Patten Recognition

Lecture 9

Lecturer: Xinchao Wang

xinchao@nus.edu.sg

NUS National University of Singapore

What we have learned so far…

- Classification using single models
  - Bayesian
  - LDA
  - SVM

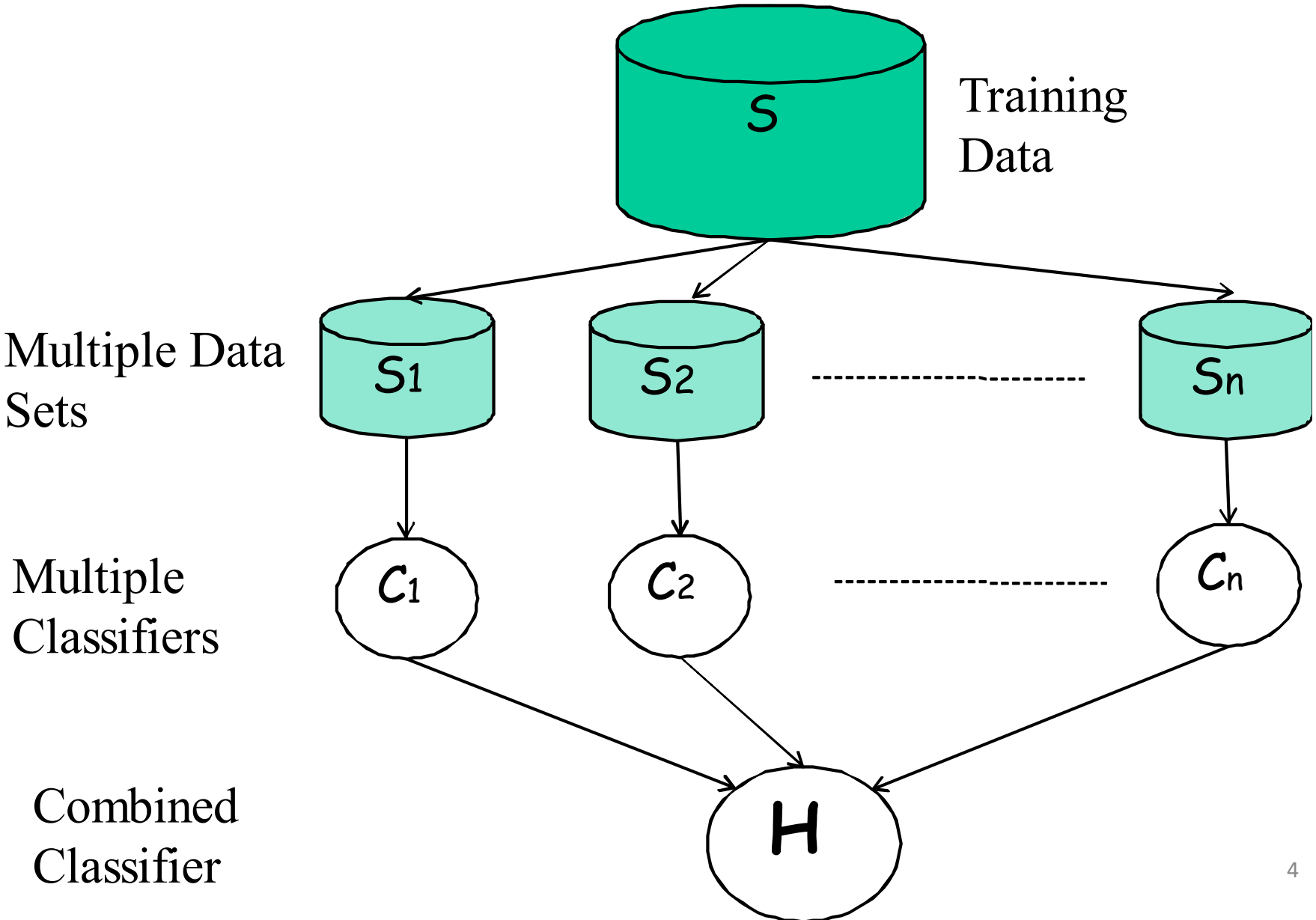
- Can we do better than single models?
  - Yes!
  - Using **multiple models** when possible!

Ensemble Methods

- Bagging (Breiman 1994,…)
  - Random forests (Breiman 2001,…)
- Boosting (Freund and Schapire 1995, Friedman et al. 1998,…)

Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data)

General Idea of Bagging



Ensemble Classifiers

- Basic idea: Build different “experts” and let them vote
- Advantages:
  - Improve predictive performance
  - Different types of classifiers can be directly included
  - Easy to implement
  - Not too much parameter tuning
- Disadvantages:
  - The combined classifier is not transparent (black box)
  - Not a compact representation

Why do they work?

- Suppose there are 25 base classifiers
- Each classifier has error rate $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

Bagging = Bootstrap Aggregating

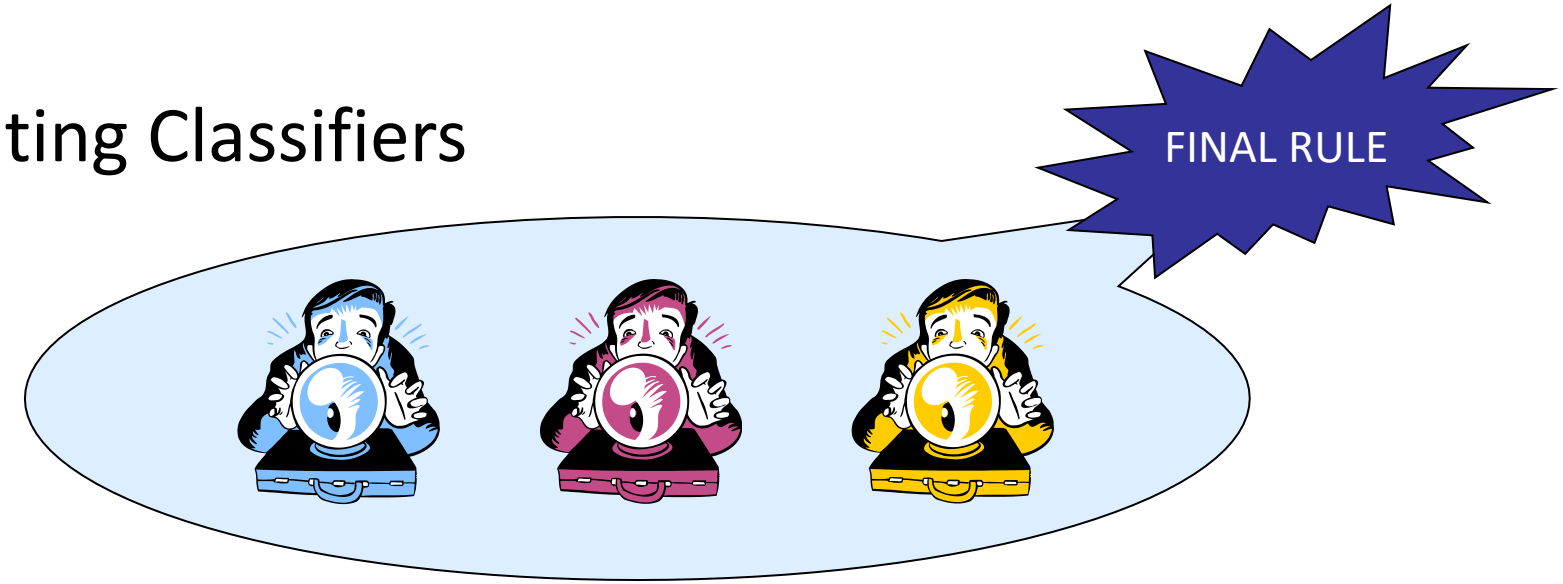
- Training
  - Given a dataset S, at each iteration i, a training set S_i is sampled with replacement from S (i.e. bootstraping)
  - A classifier C_i is learned for each S_i
- Classification: given an unseen sample X
  - Each classifier C_i returns its class prediction
  - The bagged classifier H counts the votes and assigns the class with the most votes to X

Bagging

- Bagging works because it reduces variance by voting/averaging
  - Usually, the more classifiers the better
- Problem: we only have one dataset
- Solution: generate new ones of size n by bootstrapping, i.e. sampling with replacement
- Can help a lot if data is noisy

Aggregating Classifiers

➔Aggregating Classifiers



- Breiman (1996) found gains in accuracy by aggregating predictors built from reweighed versions of the learning set

Bagging

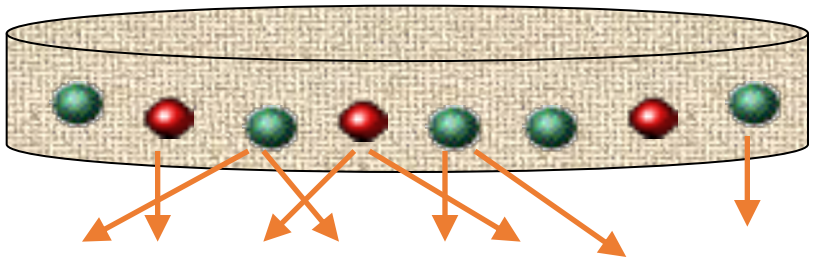
- *Bagging* = Bootstrap Aggregating
- Reweighing of the learning sets is done by drawing at random with replacement from the learning sets
- Predictors are aggregated by voting

The Bagging Algorithm

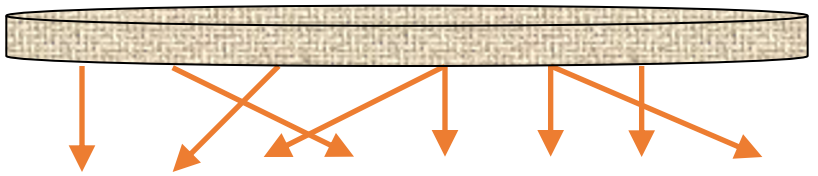
- B bootstrap samples
- From which we derive:
  - B Classifiers $\in \{-1, 1\}: c^1, c^2, c^3, \ldots, c^B$
  - B Estimated probabilities $\in [0, 1]: p^1, p^2, p^3, \ldots, p^B$
- The aggregate classifier becomes:

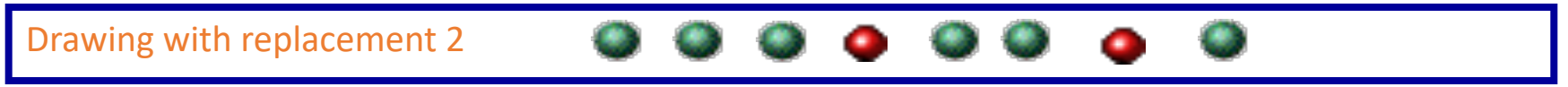
$$c_{bag}(x) = sign\left(\frac{1}{B}\sum_{b=1}^{B} c^b(x)\right) \quad \text{or} \quad p_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} p^b(x)$$

Weighting



Classifier 1





Classifier 2

Aggregation

Sign

Classifier 1

+

Classifier 2

+

Classifier 3

+

...

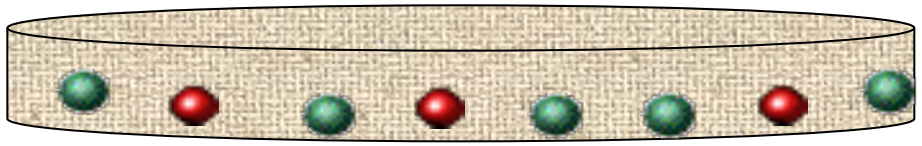
+

Classifier T

=

Final rule

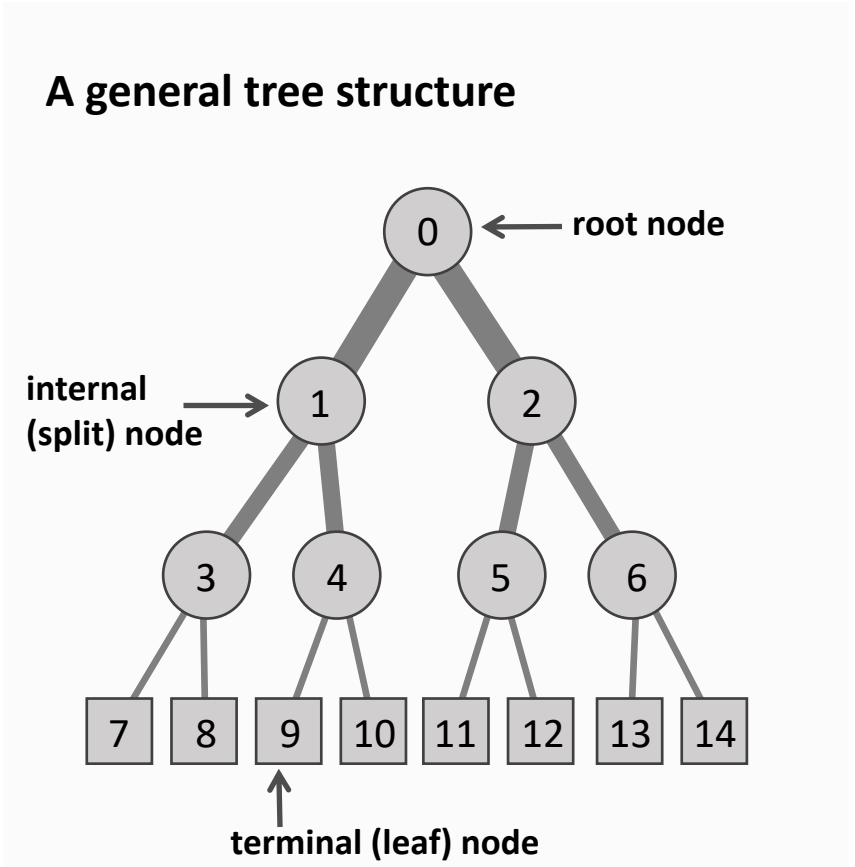
Initial set

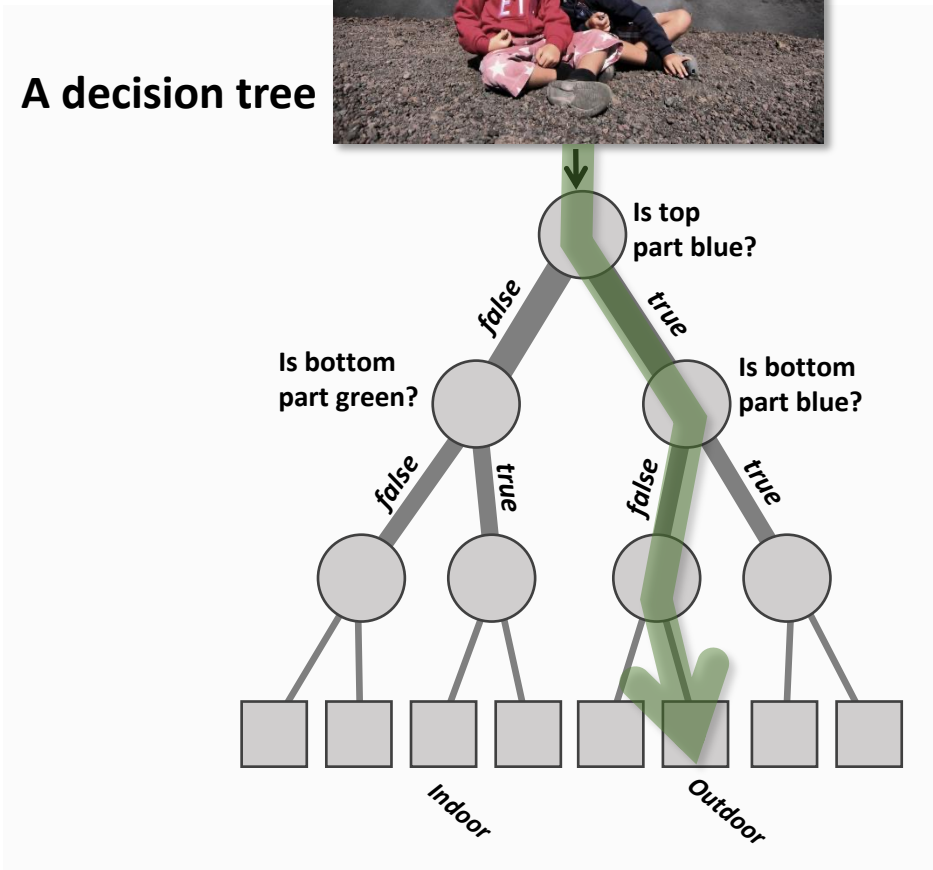


Random Forests

Breiman L. Random forests. Machine Learning 2001;45(1):5-32
http://stat-www.berkeley.edu/users/breiman/RandomForests/

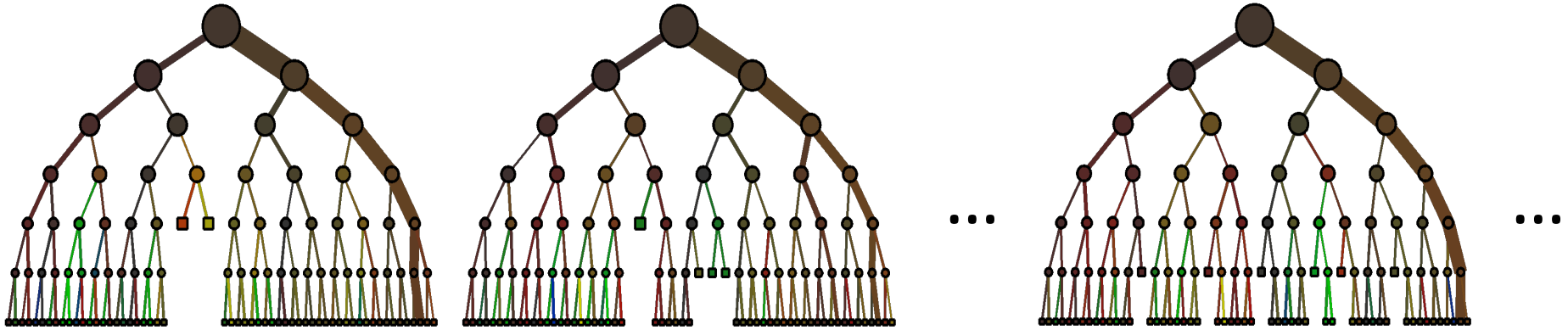
Decision Trees and Decision Forests





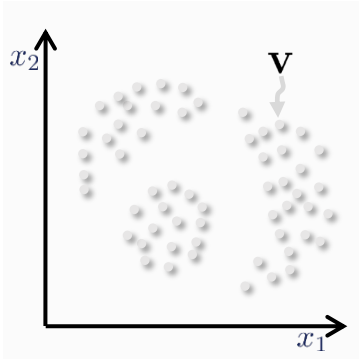
A forest is an ensemble of trees. The trees are all slightly different from one another

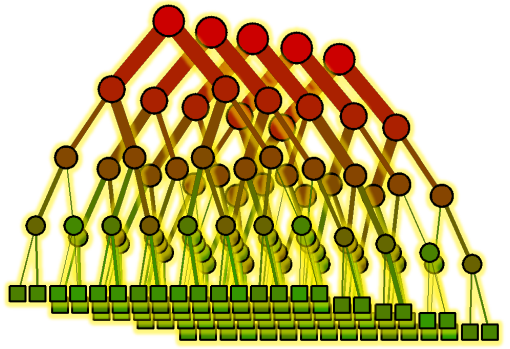
Decision Tree Training (Offline)



How many trees?
How different?
How to fuse their outputs?

Random Forest Model





Basic notation

Input data point	e.g. $\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$	*Collection of feature responses x_i. d=?*
Output/label space	e.g. $\in \{c_k\}$? $\mathbb{R}$?	*Categorical, continuous?*
Feature response selector	ϕ	*Features can be e.g. wavelets? Pixel intensities? Context?*

Forest model

tree

Node test parameters	$\theta \in \mathcal{T}$	*Parameters related to each split node: i) which features, ii) what geometric primitive, iii) thresholds.*
Node objective function (train.)	e.g. $I = I(\mathcal{S}_j, \theta)$	*The "energy" to be minimized when training the j-th split node*
Node weak learner	e.g. $h(\mathbf{v}, \theta_j) \in \{\texttt{true}, \texttt{false}\}$	*The test function for splitting data at a node j.*
Leaf predictor model	e.g. $p(c\|\mathbf{v})$	*Point estimate? Full distribution?*
Randomness model (train.)	e.g. *1. Bagging, 2. Randomized node optimization*	*How is randomness injected during training? How much?*
Stopping criteria (train.)	e.g. *max tree depth =* D	*When to stop growing a tree during training*

ensemble

Forest size	T	*Total number of trees in the forest*
The ensemble model	e.g. $p(c\|\mathbf{v}) = \frac{1}{T}\sum_t^T p_t(c\|\mathbf{v})$	*How to compute the forest output from that of individual trees?*

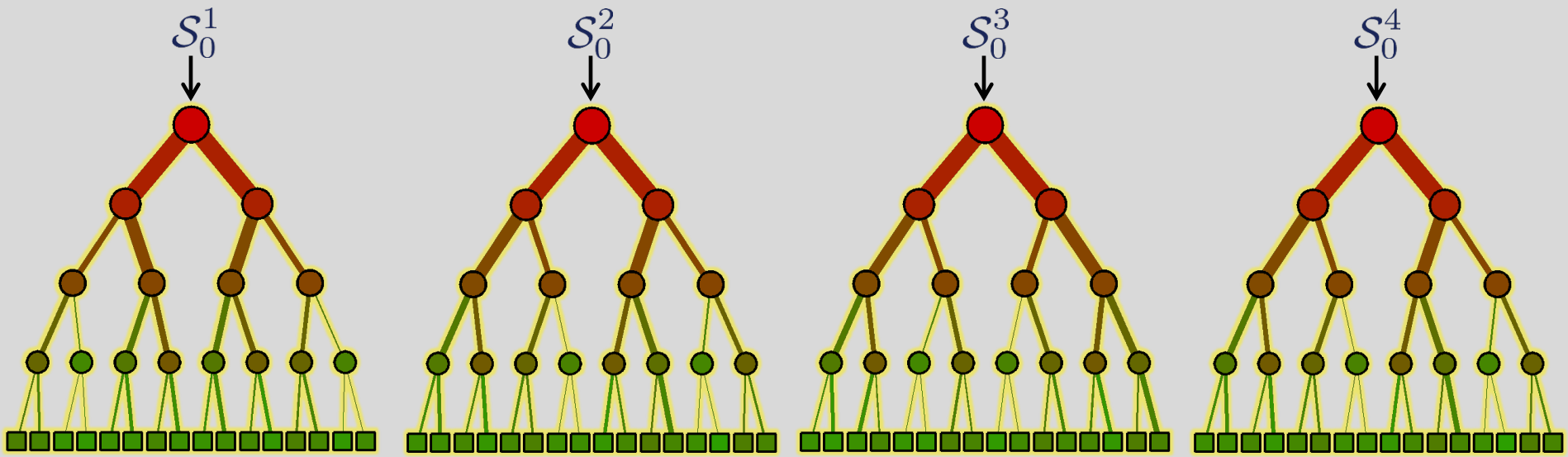
Randomness Model

1) Bagging (randomizing the training set)

$\mathcal{S}_0$ The full training set

$\mathcal{S}_0^t \subset \mathcal{S}_0$ The randomly sampled subset of training data made available for the tree *t*

Forest training



Efficient training

Randomness Model

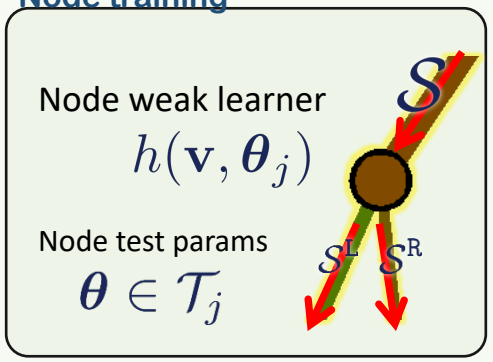
2) Randomized node optimization (RNO)

$\mathcal{T}$ The full set of all possible node test parameters

$\mathcal{T}_j \subset \mathcal{T}$ For each node the set of randomly sampled features

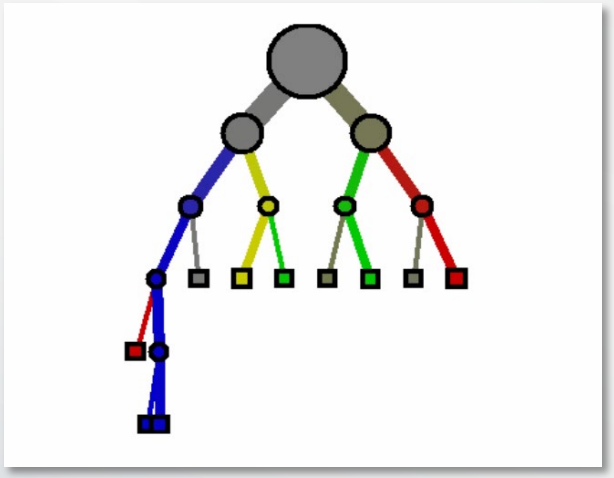
$\rho = |\mathcal{T}_j|$ Randomness control parameter.
For $\rho = |\mathcal{T}|$ no randomness and maximum tree correlation.
For $\rho = 1$ max randomness and minimum tree correlation.

Node training

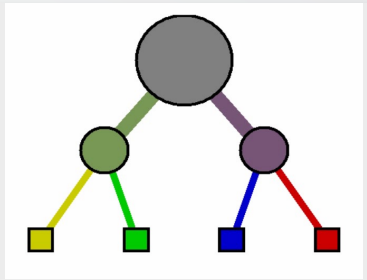


The effect of ρ

Small value of ρ ; little tree correlation.

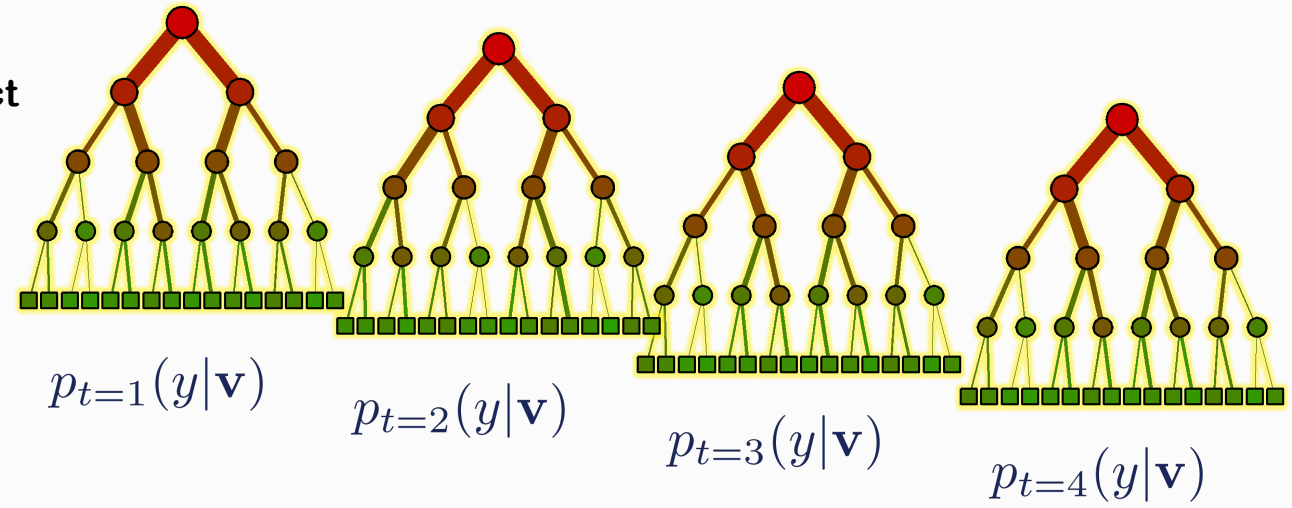


Large value of ρ ; large tree correlation.

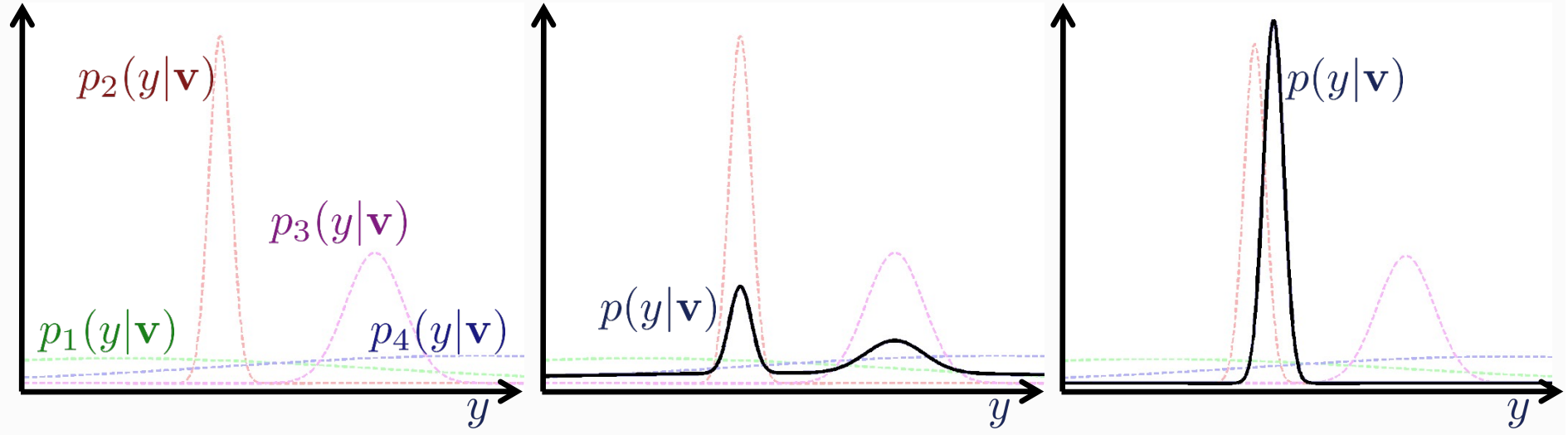


The Ensemble Model

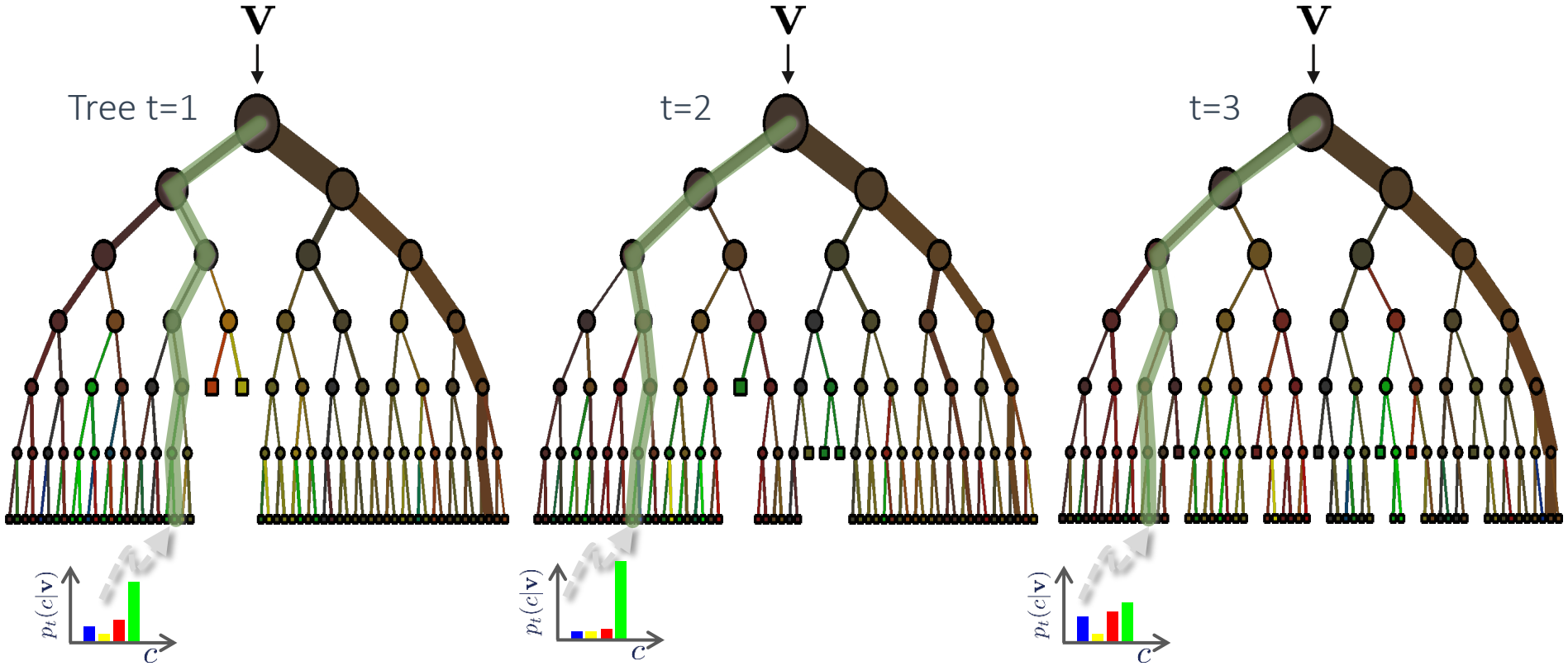
An example forest to predict continuous variables



$$p(y|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(y|\mathbf{v}) \qquad p(y|\mathbf{v}) = \frac{1}{Z}\prod_{t}^{T} p_t(y|\mathbf{v})$$

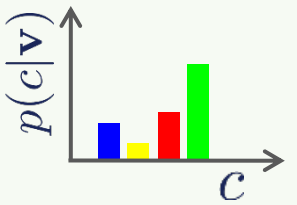


Classification Forest: Ensemble Model

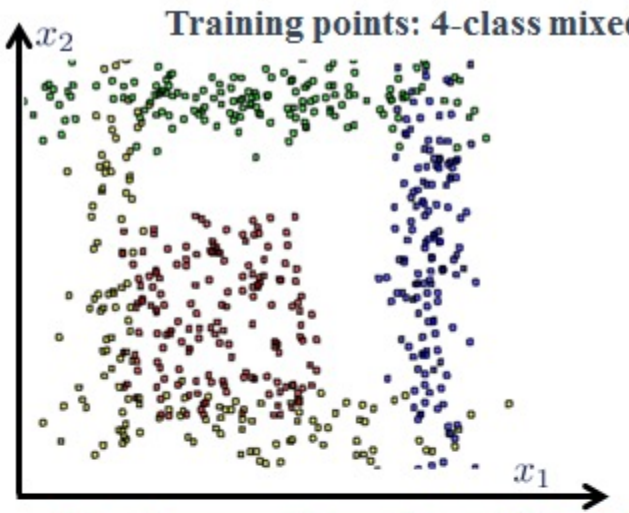


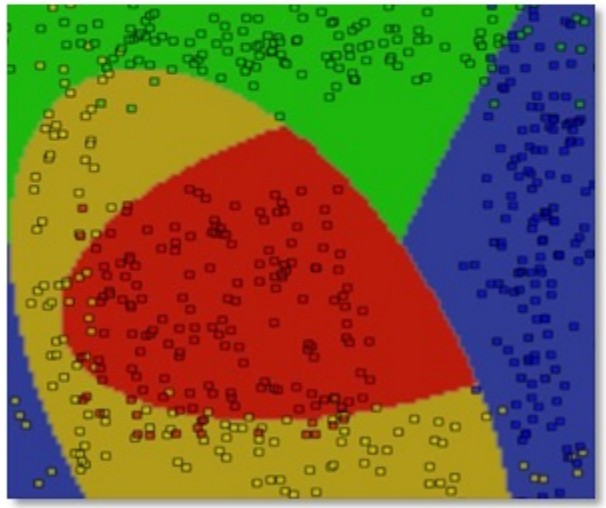
The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$

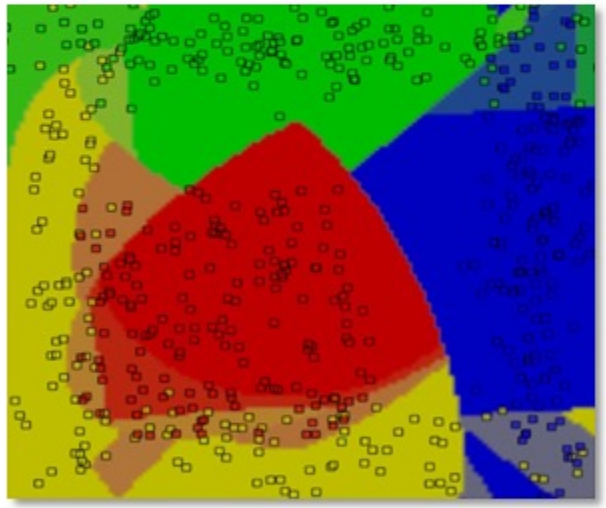


Effect of Tree Depth

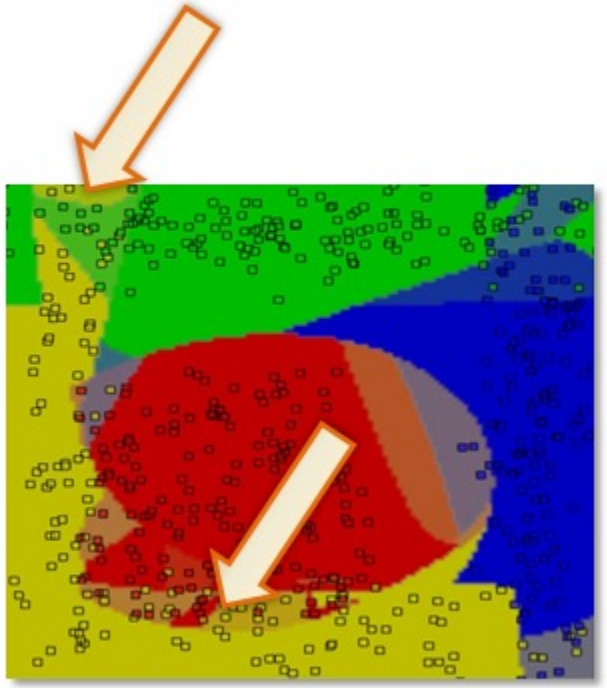




T=200, D=3, w. l. = conic



T=200, D=6, w. l. = conic



T=200, D=15, w. l. = conic

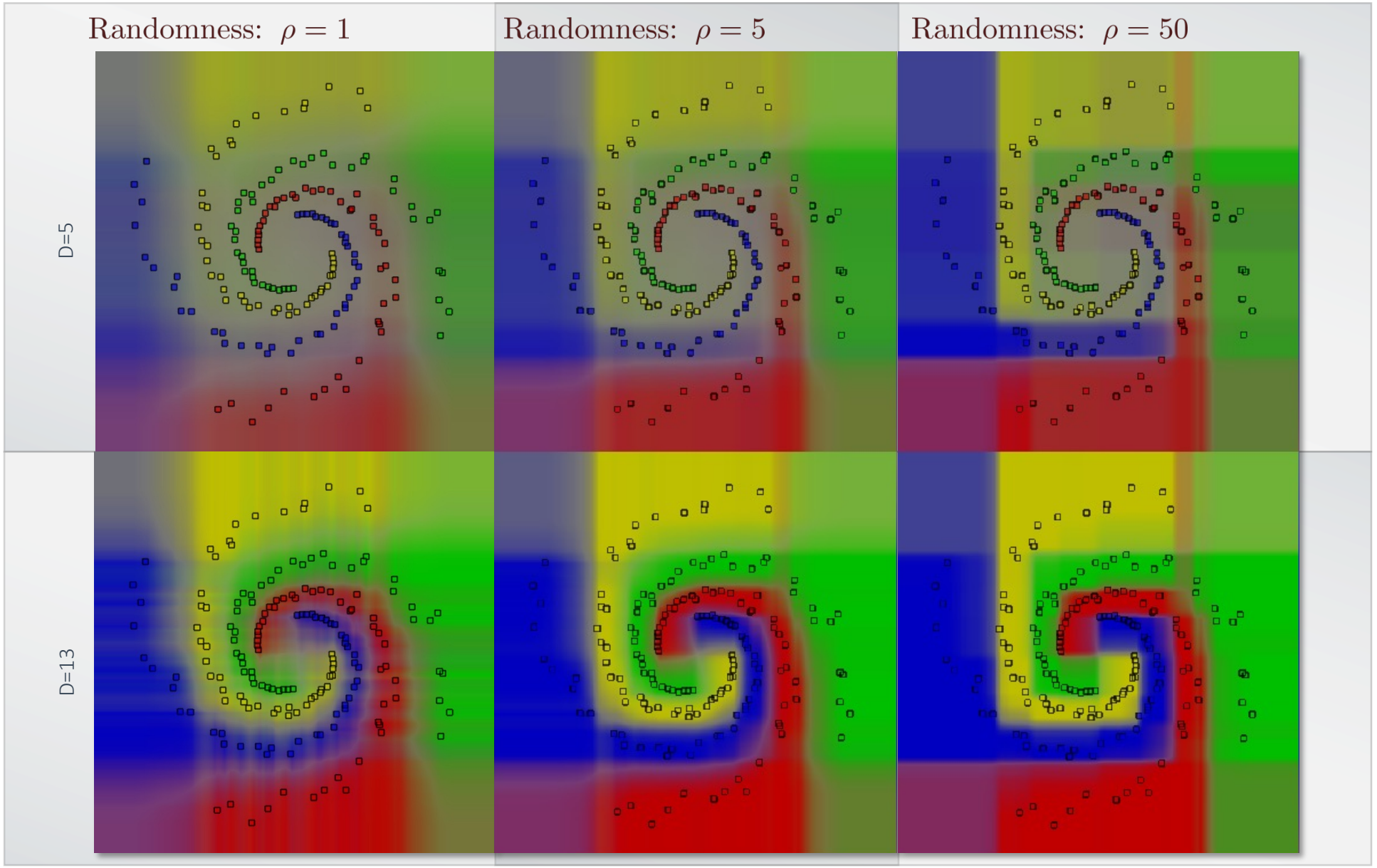
max tree depth, D

underfitting

overfitting

Effect of Weak Learner Model and Randomness

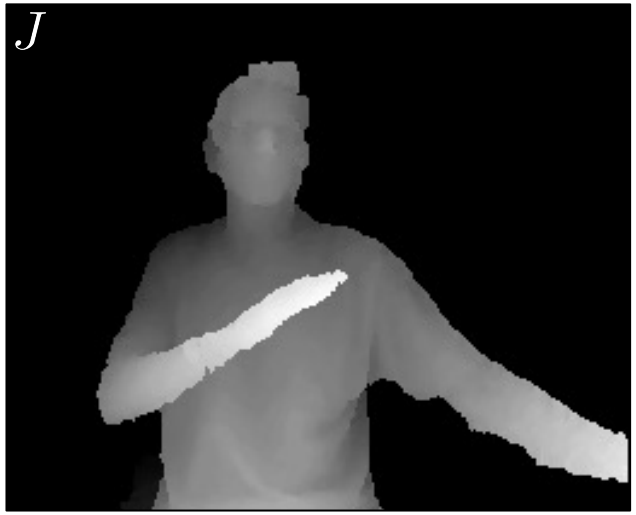
Testing posteriors



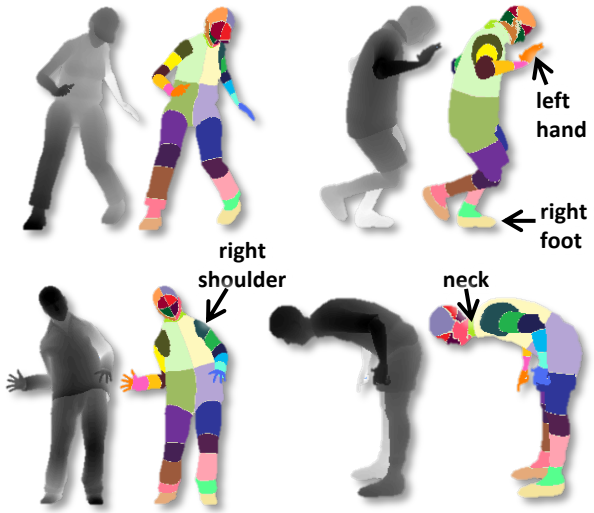
Weak learner: axis aligned

Parameters: T=400 predictor model = prob.

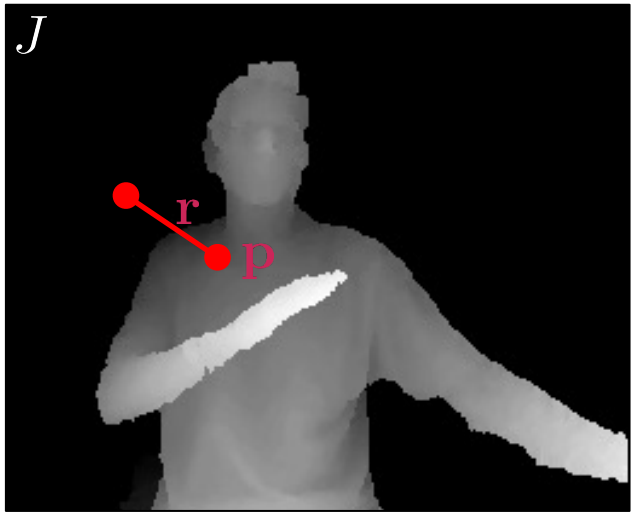
Body tracking in Microsoft Kinect for XBox 360



Input depth image



Training labelled data

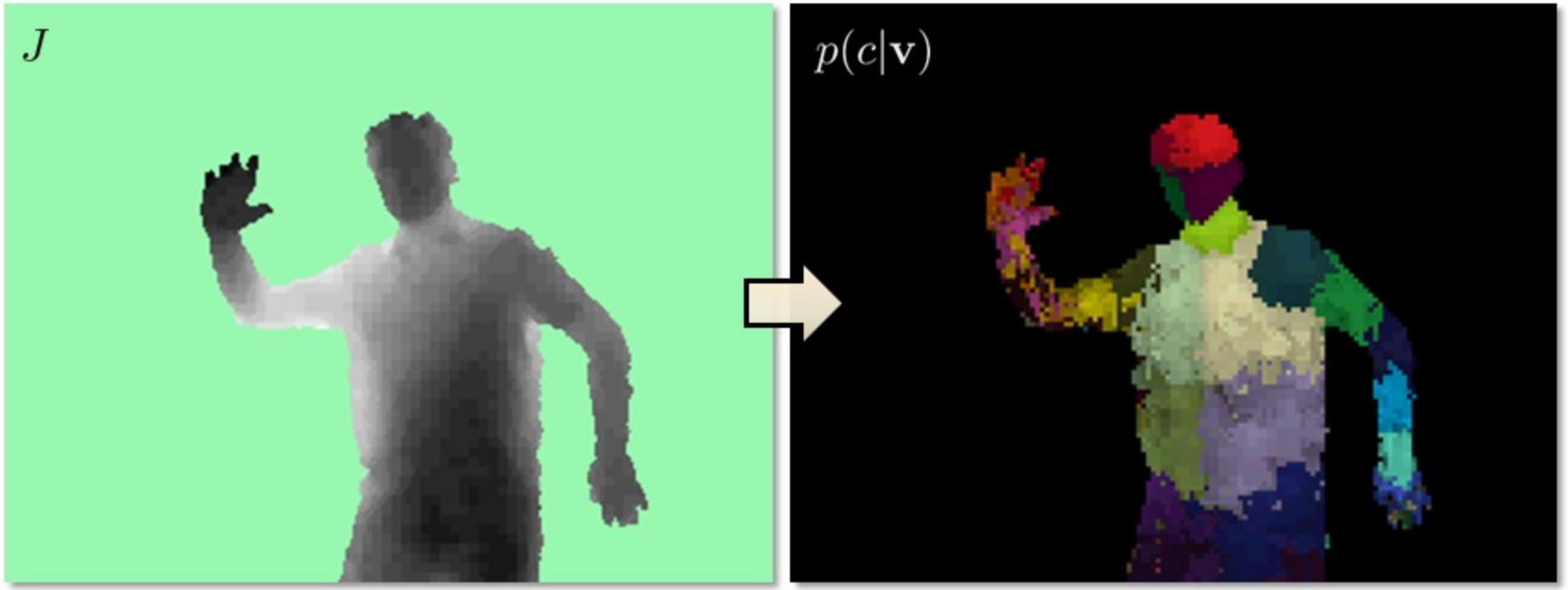


Visual features

Classification forest

Labels are categorical	$c \in \{\mathtt{l.hand}, \mathtt{r.hand}, \mathtt{head}, \ldots\}$
Input data point	$\mathbf{p} \in \mathbb{R}^2$
Visual features	$\mathbf{v}(\mathbf{p}) = (x_1, \ldots, x_i, \ldots, x_d) \in \mathbb{R}^d$
Feature response	$x_i = J(\mathbf{p}) - J\left(\mathbf{p} + \frac{\mathbf{r}_i}{J(\mathbf{p})}\right)$
Predictor model	$p(c\|\mathbf{v})$
Objective function	$I = H(\mathcal{S}_j) - \sum_{i=\mathtt{L},\mathtt{R}} \frac{\lvert\mathcal{S}_j^i\rvert}{\lvert\mathcal{S}_j\rvert} H(\mathcal{S}_j^i)$
Node parameters	$\boldsymbol{\theta} = (\mathbf{r}, \tau)$
Node training	$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta} \in \mathcal{T}_j} I(\mathcal{S}_j, \boldsymbol{\theta})$
Weak learner	$h(\mathbf{v}, \boldsymbol{\theta}) = [\phi(\mathbf{v}, \mathbf{r}) > \tau]$

Body tracking in Microsoft Kinect for XBox 360



Input depth image (bg removed)

Inferred body parts posterior

Advantages of Random Forests

- Very high accuracy – not easily surpassed by other algorithms
- Efficient on large datasets
- Can handle thousands of input variables without variable deletion
- Effective method for estimating missing data, also maintains accuracy when a large proportion of the data are missing
- Robust to label noise
- Can be used in clustering, locating outliers and semi-supervised learning

L. Breiman's web page

Boosting

Boosting Resources

- Slides Credits:
  - Tutorial by Rob Schapire
  - Tutorial by Yoav Freund
  - Slides by Carlos Guestrin
  - Tutorial by Paul Viola
  - Tutorial by Ron Meir
  - Slides by Aurélie Lemmens
  - Slides by Zhuowen Tu

Code

- Antonio Torralba (Object detection)
  - http://people.csail.mit.edu/torralba/shortCourseRLOC/boosting/boosting.html
- GML AdaBoost
  - http://graphics.cs.msu.ru/ru/science/research/machinelearning/adaboosttoolbox

Boosting

- Invented independently by Schapire (1989) and Freund (1990)
  - Later joined forces

- Main idea: train a strong classifier by combining weak classifiers
  - Practically useful
  - Theoretically interesting

Boosting

- Given a set of weak learners, run them multiple times on (***reweighted***) training data, then let learned classifiers vote
- At each iteration *t:*
  - Weight each training example by how ***incorrectly*** it was classified
  - Learn a hypothesis – h_t
    - The one with the smallest error
  - Choose a strength for this hypothesis – α_t
- Final classifier: weighted combination of weak learners

Learning from Weighted Data

- Sometimes not all data points are equal
  - Some data points are more equal than others
- Consider a weighted dataset
  - D(i) – weight of i th training example $(\mathbf{x_i}, \mathbf{y_i})$
  - Interpretations:
    - i th training example counts as D(i) examples
    - If I were to "resample" data, I would get more samples of "heavier" data points
- Now, in all calculations the *i th* training example counts as D(i) "examples"

Definition of Boosting

- Given training set $(\mathbf{x_1},\mathbf{y_1}),\ldots, (\mathbf{x_m},\mathbf{y_m})$
- $y_i \in \{-1,+1\}$ correct label of instance $x_i \in X$
- For t=1,…,T
  - construct distribution D_t on {1,…,m}
  - find weak hypothesis
  - h_t: X→ {-1,+1} with small error ε_t on D_t

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

- Output final hypothesis H_{final}

AdaBoost

- Constructing D_t
  - D_1=1/m
  - Given D_t and h_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \cdot \exp(-\alpha_t\, y_i\, h_t(x_i))$$

where Z_t is a normalization constant

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

$$\alpha_t = \tfrac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$$

- Final hypothesis:

$$H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$$

The AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

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

The AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$. ← with minimum ϵ_t
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

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

The AdaBoost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

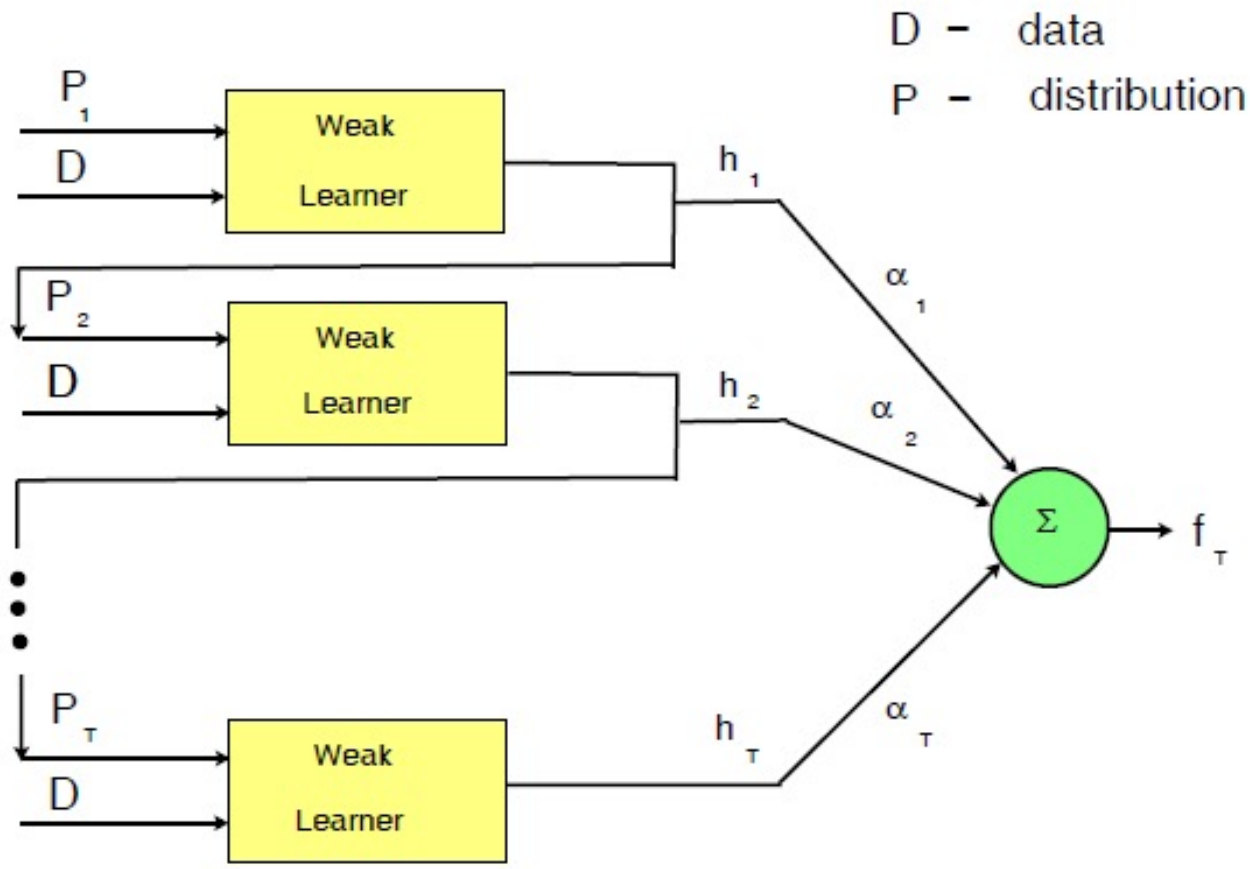
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

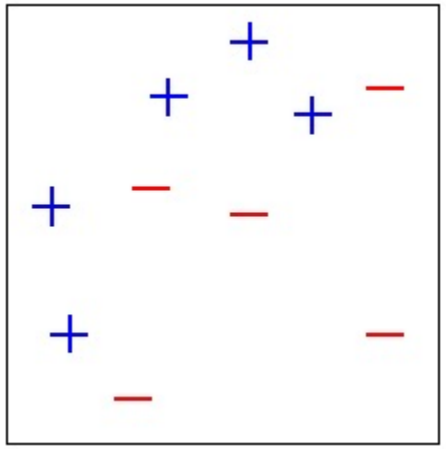
$$\epsilon_t = \frac{1}{\sum_{i=1}^{n} D_t(i)} \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)$$

The AdaBoost Algorithm

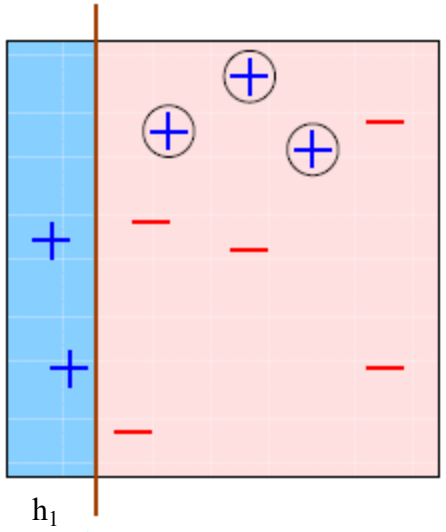


Toy Example

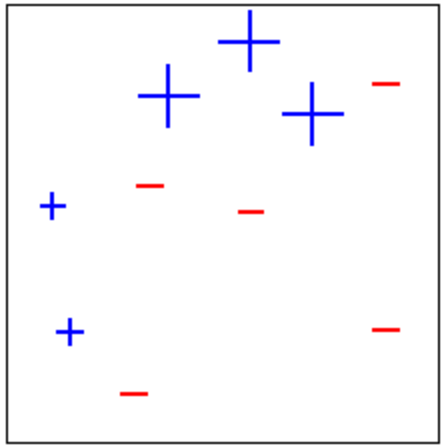
D_1



Toy Example: Round 1

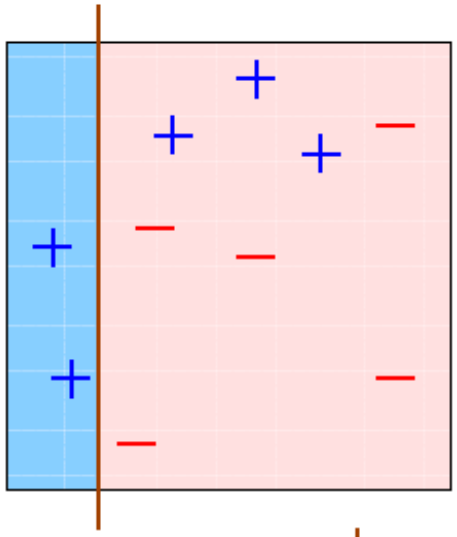


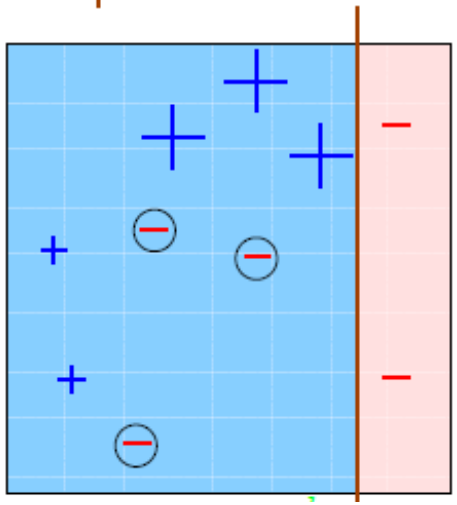
ε_1=0.3
α_1=0.42



D_2

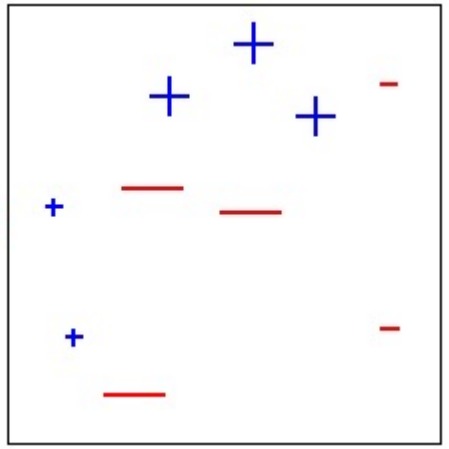
Toy Example: Round 2





h_2

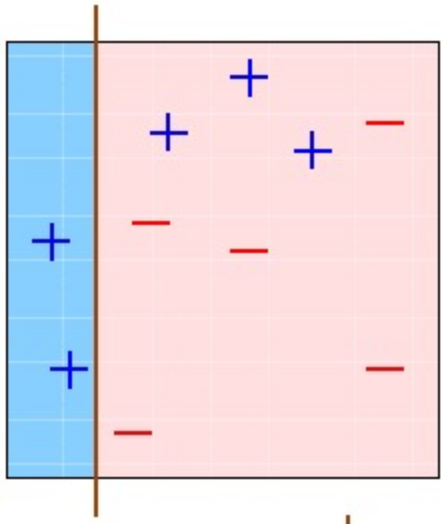
D_3

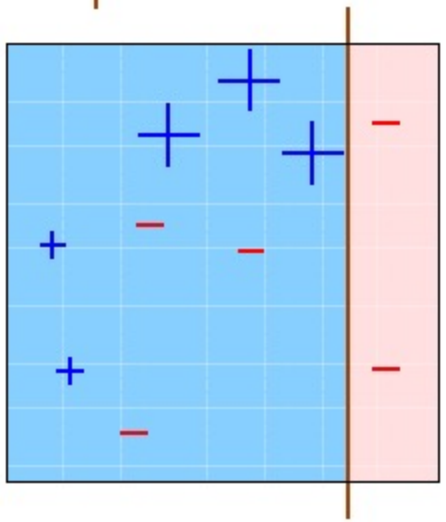


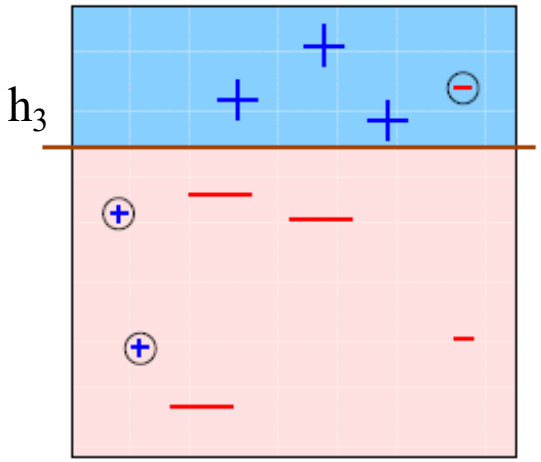
$\varepsilon_2=0.21$

$\alpha_2=0.65$

Toy Example: Round 3

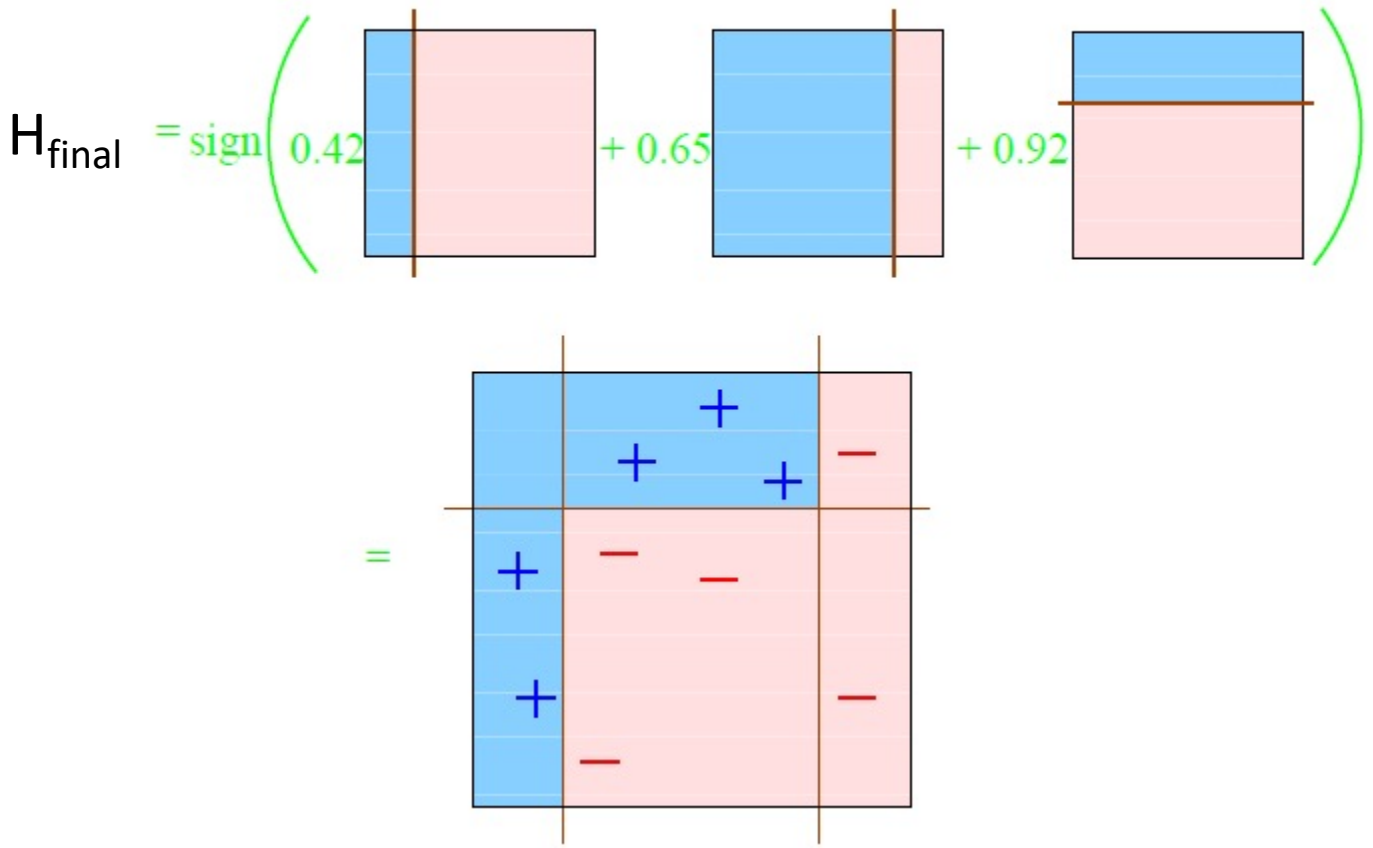






ε_3=0.14
α_3=0.92

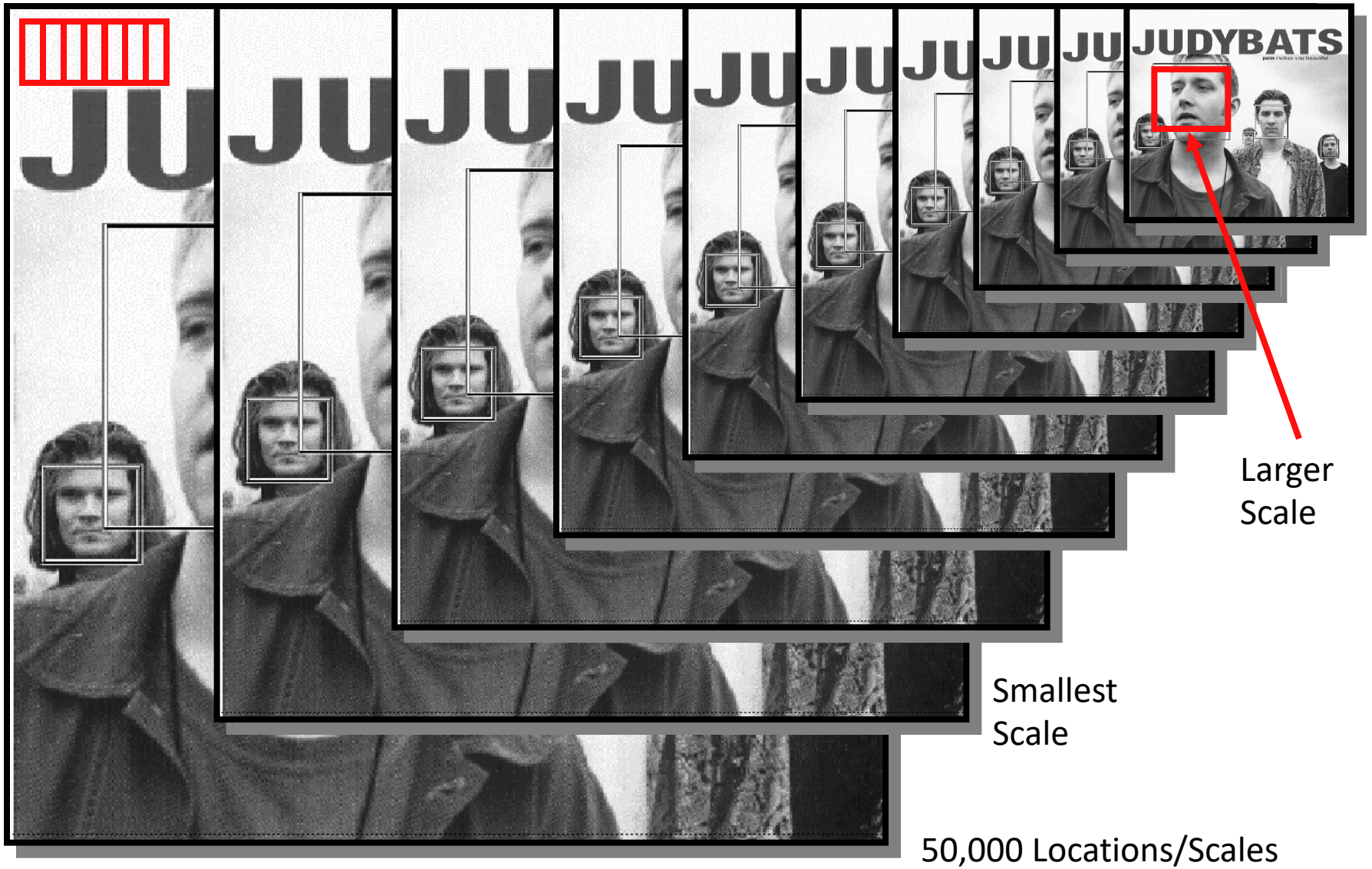
Toy Example: Final Hypothesis



Applications of Boosting
Viola-Jones Face Detector

Real time face detection using a classifier cascade [Viola and Jones, 2001 and 2004]

The Classical Face Detection Process



Classifier is Trained on Labeled Data

- Training Data
  - 5000 faces
    - All frontal
  - 10^8 non faces
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose (rotation both in plane and out)

Classifier Cascade (Viola-Jones)

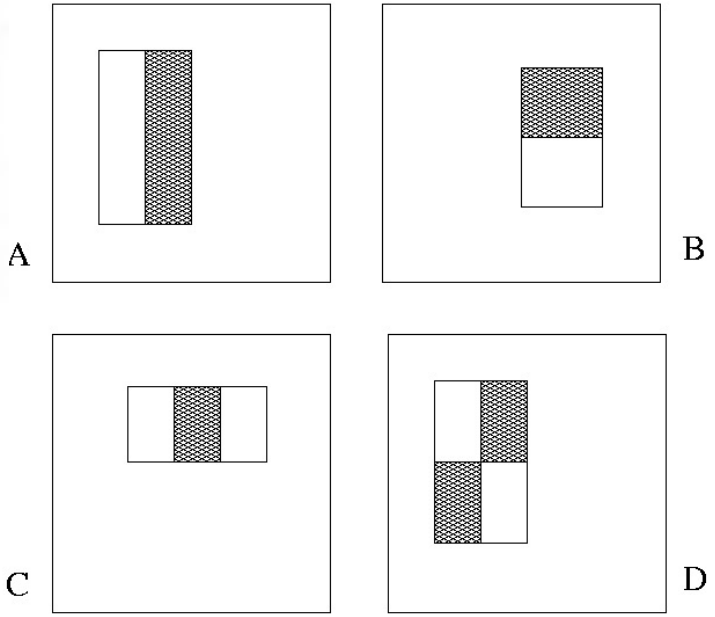
- For real problems results are only as good as the features used...
  - This is the main piece of ad-hoc (or domain) knowledge

- Rather than the pixels, use a very large set of simple functions
  - Sensitive to edges and other critical features of the image
  - Computed at multiple scales

- Introduce a threshold to yield binary features
  - Binary features seem to work better in practice
  - In general, convert continuous features to binary by quantizing

Boosted Face Detection: Image Features

"Rectangle filters"



$$h_t(x_i) = \begin{cases} \alpha_t & \text{if } f_t(x_i) > \theta_t \\ \beta_t & \text{otherwise} \end{cases}$$

$$C(x) = \theta\left(\sum_t h_t(x) + b\right)$$

Feature Selection

- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter
  - Select best filter/threshold (= Feature)
  - Reweight examples

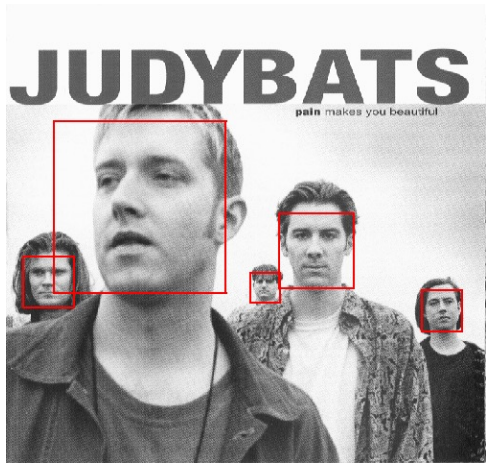
Building Fast Classifiers

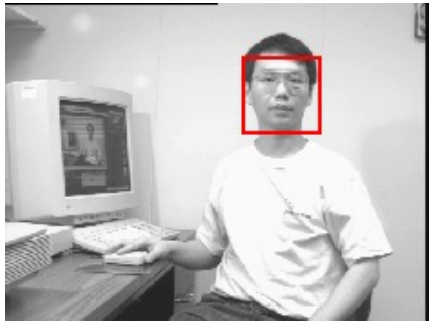
- In general, simple classifiers are more efficient, but they are also weaker

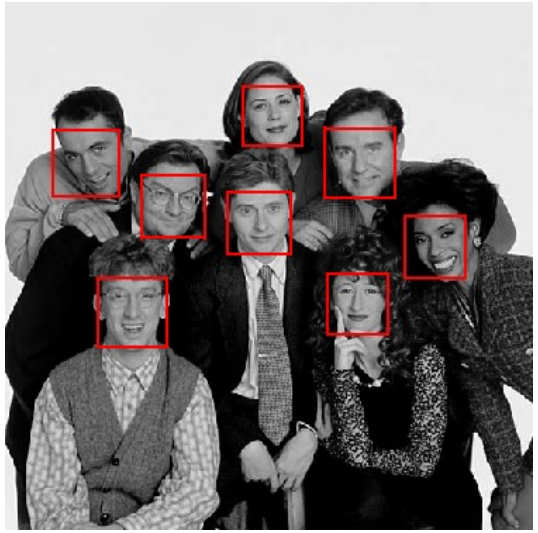
- We could define a computational risk hierarchy
  - A nested set of classifier classes

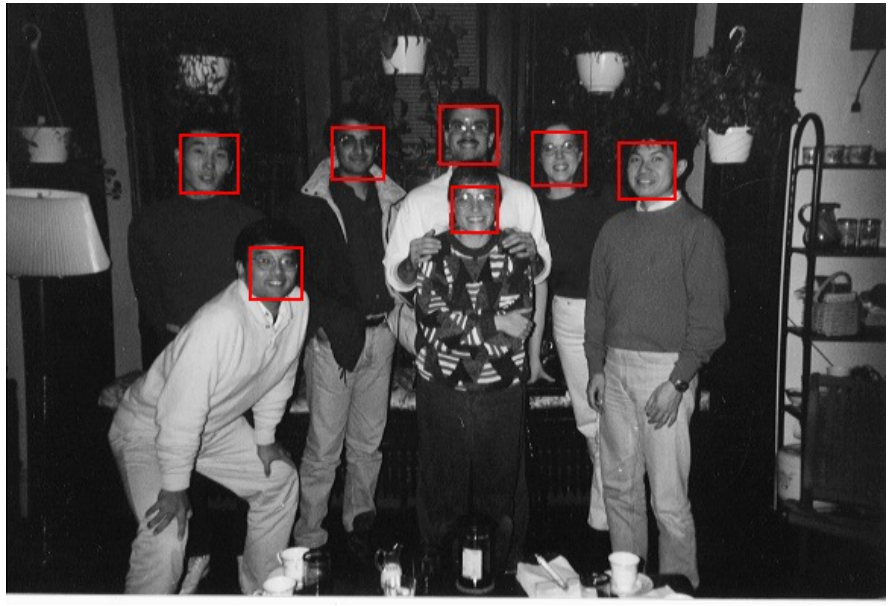
- The training process is reminiscent of boosting…
  - Previous classifiers reweight the examples used to train subsequent classifiers

Output of Face Detector on Test Images



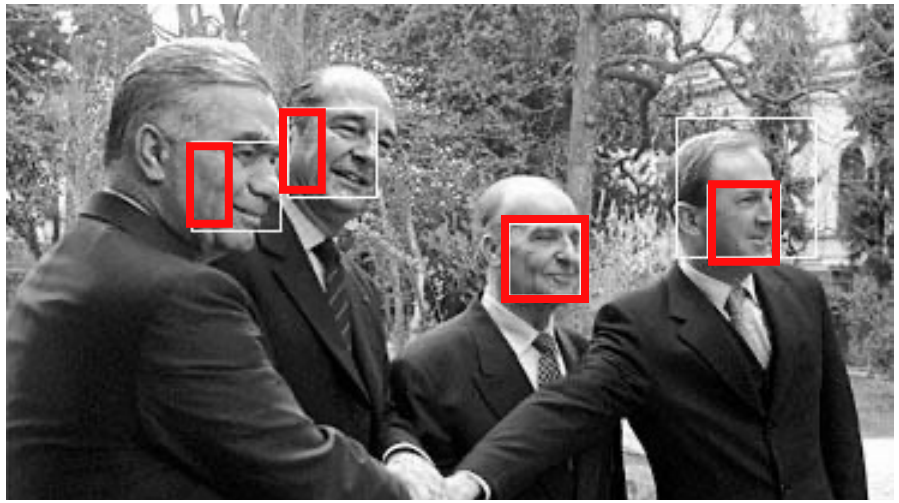






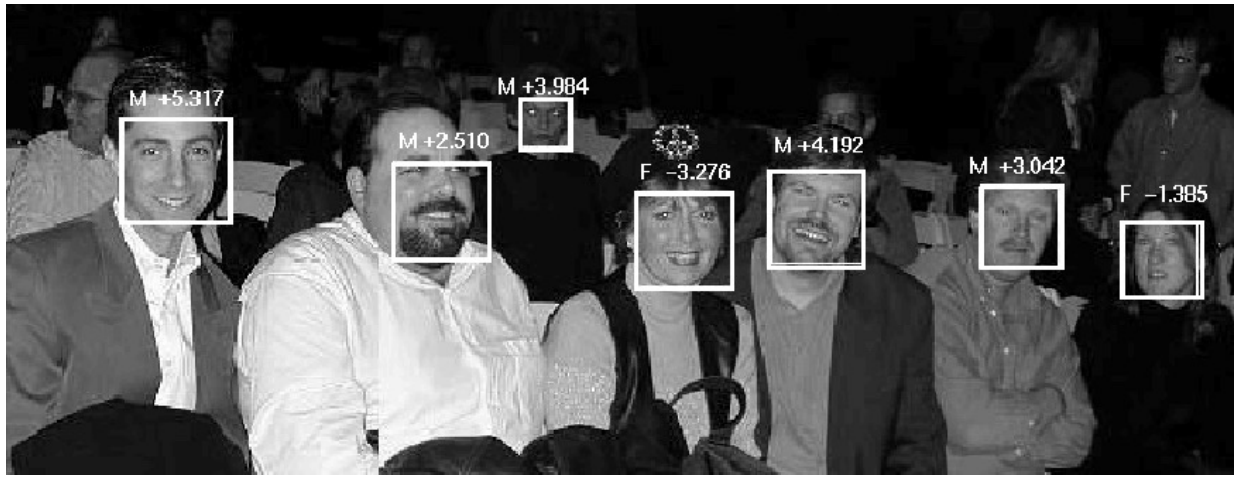
Solving other “Face” Tasks

Facial Feature Localization



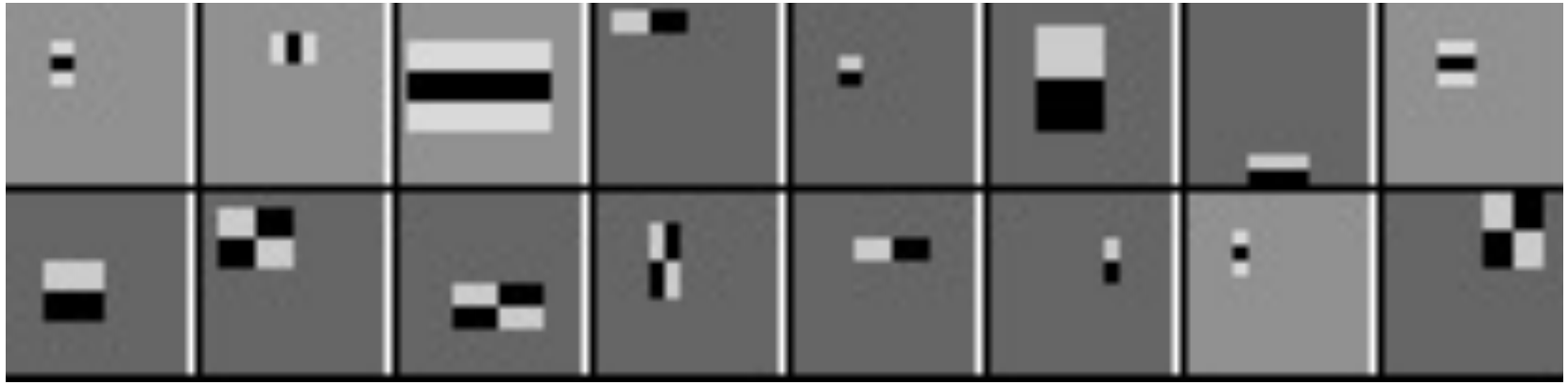
Profile Detection

Demographic Analysis

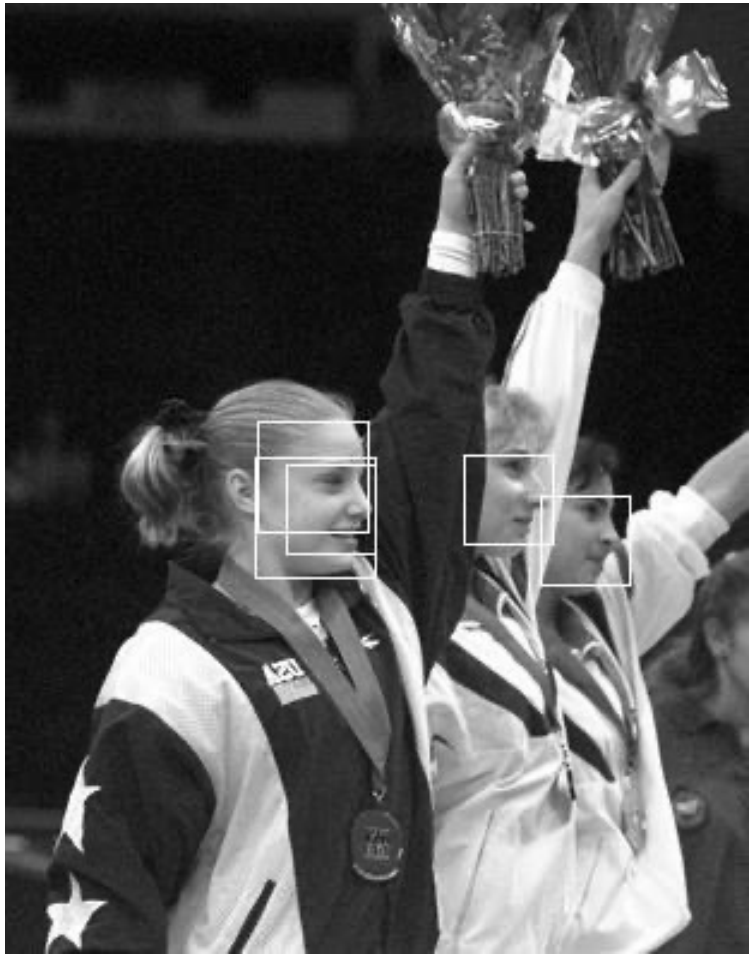


Feature Localization

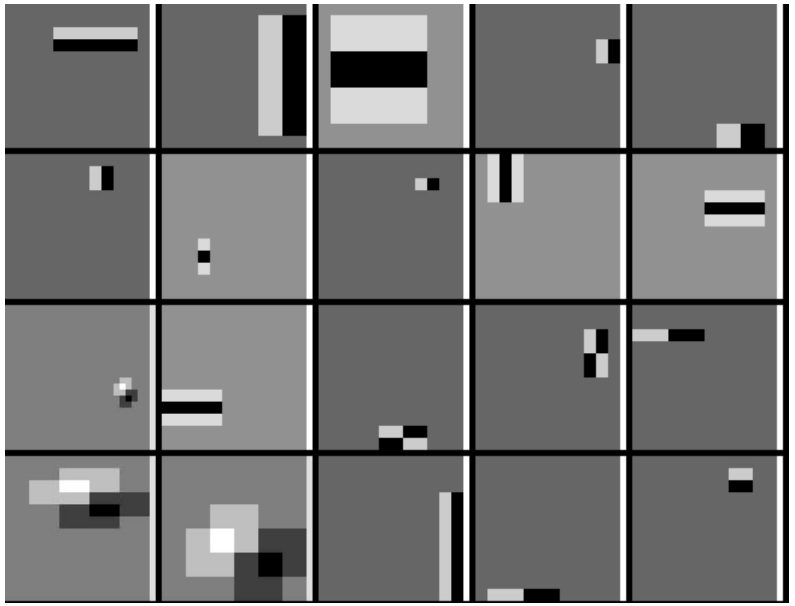
- Learned features reflect the task



Profile Detection



Profile Features



Summary…

- Bagging
  - Random Forest/Decision Forest
- Boosting
  - Viola-Jones Face Detector