Boosting, face detection

Class 7. 14 Sep 2010

Instructor: Bhiksha Raj

Administrivia: Projects

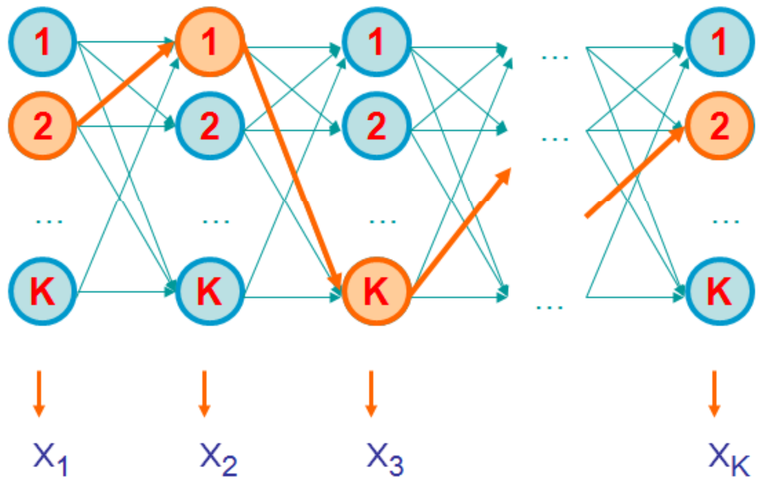
- Only 1 group so far
  - Plus one individual
- Notify us about your teams ASAP
  - Or at least that you are *trying* to form a team
  - Otherwise, on 1st we will assign teams by lots
- Inform us about the project you will be working on

Administrivia: Homeworks

- Trick question: When is the homework due?
- Second homework: up next week.

Lecture by Raffay Hamid on Thursday

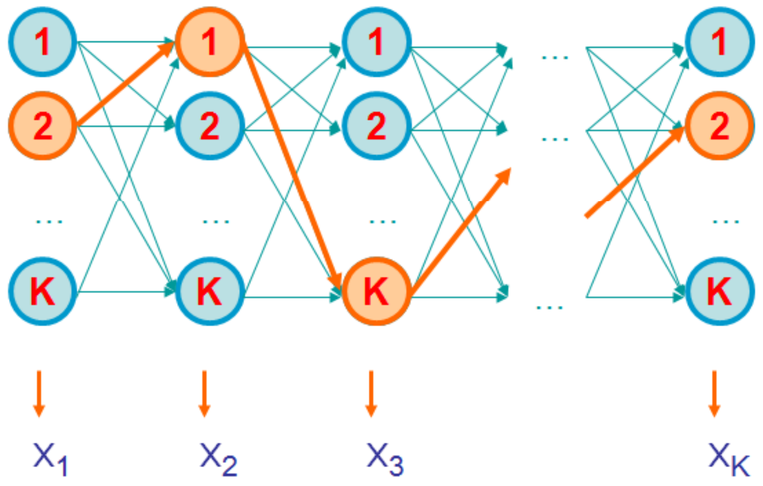
ACTIVITY RECOGNITION



- In this lecture, we will learn how to apply machine learning techniques to temporal processes. For instance, we might be interested in "beating the casino", by figuring out how are a pair of dice loaded by analyzing the sequence of their outcomes (this would help us hedge our bets more intelligently). Or, we could be interested in finding out the general topic of an article, by analyzing a small (say two to three sentences long) sequence of words taken from that article. Finally, we might be interested in predicting what's the most likely work one would speak, given a sequence of words one has just spoken (this might be useful for designing more intelligent automatic phone response systems).

Lecture by Raffay Hamid on Thursday

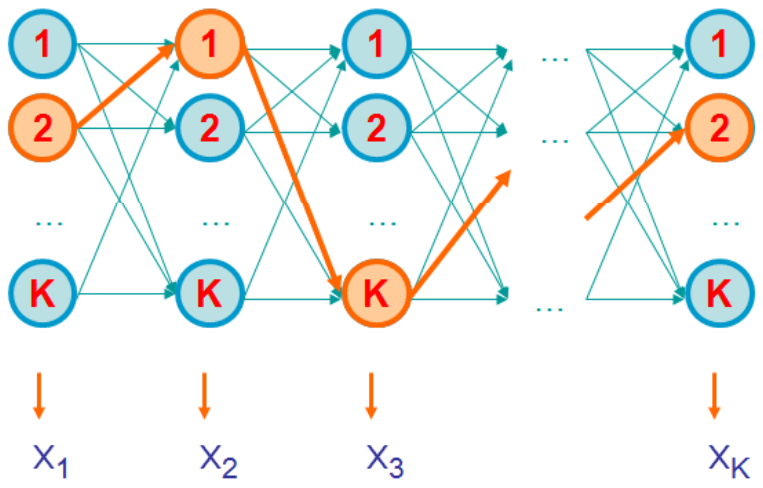
ACTIVITY RECOGNITION



- The particular method we'll discuss consists of what are called the Markov Models. We will briefly go over the mathematical background of the Markov Models, making our segue into their slightly more elaborate cousins called the Hidden Markov Models (HMMs). We will attempt to cover the three basic questions of HMMs: (i) Evaluation, (ii) Decoding, and (iii) Learning (the meaning of these terms would hopefully become more clear at the end of our discussion).
- We will also attempt to cover some of the practical applications of HMMs, with emphasis on their application on Human action recognition observed through video.

Lecture by Raffay Hamid on Thursday

ACTIVITY RECOGNITION



- *Must Read* references:

- Please read the following before the class:
- http://www.stanford.edu/class/cs229/section/cs229-hmm.pdf
- http://ai.stanford.edu/~serafim/CS262_2009/index.php
    - look for the lectures on HMMs

Project Idea 1: Mario Berges

- ***marioberges@cmu.edu***
- Sparse coding and disaggregation of low-resolution aggregate power data for a house

- We have low-res aggregate power measurements (e.g., 1Hz whole-house measurements) for a couple of homes for some months.

- Explore unsupervised approaches to decompose that data into individual appliances (or individual activities)
  - Using sparse representations and finding the best projection of the data into it.

Project Idea 2: Mario Berges

- ***Multi-resolution event detection for appliance state-transitions.***

- We have aggregate and appliance-level datasets of power measurements in which many appliance state-transitions take place.

- Each appliance state-transition may have a different "time constant", that determines how long it takes for the load to reach steady-state.

- Detecting the transitions is challenging due to these differences.
  - Use a multi-resolution approach that looks at changes in various time-scales

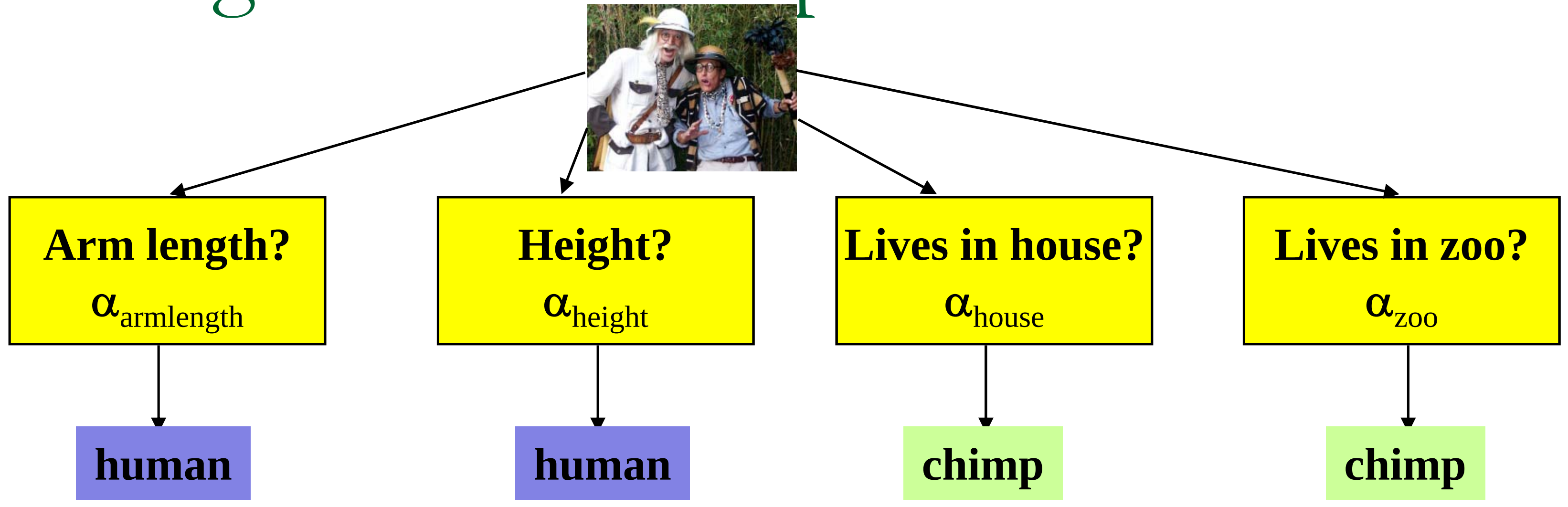
- Explore supervised algorithms to detect these changes
  - Or if there are invariant representations that are not affected by the time-scale.

A Quick Intro to Boosting

Introduction to Boosting

- An *ensemble* method that sequentially combines many simple ***BINARY*** classifiers to construct a final complex classifier
  - Simple classifiers are often called “weak” learners
  - The complex classifiers are called “strong” learners
- Each weak learner focuses on instances where the previous classifier failed
  - Give greater weight to instances that have been incorrectly classified by previous learners
- Restrictions for weak learners
  - Better than 50% correct
- Final classifier is *weighted* sum of weak classifiers

Boosting and the Chimpanzee Problem



- The total confidence in all classifiers that classify the entity as a chimpanzee is

$$Score_{chimp} = \sum_{classifier\ favors\ chimpanzee} \alpha_{\text{classifier}}$$

- The total confidence in all classifiers that classify it as a human is

$$Score_{human} = \sum_{classifier\ favors\ human} \alpha_{\text{classifier}}$$

- If $Score_{chimpanzee} > Score_{human}$ then the our belief that we have a chimpanzee is greater than the belief that we have a human

Boosting: A very simple idea

- One can come up with many rules to classify
  - E.g. Chimpanzee vs. Human classifier:
  - If arms == long, entity is chimpanzee
  - If height > 5’6” entity is human
  - If lives in house == entity is human
  - If lives in zoo == entity is chimpanzee
- Each of them is a reasonable rule, but makes many mistakes
  - Each rule has an intrinsic error rate
- *Combine* the predictions of these rules
  - But not equally
  - Rules that are less accurate should be given lesser weight

Formalizing the Boosting Concept

- Given a set of instances $(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$
  - x_i is the set of attributes of the i^{th} instance
  - y_1 is the class for the i^{th} instance
    - y_1 can be +1 or -1 (binary classification only)
- Given a set of classifiers $h_1, h_2, \ldots, h_T$
  - h_i classifies an instance with attributes x as $h_i(x)$
  - $h_i(x)$ is either -1 or +1 (for a binary classifier)

  - y*h(x) is 1 for all correctly classified points and -1 for incorrectly classified points
- Devise a function $f(h_1(x), h_2(x), \ldots, h_T(x))$ such that classification based on $f()$ is superior to classification by any $h_i(x)$
  - The function is succinctly represented as $f(x)$

The Boosting Concept

- A simple combiner function: Voting
  - $f(x) = \Sigma_i h_i(x)$
  - Classifier $H(x) = \text{sign}(f(x)) = \text{sign}(\Sigma_i h_i(x))$
  - Simple majority classifier
    - A simple voting scheme
- A better combiner function: Boosting
  - $f(x) = \Sigma_i \alpha_i h_i(x)$
    - Can be any real number
  - Classifier $H(x) = \text{sign}(f(x)) = \text{sign}(\Sigma_i \alpha_i h_i(x))$
  - A weighted majority classifier
    - The weight α_i for any $h_i(x)$ is a measure of our trust in $h_i(x)$

The ADABoost Algorithm

- Adaboost is ADAPTIVE boosting

- The combined classifier is a *sequence* of weighted classifiers
- We learn classifier weights in an adaptive manner

- Each classifier's weight optimizes performance on data whose weights are in turn adapted to the accuracy with which they have been classified

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum}\,\{D_t\,(x_i)\,½(1 - y_i\,h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln((1 - \varepsilon_t) / \varepsilon_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i)\exp(-\,\alpha_t\,y_i\,h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\,\alpha_t\,h_t(x))$

First, some example data

= 0.3 E1 - 0.6 E2

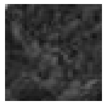
= 0.5 E1 - 0.5 E2

= 0.7 E1 - 0.1 E2

= 0.6 E1 - 0.4 E2

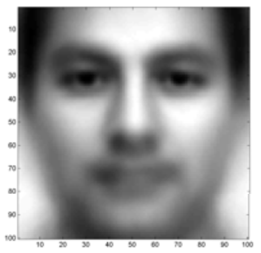
= 0.2 E1 + 0.4 E2

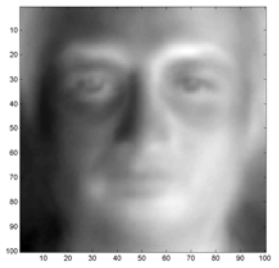
= -0.8 E1 - 0.1 E2

 **= 0.4 E1 - 0.9 E2**

= 0.2 E1 + 0.5 E2

Image = a*E1 + b*E2 → a = Image.E1/|Image|

E_1 

E_2 

- Face detection with multiple Eigen faces
- Step 0: Derived top 2 Eigen faces from eigen face training data
- Step 1: On a (different) set of examples, express each image as a linear combination of Eigen faces
  - Examples include both faces and non faces
  - Even the non-face images will are explained in terms of the eigen faces

Training Data

= 0.3 E1 - 0.6 E2

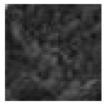
= 0.5 E1 - 0.5 E2

= 0.7 E1 - 0.1 E2

= 0.6 E1 - 0.4 E2

= 0.2 E1 + 0.4 E2

= -0.8 E1 - 0.1 E2

 = 0.4 E1 - 0.9 E2

= 0.2 E1 + 0.5 E2

ID	E1	E2.	Class
A	0.3	-0.6	+1
B	0.5	-0.5	+1
C	0.7	-0.1	+1
D	0.6	-0.4	+1
E	0.2	0.4	-1
F	-0.8	-0.1	-1
G	0.4	-0.9	-1
H	0.2	0.5	-1

Face = +1
Non-face = -1

The ADABoost Algorithm

Initialize $D_1(x_i) = 1/N$

- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum } \{D_t(x_i) \tfrac{1}{2}(1 - y_i h_t(x_i))\}$
  - Set $\alpha_t = \tfrac{1}{2} \ln ((1 - \varepsilon_t) / \varepsilon_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- \alpha_t y_i h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t \alpha_t h_t(x))$

Training Data

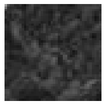
= 0.3 E1 - 0.6 E2

= 0.2 E1 + 0.4 E2

= 0.5 E1 - 0.5 E2

= -0.8 E1 - 0.1 E2

= 0.7 E1 - 0.1 E2

 = 0.4 E1 - 0.9 E2

= 0.6 E1 - 0.4 E2

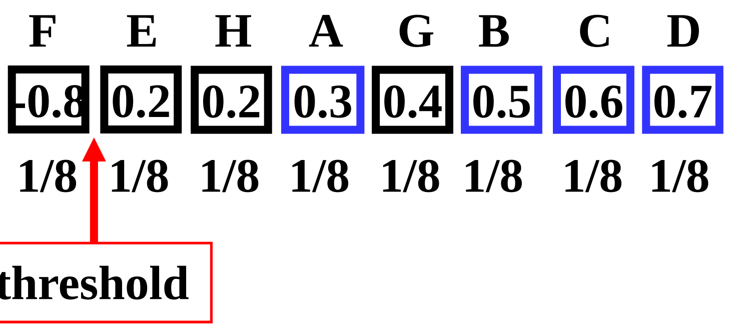
= 0.2 E1 + 0.5 E2

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum}\ \{D_t\ (x_i)\ ½(1 – y_i\ h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln\ (\varepsilon_t / (1 – \varepsilon_t))$
  - For $i = 1\ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- \alpha_t\ y_i\ h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

The E1"Stump"



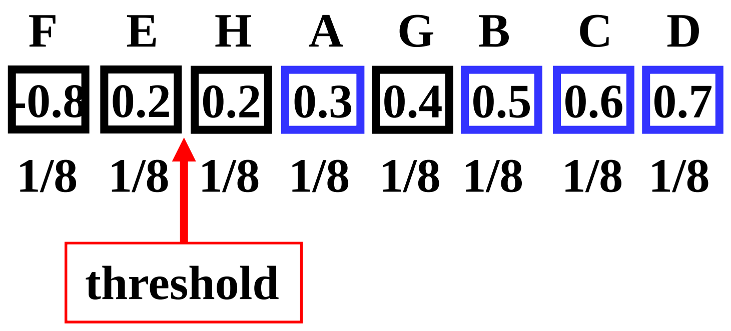
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

Sign = +1, error = 3/8
Sign = -1, error = 5/8

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E1 "Stump"



Sign = +1, error = 2/8

Sign = -1, error = 6/8

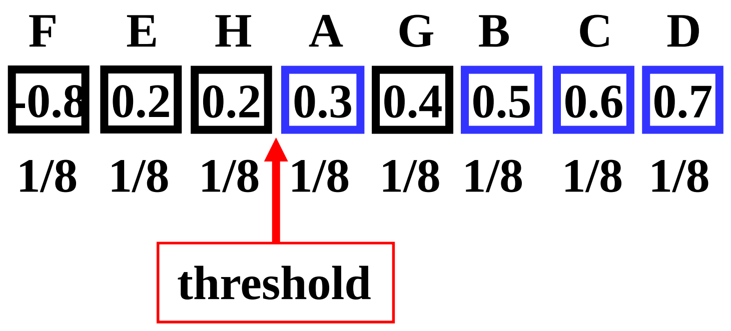
Classifier based on E1:

```
if ( sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1
```

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E1 “Stump”



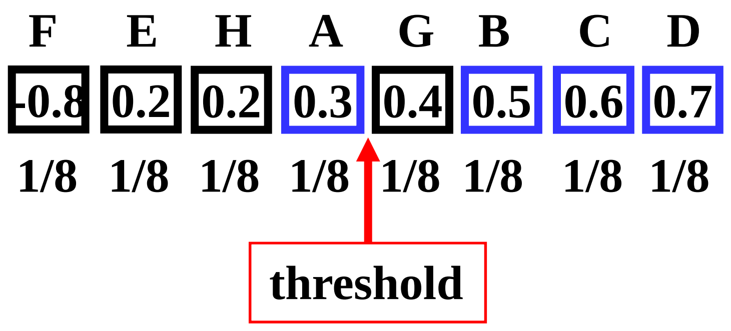
Sign = +1, error = 1/8
Sign = -1, error = 7/8

Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E1 "Stump"



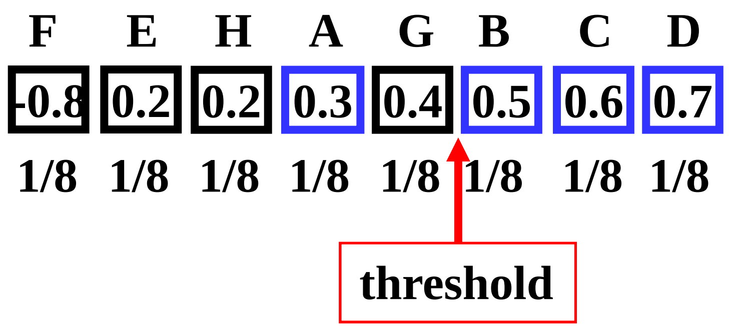
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

Sign = +1, error = 2/8
Sign = -1, error = 6/8

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E1 "Stump"



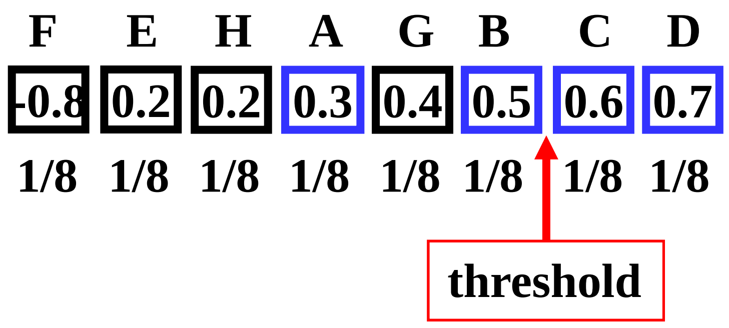
Sign = +1, error = 1/8
Sign = -1, error = 7/8

Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E1 "Stump"



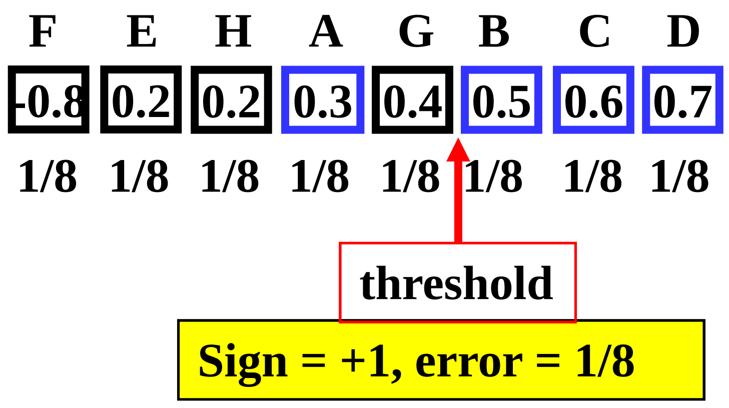
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

Sign = +1, error = 2/8
Sign = -1, error = 6/8

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The Best E1 "Stump"

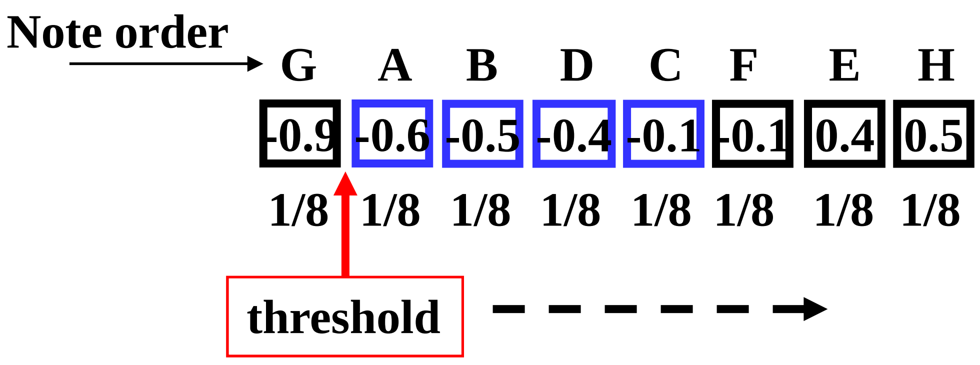


Classifier based on E1:
```
if ( sign*wt(E1) > thresh) > 0)
    face = true
```

Sign = +1
Threshold = 0.45

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The E2"Stump"



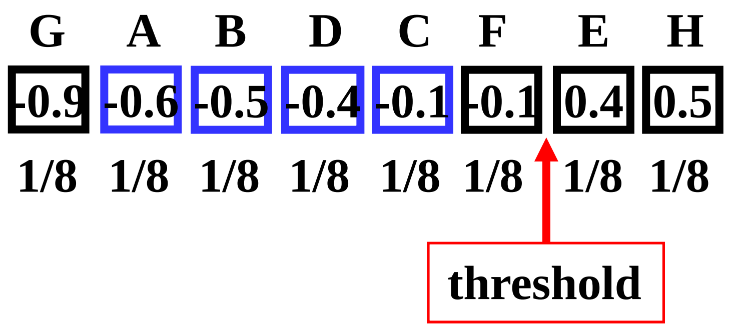
Sign = +1, error = 3/8
Sign = -1, error = 5/8

Classifier based on E2:
if (sign*wt(E2) > thresh) > 0)
    face = true

sign = +1 or -1

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The Best E2"Stump"



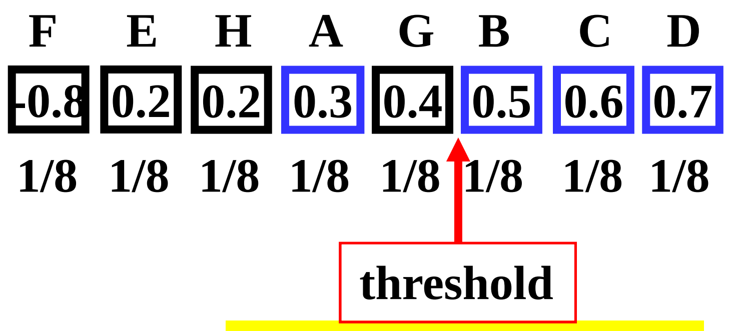
Sign = -1, error = 2/8

Classifier based on E2:
if (sign*wt(E2) > thresh) > 0)
    face = true

sign = -1
Threshold = 0.15

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	-0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The Best "Stump"



Sign = +1, error = 1/8

The Best overall classifier based on a single feature is based on E1

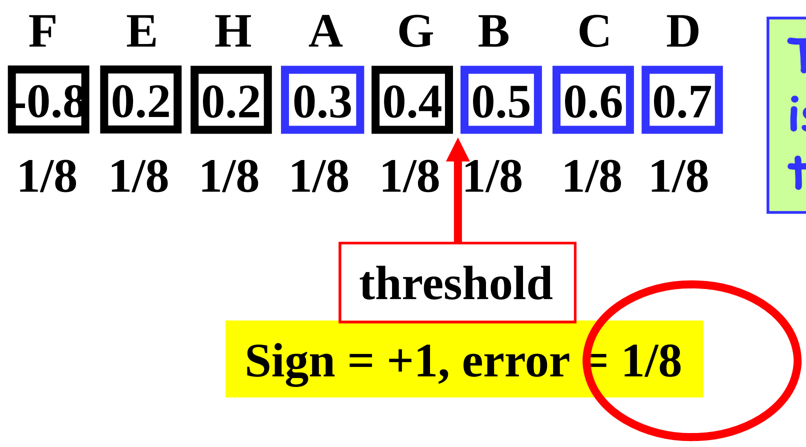
If (wt(E1) > 0.45) → Face

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum}\ \{D_t\ (x_i)\ ½(1 - y_i\ h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln\ (\varepsilon_t /(1 - \varepsilon_t))$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- \alpha_t\ y_i\ h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

The Best Error



Sign = +1, error = 1/8

The Error of the classifier is the sum of the weights of the misclassified instances

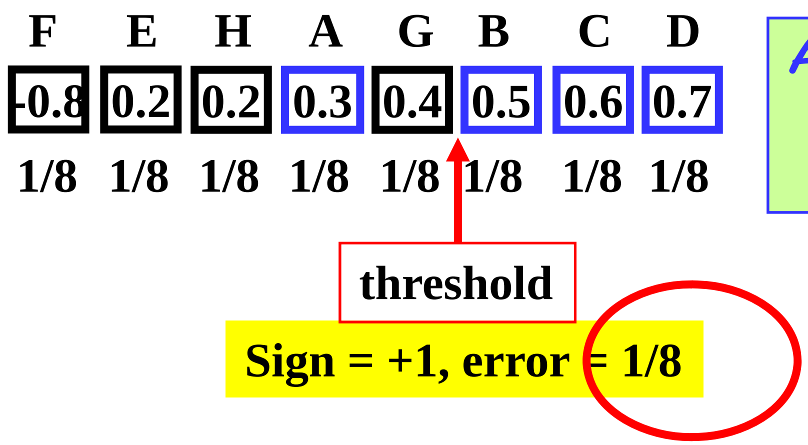
ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	1/8
B	0.5	-0.5	+1	1/8
C	0.7	-0.1	+1	1/8
D	0.6	-0.4	+1	1/8
E	0.2	0.4	-1	1/8
F	-0.8	0.1	-1	1/8
G	0.4	-0.9	-1	1/8
H	0.2	0.5	-1	1/8

NOTE: THE ERROR IS THE SUM OF THE WEIGHTS OF MISCLASSIFIED INSTANCES

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum}\,\{D_t\,(x_i)\,½(1 – y_i\,h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln((1 – \varepsilon_t) / \varepsilon_t)$
  - For $i = 1\ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(-\,\alpha_t\,y_i\,h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\,\alpha_t\,h_t(x))$

Computing Alpha



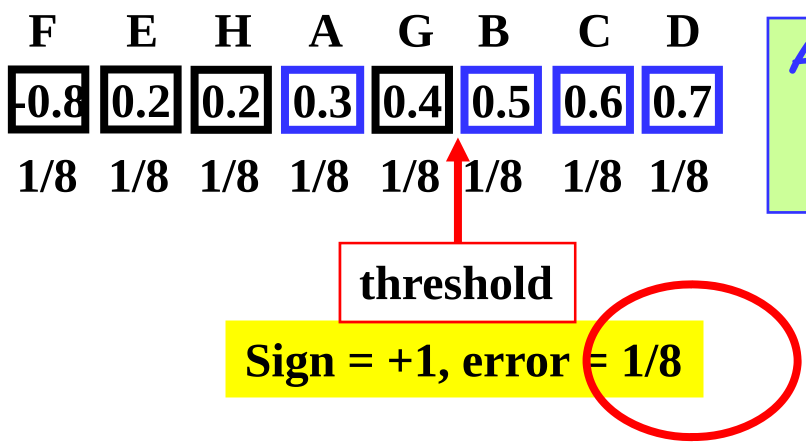
F	E	H	A	G	B	C	D
-0.8	0.2	0.2	0.3	0.4	0.5	0.6	0.7
1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

Sign = +1, error = 1/8

Alpha = 0.5ln((1-1/8) / (1/8))

= 0.5 ln(7) = 0.97

The Boosted Classifier Thus Far



Alpha = 0.5ln((1-1/8) / (1/8))

= 0.5 ln(7) = 0.97

h1(X) = wt(E1) > 0.45 ? +1 : -1

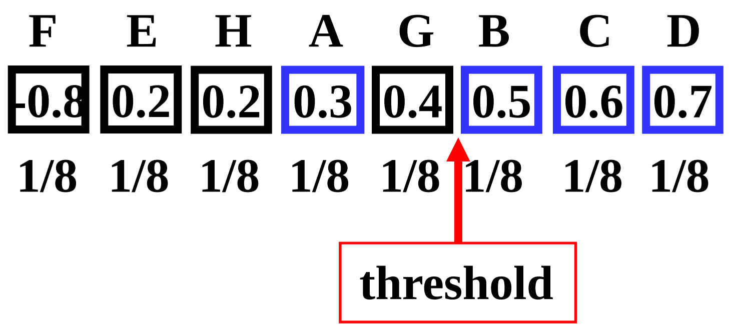
H(X) = sign(0.97 * h1(X))

It's the same as h1(x)

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Average}\,\{½\,(1 - y_i\, h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln ((1 - \varepsilon_t) / \varepsilon_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(-\,\alpha_t\, y_i\, h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\, \alpha_t\, h_t(x))$

The Best Error



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

$$\exp(\alpha_t) = \exp(0.97) = 2.63$$
$$\exp(-\alpha_t) = \exp(-0.97) = 0.38$$

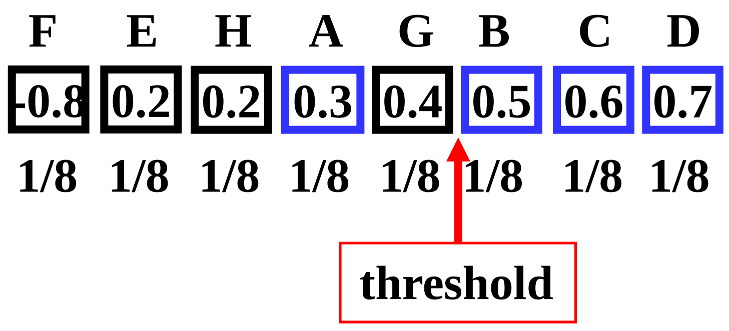
ID	E1	E2.	Class	Weight	Weight
A	0.3	-0.6	+1	1/8 * 2.63	0.33
B	0.5	-0.5	+1	1/8 * 0.38	0.05
C	0.7	-0.1	+1	1/8 * 0.38	0.05
D	0.6	-0.4	+1	1/8 * 0.38	0.05
E	0.2	0.4	-1	1/8 * 0.38	0.05
F	-0.8	0.1	-1	1/8 * 0.38	0.05
G	0.4	-0.9	-1	1/8 * 0.38	0.05
H	0.2	0.5	-1	1/8 * 0.38	0.05

Multiply the correctly classified instances by 0.38
Multiply incorrectly classified instances by 2.63

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Average}\ \{½\ (1 - y_i\ h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln\ ((1 - \varepsilon_t) / \varepsilon_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- \alpha_t\ y_i\ h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

The Best Error

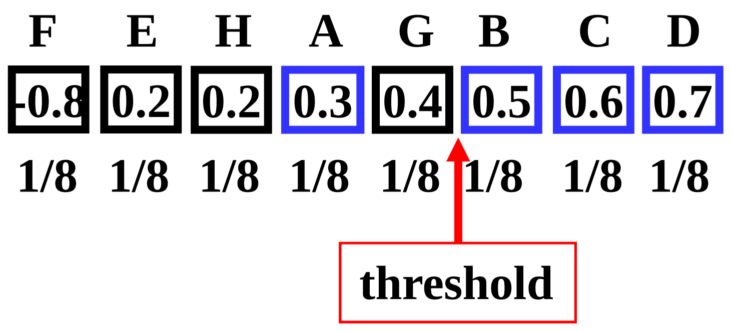


D' = D / sum(D)

ID	E1	E2.	Class	Weight	Weight	Weight
A	0.3	-0.6	+1	1/8 * 2.63	0.33	0.48
B	0.5	-0.5	+1	1/8 * 0.38	0.05	0.074
C	0.7	-0.1	+1	1/8 * 0.38	0.05	0.074
D	0.6	-0.4	+1	1/8 * 0.38	0.05	0.074
E	0.2	0.4	-1	1/8 * 0.38	0.05	0.074
F	-0.8	0.1	-1	1/8 * 0.38	0.05	0.074
G	0.4	-0.9	-1	1/8 * 0.38	0.05	0.074
H	0.2	0.5	-1	1/8 * 0.38	0.05	0.074

Multiply the correctly classified instances by 0.38
Multiply incorrectly classified instances by 2.63
Normalize to sum to 1.0

The Best Error



D’ = D / sum(D)

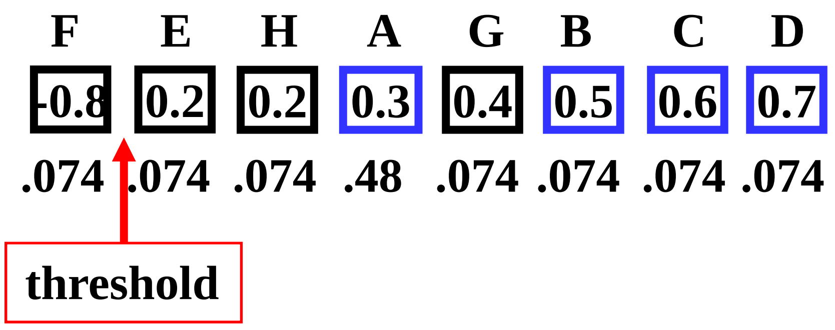
ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

Multiply the correctly classified instances by 0.38
Multiply incorrectly classified instances by 2.63
Normalize to sum to 1.0

The ADABoost Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Average}\,\{½\,(1 - y_i\, h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln (\varepsilon_t /(1 - \varepsilon_t))$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- \alpha_t\, y_i\, h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t\, \alpha_t\, h_t(x))$

E1 classifier



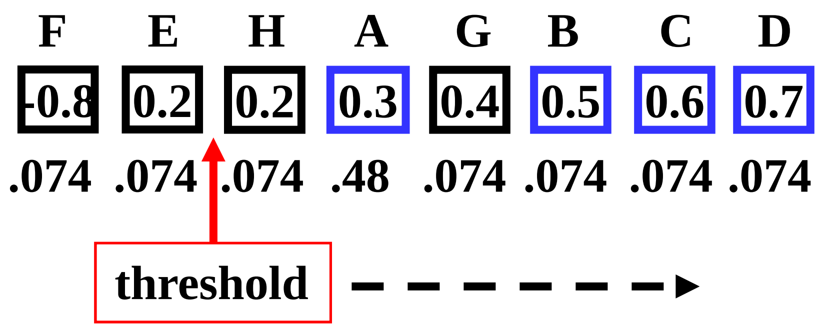
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1

Sign = +1, error = 0.222
Sign = -1, error = 0.778

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

E1 classifier



```
Classifier based on E1:
if ( sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1
```

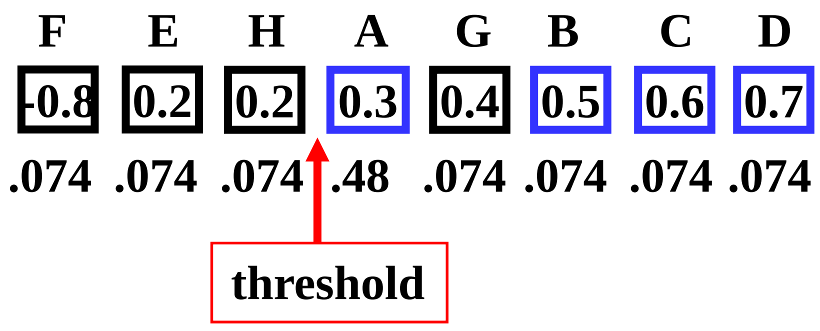
Sign = +1, error = 0.148
Sign = -1, error = 0.852

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

The Best E1 classifier

Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

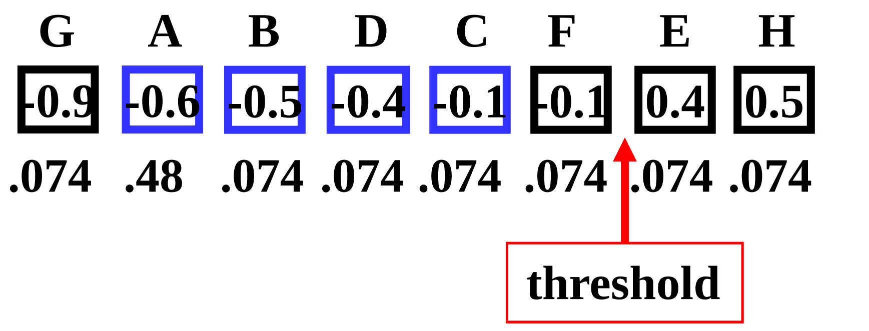
sign = +1 or -1



Sign = +1, error = 0.074

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

The Best E2 classifier



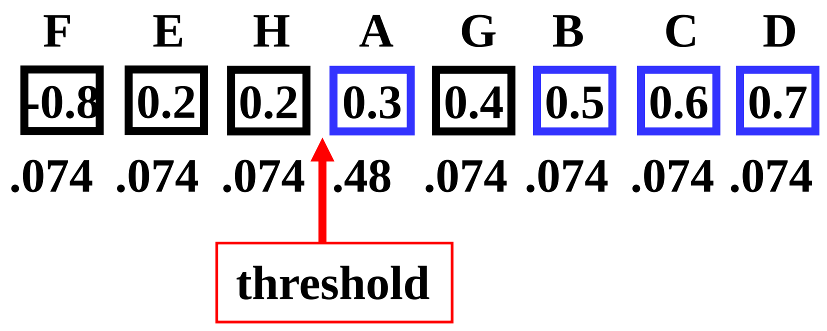
Classifier based on E2:
if (sign*wt(E2) > thresh) > 0)
    face = true

sign = +1 or -1

Sign = -1, error = 0.148

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	-0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

The Best Classifier



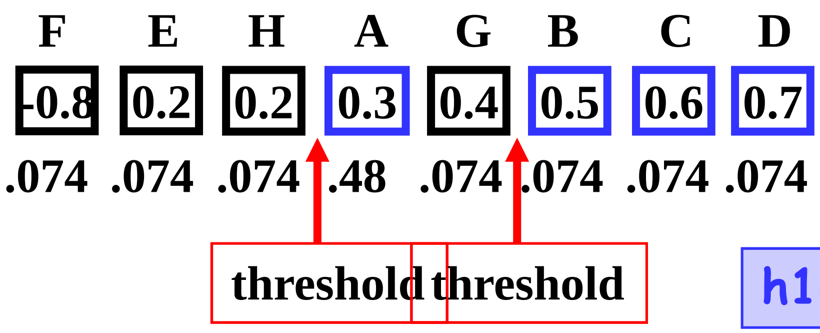
Sign = +1, error = 0.074

Classifier based on E1:
if (wt(E1) > 0.45) face = true

Alpha = 0.5ln((1-0.074) / 0.074)
= 1.26

ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

The Boosted Classifier Thus Far

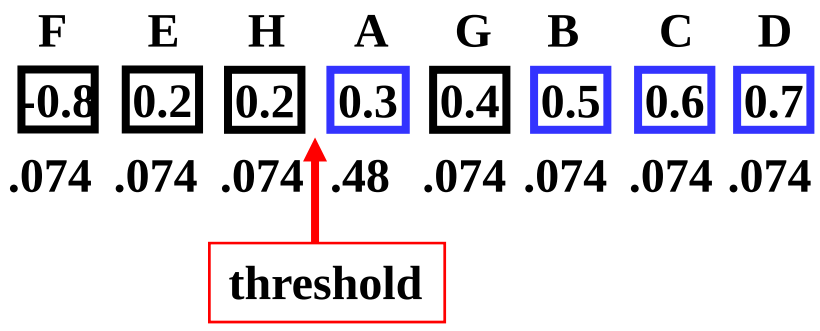


h1(X) = wt(E1) > 0.45 ? +1 : -1

h2(X) = wt(E1) > 0.25 ? +1 : -1

H(X) = sign(0.97 * h1(X) + 1.26 * h2(X))

Reweighting the Data



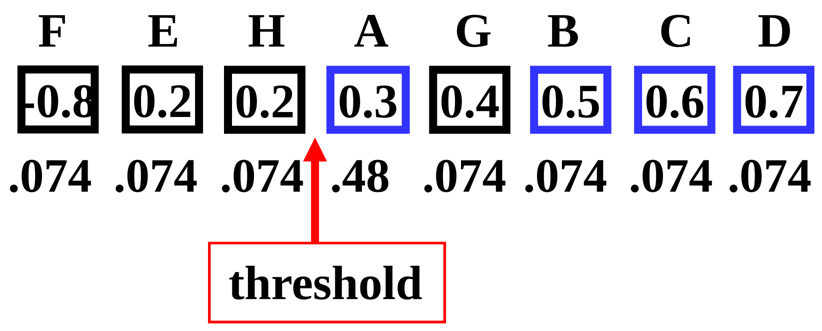
Sign = +1, error = 0.074

Exp(alpha) = exp(2.36) = 10
Exp(-alpha) = exp(-2.36) = 0.1

ID	E1	E2.	Class	Weight	
A	0.3	-0.6	+1	0.48*0.1	0.06
B	0.5	-0.5	+1	0.074*0.1	0.01
C	0.7	-0.1	+1	0.074*0.1	0.01
D	0.6	-0.4	+1	0.074*0.1	0.01
E	0.2	0.4	-1	0.074*0.1	0.01
F	-0.8	0.1	-1	0.074*0.1	0.01
G	0.4	-0.9	-1	0.074*10	0.86
H	0.2	0.5	-1	0.074*0.1	0.01

RENORMALIZE

Reweighting the Data



NOTE: THE WEIGHT OF "G" WHICH WAS MISCLASSIFIED BY THE SECOND CLASSIFIER IS NOW SUDDENLY HIGH

Sign = +1, error = 0.074

ID	E1	E2.	Class	Weight	
A	0.3	-0.6	+1	0.48*0.1	0.06
B	0.5	-0.5	+1	0.074*0.1	0.01
C	0.7	-0.1	+1	0.074*0.1	0.01
D	0.6	-0.4	+1	0.074*0.1	0.01
E	0.2	0.4	-1	0.074*0.1	0.01
F	-0.8	0.1	-1	0.074*0.1	0.01
G	0.4	-0.9	-1	0.074*10	0.86
H	0.2	0.5	-1	0.074*0.1	0.01

RENORMALIZE

AdaBoost

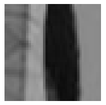
- In this example both of our first two classifiers were based on E1
  - Additional classifiers may switch to E2
- In general, the reweighting of the data will result in a different feature being picked for each classifier
- This also automatically gives us a *feature selection* strategy
  - In this data the wt(E1) is the most important feature

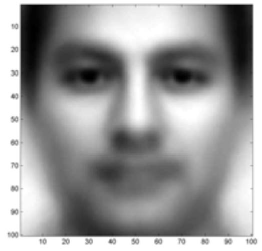
AdaBoost

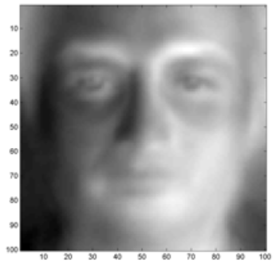
- NOT required to go with the best classifier so far
- For instance, for our second classifier, we might use the best E2 classifier, even though its worse than the E1 classifier
  - So long as its right more than 50% of the time

- We can *continue* to add classifiers even after we get 100% classification of the training data
  - Because the weights of the data keep changing
  - Adding new classifiers beyond this point is often a good thing to do

ADA Boost

 = 0.4 E1 - 0.4 E2

E_1 

E_2 

- The final classifier is
  - $H(x) = \text{sign}(\Sigma_t \, \alpha_t \, h_t(x))$

- The output is 1 if the total weight of all weak learners that classify *x* as 1 is greater than the total weight of all weak learners that classify it as -1

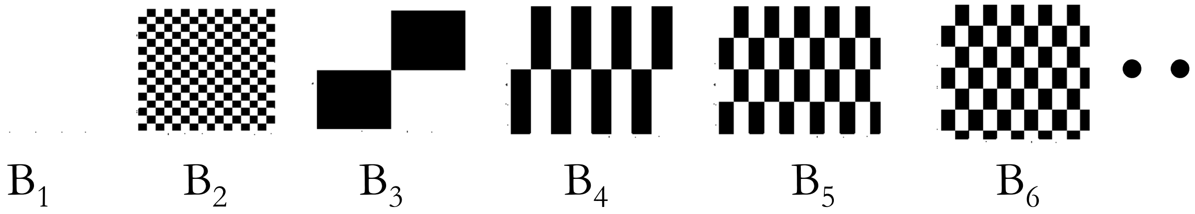
Boosting and Face Detection

- Boosting forms the basis of the most common technique for face detection today: The Viola-Jones algorithm.

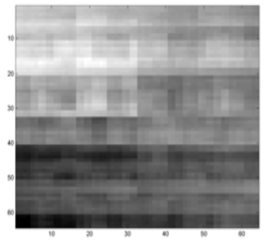
The problem of face detection

- Defining Features
  - Should we be searching for noses, eyes, eyebrows etc.?
    - Nice, but expensive
  - Or something simpler
- Selecting Features
  - Of all the possible features we can think of, which ones make sense
- Classification: Combining evidence
  - How does one combine the evidence from the different features?

Features: The Viola Jones Method



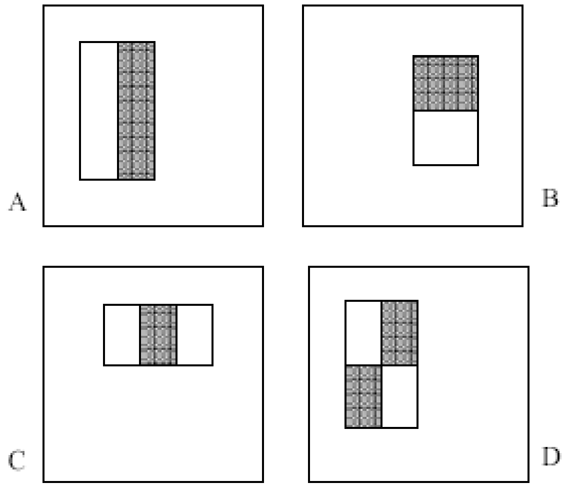
$$\text{Image} \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

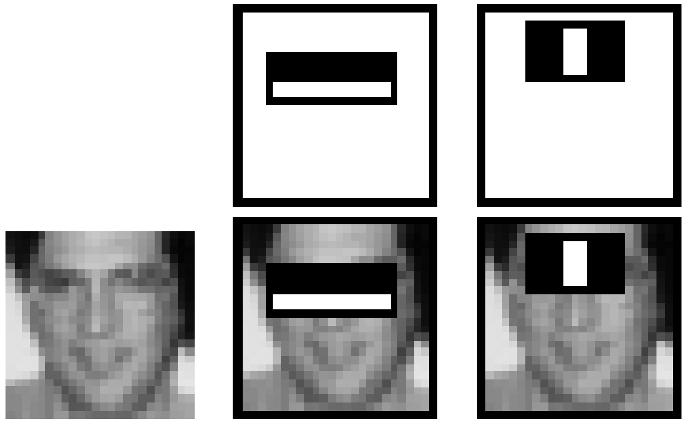


- Integral Features!!
  - Like the Checkerboard
- The same principle as we used to decompose images in terms of checkerboards:
  - The image of any object has changes at various scales
  - These can be represented coarsely by a checkerboard pattern
- The checkerboard patterns must however now be *localized*
  - Stay within the region of the face

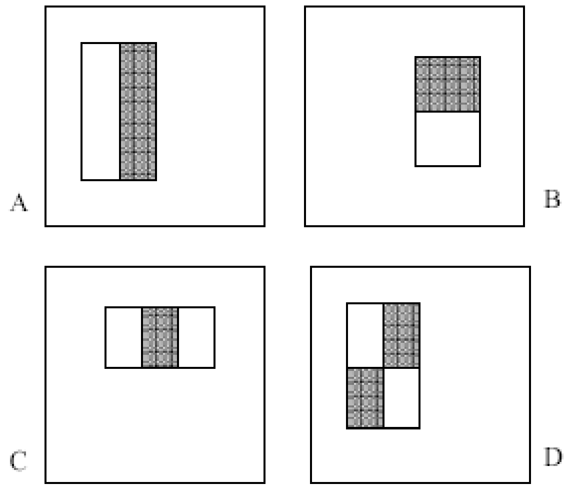
Features

- Checkerboard Patterns to represent facial features
  - The white areas are subtracted from the black ones.
  - Each checkerboard explains a *localized* portion of the image
- Four types of checkerboard patterns (only)



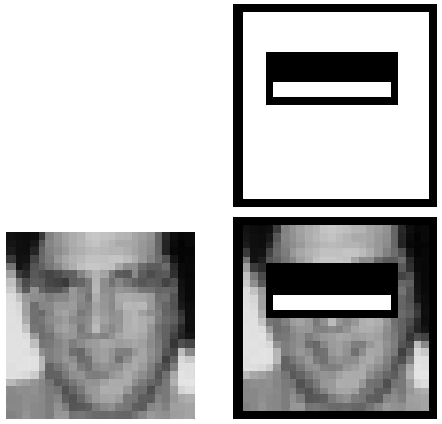


“Integral” features



- Each checkerboard has the following characteristics
  - Length
  - Width
  - Type
    - Specifies the number and arrangement of bands
- The four checkerboards above are the four used by Viola and Jones

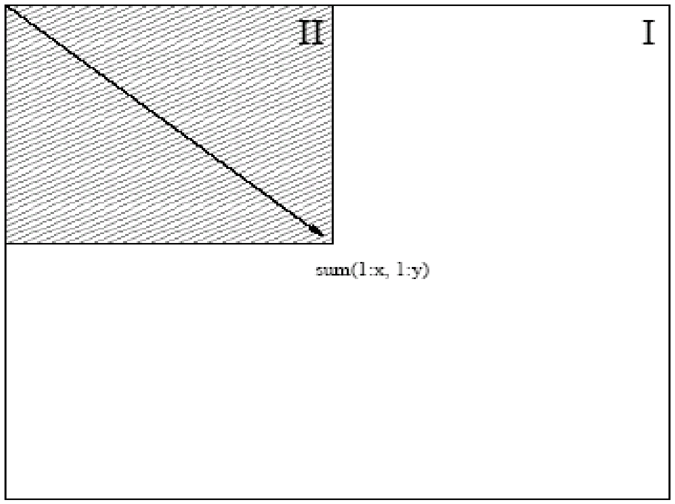
Explaining a portion of the face with a checker..



- How much is the difference in average intensity of the image in the black and white regions
  - Sum(pixel values in white region) – Sum(pixel values in black region)
- This is actually the dot product of the region of the face covered by the rectangle and the checkered pattern itself
  - White = 1, Black = -1

Integral images

- Summed area tables



- For each pixel store the sum of ALL pixels to the left of and above it.

Fast Computation of Pixel Sums

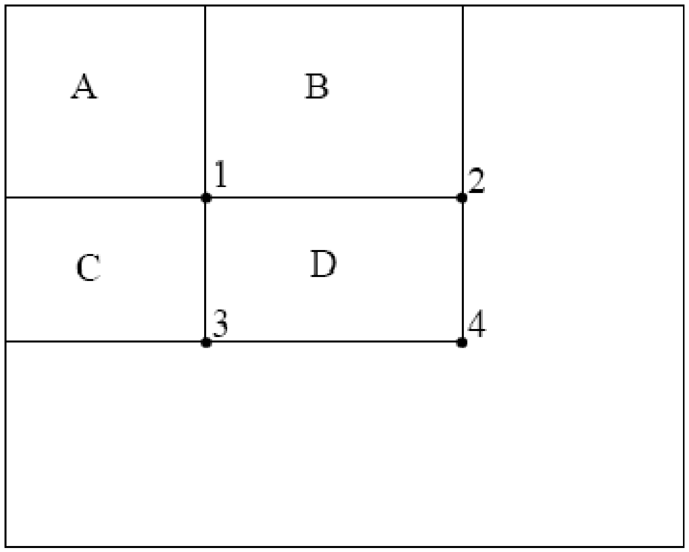
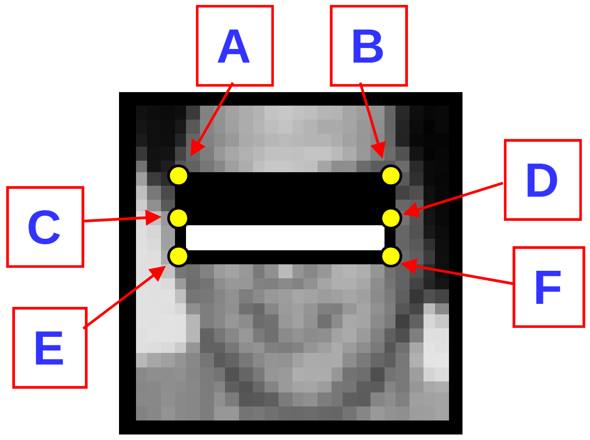


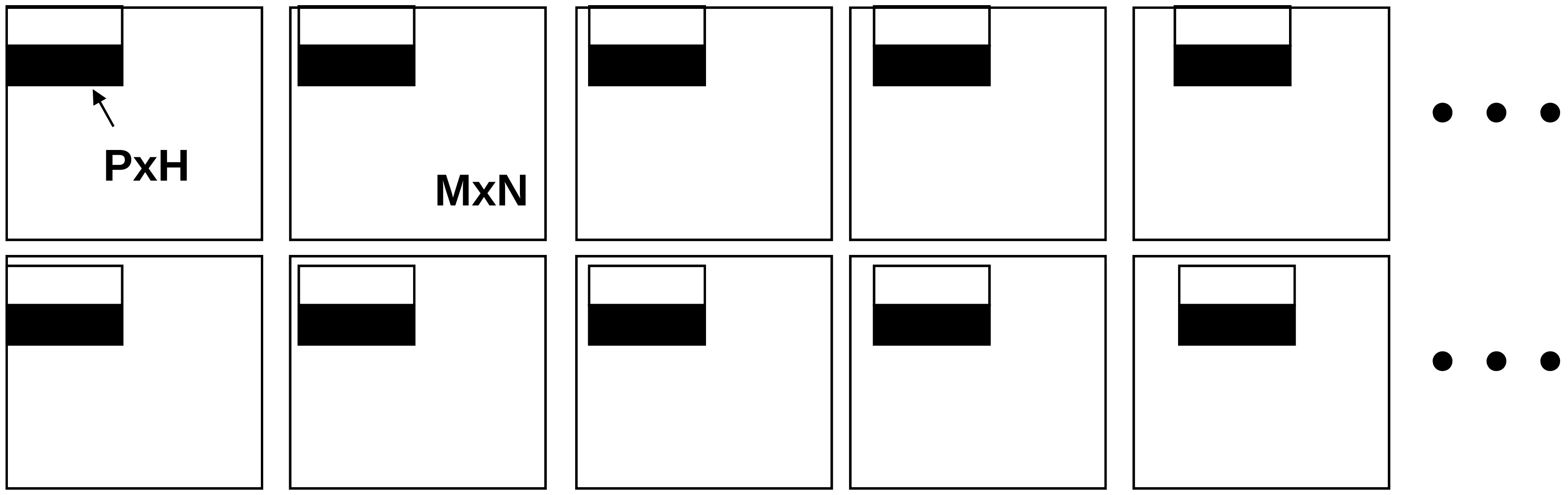
Figure 3: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

A Fast Way to Compute the Feature



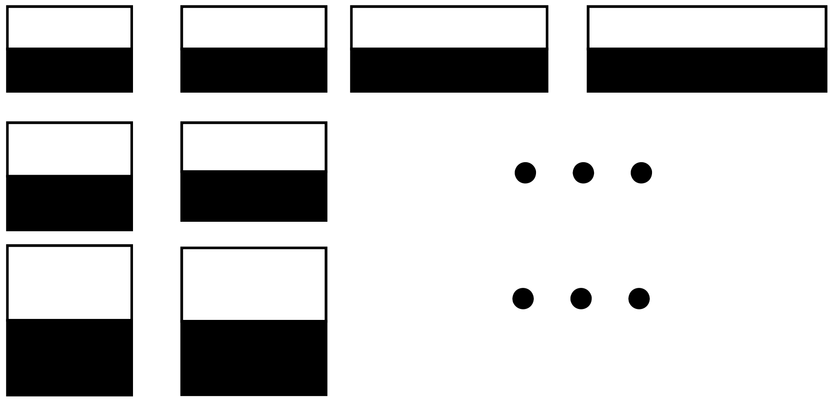
- Store pixel table for every pixel in the image
  - The sum of all pixel values to the left of and above the pixel
- Let A, B, C, D, E, F be the pixel table values at the locations shown
  - Total pixel value of black area = D + A – B – C
  - Total pixel value of white area = F + C – D – E
  - Feature value = (F + C – D – E) – (D + A – B – C)

How many features?



- Each checker board of width P and height H can start at
  - (0,0), (0,1),(0,2), … (0, N-P)
  - (1,0), (1,1),(1,2), … (1, N-P)
  - ..
  - (M-H,0), (M-H,1), (M-H,2), … (M-H, N-P)
- (M-H)*(N-P) possible starting locations
  - Each is a unique checker feature
    - E.g. at one location it may measure the forehead, at another the chin

How many features

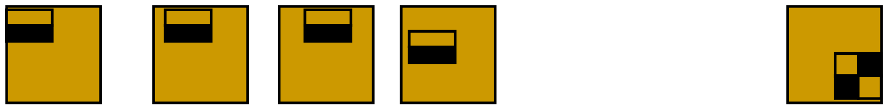


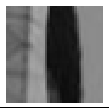
- Each feature can have many sizes
  - Width from (min) to (max) pixels
  - Height from (min ht) to (max ht) pixels
- At each size, there can be many starting locations
  - Total number of possible checkerboards of one type:
    No. of possible sizes x No. of possible locations
- There are four types of checkerboards
  - Total no. of possible checkerboards: VERY VERY LARGE!

Learning: No. of features

- Analysis performed on images of 24x24 pixels only
  - Reduces the no. of possible features to about 180000
- Restrict checkerboard size
  - Minimum of 8 pixels wide
  - Minimum of 8 pixels high
    - Other limits, e.g. 4 pixels may be used too
  - Reduces no. of checkerboards to about 50000

No. of features



	F1	F2	F3	F4	…..	F180000
	7	9	2	-1	…..	12
	-11	3	19	17	…..	2

- Each possible checkerboard gives us one feature
- A total of up to 180000 features derived from a 24x24 image!
- Every 24x24 image is now represented by a set of 180000 numbers
  - This is the set of features we will use for classifying if it is a face or not!

The Classifier

- The Viola-Jones algorithm uses a simple Boosting based classifier
- Each “weak learner” is a simple threshold
- At each stage find the best feature to classify the data with
  - I.e the feature that gives us the best classification of all the training data
    - Training data includes many examples of faces and non-face images
  - The classification rule is of the kind
    - If feature > threshold, face (or if feature < threshold, face)
    - The optimal value of “threshold” must also be determined.

The Weak Learner

- Training (for each weak learner):
  - For each feature f (of all 180000 features)
    - Find a threshold θ(f) and polarity *p*(f) (*p*(f) = *-1 or p*(f) = *1*) such that
      (f > *p*(f)*θ(f)) performs the best classification of faces
      - Lowest overall error in classifying all training data
        - Error counted over *weighted* samples
    - Let the optimal overall error for *f* be error(*f*)
  - Find the feature f' such that error(f') is lowest
  - The weak learner is the test (f' > *p*(f')*θ(f')) => face

- Note that the procedure for learning weak learners also identifies the most useful features for face recognition

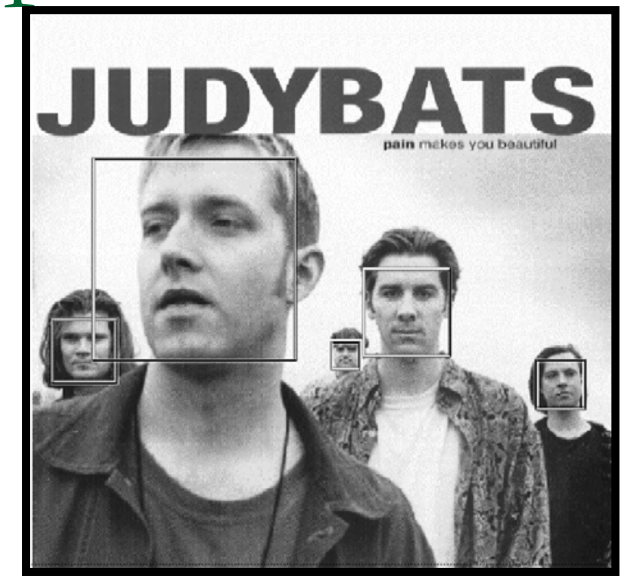
The Viola Jones Classifier

- A boosted threshold-based classifier
- First weak learner: Find the best feature, and its optimal threshold
  - Second weak learner: Find the best feature, for the weighted training data, and its threshold (weighting from one weak learner)
    - Third weak learner: Find the best feature for the reweighted data and its optimal threshold (weighting from two weak learners)
      - Fourth weak learner: Find the best feature for the reweighted data and its optimal threhsold (weighting from three weak learners)
        - ..

To Train

- Collect a large number of histogram equalized facial images
  - Resize all of them to 24x24
  - These are our “face” training set
- Collect a much much much larger set of 24x24 non-face images of all kinds
  - Each of them is histogram equalized
  - These are our “non-face” training set
- Train a boosted classifier

The Viola Jones Classifier



- During tests:
  - Given any new 24x24 image
    - $H(f) = \text{Sign}(\Sigma_f \alpha_f (f > p_f \theta(f)))$
    - Only a small number of features (f < 100) typically used
- Problems:
  - Only classifies 24 x 24 images entirely as faces or non-faces
    - Typical pictures are much larger
    - They may contain many faces
    - Faces in pictures can be much larger or smaller
  - Not accurate enough

Multiple faces in the picture

- Scan the image
  - Classify each 24x24 rectangle from the photo
  - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform (N-24)*(M-24) classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Multiple faces in the picture

- Scan the image
  - Classify each 24x24 rectangle from the photo
  - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform (N-24)*(M-24) classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Multiple faces in the picture

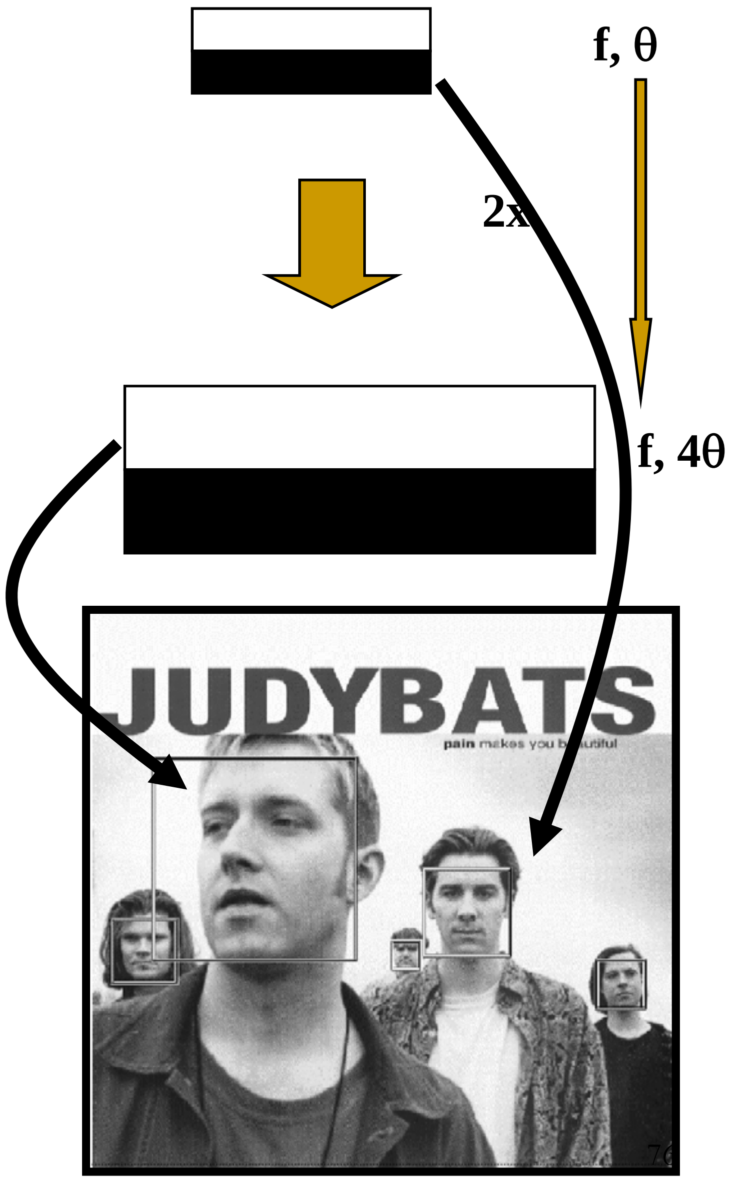
- Scan the image
    - Classify each 24x24 rectangle from the photo
    - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform (N-24)*(M-24) classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Multiple faces in the picture

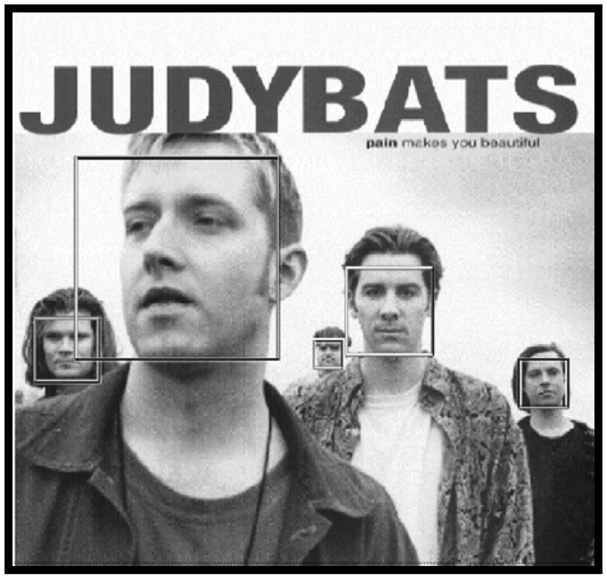
- Scan the image
  - Classify each 24x24 rectangle from the photo
  - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform (N-24)*(M-24) classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Face size solution

- We already have a classifier
  - That uses weak learners
- *Scale each classifier*
  - Every weak learner
  - Scale its size up by factor α. Scale the threshold up to $\alpha^2\theta$.
  - Do this for many scaling factors



Overall solution



- Scan the picture with classifiers of size 24x24
- Scale the classifier to 26x26 and scan
- Scale to 28x28 and scan etc.

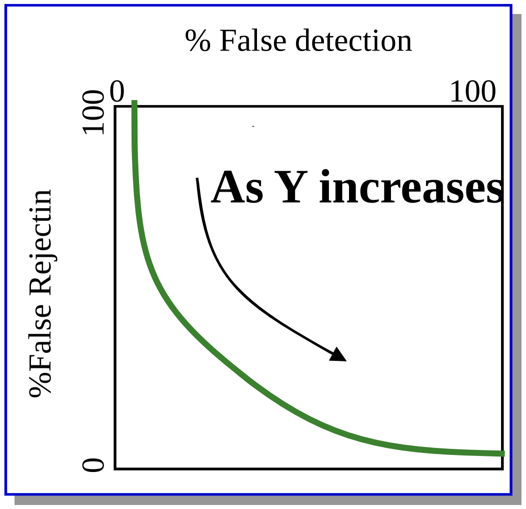
- Faces of different sizes will be found at different scales

False Rejection vs. False detection

- False Rejection: There's a face in the image, but the classifier misses it
    - Rejects the hypothesis that there's a face
- False detection: Recognizes a face when there is none.

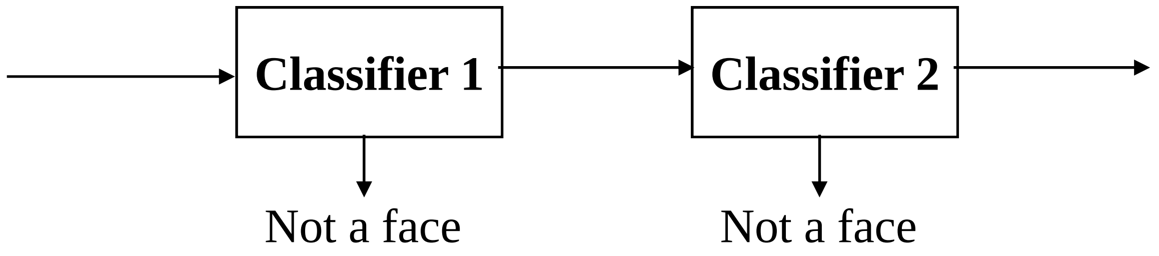
- Classifier:
    - Standard boosted classifier: $H(x) = \mathrm{sign}(\Sigma_t\, \alpha_t\, h_t(x))$
    - Modified classifier $H(x) = \mathrm{sign}(\Sigma_t\, \alpha_t\, h_t(x) + Y)$
        - Y is a bias that we apply to the classifier.
        - If Y is large, then we assume the presence of a face even when we are not sure
    - By increasing Y, we can reduce false rejection, while increasing false detection
        - Many instances for which $\Sigma_t\, \alpha_t\, h_t(x)$ is negative get classified as faces

ROC



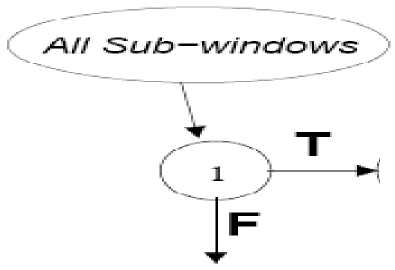
- Ideally false rejection will be 0%, false detection will also be 0%
- As Y increases, we reject faces less and less
  - But accept increasing amounts of garbage as faces
- Can set Y so that we rarely miss a face

Problem: Not accurate enough, too slow



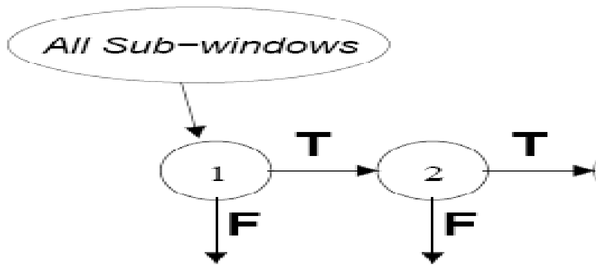
- If we set Y high enough, we will never miss a face
  - But will classify a lot of junk as faces
- Solution: Classify the output of the first classifier with a second classifier
  - And so on.

Cascaded Classifiers



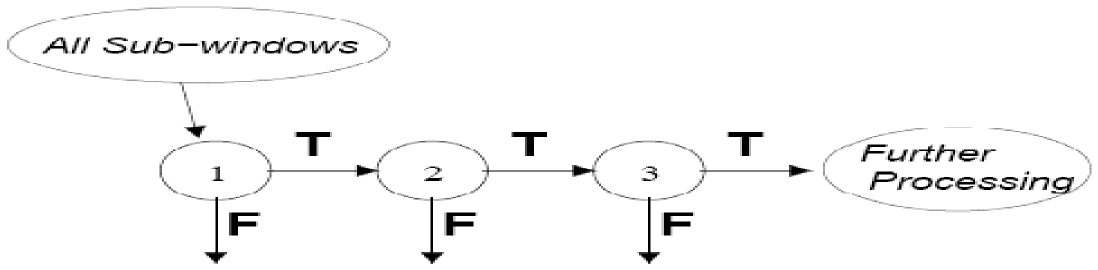
- Build the first classifier to have near-zero false rejection rate
  - But will reject a large number of non-face images

Cascaded Classifiers



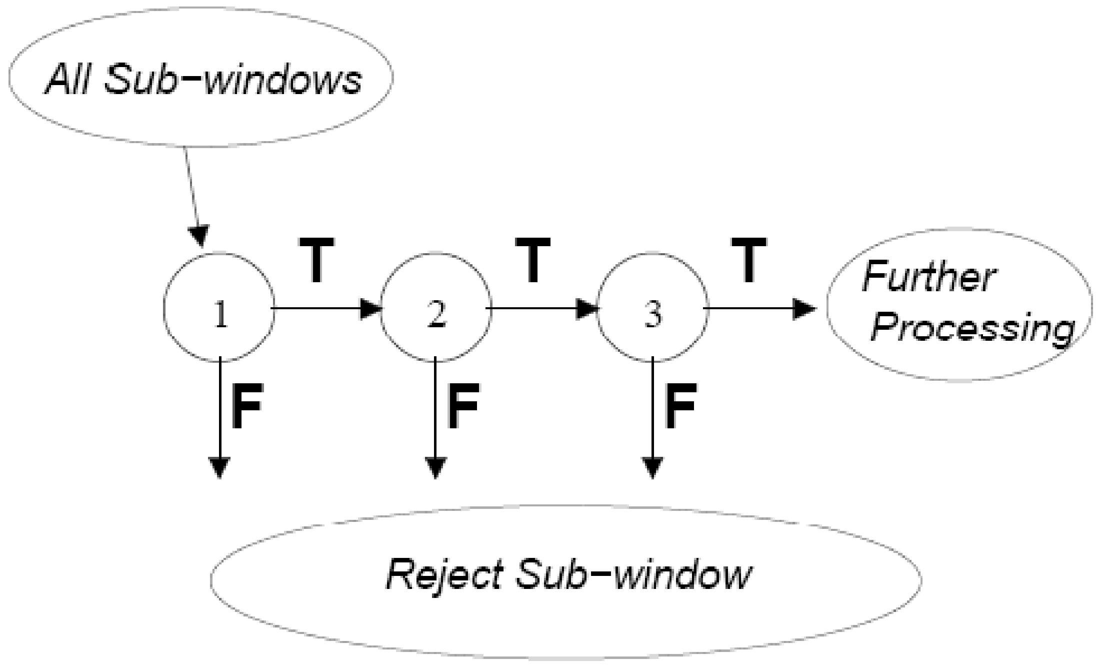
- Build the first classifier to have near-zero false rejection rate
  - But will reject a large number of non-face images
- Filter all training data with this classifier
- Build a second classifier on the data that have been passed by the first classifier, to have near-zero false rejection rate
  - This classifier will be different from the first one
    - Different data set

Cascaded Classifiers

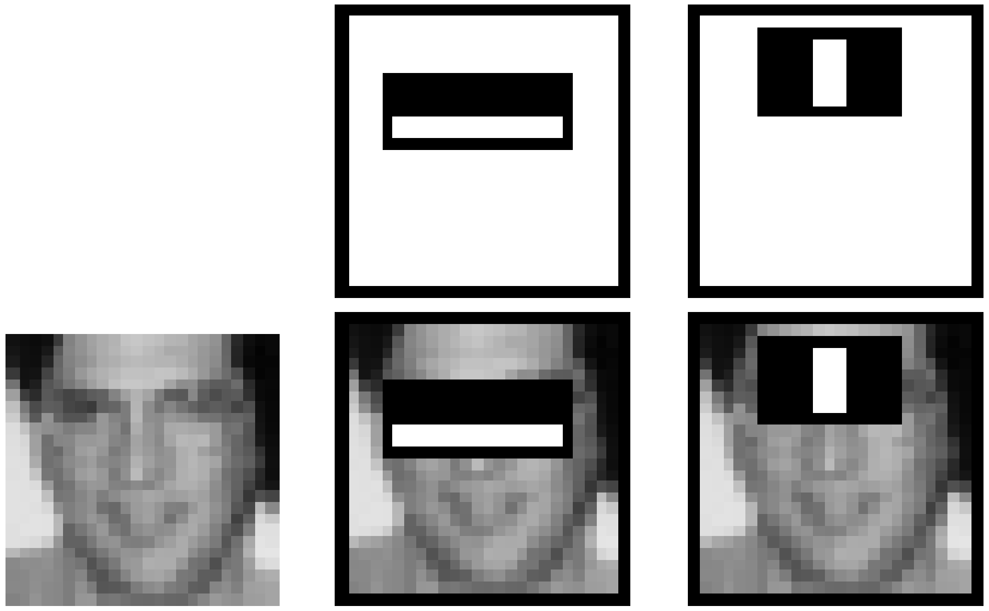


- Build the first classifier to have near-zero false rejection rate
  - But will reject a large number of non-face images
- Filter all training data with this classifier
- Build a second classifier on the data that have been passed by the first classifier, to have near-zero false rejection rate
  - This classifier will be different from the first one
    - Different data set
- Filter all training data with the cascade of the first two classifiers
- Build a third classifier on data passed by the cascade..
  - And so on..

Final Cascade of Classifiers



Useful Features Learned by Boosting



Detection in Real Images

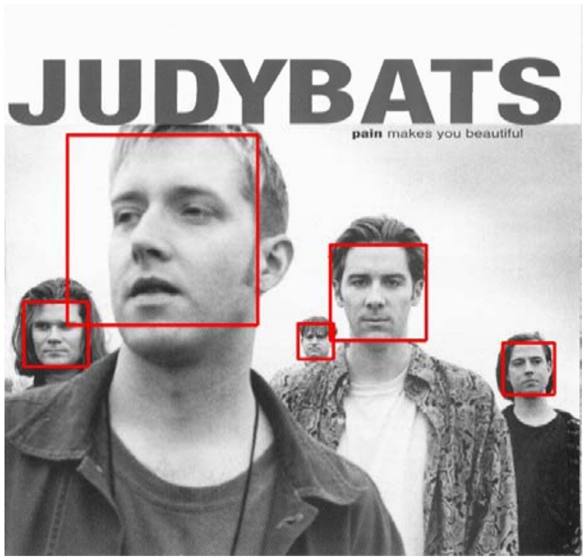
- Basic classifier operates on 24 x 24 subwindows
- Scaling:
  - Scale the detector (rather than the images)
  - Features can easily be evaluated at any scale
  - Scale by factors of 1.25
- Location:
  - Move detector around the image (e.g., 1 pixel increments)
- Final Detections
  - A real face may result in multiple nearby detections
  - Postprocess detected subwindows to combine overlapping detections into a single detection

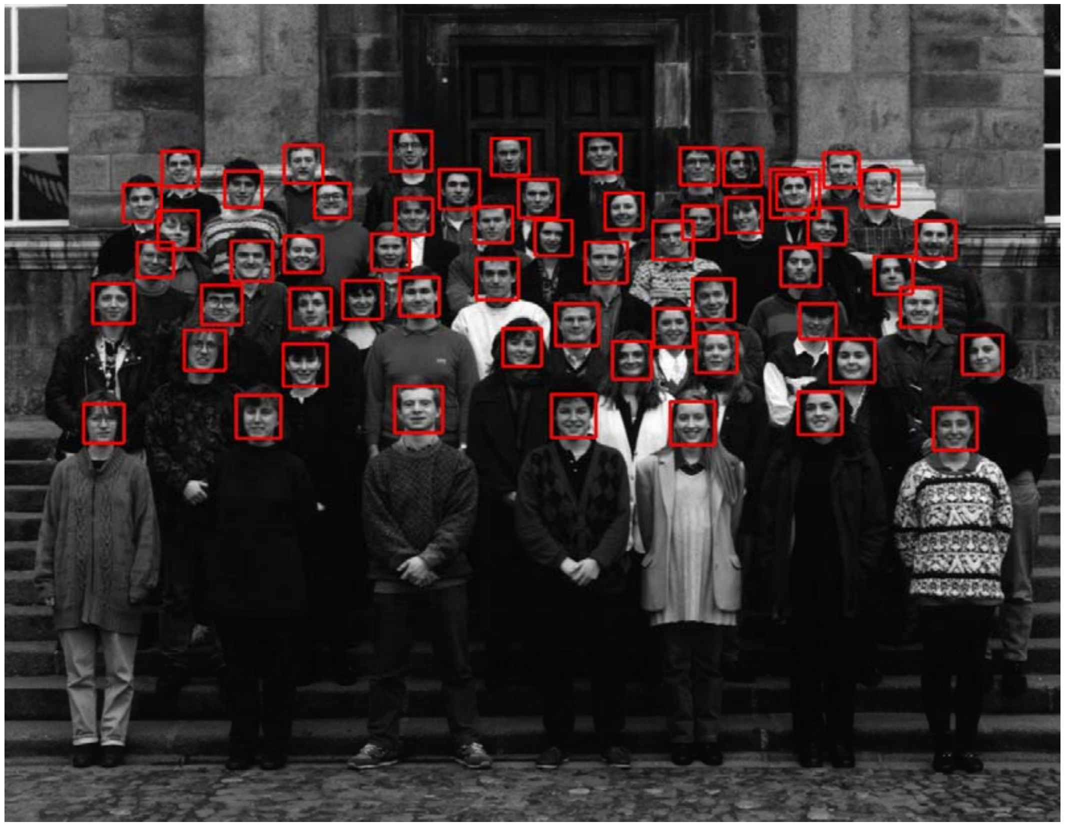
Training

- In paper, 24x24 images of faces and non faces (positive and negative examples).

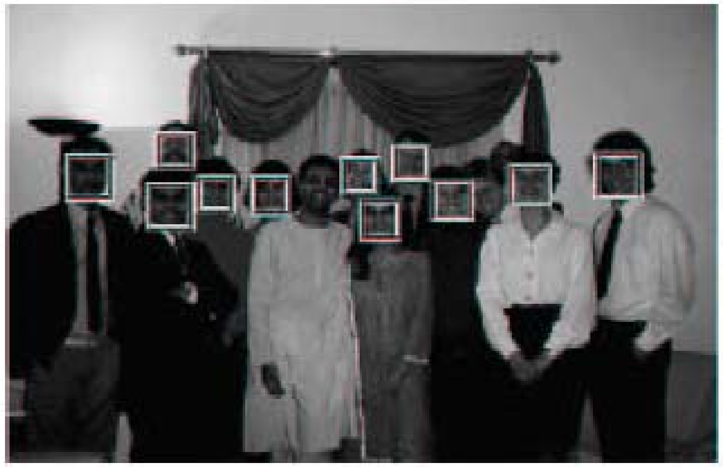
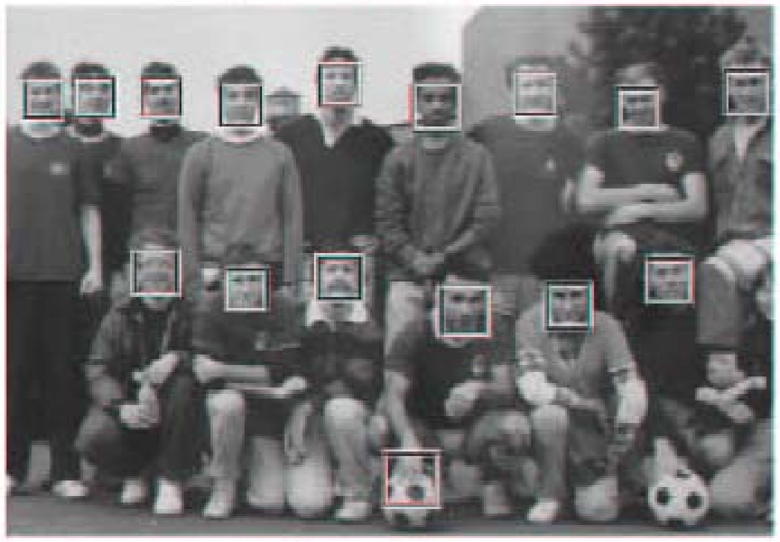
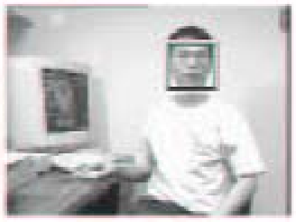
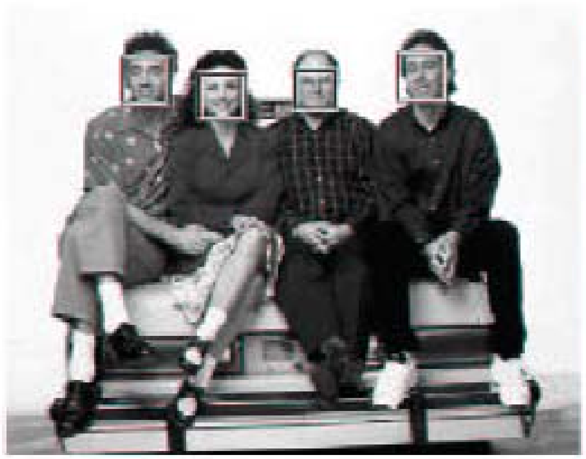
Sample results using the Viola-Jones Detector

- Notice detection at multiple scales





More Detection Examples

Practical implementation

- Details discussed in Viola-Jones paper
- Training time = weeks (with 5k faces and 9.5k non-faces)
- Final detector has 38 layers in the cascade, 6060 features
- 700 Mhz processor:
  - Can process a 384 x 288 image in 0.067 seconds (in 2003 when paper was written)