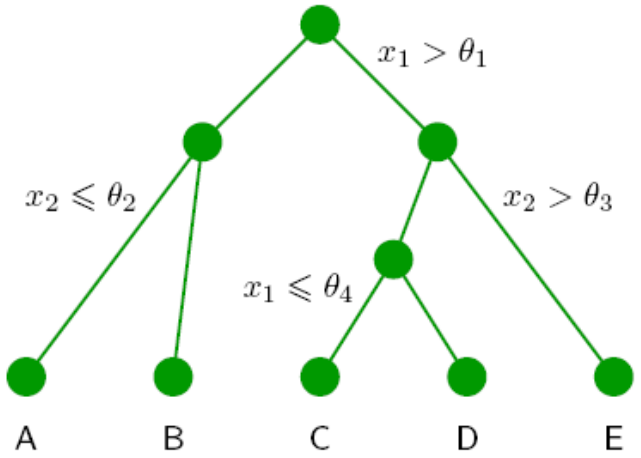
Decision Trees

Random Forests & Ferns

Which level of classification tree?

- Deeper trees result in more complex classifiers



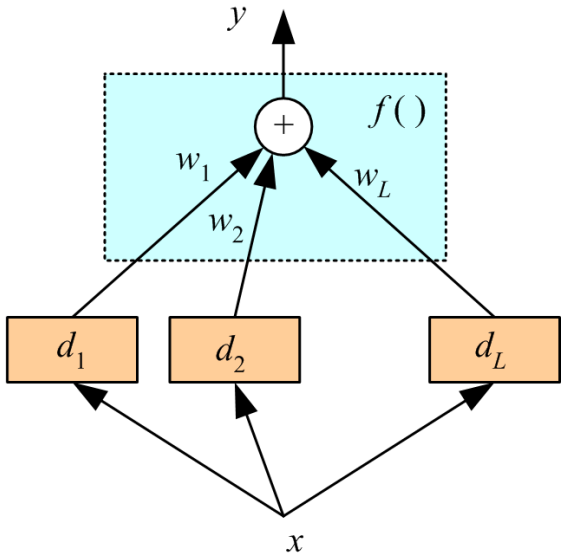


- After some point overfitting occurs

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

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



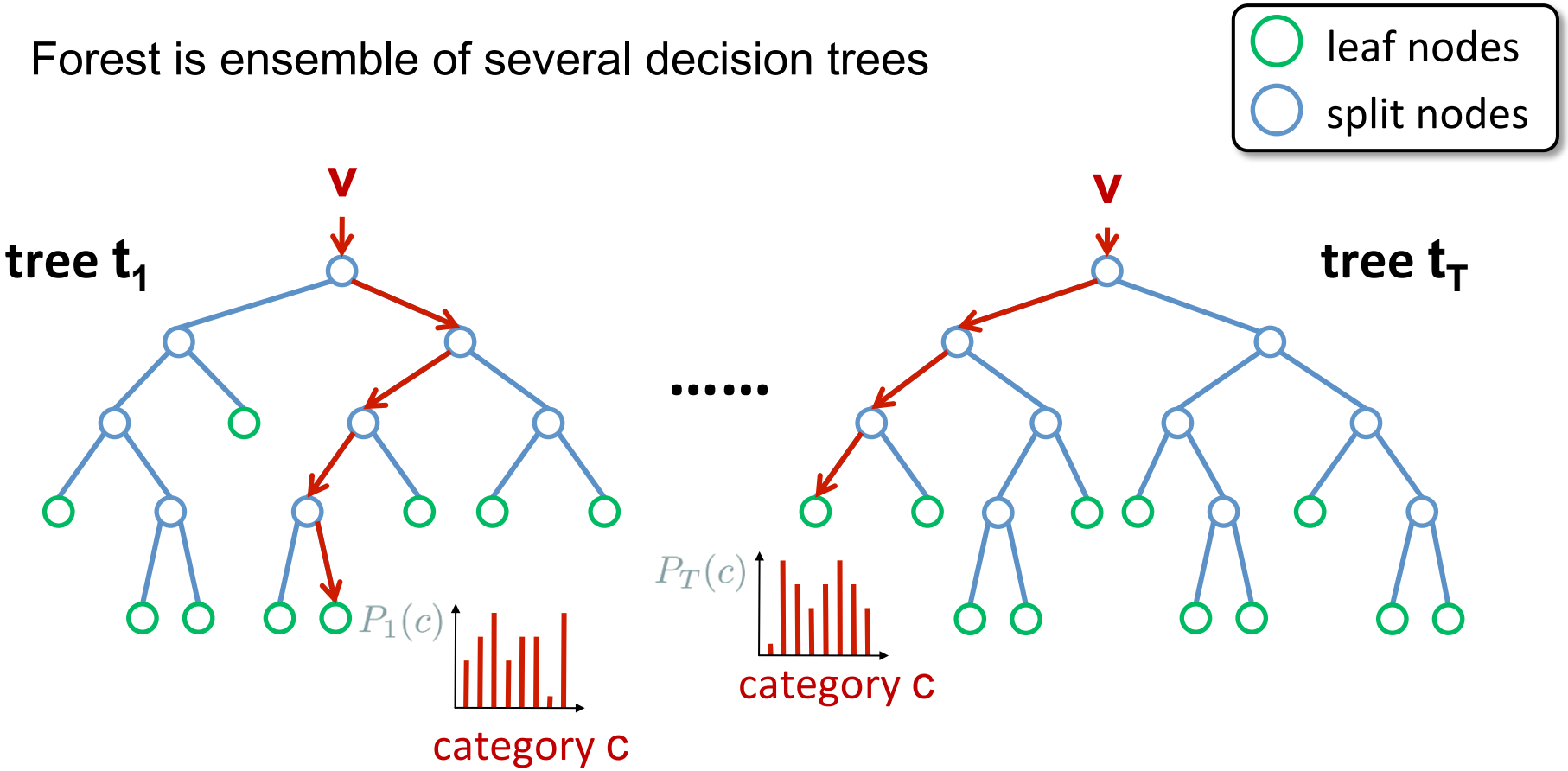
If errors have zero mean and are uncorrelated: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] = 0$

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

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

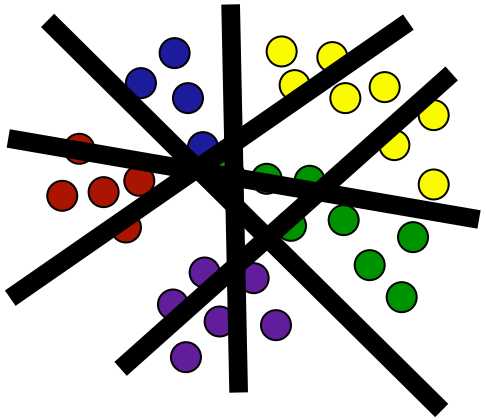
$$\begin{aligned} \text{left split } I_{\mathrm{l}} &= \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \text{right split } I_{\mathrm{r}} &= I_n \setminus I_{\mathrm{l}} \end{aligned}$$

- Features f(v) chosen 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 purity criterion
- Design choices: how many thresholds? Which features? When should we stop growing?

Why do random tests work?

For a small number of classes

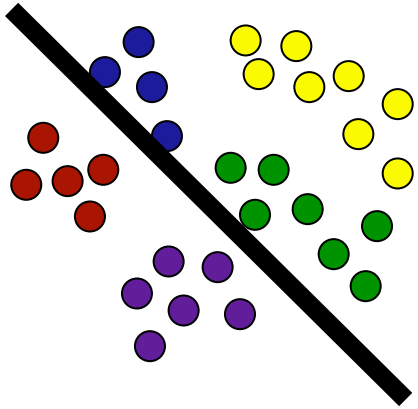
- we can try several tests, and
- retain the best one according to some criterion.



Why do random tests work?

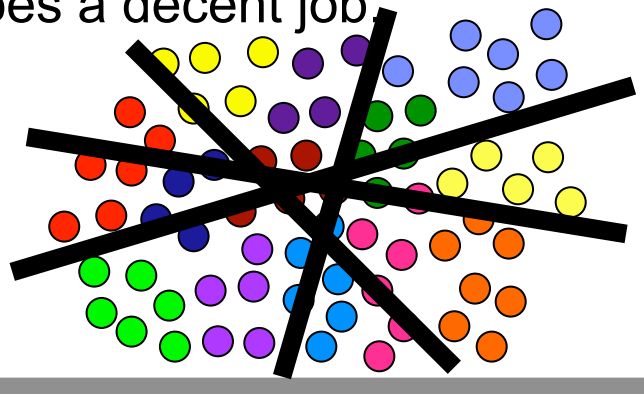
For a small number of classes

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



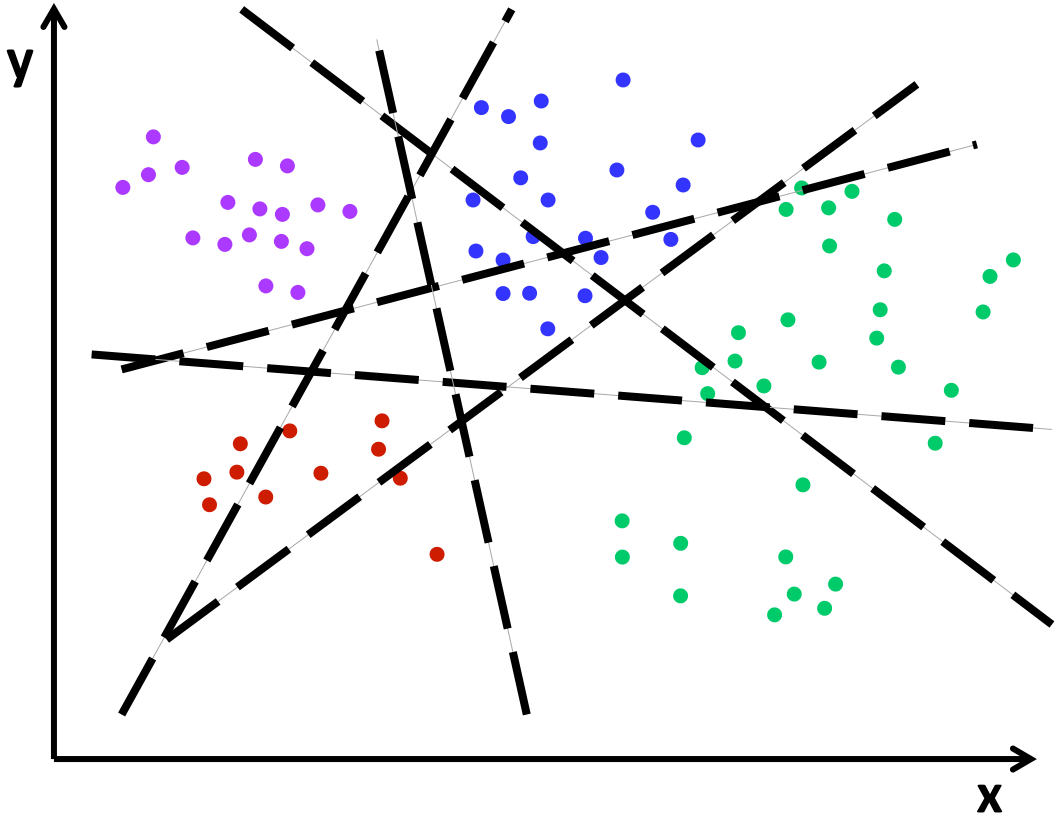
When the number of classes is large

- any test does a decent job:



Toy Learning Example

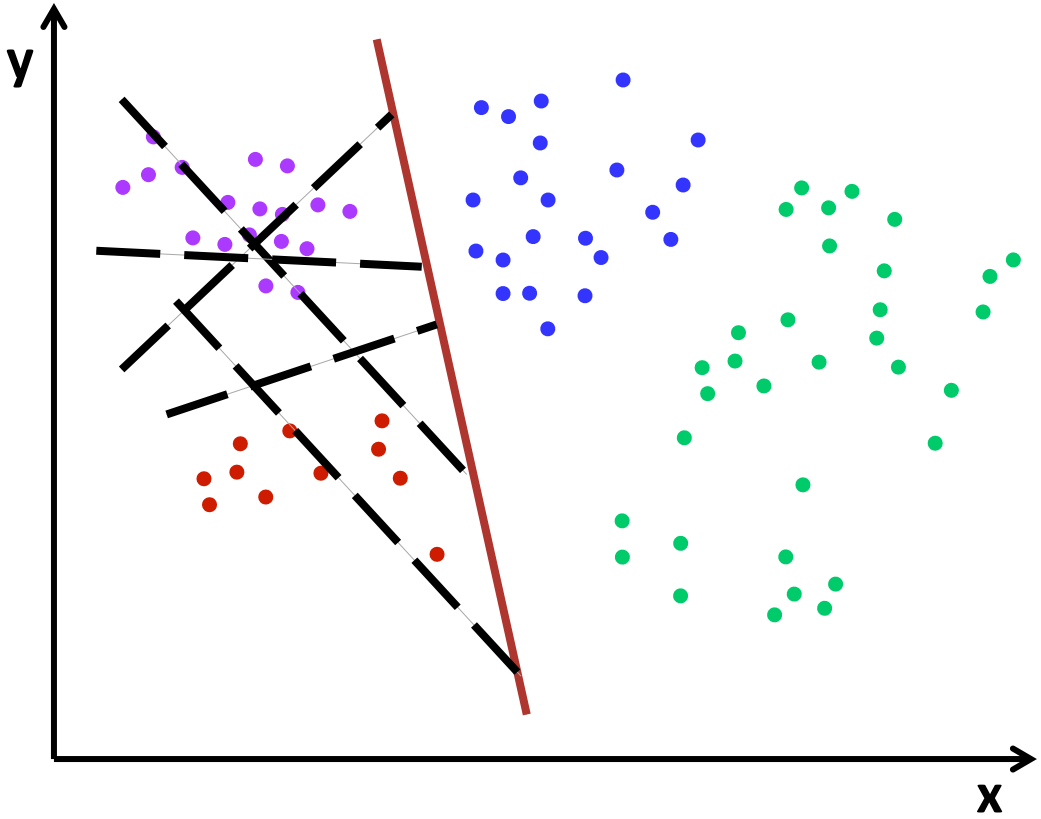
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **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

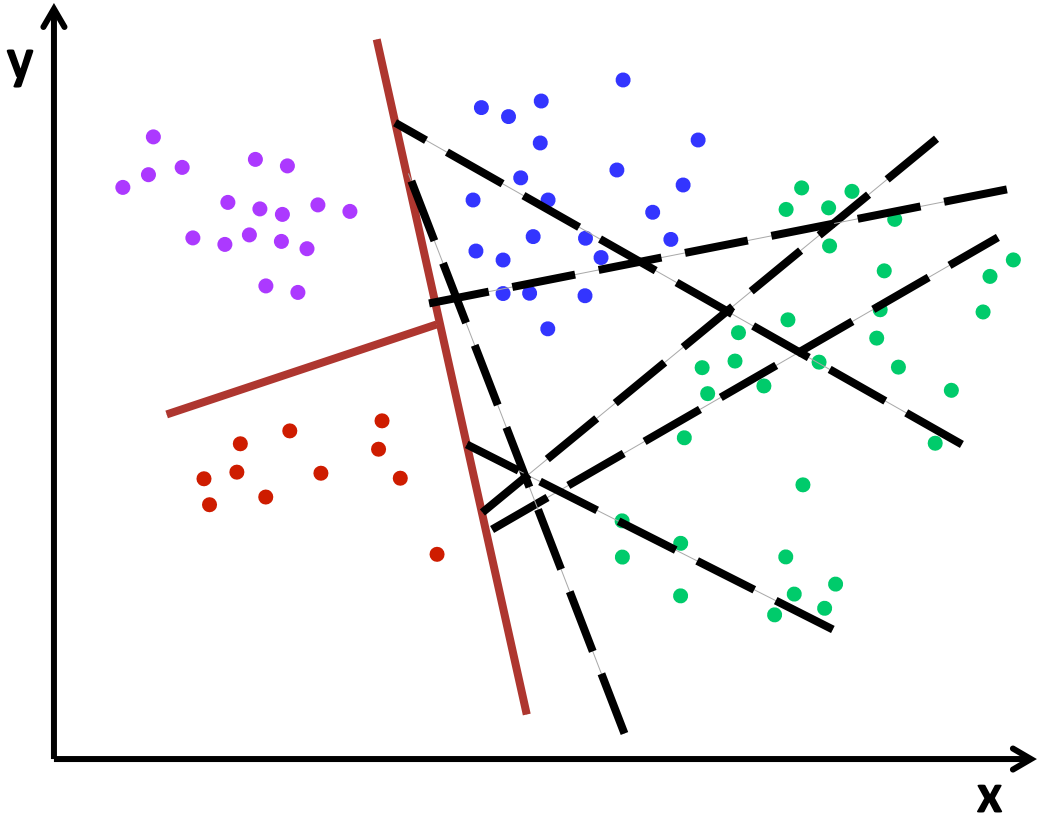
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **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

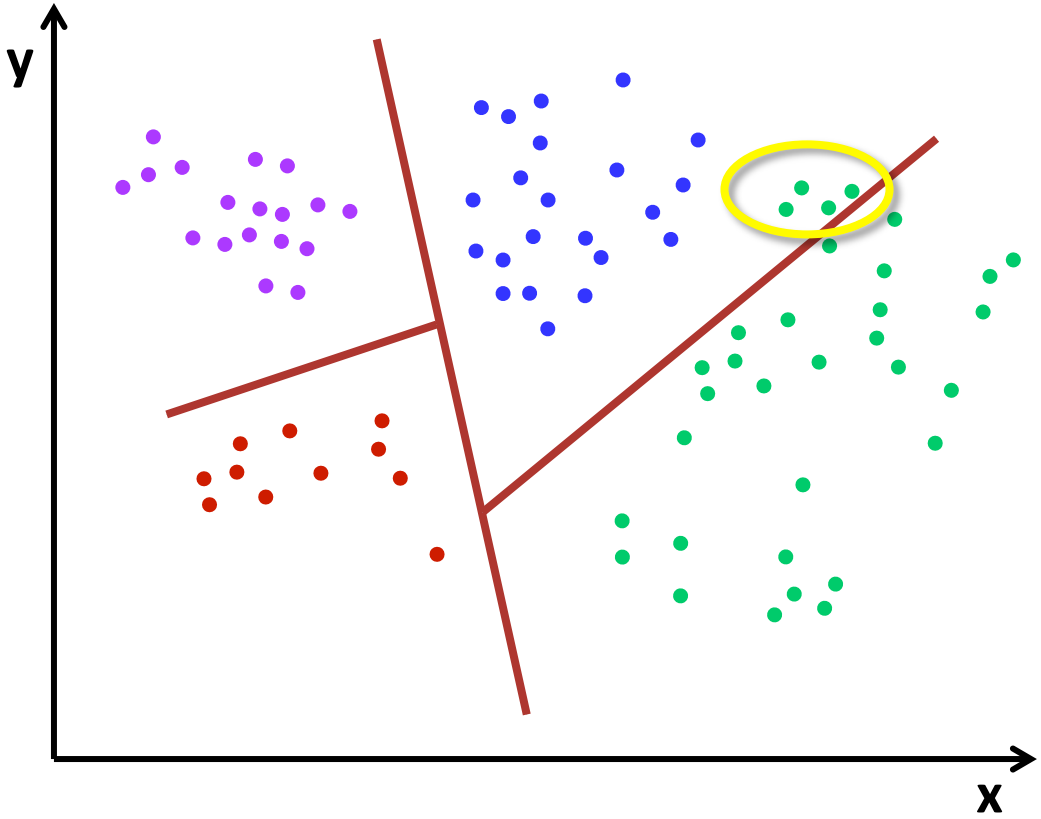
- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- **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**

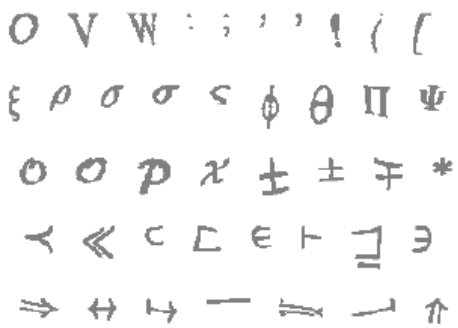
Toy Learning Example

- 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

Randomized Forests in Vision

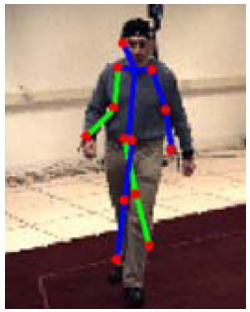


[Amit & Geman, 97]
digit recognition

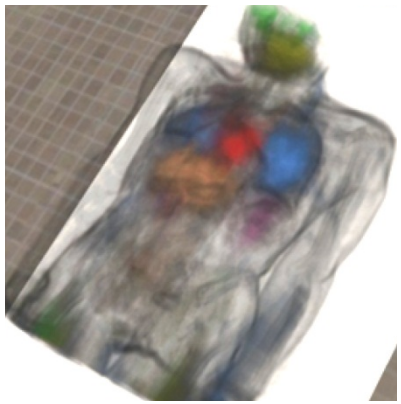
[Lepetit *et al.*, 06]
keypoint recognition

[Moosmann *et al.*, 06]
visual word clustering

[Shotton *et al.*, 08]
object segmentation



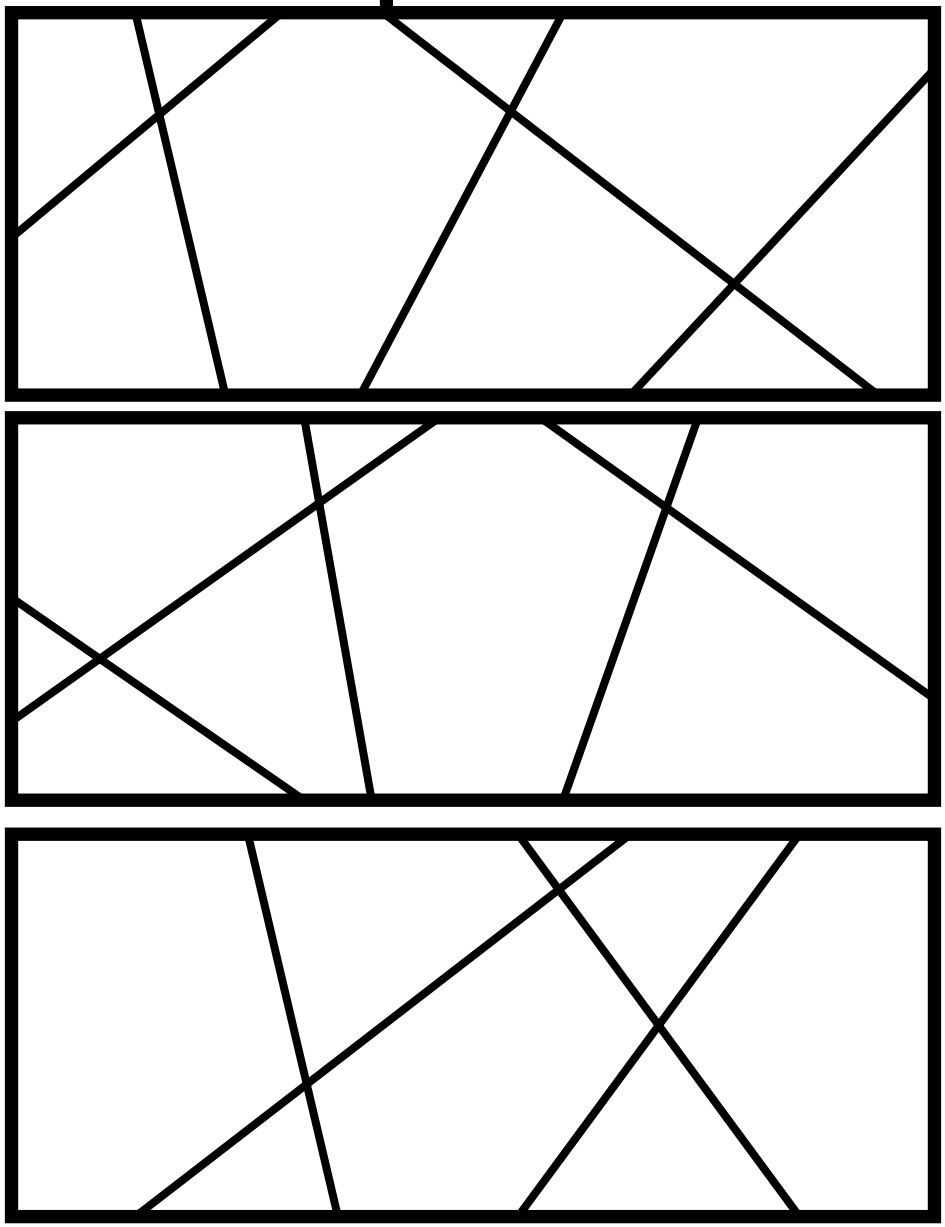
[Rogez *et al.*, 08]
pose estimation



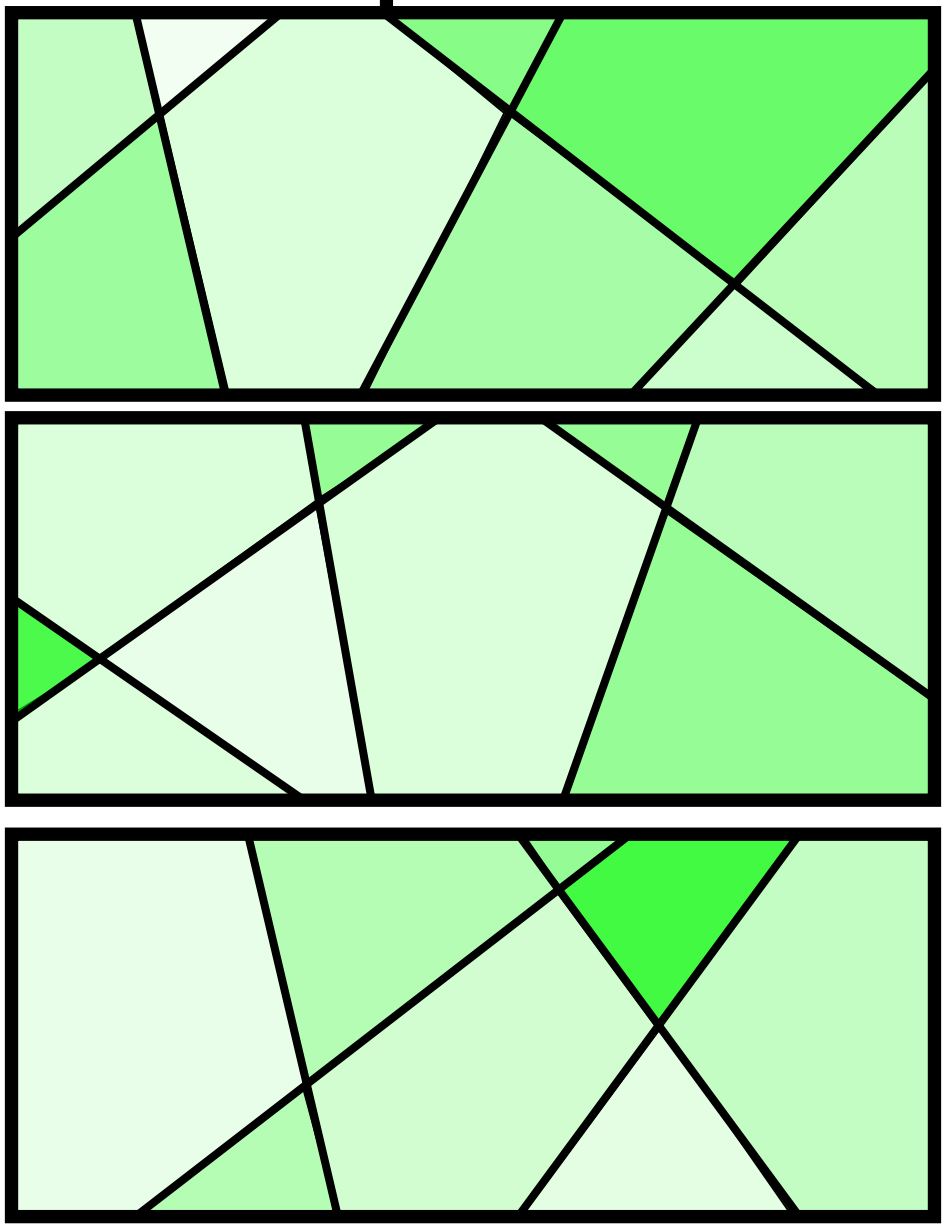
[Criminisi *et al.*, 09]
organ detection

(Among many others...)

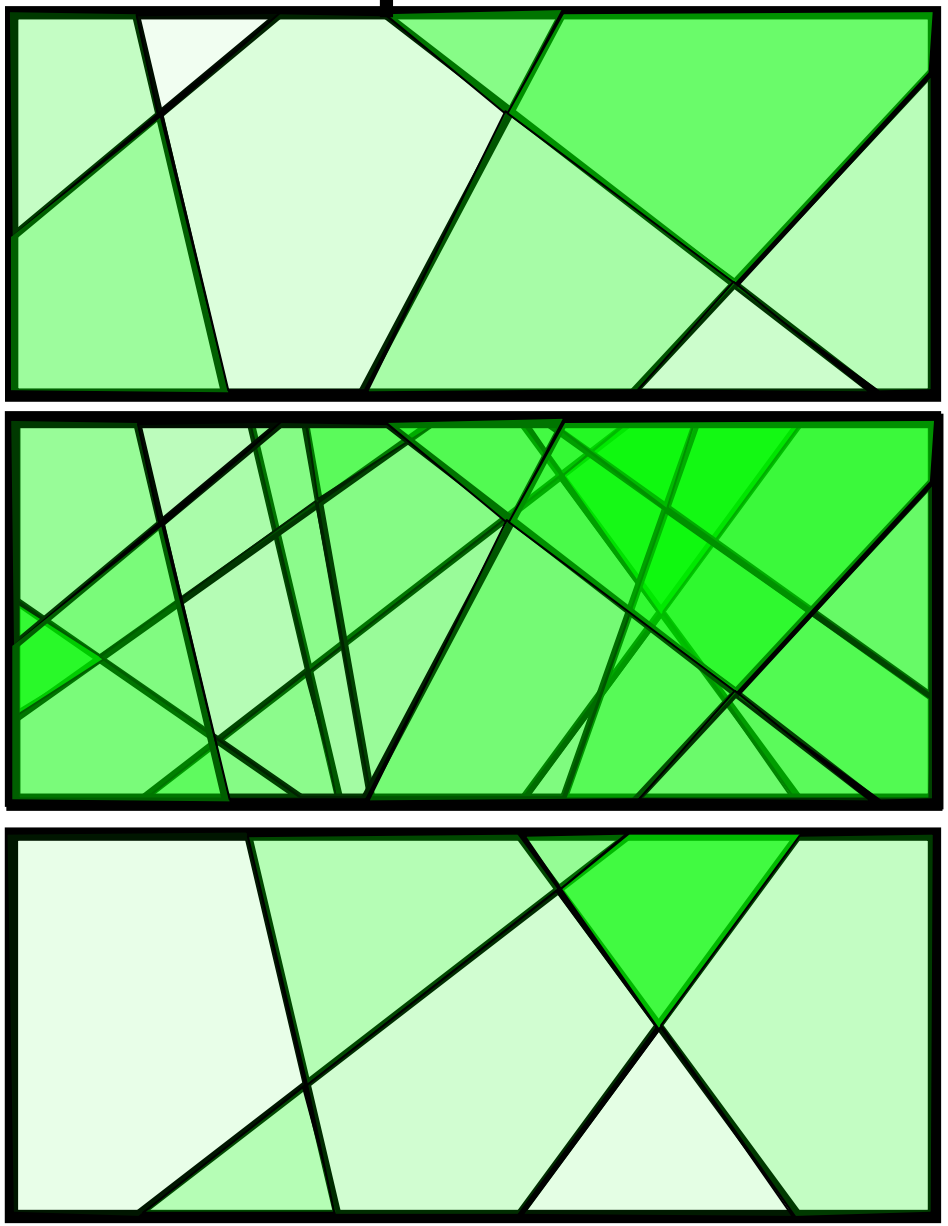
A Graphical Interpretation



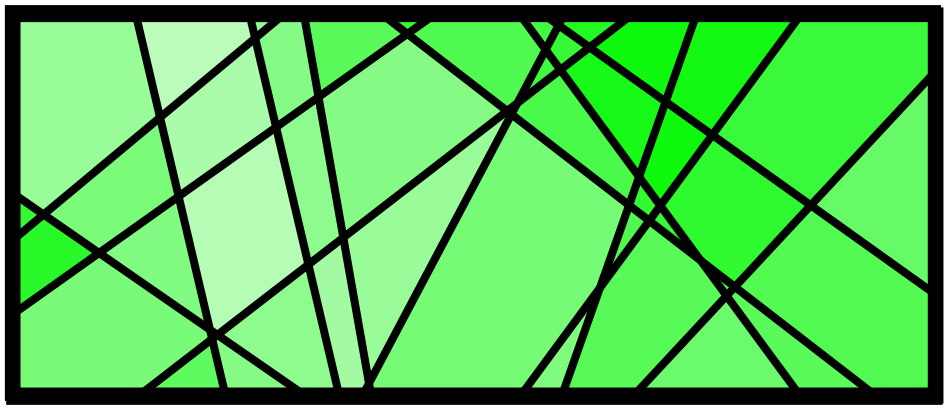
A Graphical Interpretation



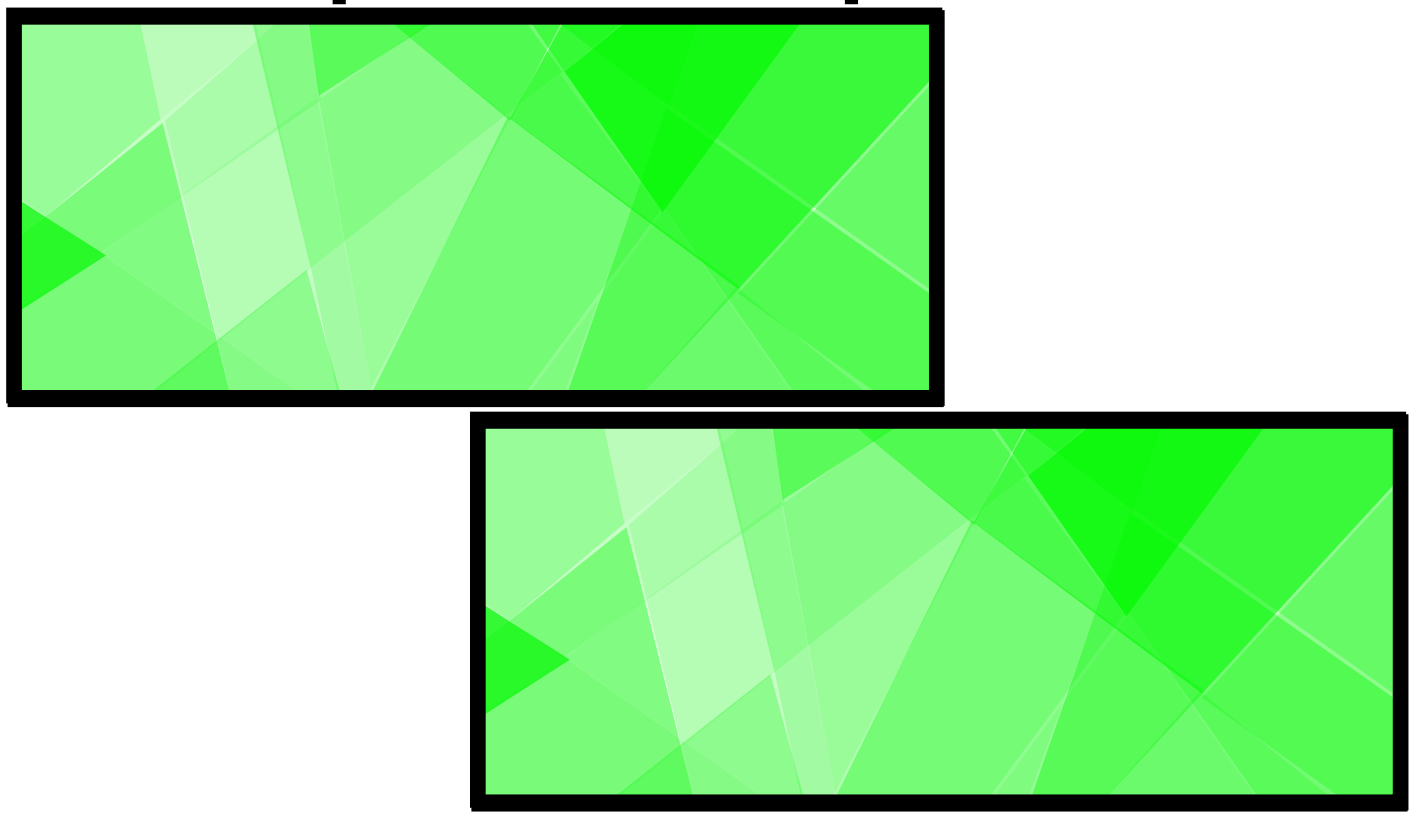
A Graphical Interpretation



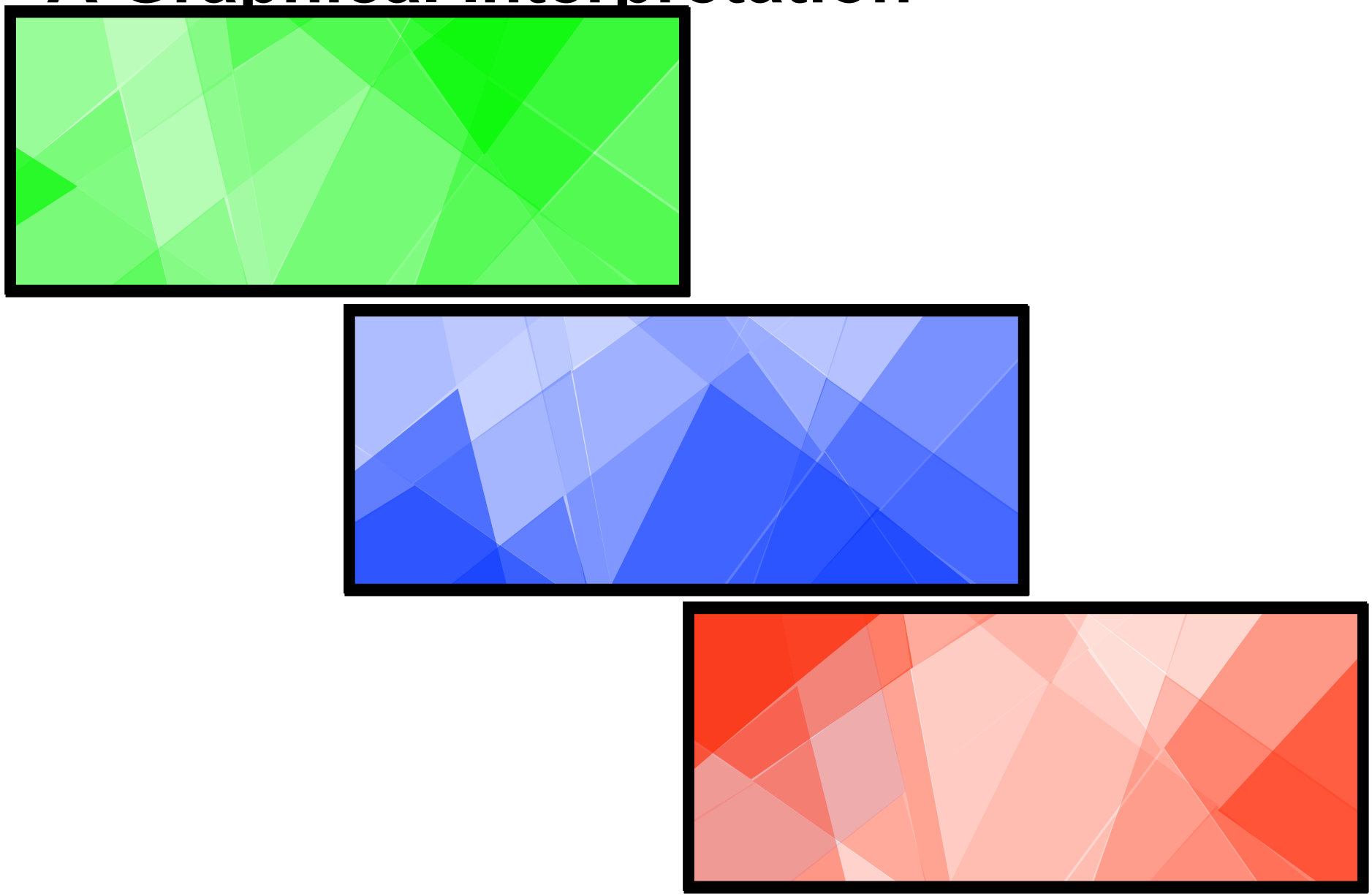
A Graphical Interpretation



A Graphical Interpretation



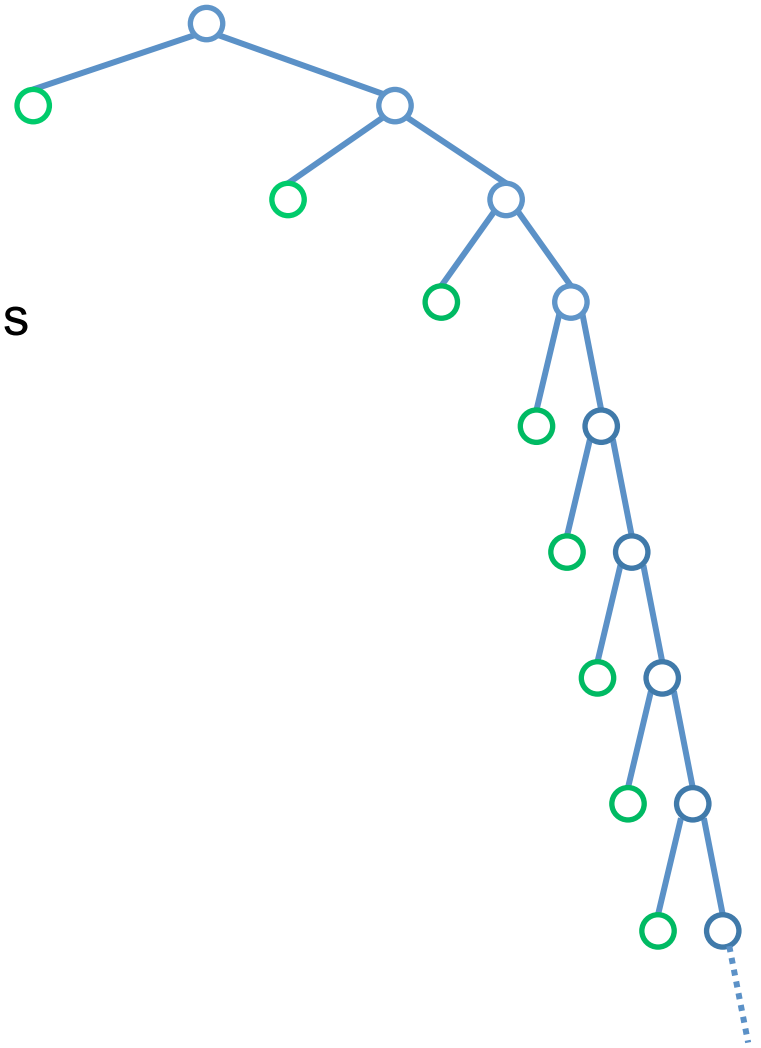
A Graphical Interpretation



Relation to Cascades [Viola & Jones 04]

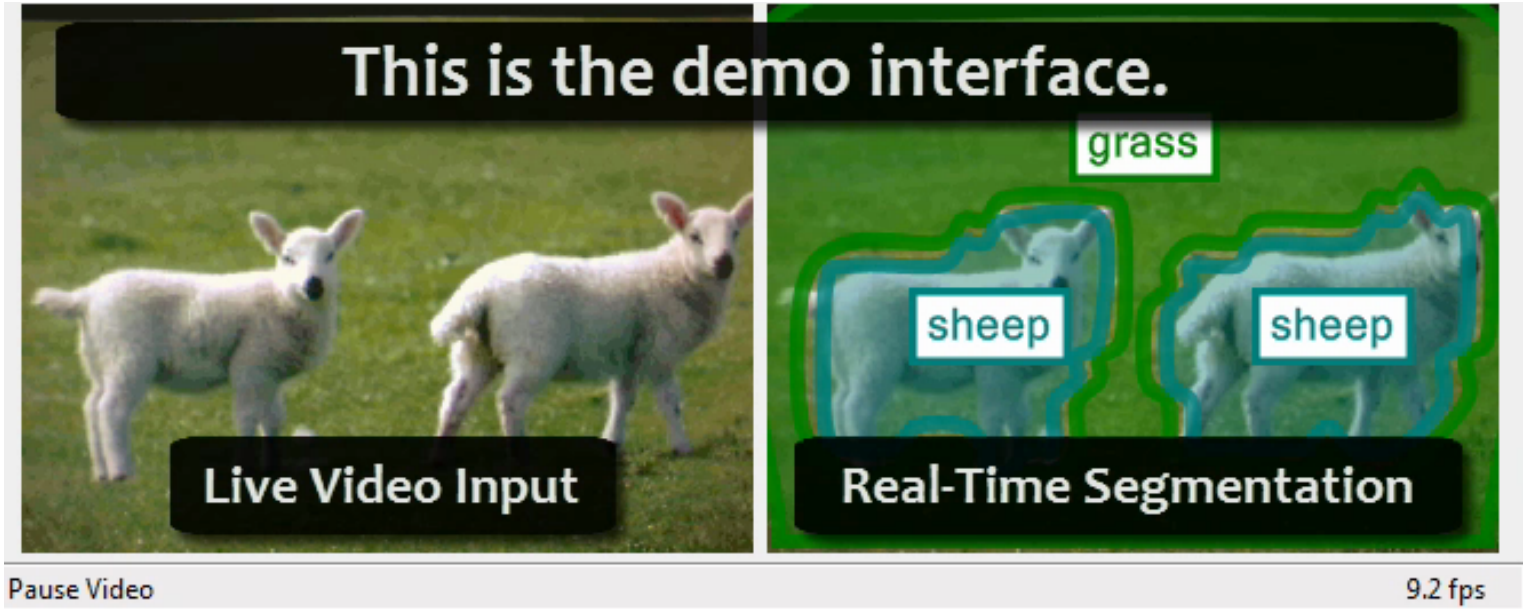
- Boosted Cascades
  - very unbalanced tree
  - good for unbalanced binary problems
    e.g. sliding window object detection

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



Real-Time Object Segmentation [Shotton *et al.* 2008]

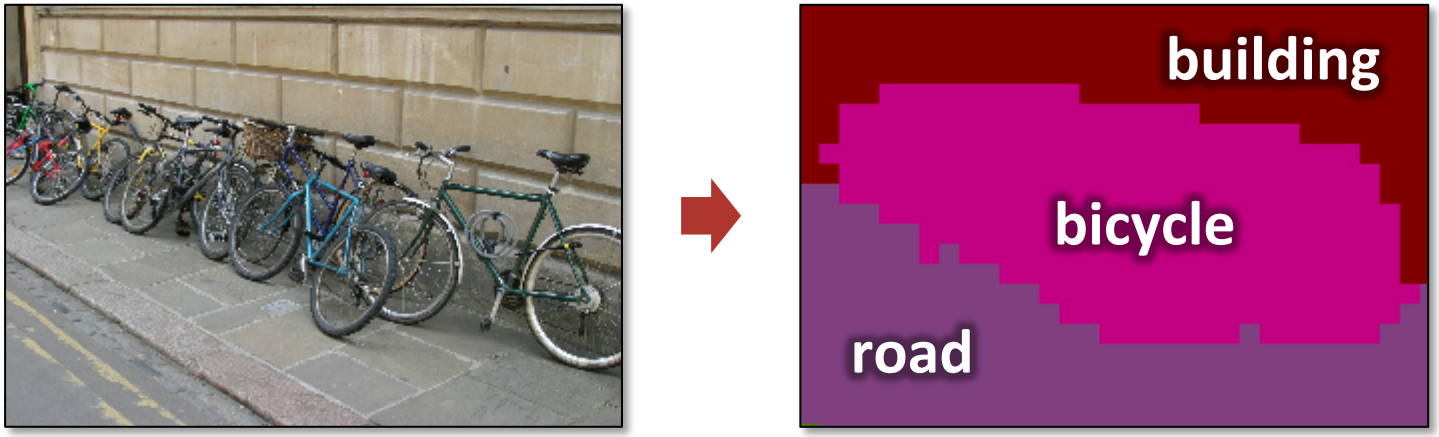
- Segment image and label segments in real-time



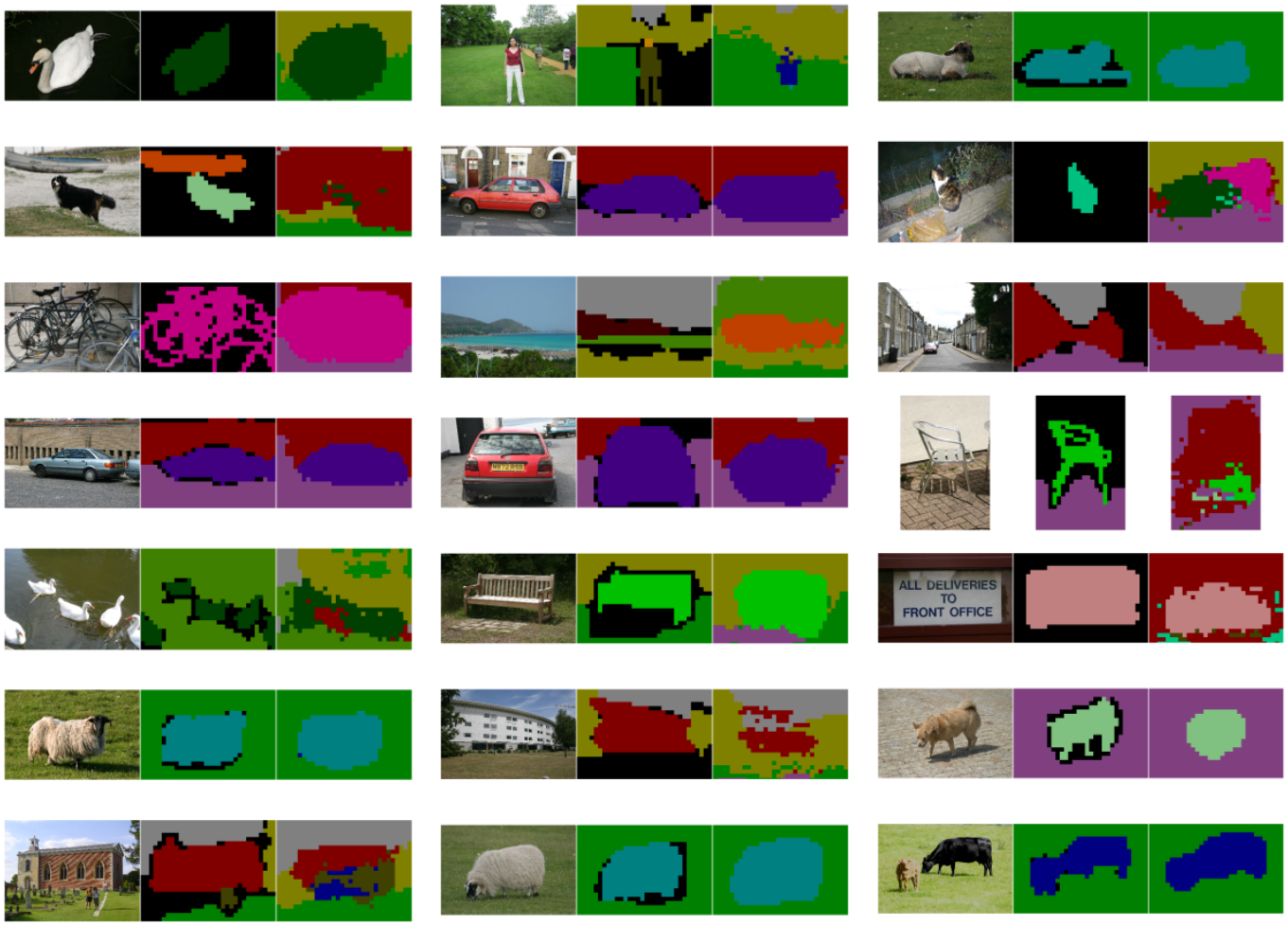
CVPR 2008 Best Demo Award

Segmentation Forest

- Object segmentation



MSRC Dataset Results



building	grass	tree	cow	sheep	sky	airplane	water	face	car	boat
bicycle	flower	sign	bird	book	chair	road	cat	dog	body	

$UCCE$$ $TORY: Kinect

12.1 million sold by end 2011

GUINNESS WORLD RECORDS™

Kinect's pipeline

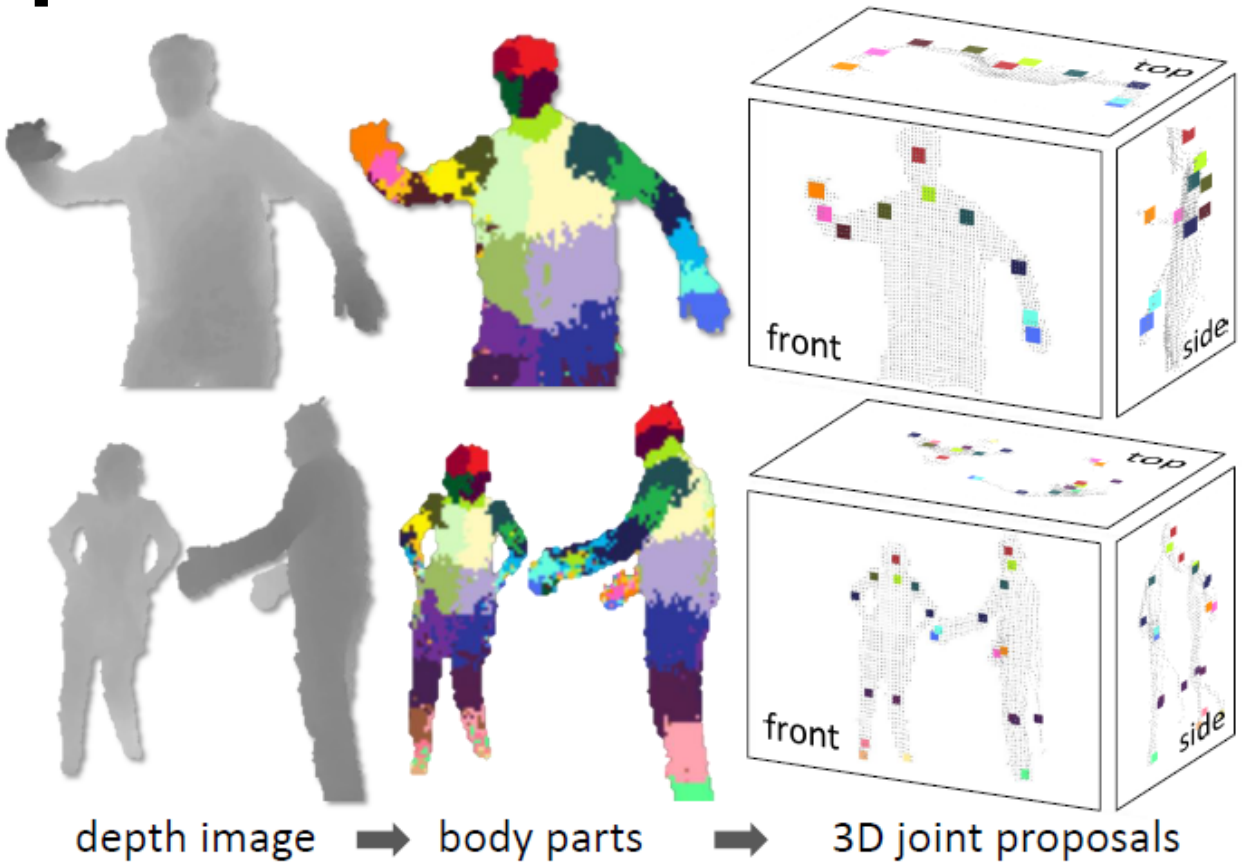
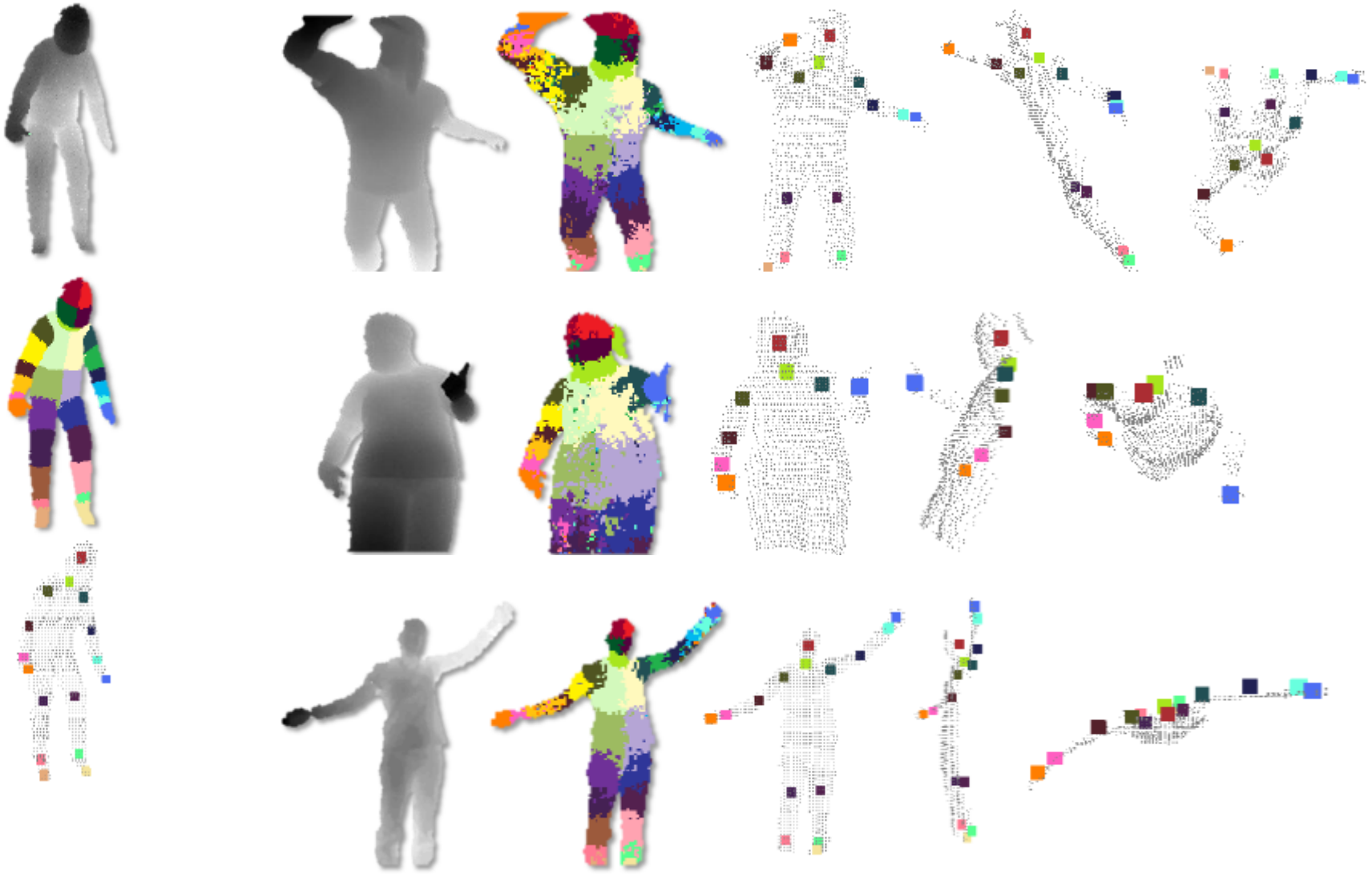


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

- Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake,
  Real-Time Human Pose Recognition in Parts from a Single Depth Image, in *CVPR*, IEEE, June 2011

- **Abstract:** We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that **maps the difficult pose estimation problem into a simpler per-pixel classification problem.** Our **large and highly varied training dataset** allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.
- **The system runs at 200 frames per second on consumer hardware**. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.



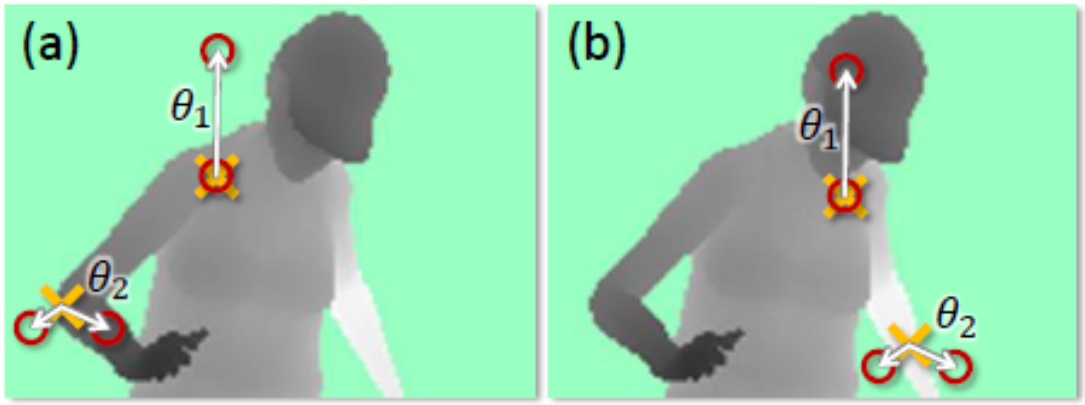


Figure 3. **Depth image features.** The yellow crosses indicates the pixel x being classified. The red circles indicate the offset pixels as defined in Eq. 1. In (a), the two example features give a large depth difference response. In (b), the same two features at new image locations give a much smaller response.

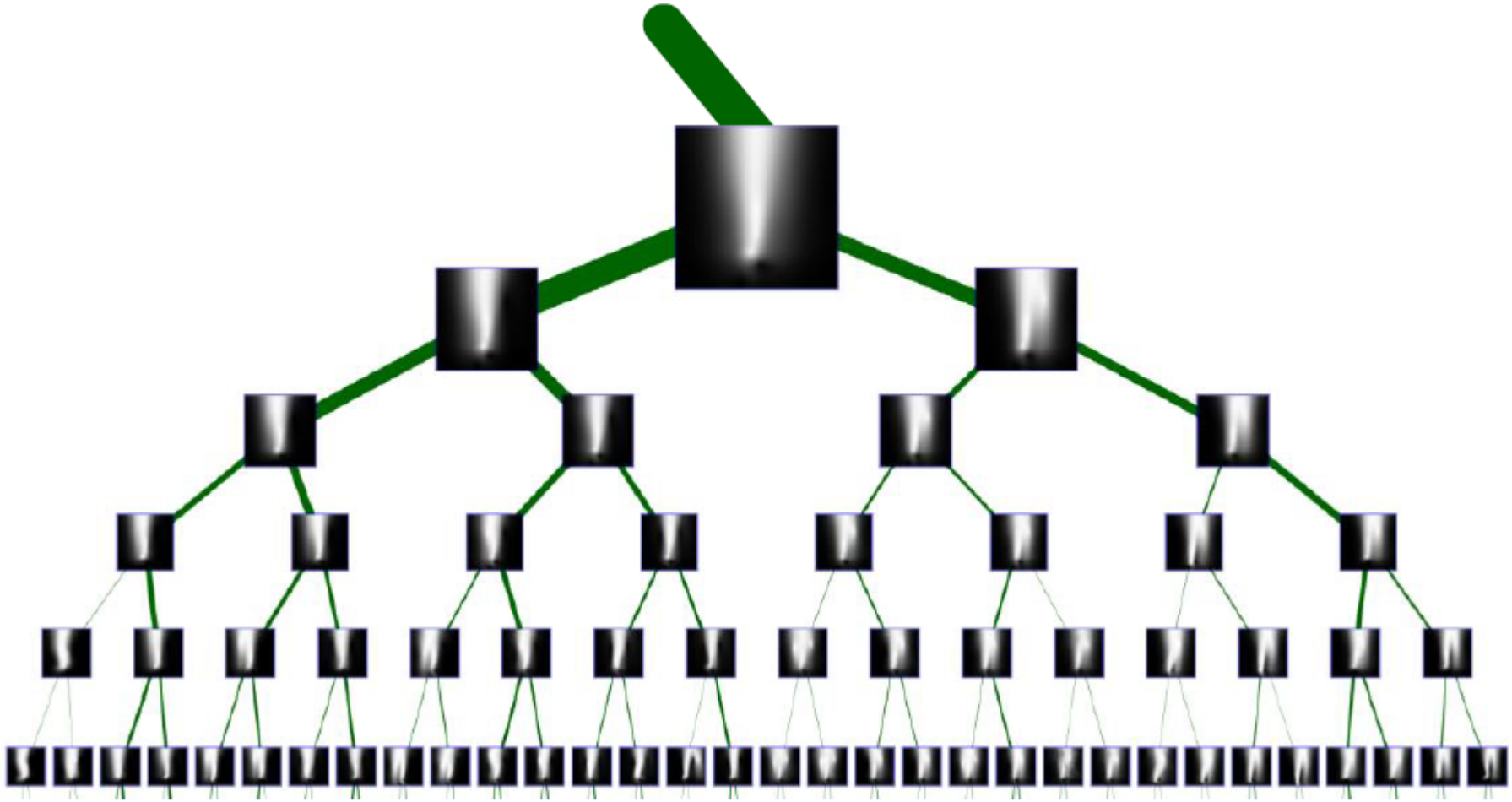
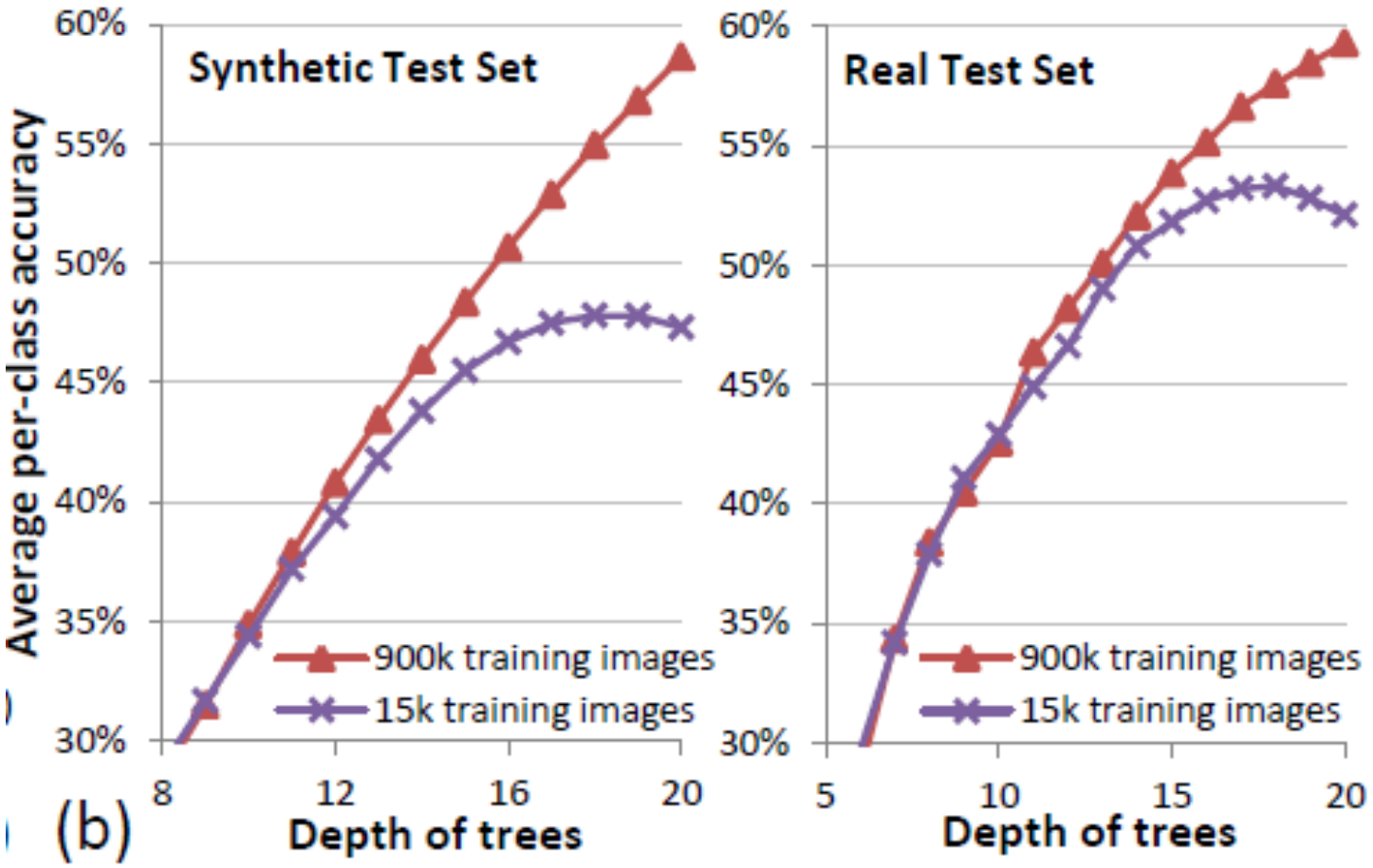


Figure 11. **Visualization of a trained decision tree.** Two separate subtrees are shown. A depth image patch centered on each pixel is taken, depth normalized, and binarized to a foreground/background silhouette. The patches are averaged across all pixels that reached any given tree node. The thickness of the edges joining the tree nodes is proportional to the number of pixels, and here shows fairly balanced trees. All pixels from 15k images are used to build the visualization shown.


Synthetic Test Set
Real Test Set
Average per-class accuracy
60%
55%
50%
45%
40%
35%
30%
900k training images
15k training images
8
12
16
20
5
10
15
20
Depth of trees
(b)

Why the class is not over yet