CS 559: Machine Learning Fundamentals and Applications 10th Set of Notes

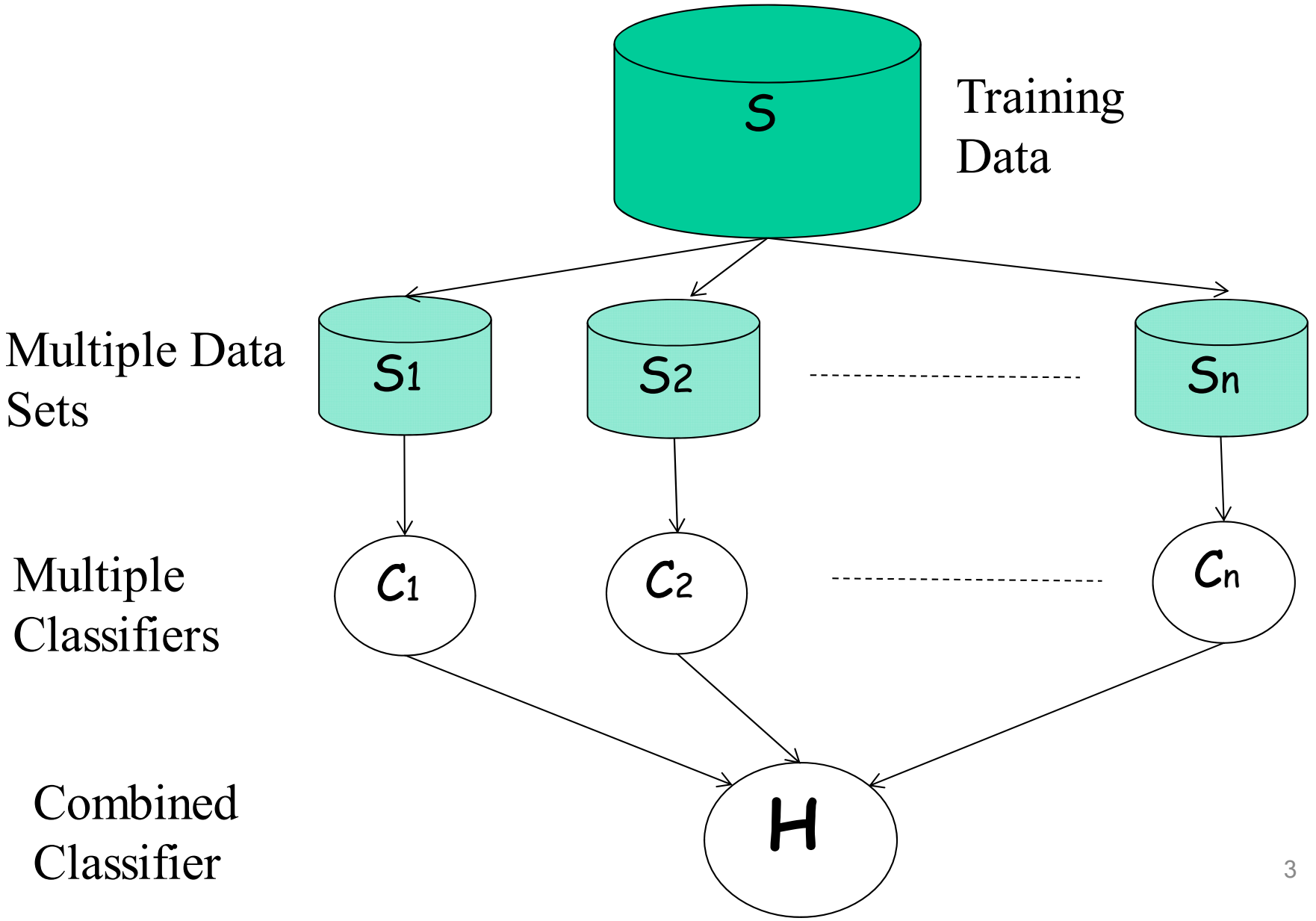
Instructor: Philippos Mordohai
Webpage: www.cs.stevens.edu/~mordohai
E-mail: Philippos.Mordohai@stevens.edu
Office: Lieb 215

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



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:

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

Bagging

- 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
  - In some pathological hypothetical situations the overall error might increase
  - 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

- Bagging and Boosting
  - ➔Aggregating Classifiers

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

Bagging

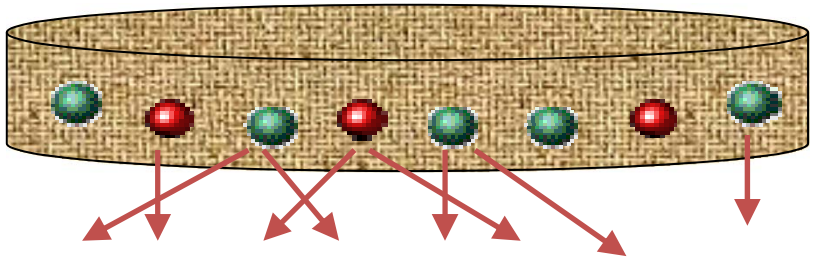
- *Bagging* = Bootstrap Aggregating
- Reweighing of the learning sets is done by drawing at random with replacement from the learning sets
- Predictors are aggregated by plurality voting

The Bagging Algorithm

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

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

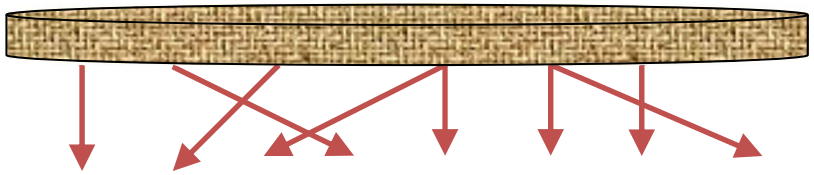
Weighting



Initial set

Drawing with replacement 1

Classifier 1



Drawing with replacement 2

Classifier 2

Aggregation

Sign

Classifier 1

+

Classifier 2

+

Classifier 3

+

…

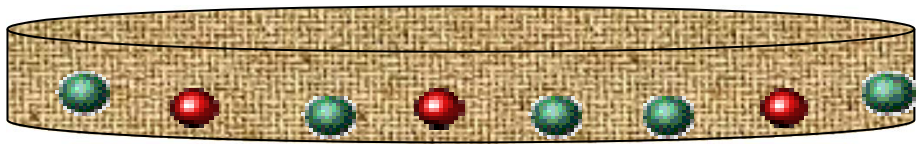
+

Classifier T

=

Final rule

Initial set

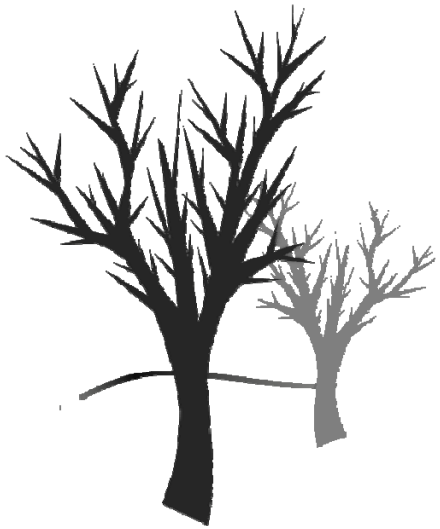


Random Forests

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

Decision Forests

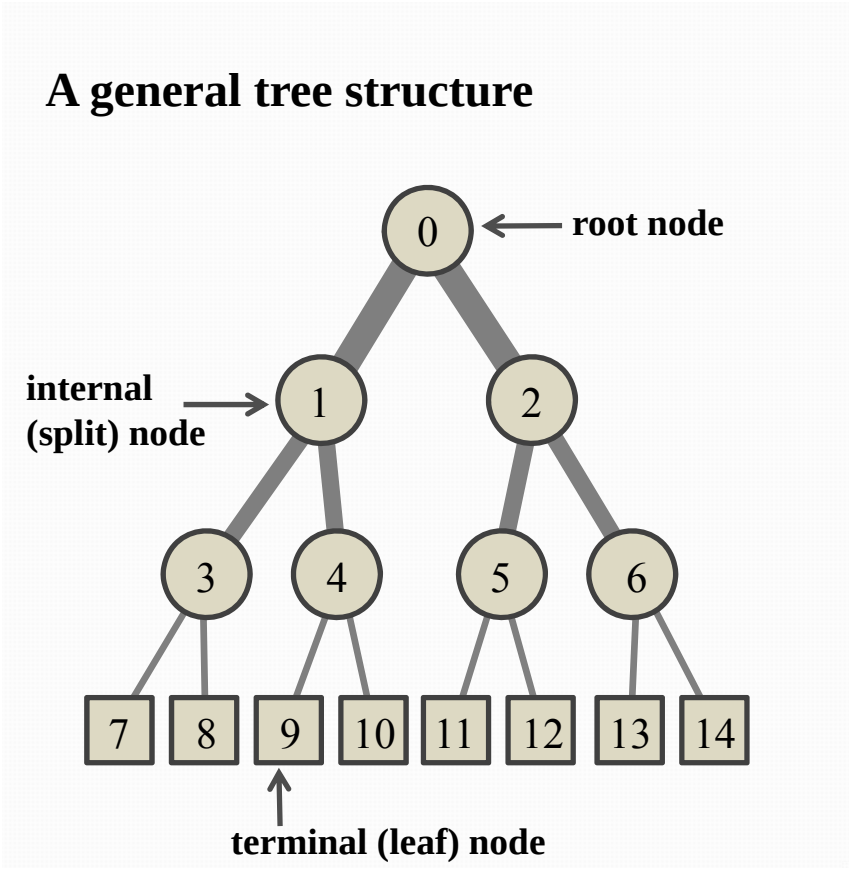
for computer vision and medical image analysis

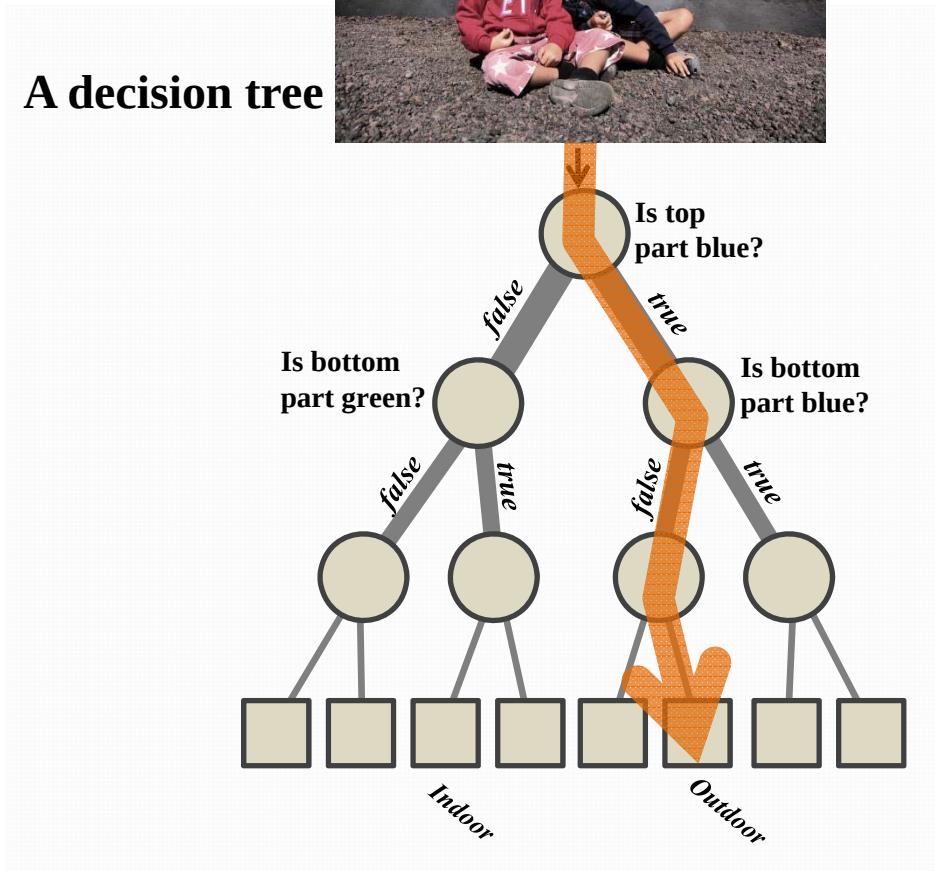


A. Criminisi, J. Shotton and E. Konukoglu

http://research.microsoft.com/projects/decisionforests

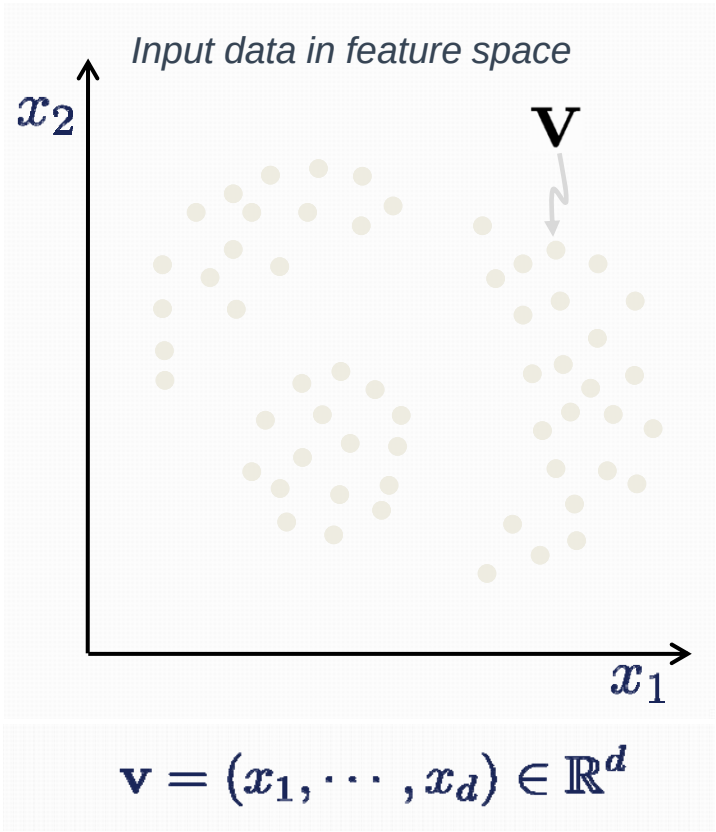
Decision Trees and Decision Forests



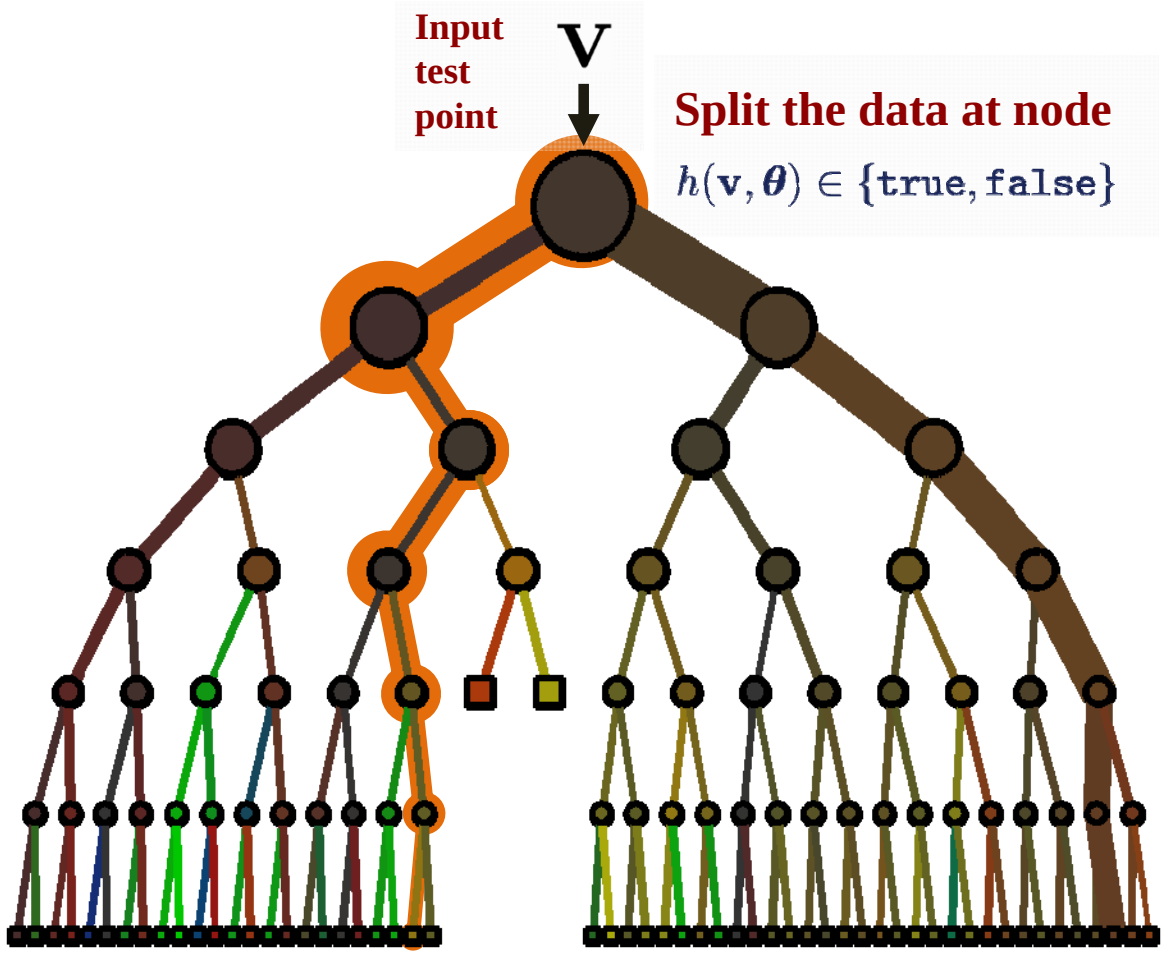


A forest is an ensemble of trees. The trees are all slightly different from one another

Decision Tree Testing (Runtime)



$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$



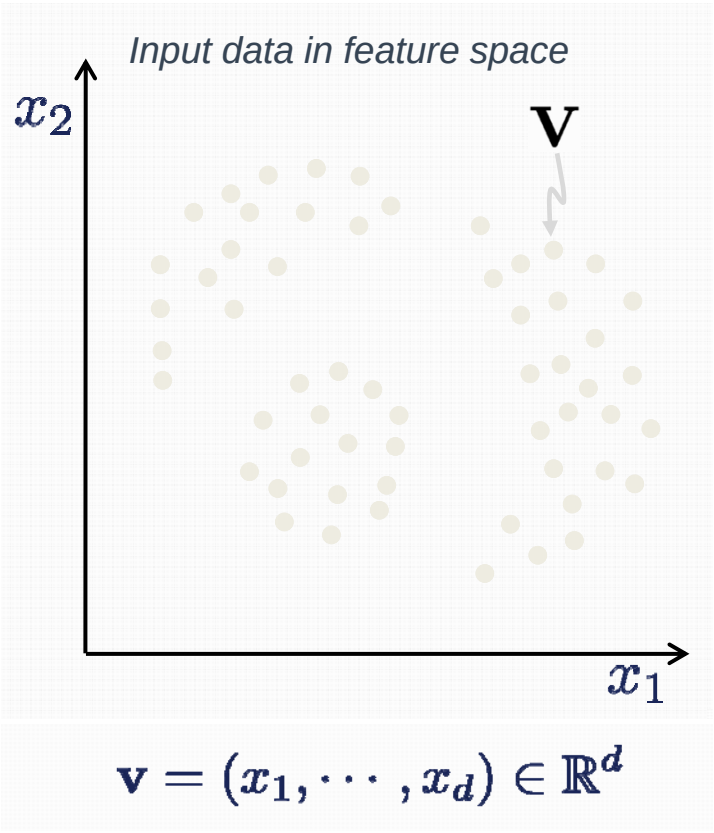
Split the data at node

$$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\texttt{true}, \texttt{false}\}$$

Prediction at leaf

$$p(c|\mathbf{v}) = \sum_j p(c|j)p(j|\mathbf{v})$$

Decision Tree Training (Offline)



$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$

Input training data

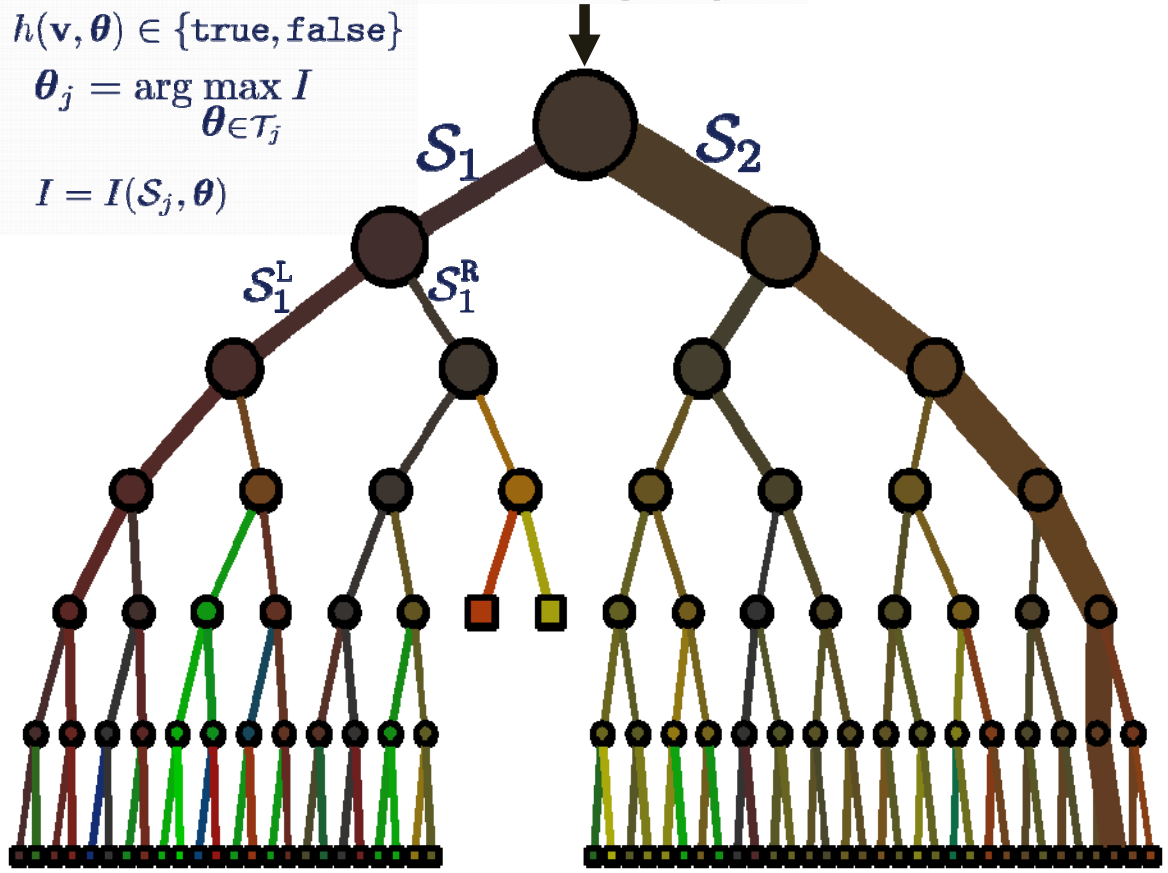
$$\mathcal{S}_0 = \{\mathbf{v}, c\}$$

How to split the data?

$$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\texttt{true}, \texttt{false}\}$$

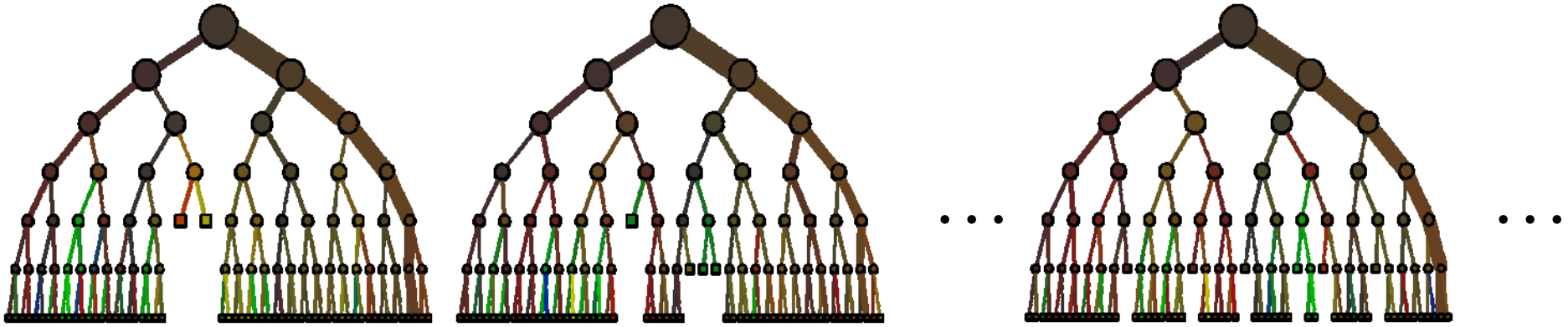
$$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta} \in \mathcal{T}_j} I$$

$$I = I(\mathcal{S}_j, \boldsymbol{\theta})$$



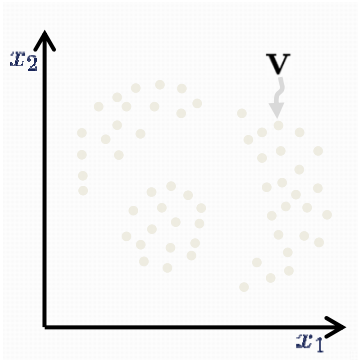
Binary tree? Ternary?
How big a tree?
What tree structure?

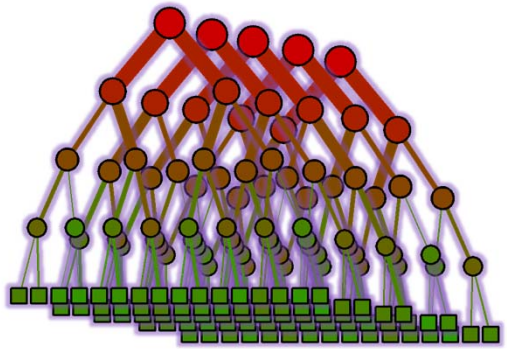
Decision Tree Training (Offline)



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

Decision 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		$\boldsymbol{\phi}$	*Features can be e.g. wavelets? Pixel intensities? Context?*

Forest model

tree

Node test parameters		$\boldsymbol{\theta} \in \mathcal{T}$	*Parameters related to each split node: i) which features, ii) what geometric primitive, iii) thresholds.*
Node objective function (train.)	*e.g.*	$I = I(\mathcal{S}_j, \boldsymbol{\theta})$	*The "energy" to be minimized when training the j-th split node*
Node weak learner	*e.g.*	$h(\mathbf{v}, \boldsymbol{\theta}_j) \in \{\mathtt{true}, \mathtt{false}\}$	*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?*

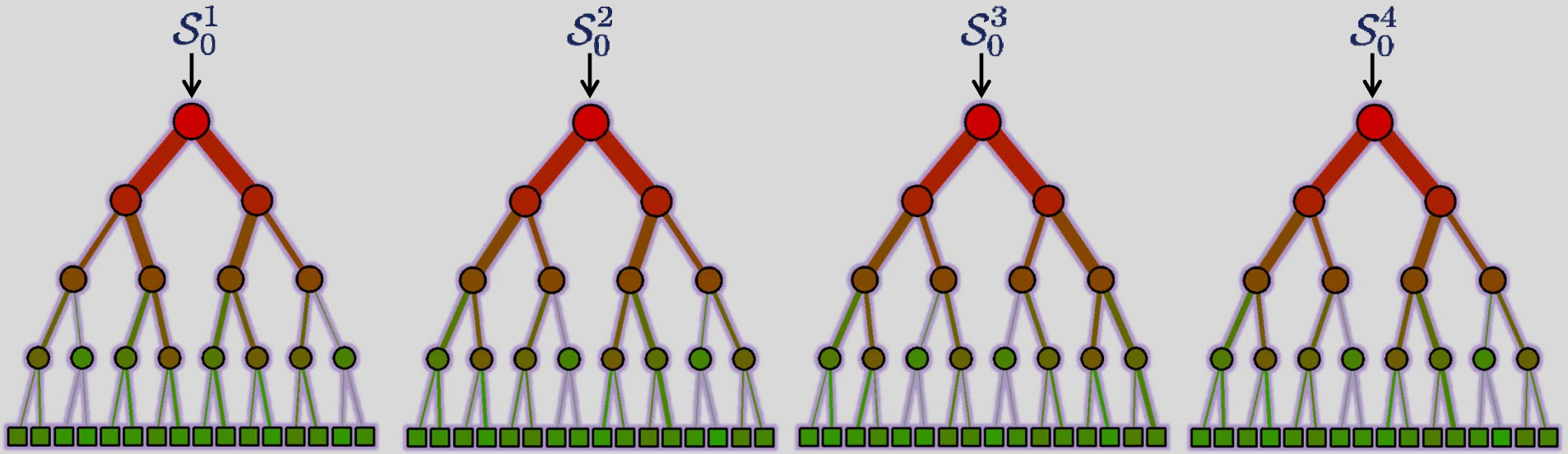
Randomness Model

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

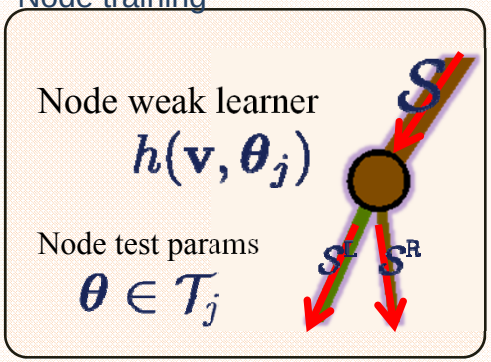
2) Randomized node optimization (RNO)

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

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

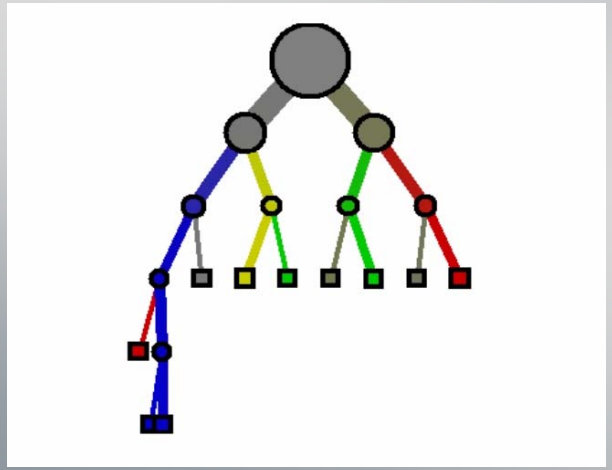
$\rho = |\mathcal{T}_j|$ Randomness control parameter.
For $\rho = |\mathcal{T}|$ no randomness and maximum tree correlation.
For $\rho = 1$ max randomness and minimum tree correlation.

Node training

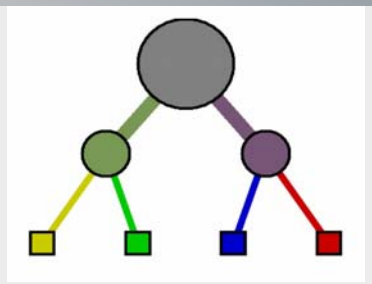


The effect of ρ

Small value of ρ; little tree correlation.

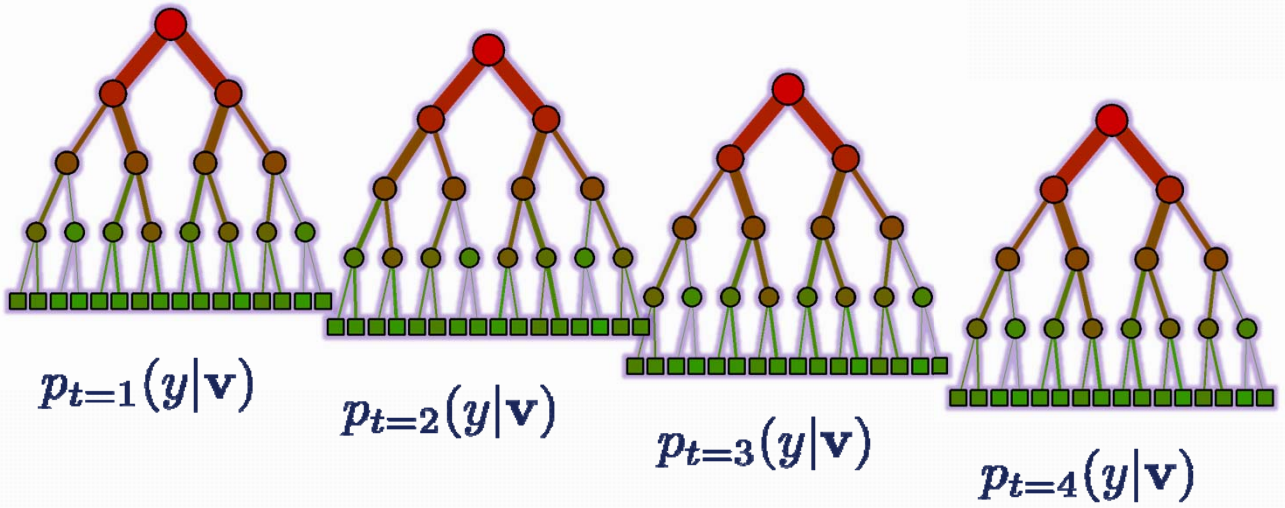


Large value of ρ; large tree correlation.

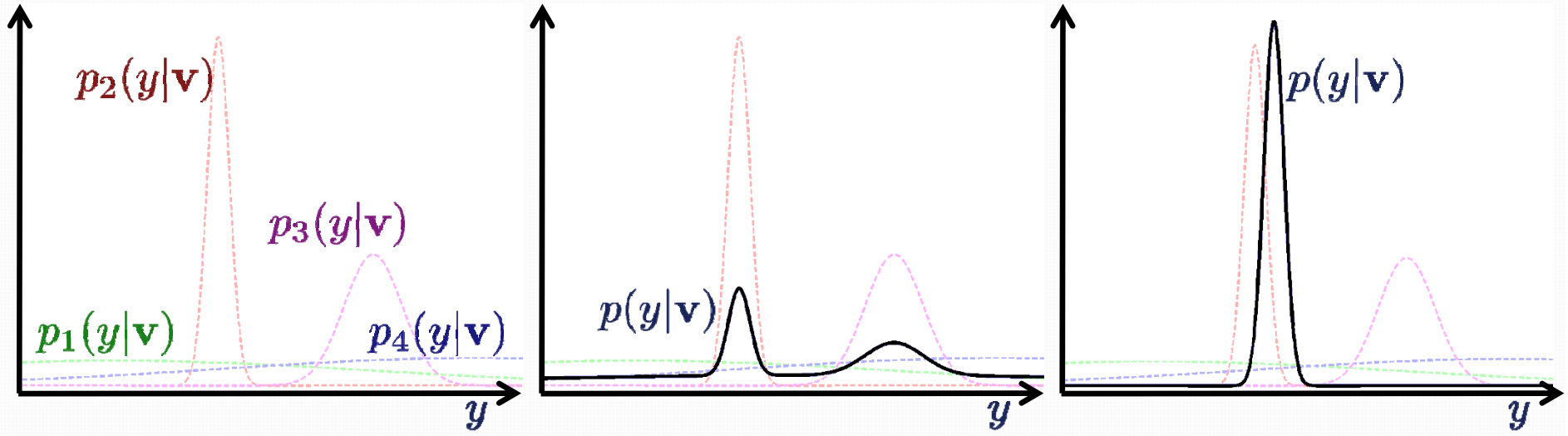


The Ensemble Model

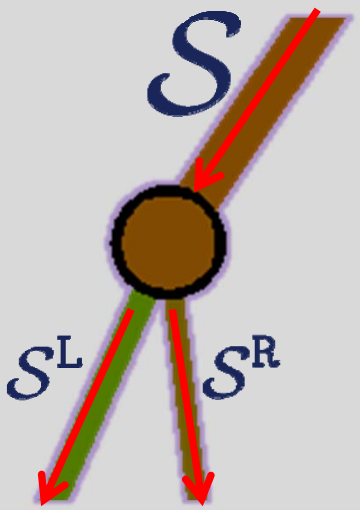
An example forest to predict continuous variables



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



Training and Information Gain



Information gain

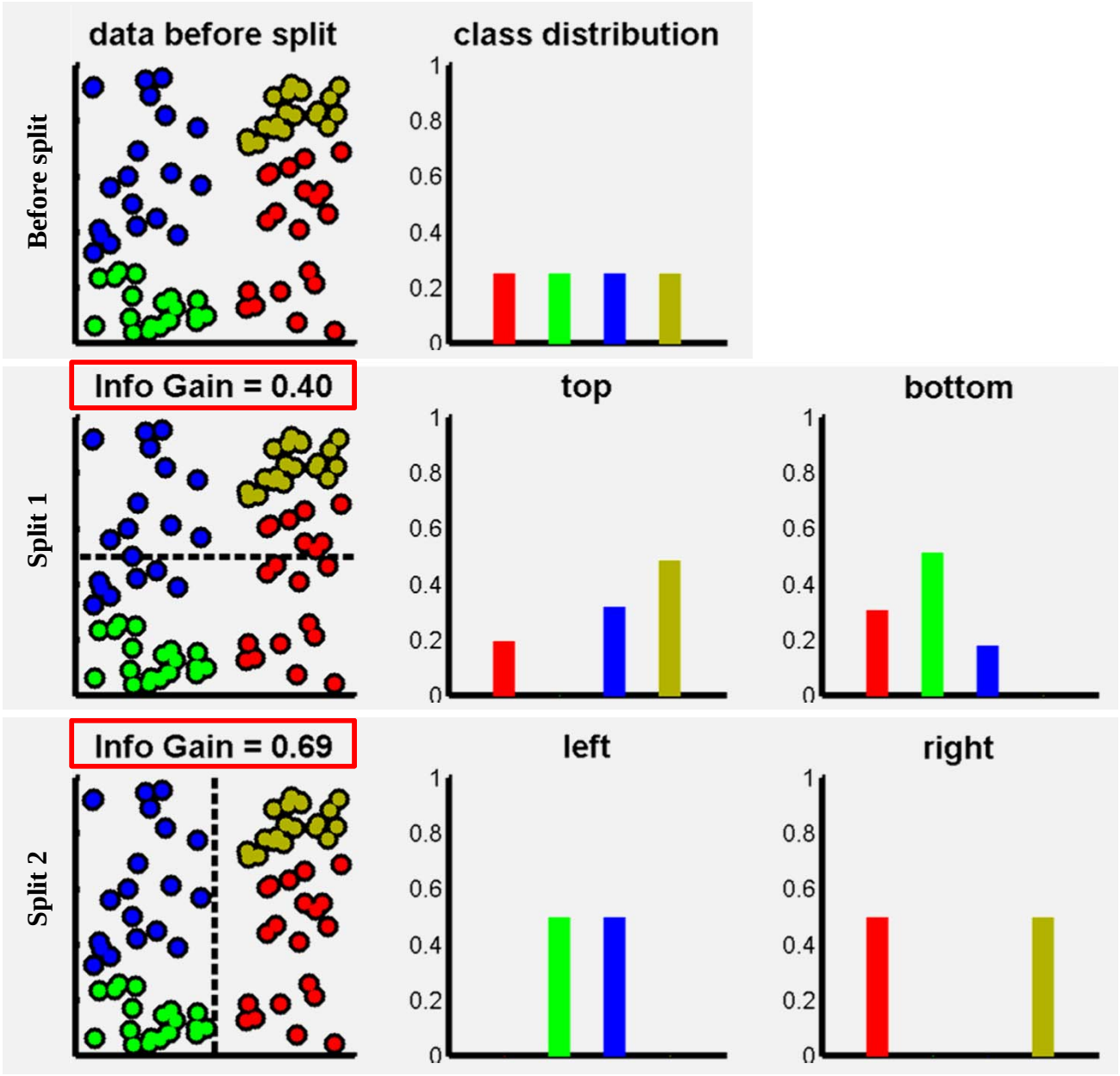
$$I(\mathcal{S},\boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i\in\{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Shannon's entropy

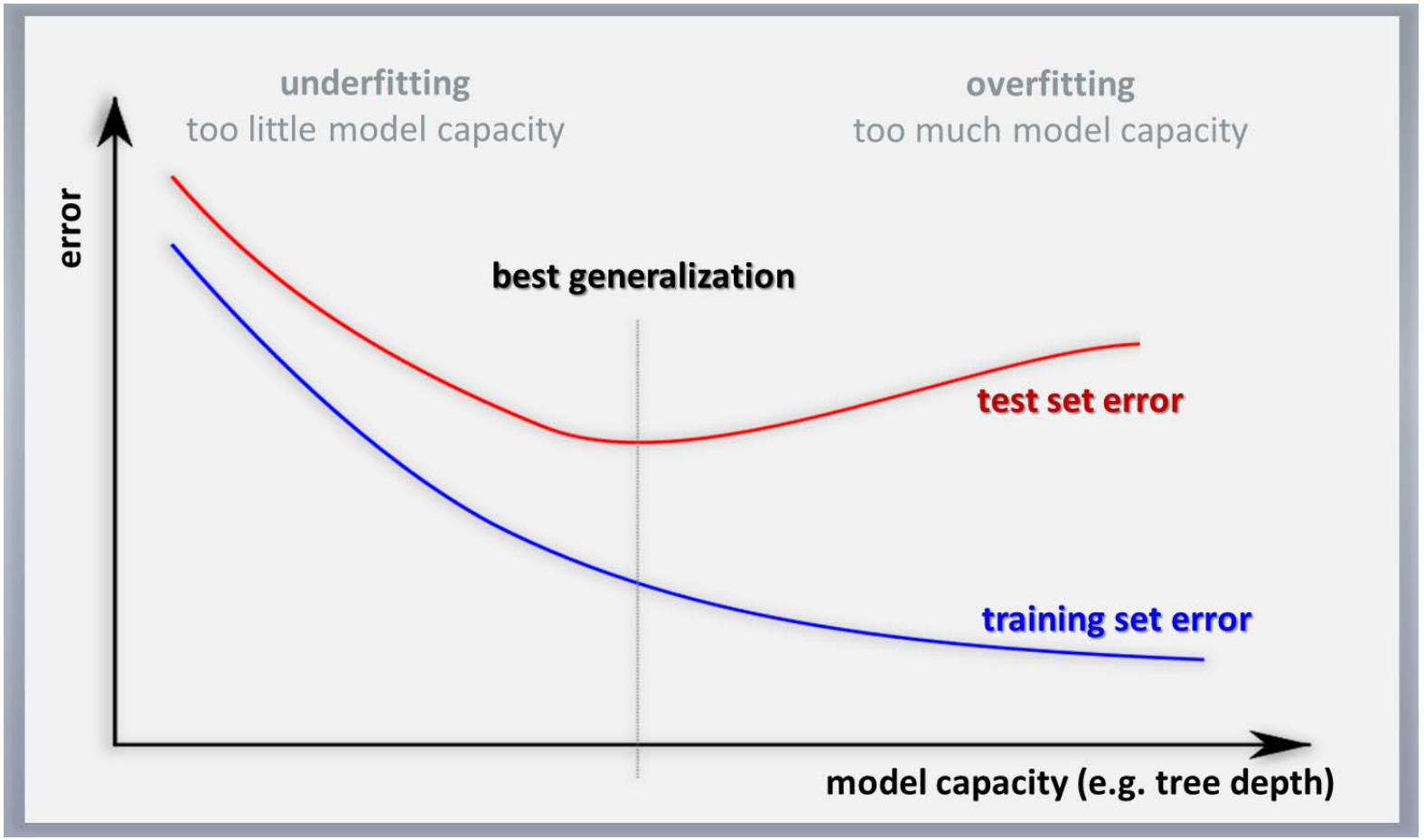
$$H(\mathcal{S}) = -\sum_{c\in\mathcal{C}} p(c)\log(p(c))$$

Node training

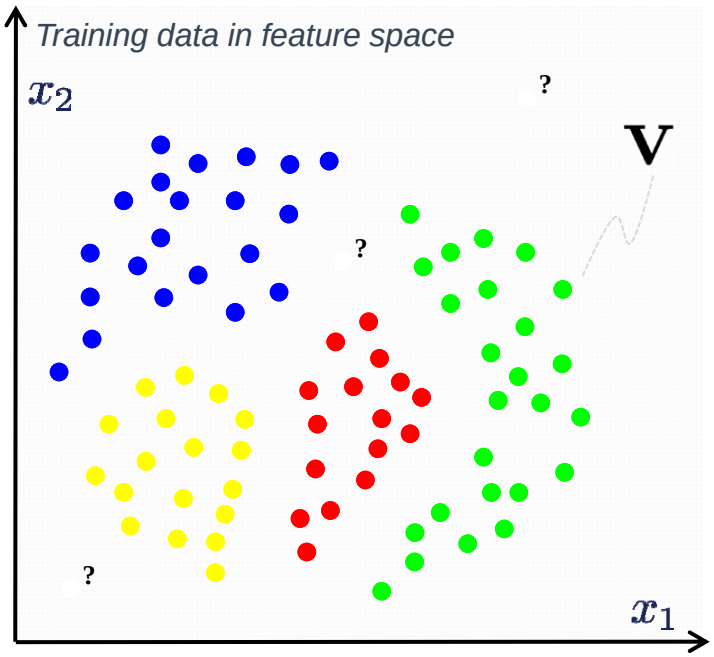
$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(\mathcal{S}_j,\boldsymbol{\theta})$$

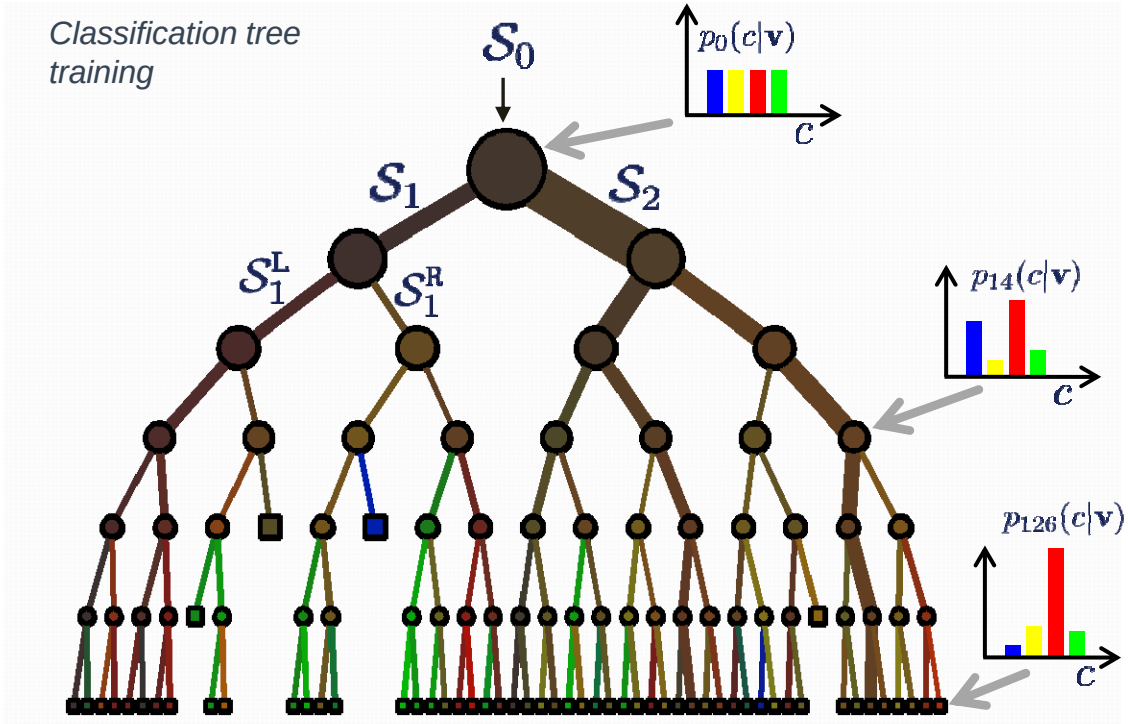


Overfitting and Underfitting



Classification Forest





Model specialization for classification

Input data point	$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$	($\boldsymbol{x_i}$ *is feature response)*
Output is categorical	$c \in \mathcal{C}$ with $\mathcal{C} = \{c_k\}$	*(discrete set)*
Node weak learner	$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\mathtt{true}, \mathtt{false}\}$	
Obj. funct. for node j	$I = H(\mathcal{S}_j) - \sum_{i=\mathrm{L,R}} \frac{\lvert\mathcal{S}_j^i\rvert}{\lvert\mathcal{S}_j\rvert} H(\mathcal{S}_j^i)$	*(information gain)*
Training node j	$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta} \in \mathcal{T}_j} I(\mathcal{S}_j, \boldsymbol{\theta})$	
Predictor model	$p(c\|\mathbf{v}) = \sum_j p(c\|j)p(j\|\mathbf{v})$	*(class posterior)*

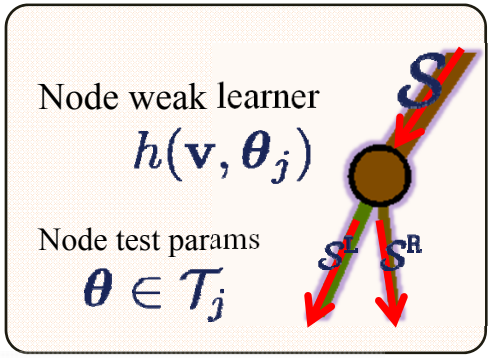
Entropy of a discrete distribution

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c))$$

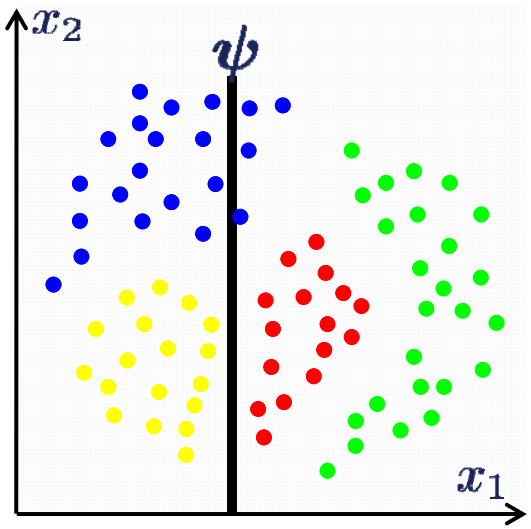
with $c(\mathbf{v}) : \mathbb{R}^d \rightarrow \mathcal{C}$

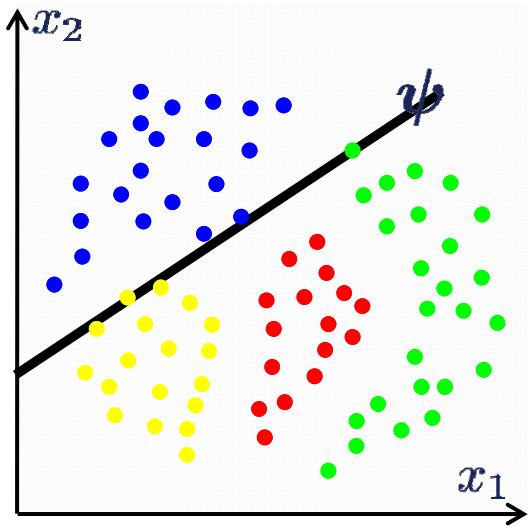
Weak Learners

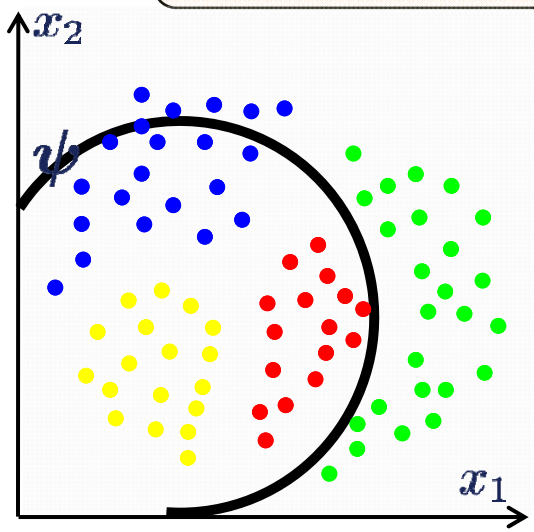
Splitting data at node j



Examples of weak learners







Weak learner: axis aligned

$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \boldsymbol{\phi}(\mathbf{v}) \cdot \boldsymbol{\psi} > \tau_2]$$

Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \; x_2 \; 1)^\top$

With $\boldsymbol{\psi} = (1 \; 0 \; \psi_3)$ or $\boldsymbol{\psi} = (0 \; 1 \; \psi_3)$

Weak learner: oriented line

$$h(\mathbf{v}, \boldsymbol{\theta}) = [\tau_1 > \boldsymbol{\phi}(\mathbf{v}) \cdot \boldsymbol{\psi} > \tau_2]$$

Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \; x_2 \; 1)^\top$

With $\boldsymbol{\psi} \in \mathbb{R}^3$ a generic line in homog. coordinates.

Weak learner: conic section

$$h(\mathbf{v}, \boldsymbol{\theta}) = \left[\tau_1 > \boldsymbol{\phi}^\top(\mathbf{v}) \; \boldsymbol{\psi} \; \boldsymbol{\phi}(\mathbf{v}) > \tau_2\right]$$

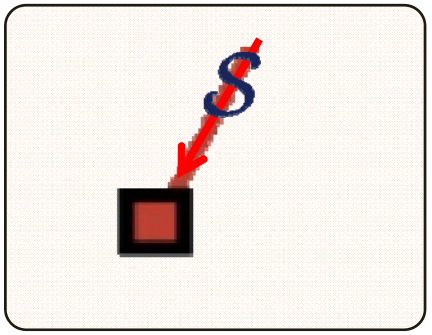
Feature response for 2D example. $\boldsymbol{\phi}(\mathbf{v}) = (x_1 \; x_2 \; 1)^\top$

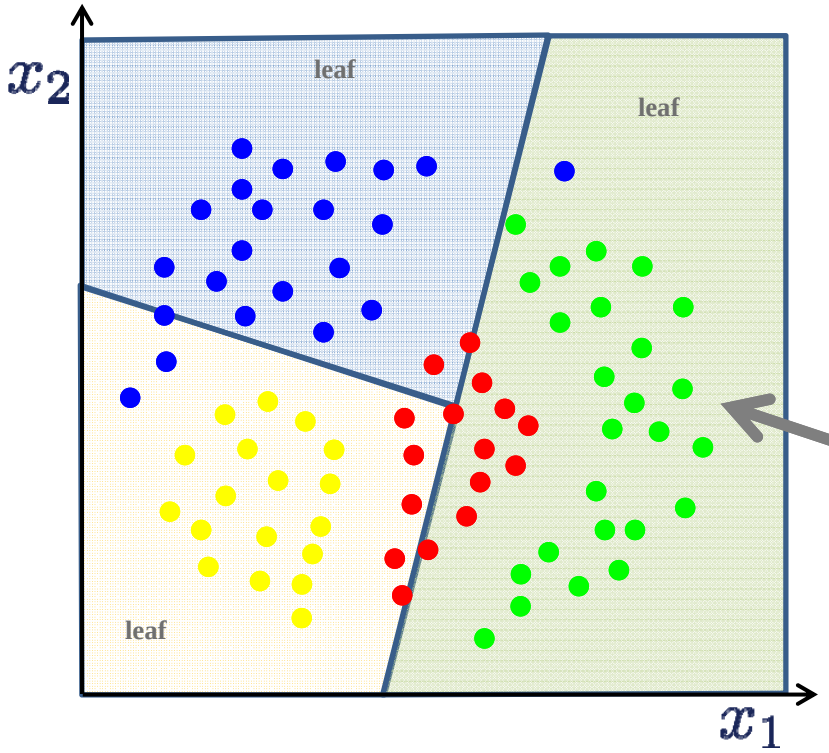
With $\boldsymbol{\psi} \in \mathbb{R}^{3\times3}$ a matrix representing a conic.

In general $\boldsymbol{\phi}$ may select only a very small subset of features $\boldsymbol{\phi}(\mathbf{v}) : \mathbb{R}^d \rightarrow \mathbb{R}^{d'+1}, \; d' << d$

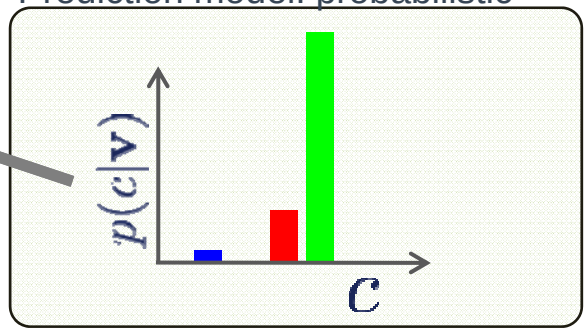
Prediction Model

What do we do at the leaf?

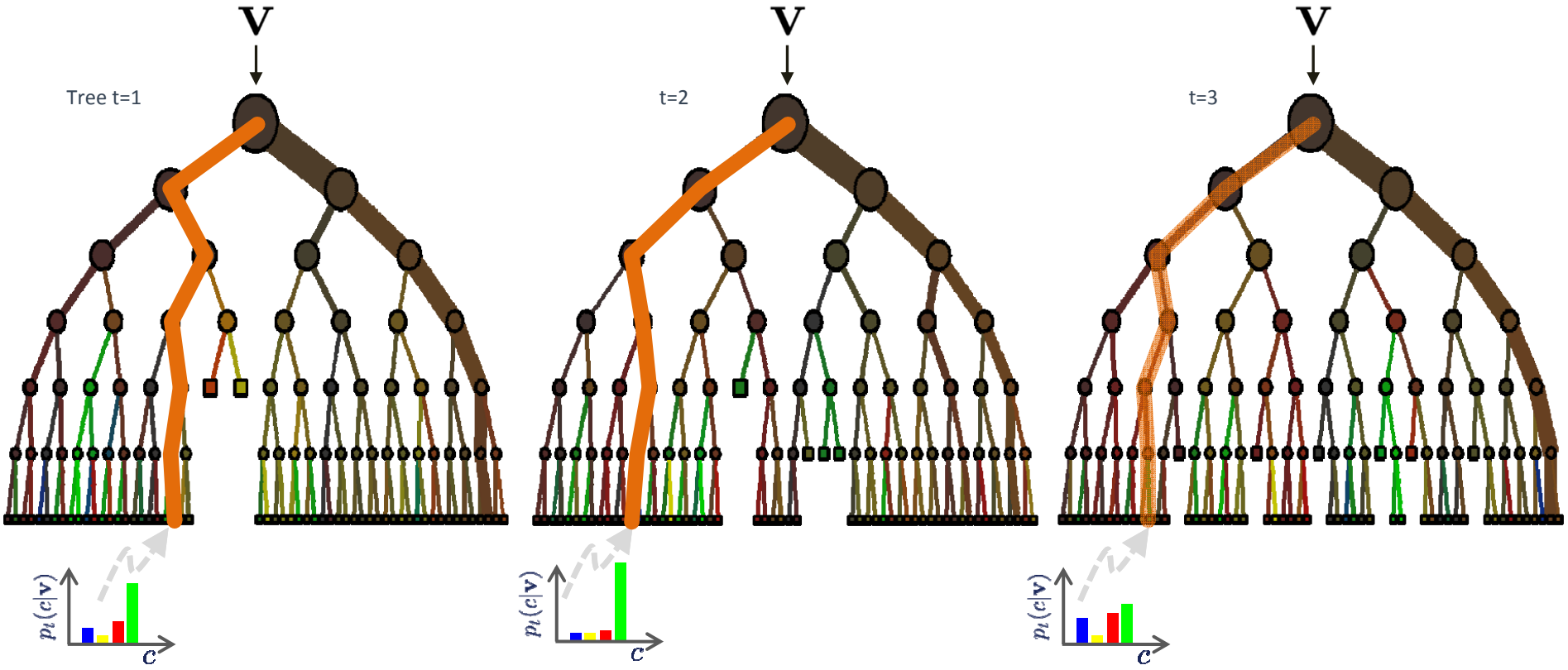




Prediction model: probabilistic

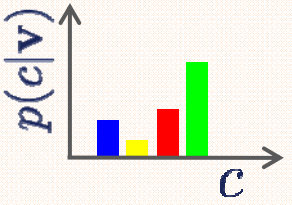


Classification Forest: Ensemble Model

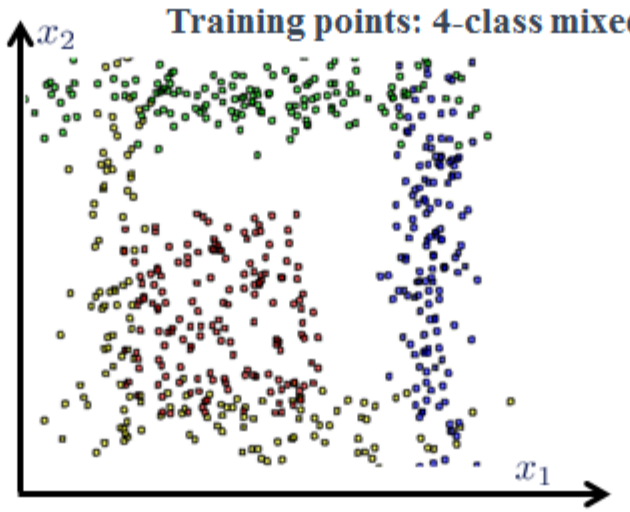


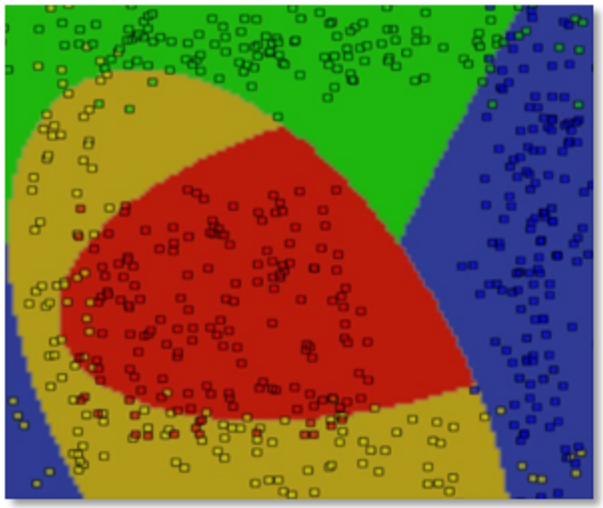
The ensemble model

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

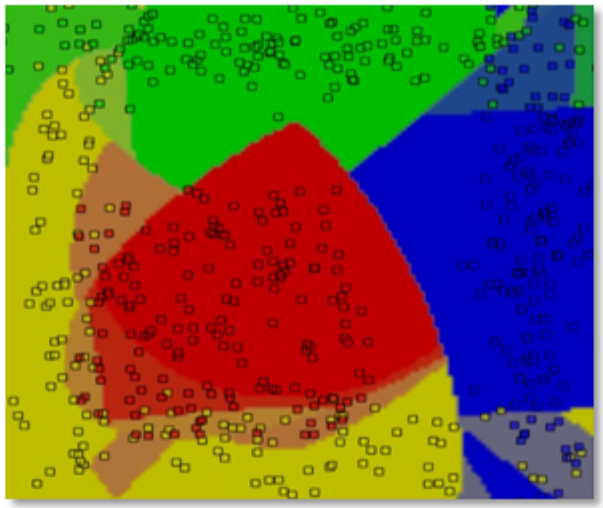


Effect of Tree Depth

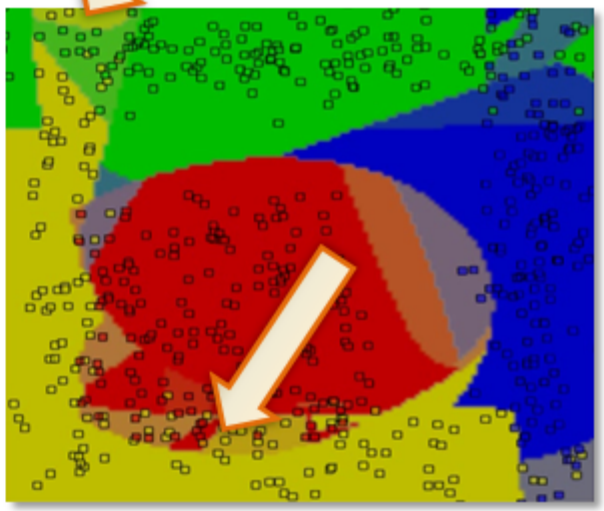




T=200, D=3, w. l. = conic



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



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

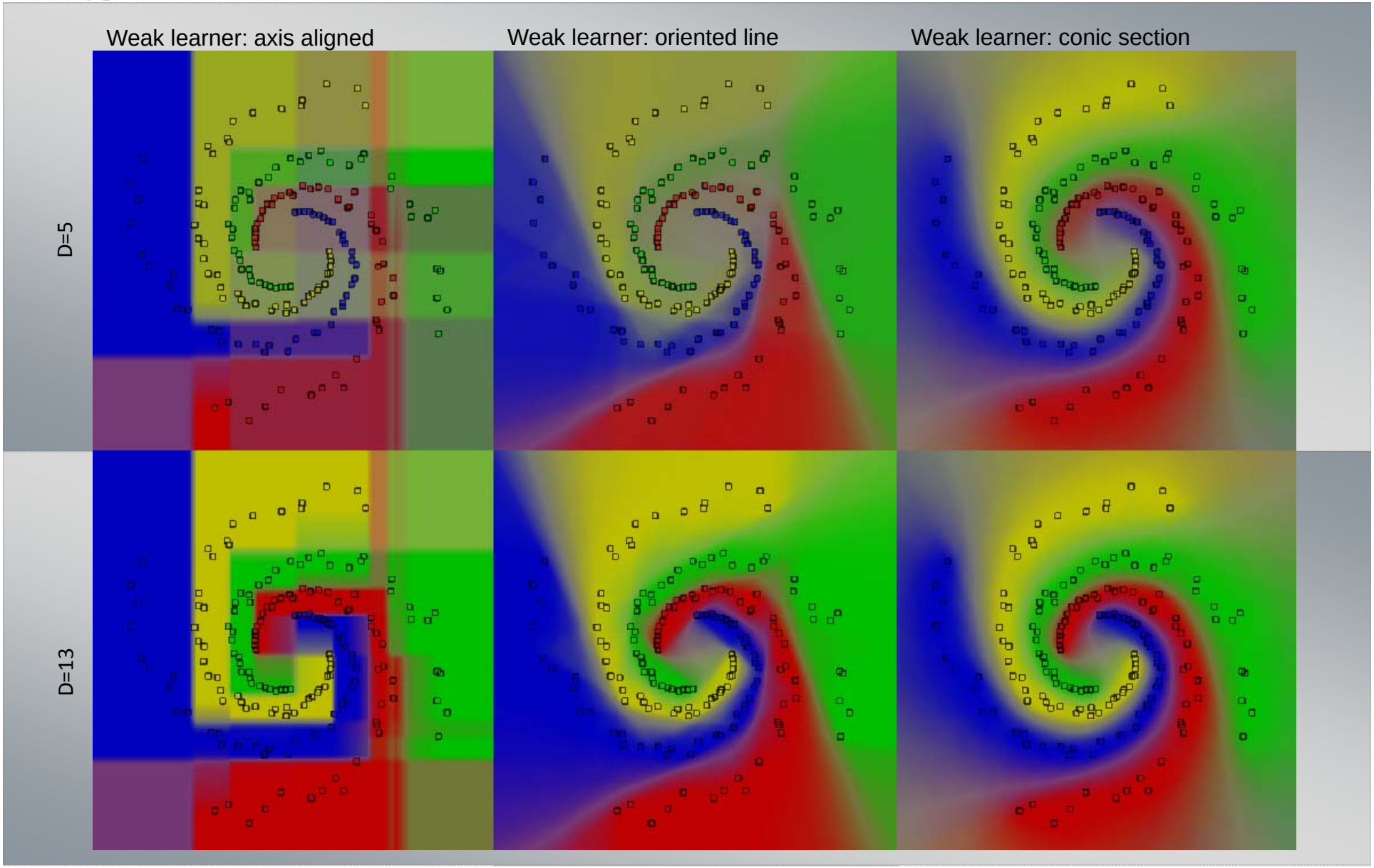
max tree depth, D

underfitting

overfitting

Effect of Weak Learner Model and Randomness

Testing posteriors

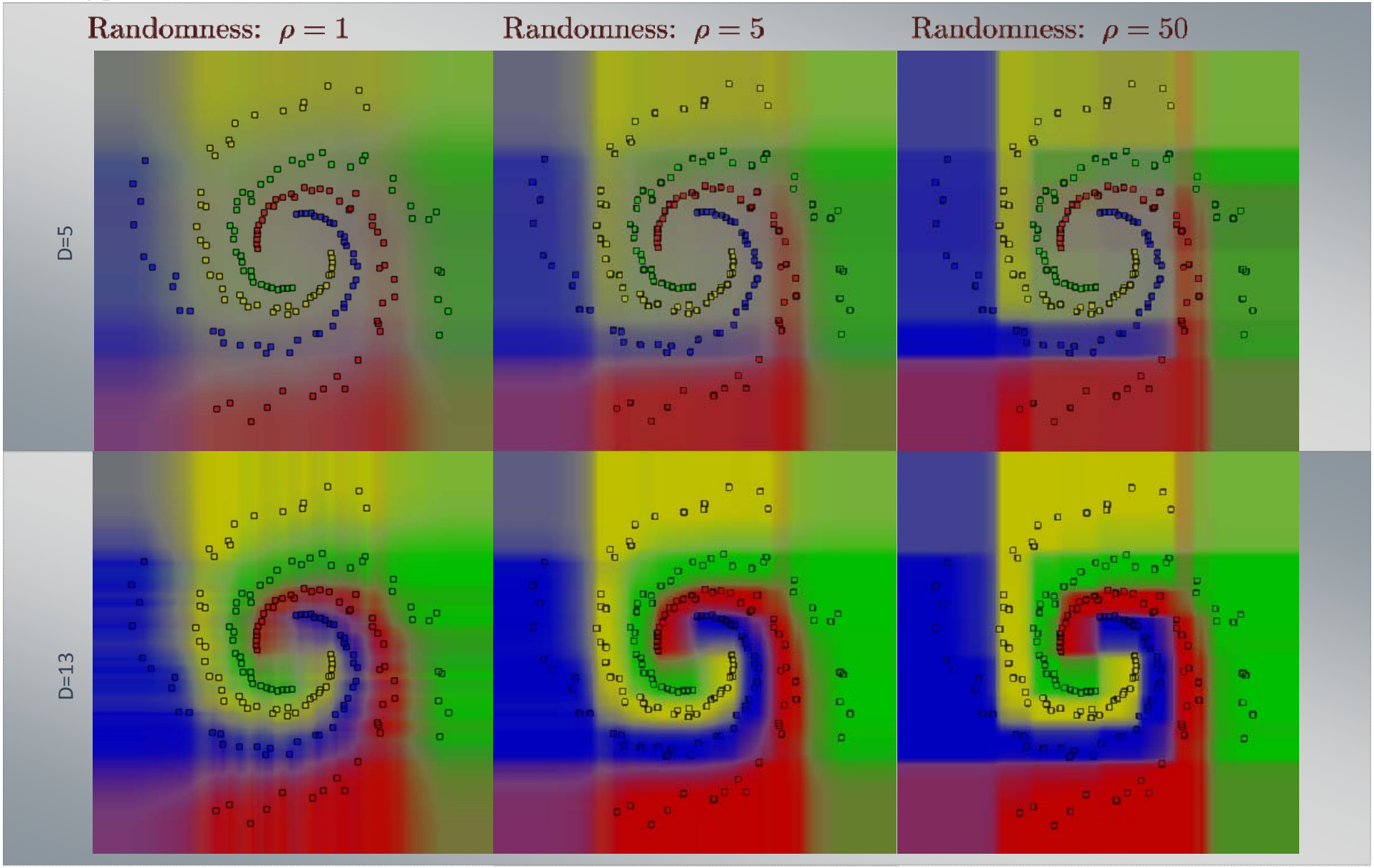


Randomness: $\rho = 500$

Parameters: T=400 predictor model = prob.

Effect of Weak Learner Model and Randomness

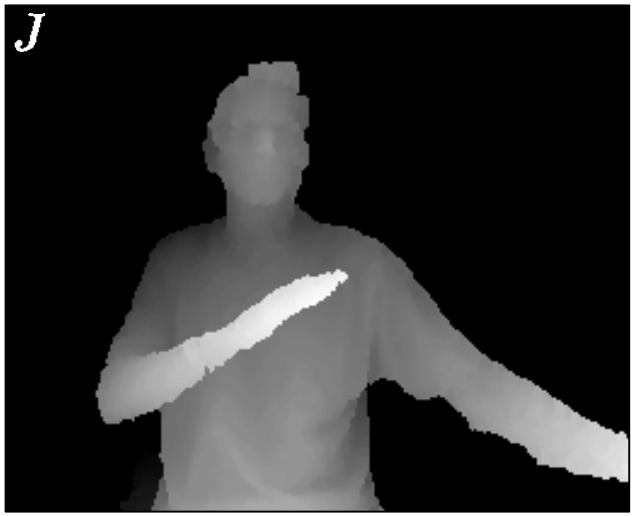
Testing posteriors



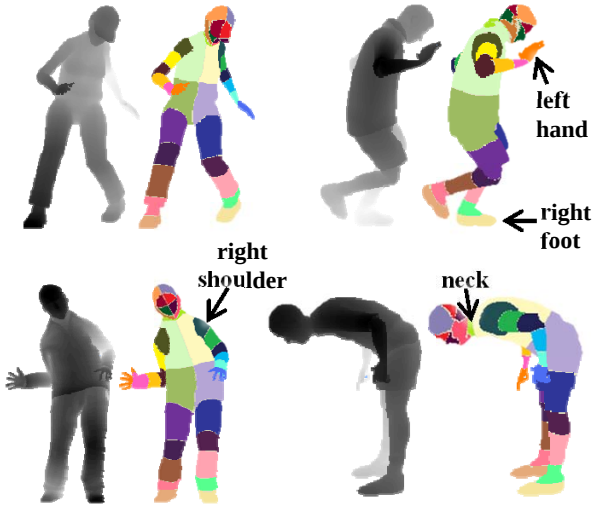
Weak learner: axis aligned

Parameters: T=400 predictor model = prob.

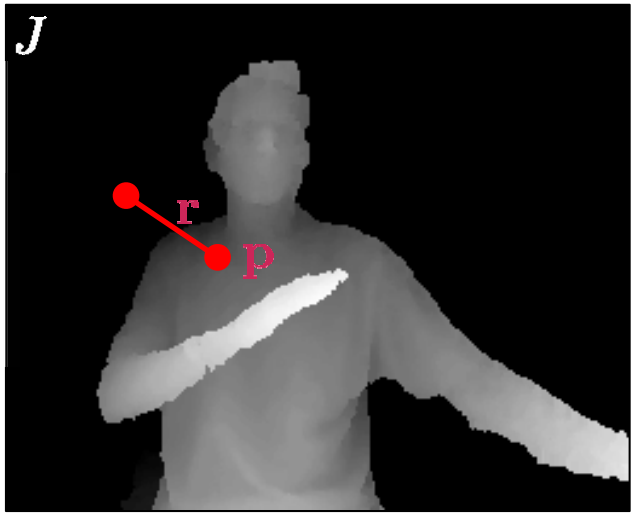
Body tracking in Microsoft Kinect for XBox 360



Input depth image



Training labelled data

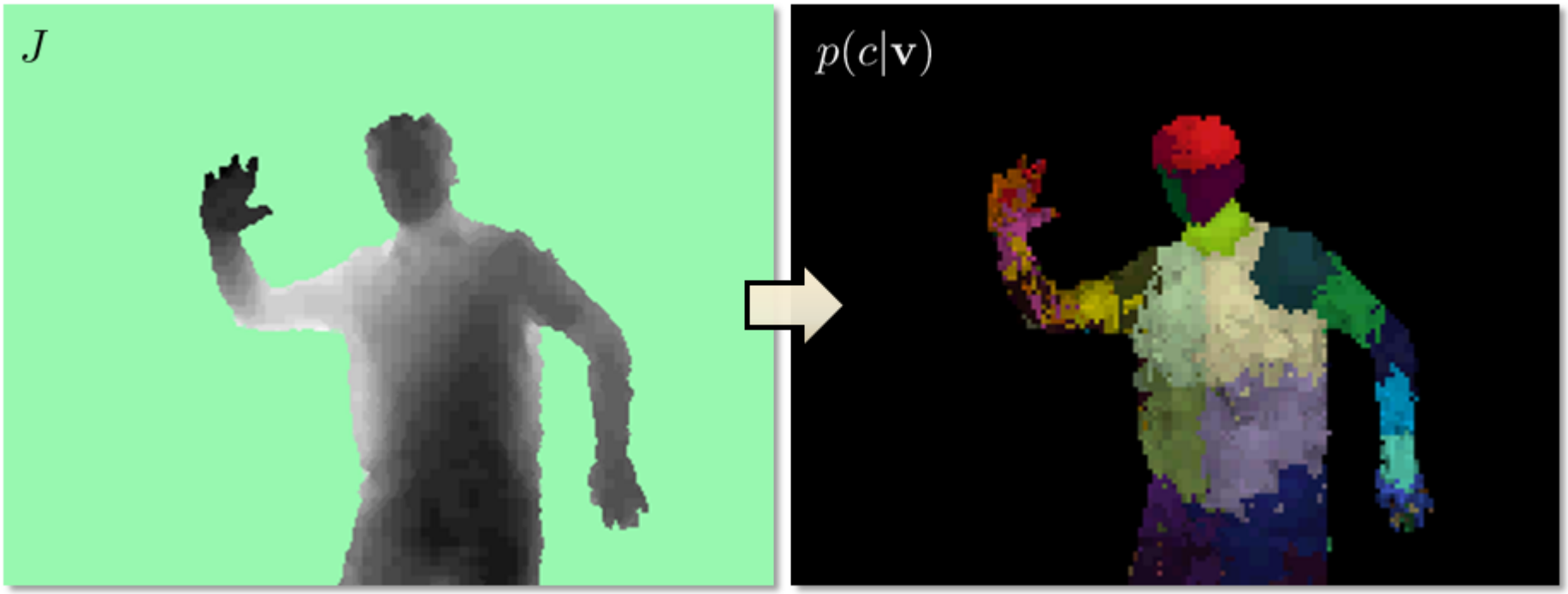


Visual features

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	$I = H(\mathcal{S}_j) - \sum_{i=\mathrm{L,R}} \frac{\|\mathcal{S}_j^i\|}{\|\mathcal{S}_j\|} H(\mathcal{S}_j^i)$
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
- Can handle categorical variables
- Robust to label noise
- Can be used in clustering, locating outliers and semi-supervised learning

L. Breiman's web page

Boosting

Boosting Resources

- Slides based on:
  - 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 $(x_1,y_1),\ldots, (x_m,y_m)$
- $y_i \in \{-1,+1\}$ 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

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

- Final hypothesis:

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

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

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:

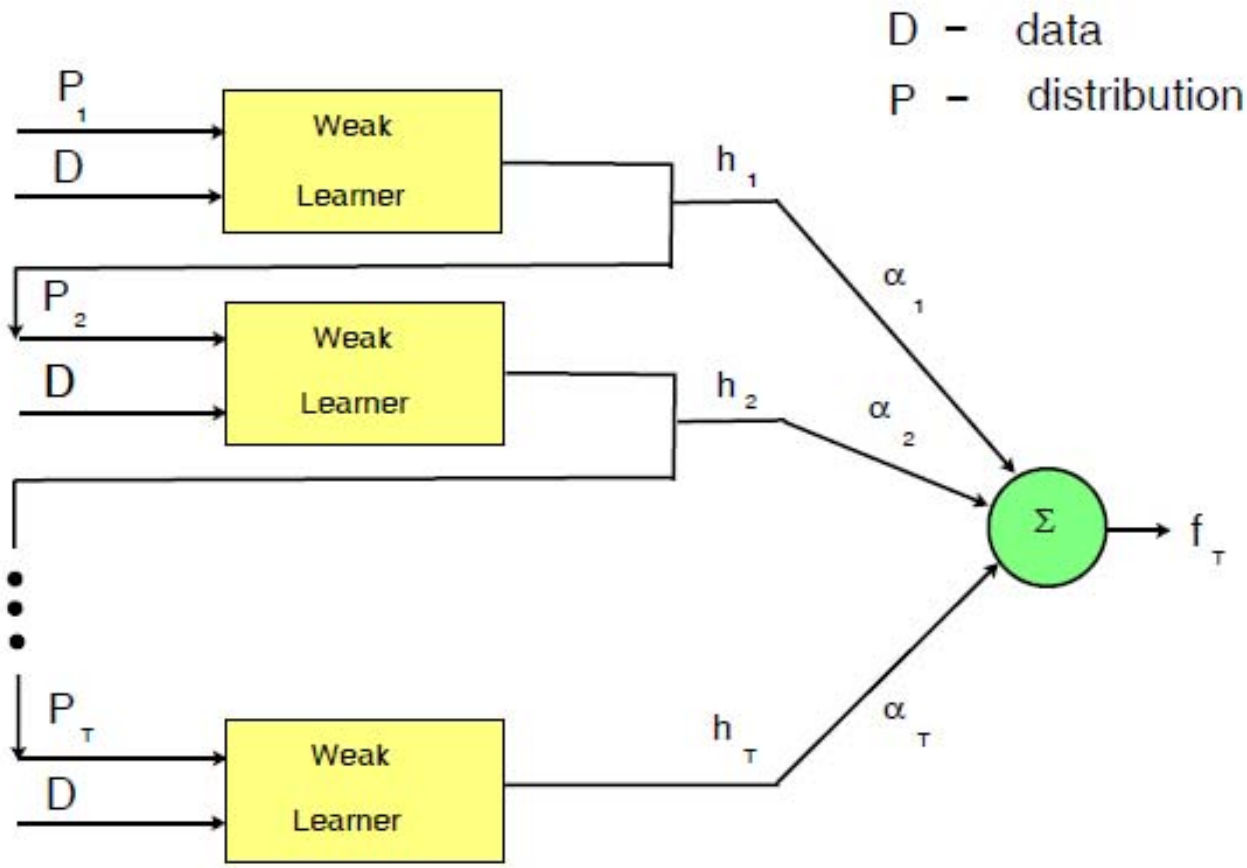
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

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

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

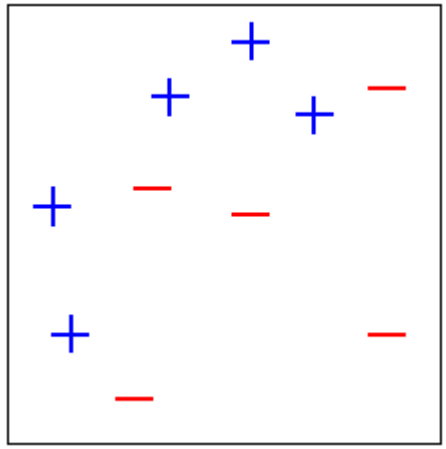
$$\epsilon_t = \frac{1}{\sum_{i=1}^{n} D_t(i)} \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)$$

The AdaBoost Algorithm

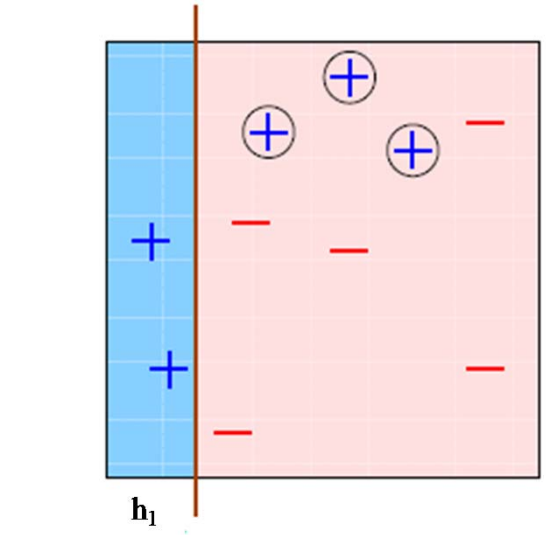


Toy Example

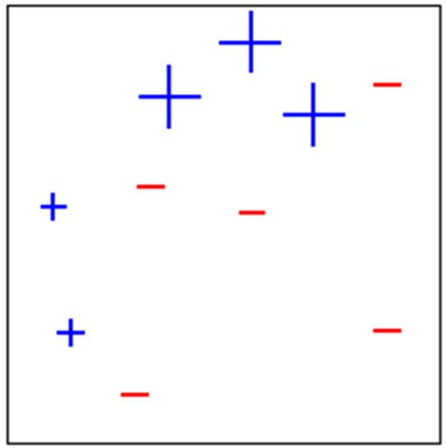
D_1



Toy Example: Round 1

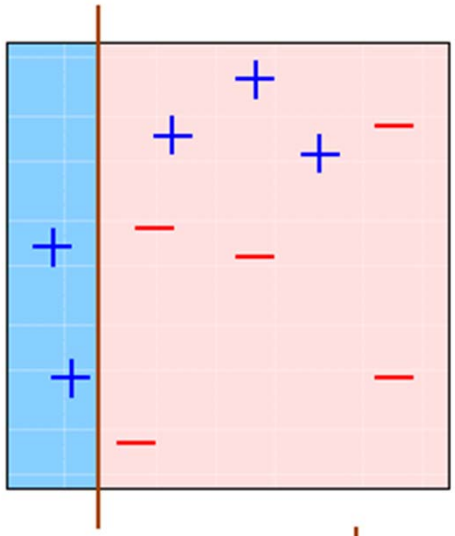


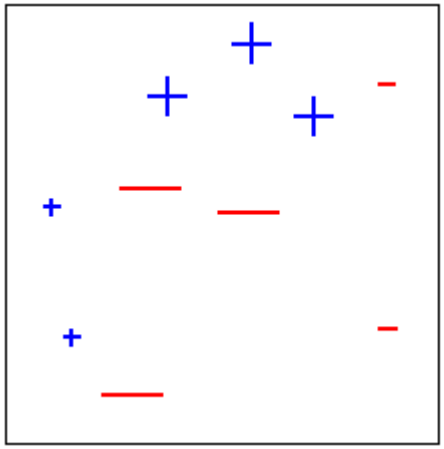
ε_1=0.3
α_1=0.42

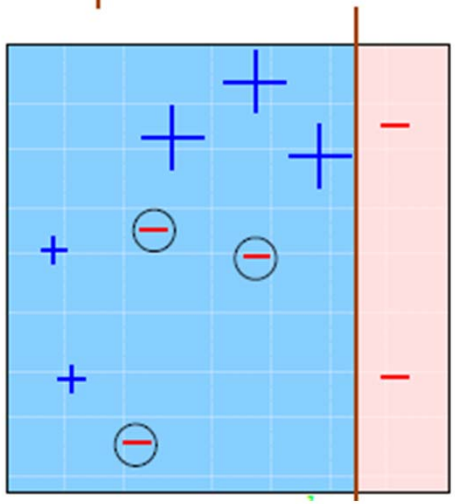


D_2

Toy Example: Round 2





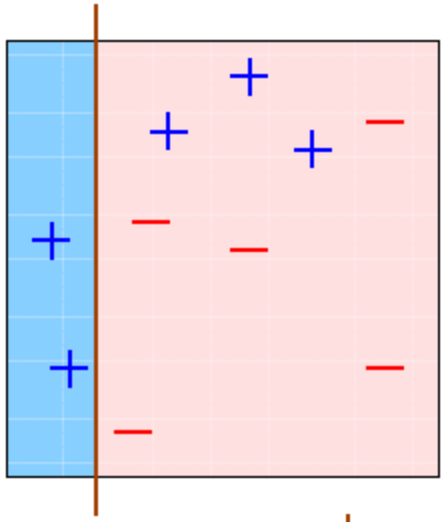


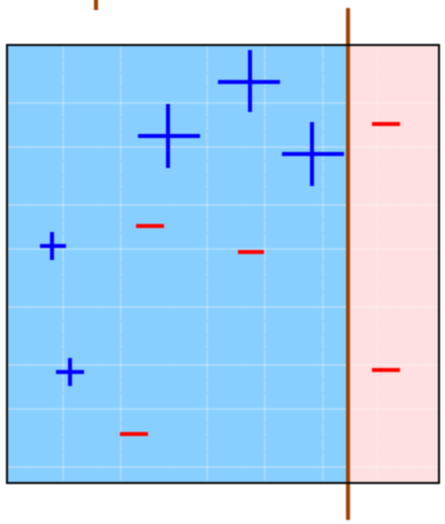
h_2

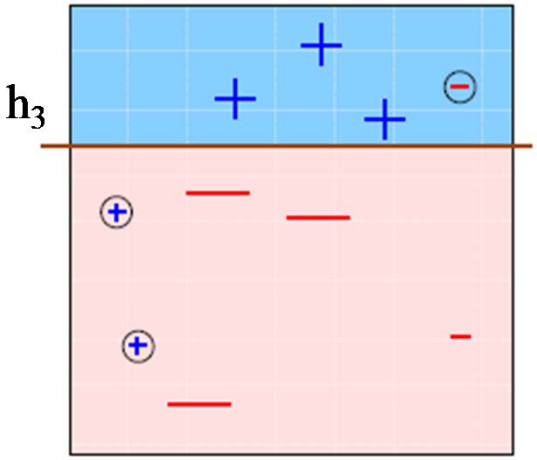
$\varepsilon_2=0.21$

$\alpha_2=0.65$

Toy Example: Round 3



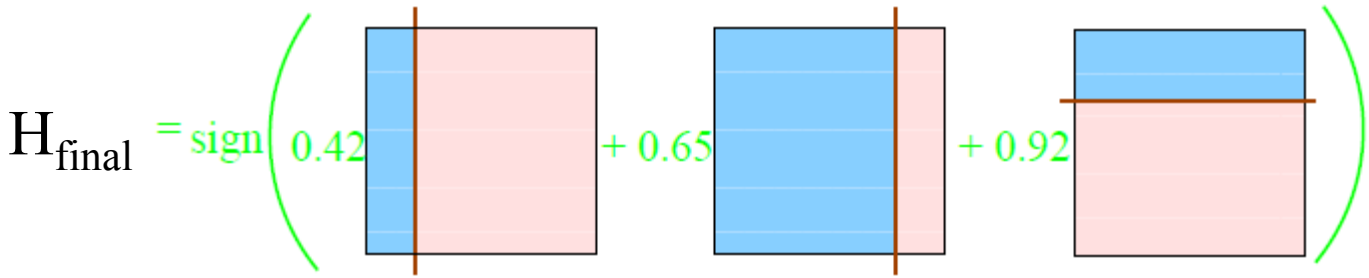


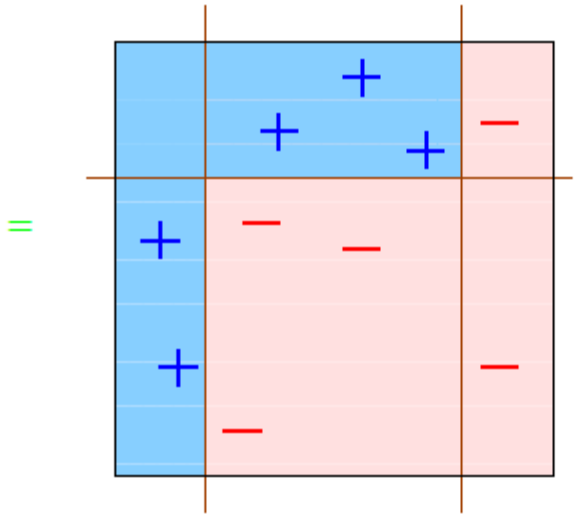


$\varepsilon_3=0.14$

$\alpha_3=0.92$

Toy Example: Final Hypothesis





How to choose Weights

Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i f(x_i))$$

where: $f(x)=\sum_t \alpha_t h_t(x); H(x)=sign(f(x))$

Notice that: $e^{-y_i f(x_i)} \geq 1 \text{ if } y_i \neq H(x_i)$

How to choose Weights

Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i f(x_i)) = \prod_t Z_t$$

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

$$D_{\text{final}}(i) = \frac{1}{m}\cdot\frac{\exp\left(-y_i\sum_t \alpha_t h_t(x_i)\right)}{\prod_t Z_t}$$

In final round:

$$= \frac{1}{m}\cdot\frac{e^{-y_i f(x_i)}}{\prod_t Z_t}$$

How to choose Weights

- If we minimize $\Pi_t Z_t$, we minimize our training error
  - We can tighten this bound greedily, by choosing α_t and h_t in each iteration to minimize Z_t

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

- For boolean target function, this is accomplished by [Freund & Schapire '97]:

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

Weak and Strong Classifiers

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$
- AdaBoost will achieve zero training error (exponentially fast):

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \prod_t Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right)$$

- Is it hard to achieve better than random training error?

Important Aspects of Boosting

- Exponential loss function
- Choice of weak learners
- Generalization and overfitting
- Multi-class boosting

Exponential Loss Function

- The exponential loss function is an upper bound of the 0-1 loss function (classification error)
- AdaBoost provably minimizes exponential loss
- Therefore, it also minimizes the upper bound of classification error

Exponential Loss Function

- AdaBoost attempts to minimize:

$$\prod_t Z_t = \frac{1}{m} \sum_i \exp(-y_i f(x_i)) \quad (*)$$

- Really a coordinate descent procedure
  - At each round add $a_t h_t$ to sum to minimize (*)
- Why this loss function?
  - upper bound on training (classification) error
  - easy to work with
  - connection to logistic regression

Coordinate Descent Explanation

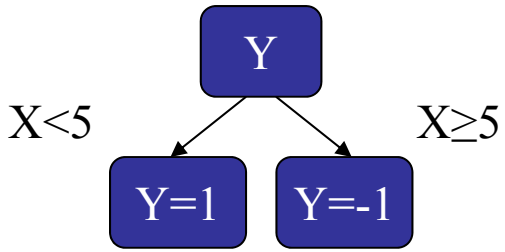
- $\{g_1, \ldots, g_N\}$ = space of all weak classifiers
- want to find $\lambda_1, \ldots, \lambda_N$ to minimize

$$L(\lambda_1, \ldots, \lambda_N) = \sum_i \exp\left(-y_i \sum_j \lambda_j g_j(x_i)\right)$$

- AdaBoost is actually doing coordinate descent on this optimization problem:
  - initially, all $\lambda_j = 0$
  - each round: choose one coordinate λ_j (corresponding to h_t) and update (increment by α_t)
  - choose update causing biggest decrease in loss
- powerful technique for minimizing over huge space of functions

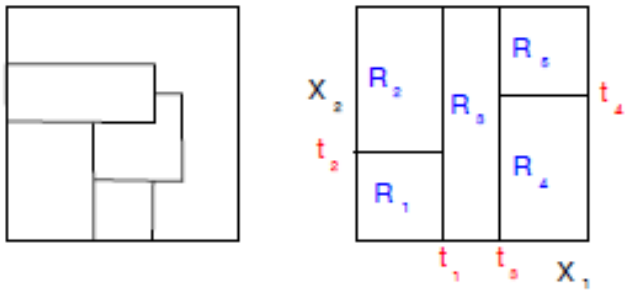
Weak Learners

- Stumps:
  - Single-axis parallel partition of space
- Decision trees:
  - Hierarchical partition of space
- Multi-layer perceptrons:
  - General nonlinear function approximators

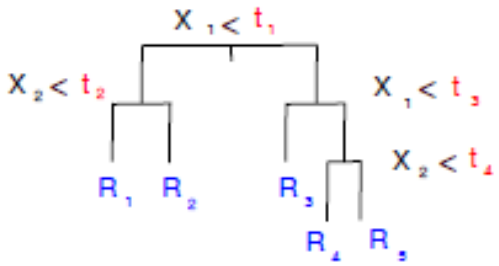


Decision Trees

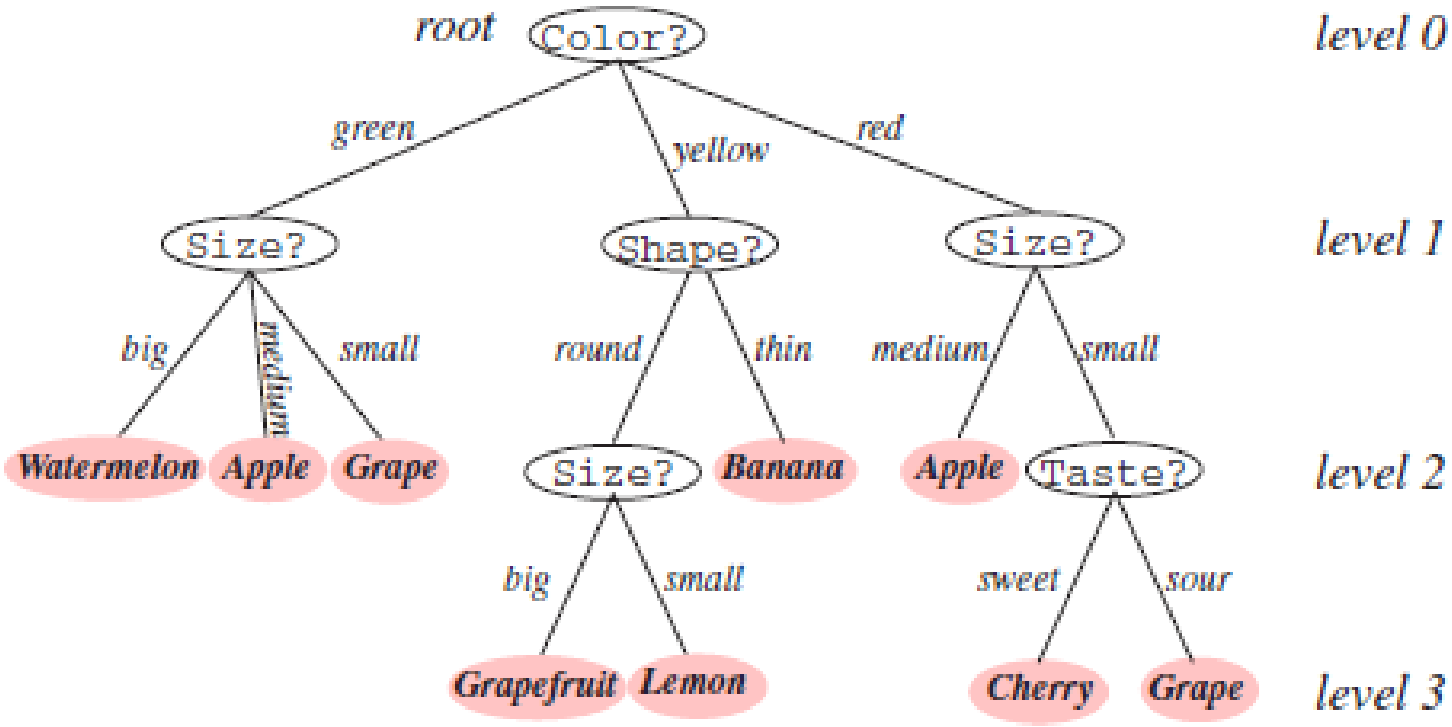
- Hierarchical and recursive partitioning of the feature space
- A simple model (e.g. constant) is fit in each region
- Often, splits are parallel to axes



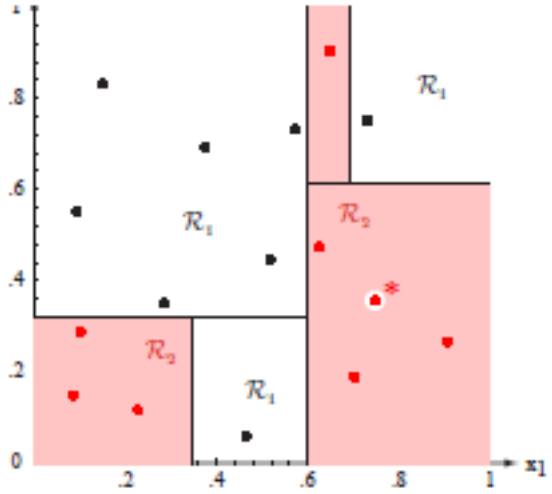
Impossible

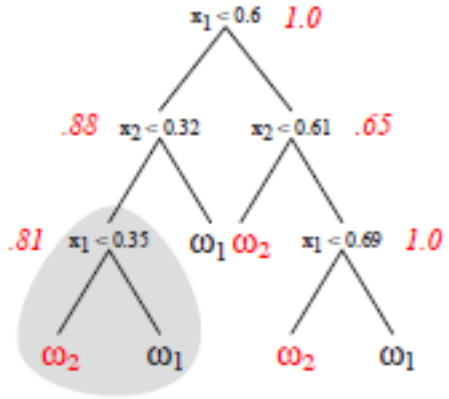


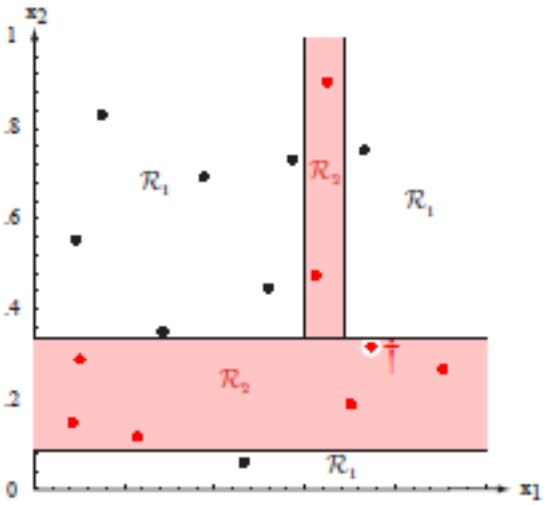
Decision Trees – Nominal Features

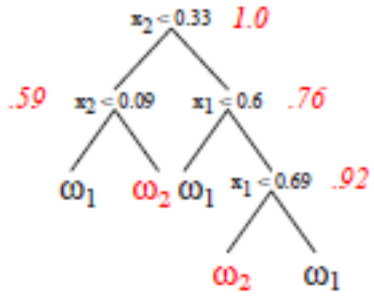


Decision Trees - Instability









Boosting: Analysis of Training Error

- Training error of final classifier is bounded by:

$$\frac{1}{m}\left|\{i : H(x_i) \neq y_i\}\right| \leq \frac{1}{m}\sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

$$e^{-y_i f(x_i)} \geq 1 \text{ if } y_i \neq H(x_i)$$

- For binary classifiers with choice of α_t as before, the error is bounded by

$$\prod_t Z_t = \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_t \gamma_t^2\right)$$

$$\gamma_t = 1/2 - \epsilon_t$$

Analysis of Training Error

- If each base classifier is slightly better than random such that there exists γ such that $\gamma_t > \gamma$ for all t
- Then the training error drops exponentially fast in T

$$\exp(-2\sum_t \gamma_t^2) \leq \exp(-2\gamma^2 T)$$

- AdaBoost is indeed a boosting algorithm in the sense that it can efficiently convert a true weak learning algorithm into a strong learning algorithm
  - Weak learning algorithm: can always generate a classifier with a weak edge for any distribution
  - Strong learning algorithm: can generate a classifier with an arbitrarily low error rate, given sufficient data

Generalization Error

$$error_{true}(H) \quad \leq \quad error_{train}(H) + \tilde{\mathcal{O}}\left(\sqrt{\frac{Td}{m}}\right)$$

- T – number of boosting rounds
- d – VC dimension of weak learner, measures complexity of classifier
  - The Vapnik-Chervonenkis (VC) dimension is a standard measure of the "complexity" of a space of binary functions
- m – number of training examples

Overfitting

- This bound suggests that boosting will overfit if run for too many rounds
- Several authors observed empirically that boosting often does *not* overfit, even when run for thousands of rounds
  - Moreover, it was observed that AdaBoost would sometimes continue to drive down the generalization error long after the training error had reached zero, clearly contradicting the bound above

Analysis of Margins

- An alternative analysis can be made in terms of the *margins* of the training examples. The margin of example (x,y) is:

$$\text{margin}_f(x,y) = \frac{yf(x)}{\sum_t |\alpha_t|} = \frac{y\sum_t \alpha_t h_t(x)}{\sum_t |\alpha_t|}$$

- It is a number in [-1, 1] and it is positive when the example is correctly classified
- Larger margins on the training set translate into a superior upper bound on the generalization error

Analysis of Margins

- It can be shown that the generalization error is at most:

$$\hat{\Pr}\left[\text{margin}_f(x,y) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right)$$

  – Independent of T
- Boosting is particularly aggressive at increasing the margin since it concentrates on the examples with the smallest margins
  – positive or negative

Error Rates and Margins

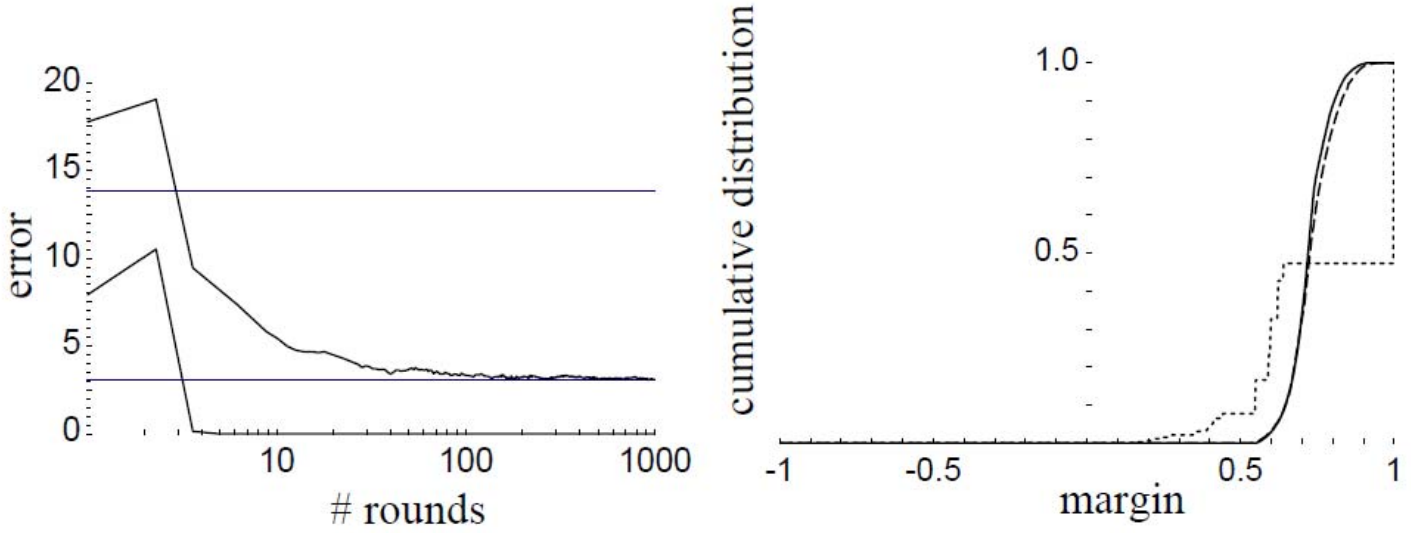


Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [69]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. *Right*: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

Margin Analysis

- Margin theory gives a qualitative explanation of the effectiveness of boosting
- Quantitatively, the bounds are rather weak
- One classifier can have a margin distribution that is better than that of another classifier, and yet be inferior in test accuracy
- Margin theory points to a strong connection between boosting and the support-vector machines

Advantages of Boosting

- Simple and easy to implement
- Flexible – can be combined with any learning algorithm
- No requirement on data being in metric space – data features don't need to be normalized, like in kNN and SVMs (this has been a central problem in machine learning)
- Feature selection and fusion are naturally combined with the same goal for minimizing an objective error function

Advantages of Boosting (cont.)

- Can show that if a gap exists between positive and negative points, generalization error converges to zero
- No parameters to tune (maybe T)
- No prior knowledge needed about weak learner
- Provably effective
- Versatile – can be applied on a wide variety of problems
- Non-parametric

Disadvantages of Boosting

- Performance of AdaBoost depends on data and weak learner
- Consistent with theory, AdaBoost can fail if
  - weak classifier too complex – overfitting
  - weak classifier too weak - underfitting
- Empirically, AdaBoost seems especially susceptible to uniform noise
- Decision boundaries are often rugged

Multi-class AdaBoost

- Assume yϵ{1,…,k}
- Direct approach (AdaBoost.M1):

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

$$\beta_t = \epsilon_t / (1 - \epsilon_t)$$

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}$$

- can prove same bound on error if $\epsilon_t \leq 1/2$
- else: abort

Limitation of AdaBoost.M1

- Achieving $\epsilon_t \leq 1/2$ may be hard if k (number of classes) is large

- [Mukherjee and Schapire, 2010]: weak learners that perform slightly better than random chance can be used in multi-class boosting framework
  - Out of scope for now

Reducing to Binary Problems

- Say possible labels are {a,b,c,d,e}
- Each training example replaced by five {-1,+1} labeled examples

$$x \ , \ \mathrm{c} \ \rightarrow \begin{cases} (x, \mathrm{a}) \ , \ -1 \\ (x, \mathrm{b}) \ , \ -1 \\ (x, \mathrm{c}) \ , \ +1 \\ (x, \mathrm{d}) \ , \ -1 \\ (x, \mathrm{e}) \ , \ -1 \end{cases}$$

AdaBoost.MH

- Formally $h_t : X \times Y \to \{-1, +1\} (\text{or } \mathbb{R})$ Used to be X→ {-1, +1}

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

where $$v_i(y) = \begin{cases} +1 & \text{if } y_i = y \\ -1 & \text{if } y_i \neq y \end{cases}$$

$$H_{\text{final}}(x) = \arg\max_{y \in Y} \sum_t \alpha_t h_t(x, y)$$

Can prove that $$\text{training error}(H_{\text{final}}) \leq \frac{k}{2} \cdot \prod Z_t$$

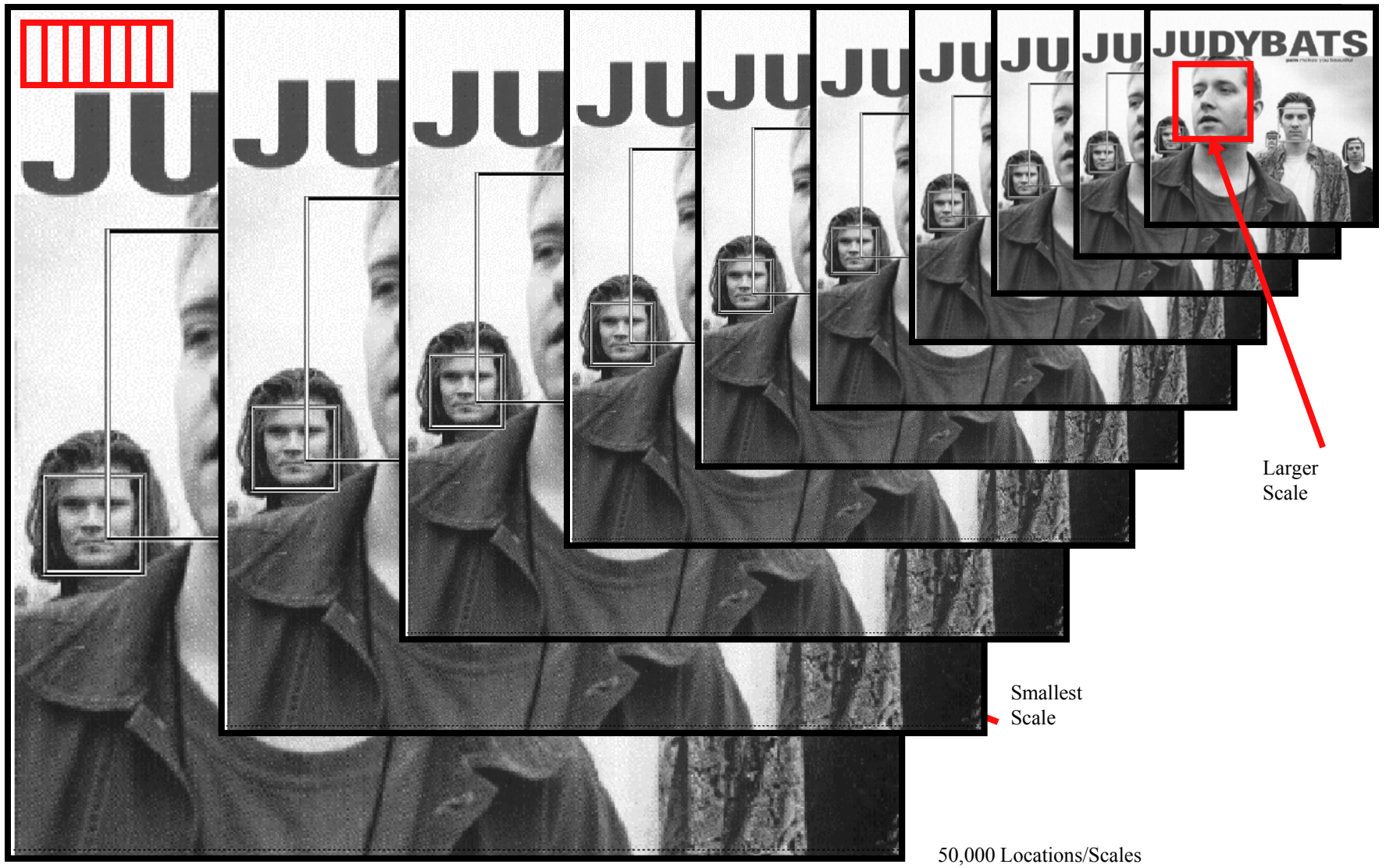
Random Forests vs. Boosting

- RF Pros:
  - More robust
  - Faster to train (no reweighting, each split is on a small subset of data and features)
  - Can handle missing/partial data
  - Easier to extend to online version
- RF Cons:
  - Feature selection process is not explicit
  - Weaker performance on small size training data
  - Weaker theoretical foundations

Applications of Boosting

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

The Classical Face Detection Process



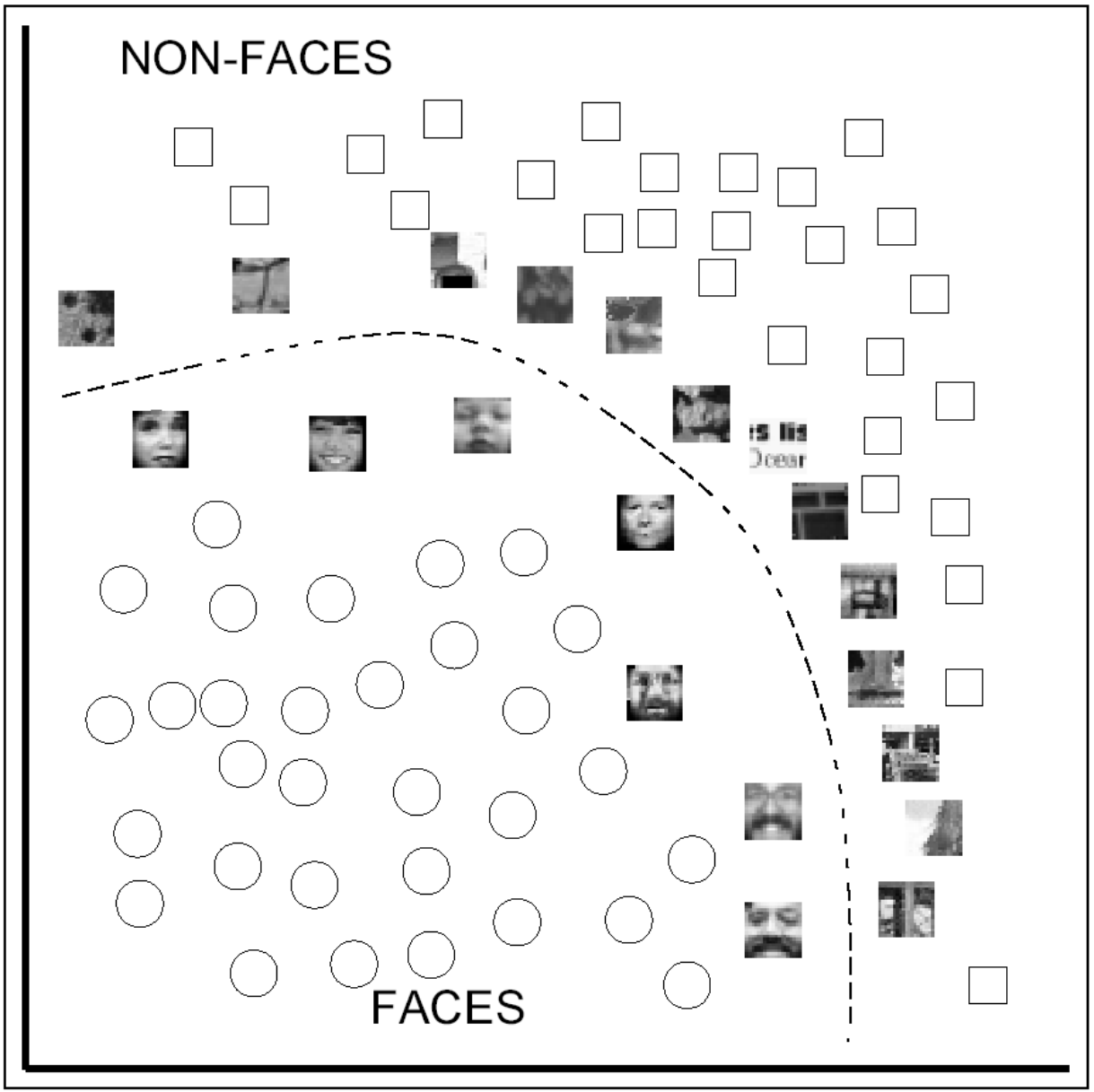
Classifier is Trained on Labeled Data

- 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)

Key Properties of Face Detection

- Each image contains 10,000 – 50,000 locations/scales
- Faces are rare 0 - 50 per image
  - 1000 times as many non-faces as faces
- Goal: Extremely small rate of false negatives: 10^{-6}

“Support Vectors”



Challenging negative examples are extremely important

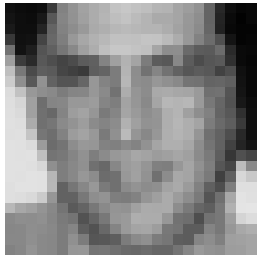
Classifier Cascade (Viola-Jones)

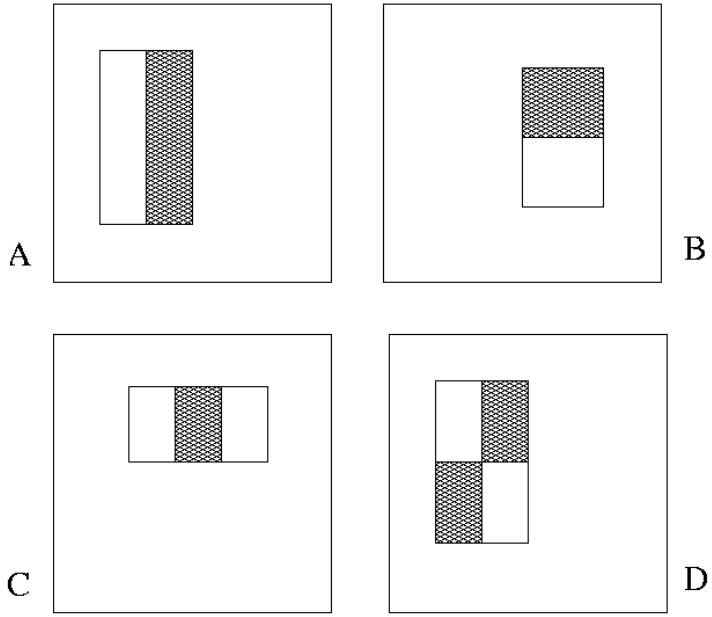
- For real problems results are only as good as the features used...
  - This is the main piece of ad-hoc (or domain) knowledge
- Rather than the pixels, use a very large set of simple functions
  - Sensitive to edges and other critical features of the image
  - Computed at multiple scales
- Introduce 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"

Similar to Haar wavelets





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

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

$$60{,}000 \times 100 = 6{,}000{,}000$$

Unique Binary Features

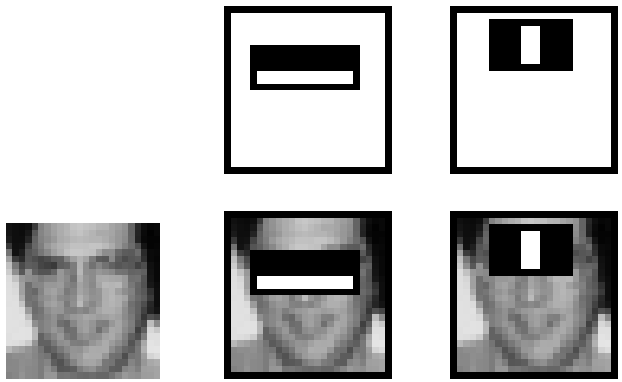
Feature Selection

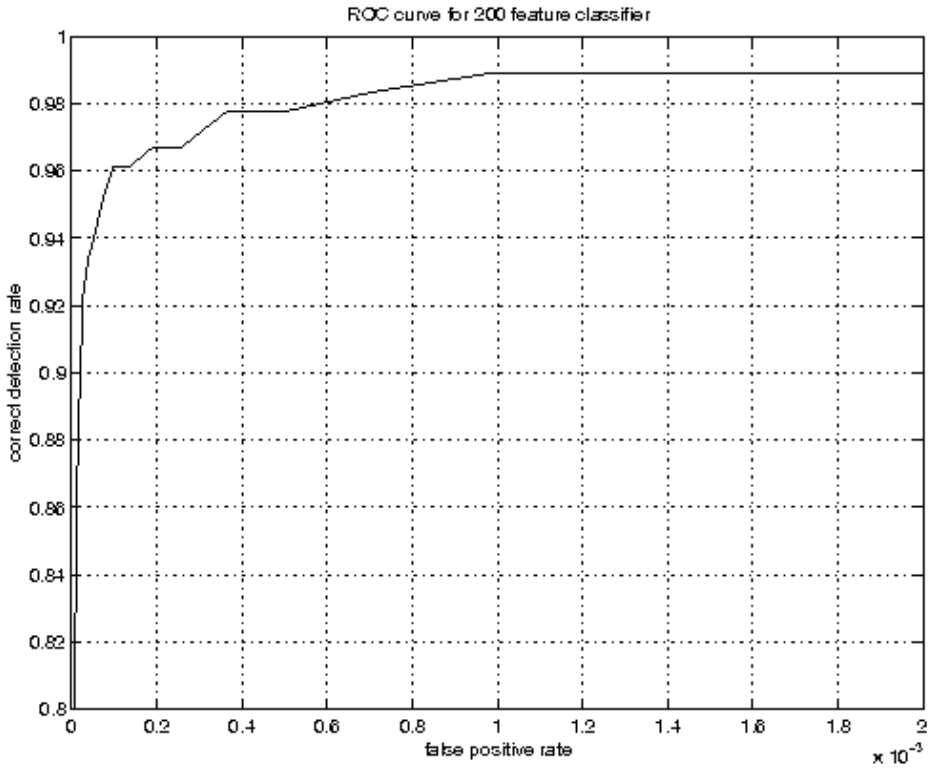
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter
  - Select best filter/threshold (= Feature)
  - Reweight examples

Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost

95% correct detection on test set with 1 in 14084
false positives.



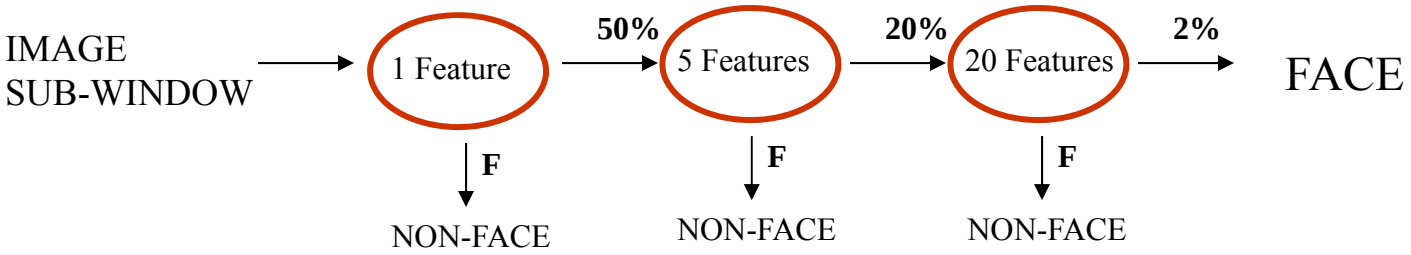


ROC curve for 200 feature classifier

Building Fast Classifiers

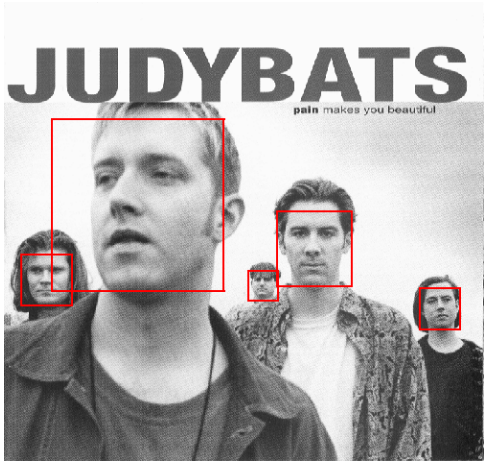
- In general, simple classifiers are more efficient, but they are also weaker
- We could define a computational risk hierarchy
  - A nested set of classifier classes
- The training process is reminiscent of boosting…
  - Previous classifiers reweight the examples used to train subsequent classifiers
- The goal of the training process is different
  - Minimize errors, but also minimize false positives

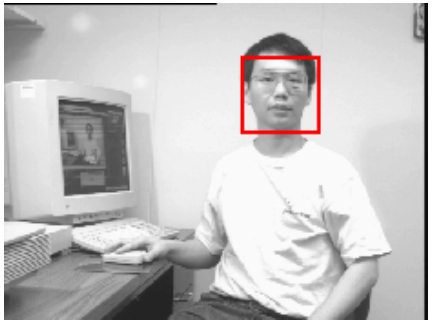
Cascaded Classifier

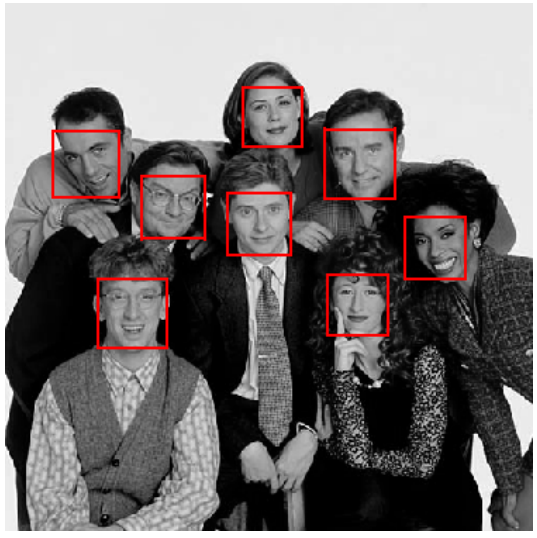


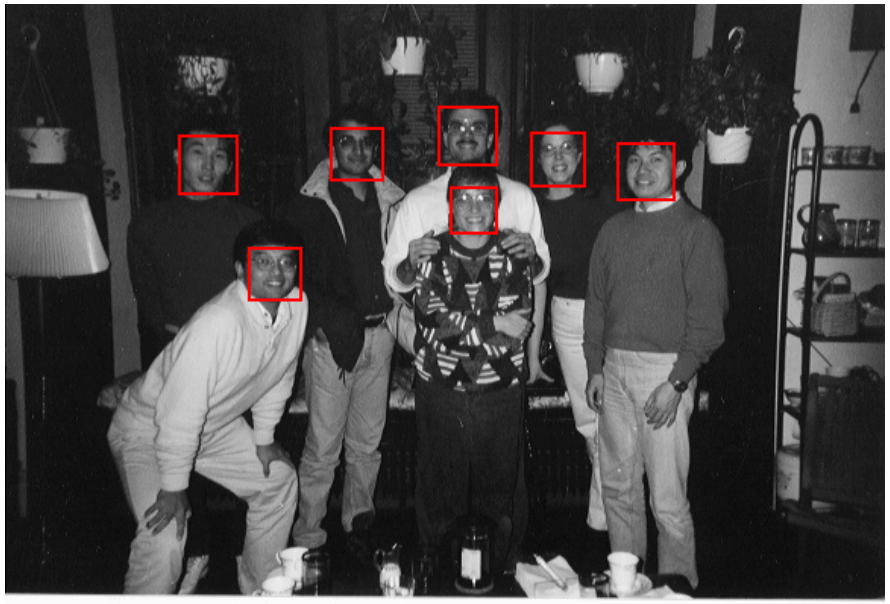
- A 1-feature classifier achieves 100% detection rate and about 50% false positive rate
- A 5-feature classifier achieves 100% detection rate and 40% false positive rate
  - using data from previous stage
- A 20-feature classifier achieve 100% detection rate with 10% false positive rate

Output of Face Detector on Test Images



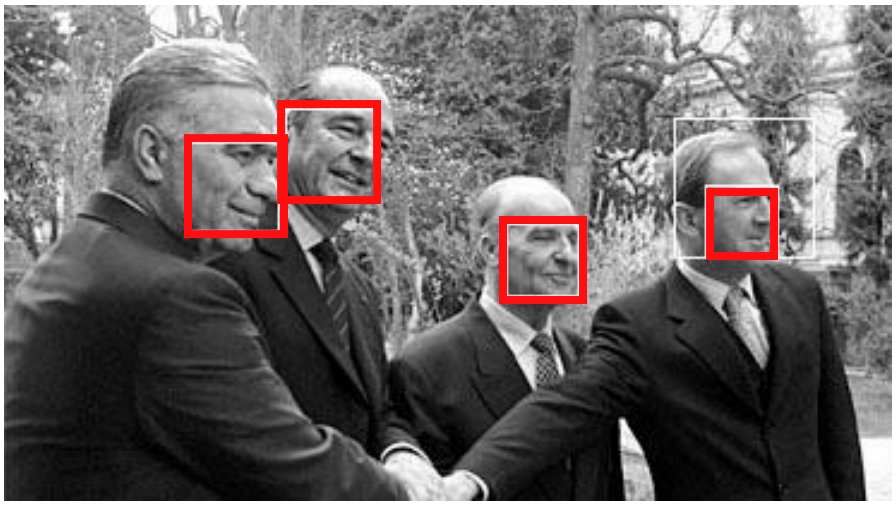






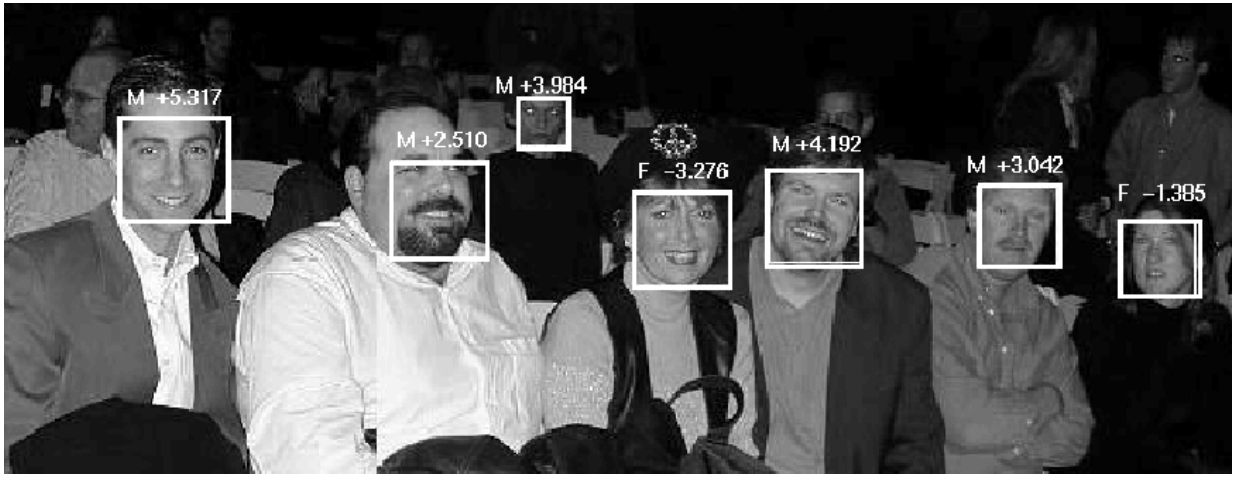
Solving other “Face” Tasks

Facial Feature Localization



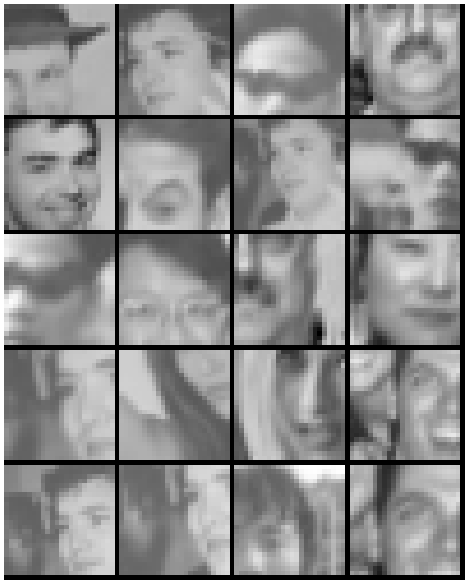
Profile Detection

Demographic Analysis



Feature Localization

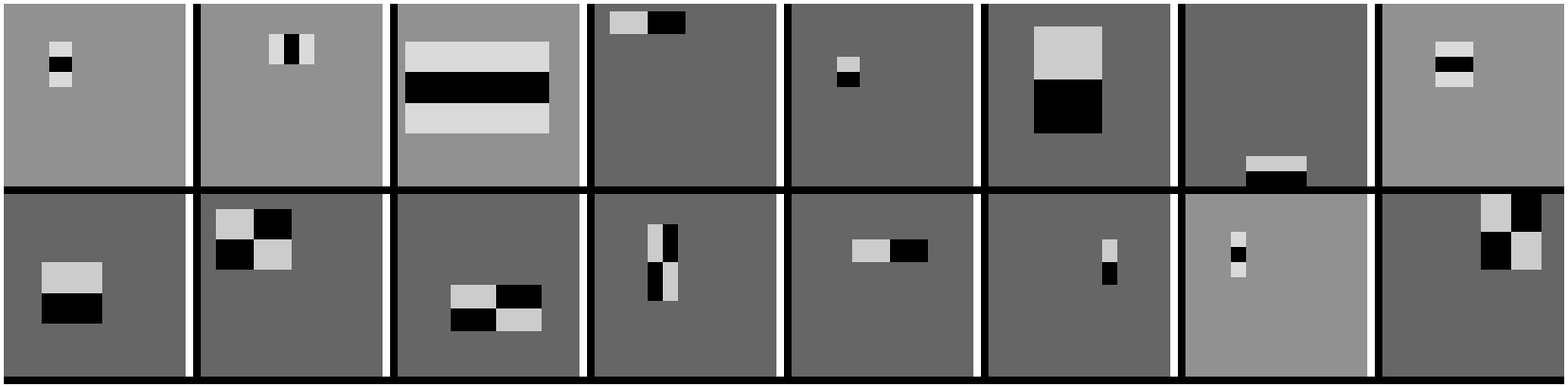
- Surprising properties of Viola-Jones framework
  - The cost of detection is not a function of image size
    - Just the number of features
  - Learning automatically focuses attention on key regions
- Conclusion: the “feature” detector can include a large contextual region around the feature



Sub-windows rejected at final stages

Feature Localization

- Learned features reflect the task



Profile Detection

Profile Features