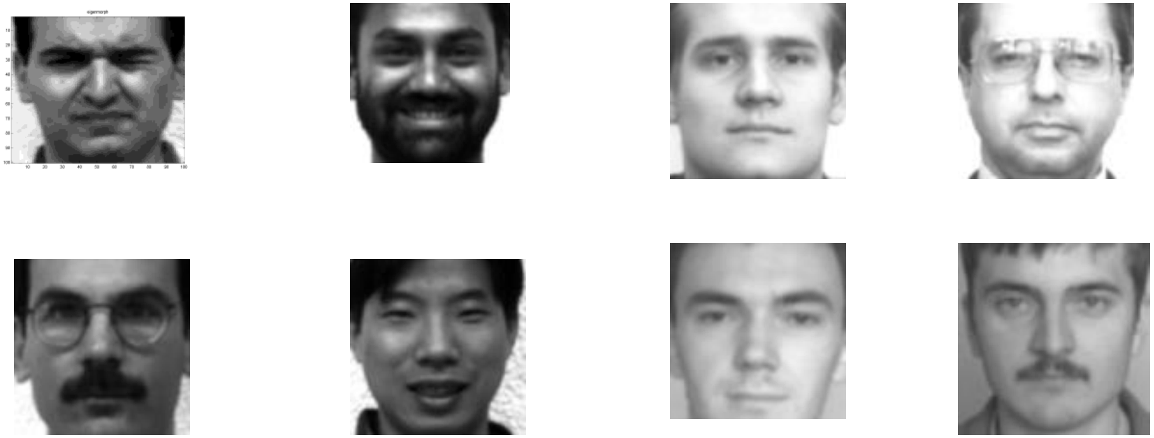
Machine Learning for Signal Processing

Detecting faces (& other objects) in images

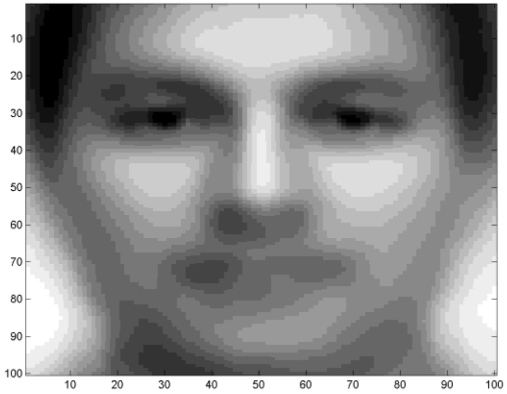
Class 8. 23 Sep 2014

Instructor: Gary Overett

Last Lecture: How to describe a face

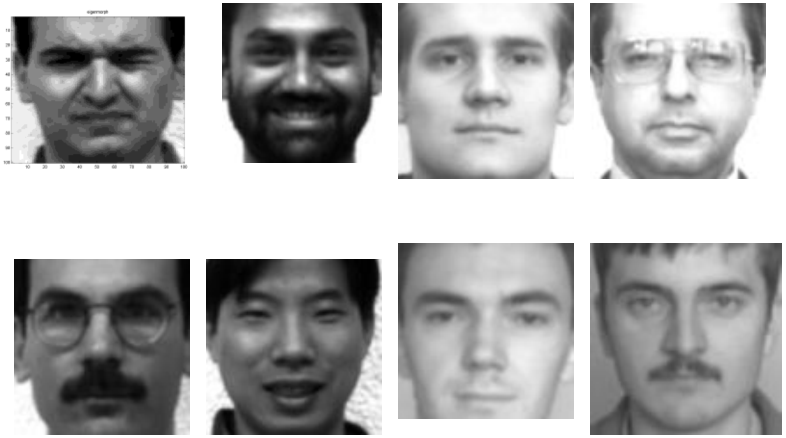


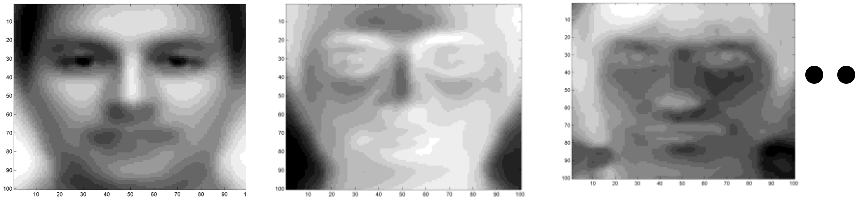
The typical face



- A “typical face” that captures the essence of “facehood”..
- The principal Eigen face..

A collection of least squares typical faces



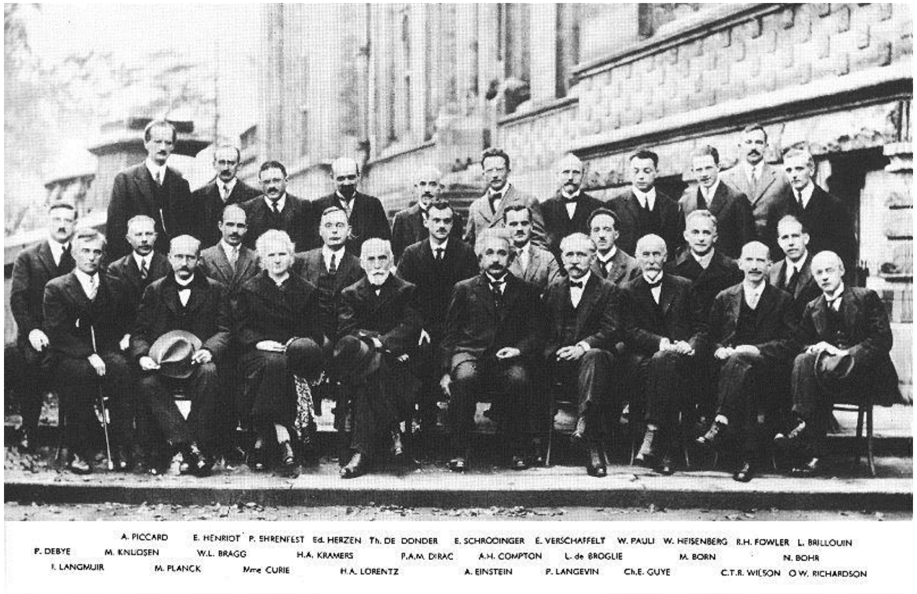


- Extension: Many Eigenfaces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + .. + w_{f,k} V_k$
  - V_2 is used to “correct” errors resulting from using only V_1
  - V_3 corrects errors remaining after correction with V_2
  - And so on..

- $V = [V_1\ V_2\ V_3]$ can be computed through Eigen analysis

Detecting Faces in Images

Detecting Faces in Images



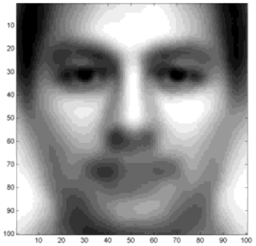
- Finding face like patterns
  - How do we find if a picture has faces in it
  - Where are the faces?

- A simple solution:
  - Define a “typical face”
  - Find the “typical face” in the image

Given an image and a ‘typical’ face how do I find the faces?

400 × 200
(RGB)

+

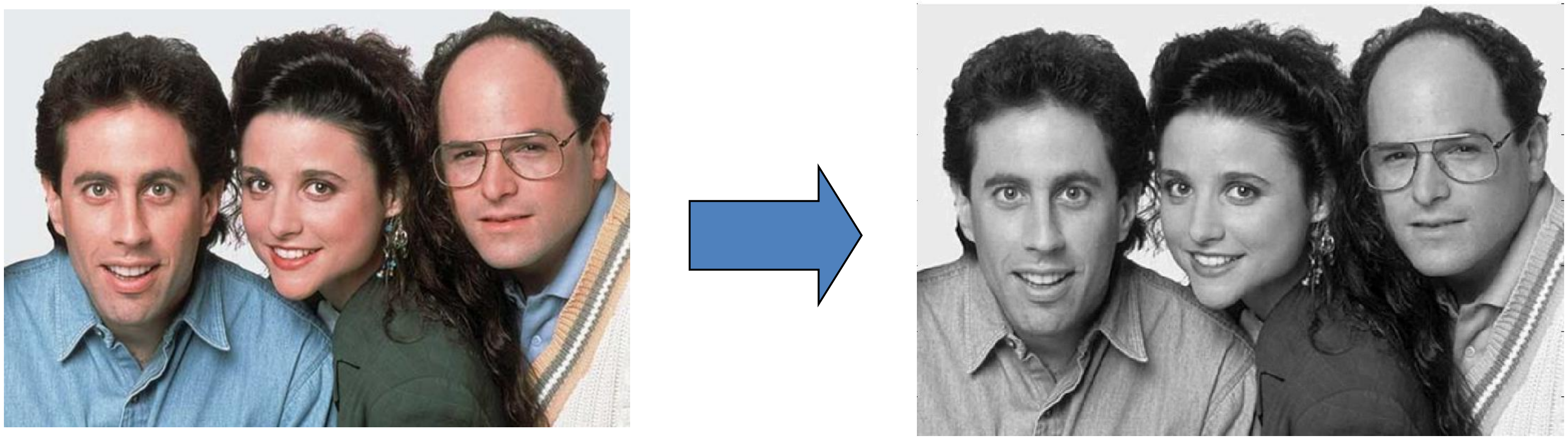


100 × 100

+

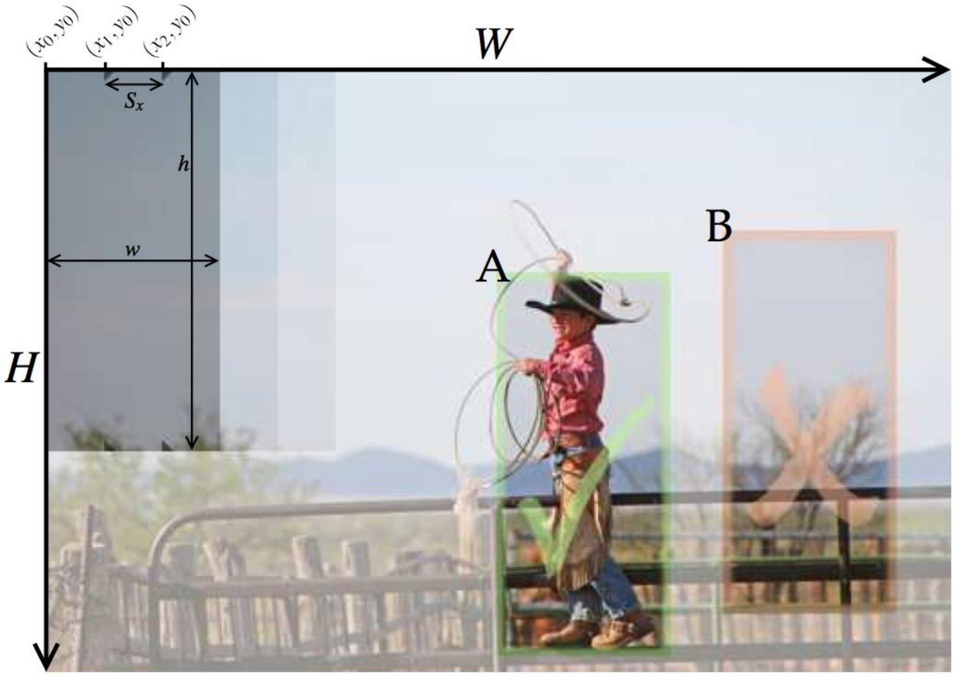
?

Finding faces in an image



- Picture is larger than the “typical face”
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - R + G + B
  - Not very useful to work in color

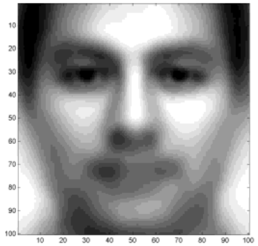
Sliding Windows



```
slidingWindows($I, W, H, w, h, S_x, S_y$)
    for $y = 0, S_y, 2S_y, 3S_y, \ldots, H - h$
        for $x = 0, S_x, 2S_x, 3S_x, \ldots, W - w$
            Query Pedestrian at I(x,y)
        endfor
    endfor
```

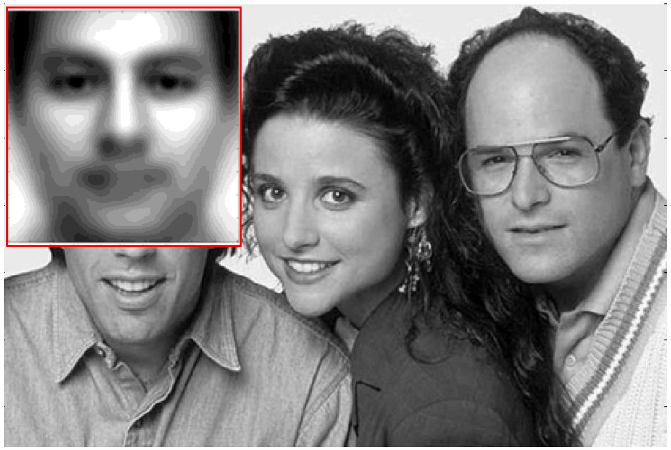
Figure 1.3: The Sliding Windows Methodology. In order to find an object instance, a detector must be run over multiple sub-regions of the image. Consequently the detector must use a minimal amount of processing for each individual window. To the right of the main image we show the basics of a sliding windows algorithm. The algorithm takes as parameters, the input image I, its associated width W and height H, the width w and height h of the detection window, and the windowing step sizes S_x and S_y. For exhaustive searching of the image a step size of 1 is used in both dimensions.

Finding faces in an image



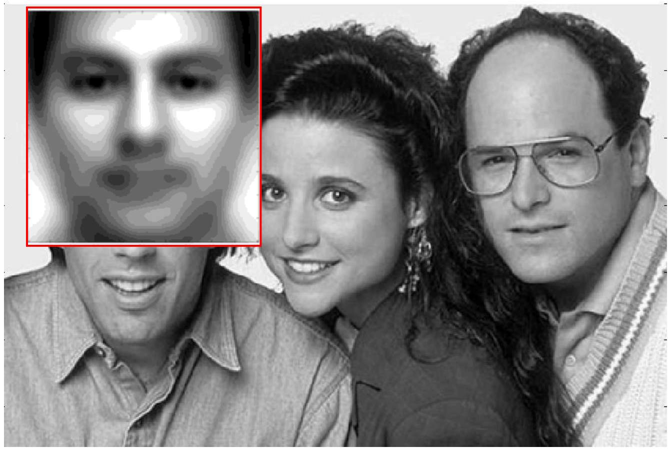
- Goal .. To find out if and where images that look like the "typical" face occur in the picture

Finding faces in an image



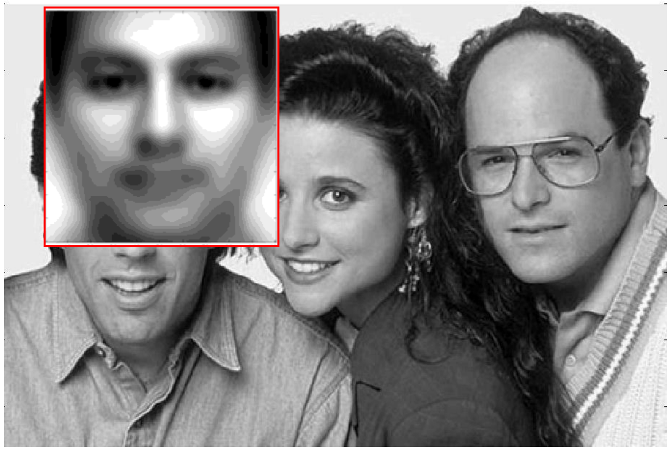
- Try to "match" the typical face to each location in the picture

Finding faces in an image



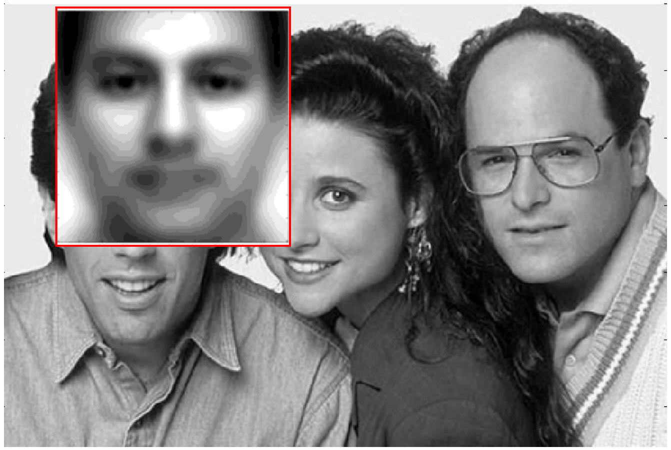
- Try to “match” the typical face to each location in the picture

Finding faces in an image



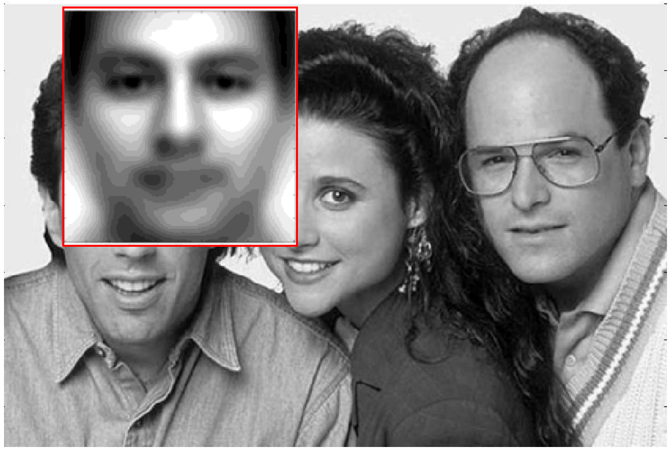
- Try to "match" the typical face to each location in the picture

Finding faces in an image



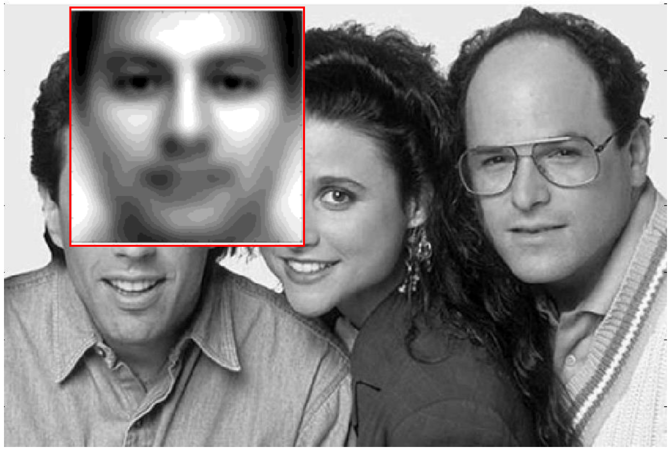
- Try to “match” the typical face to each location in the picture

Finding faces in an image



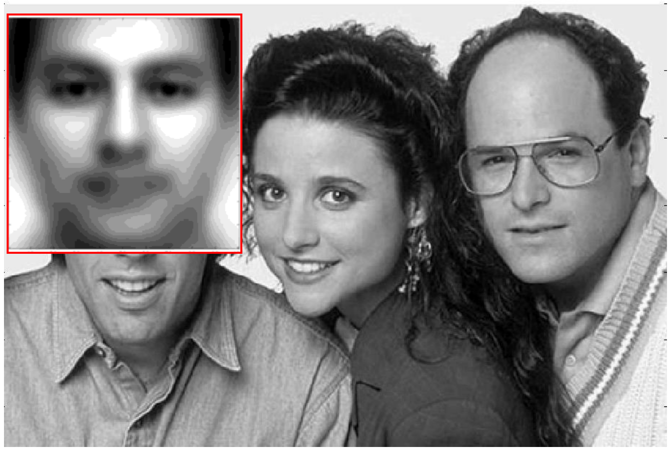
- Try to “match” the typical face to each location in the picture

Finding faces in an image



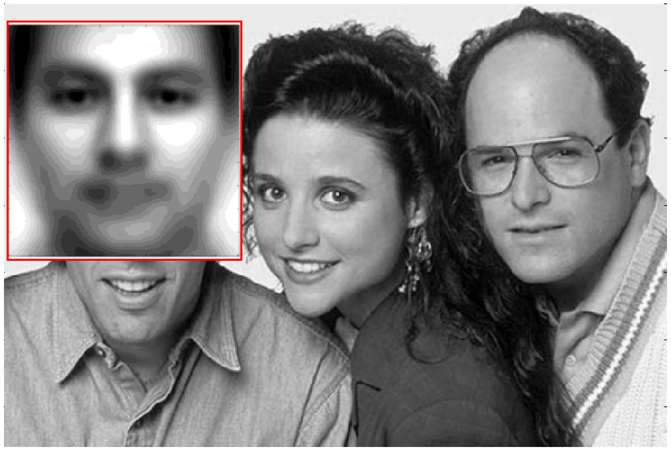
- Try to “match” the typical face to each location in the picture

Finding faces in an image



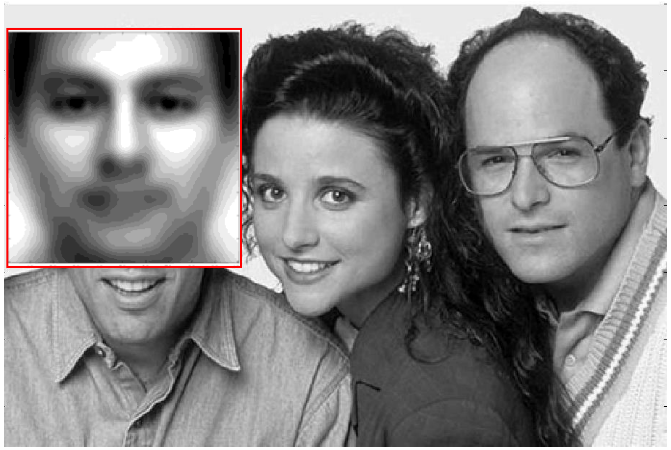
- Try to “match” the typical face to each location in the picture

Finding faces in an image



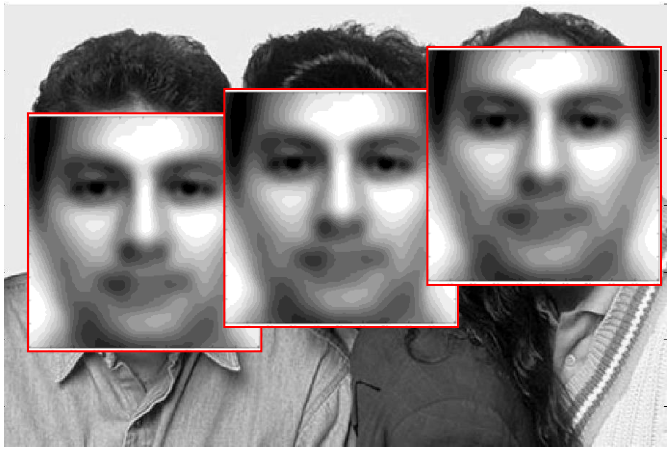
- Try to "match" the typical face to each location in the picture

Finding faces in an image



- Try to “match” the typical face to each location in the picture

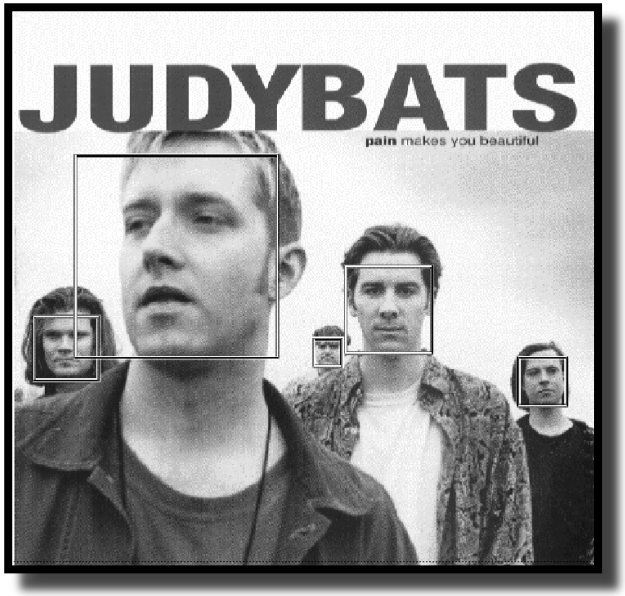
Finding faces in an image



- Try to “match” the typical face to each location in the picture
- The “typical face” will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!

Sliding windows solves only the issue of location – what about scale?

- Not all faces are the same size
- Some people have bigger faces
- The size of the face on the image changes with perspective
- Our “typical face” only represents one of these sizes



Scale-Space Pyramid

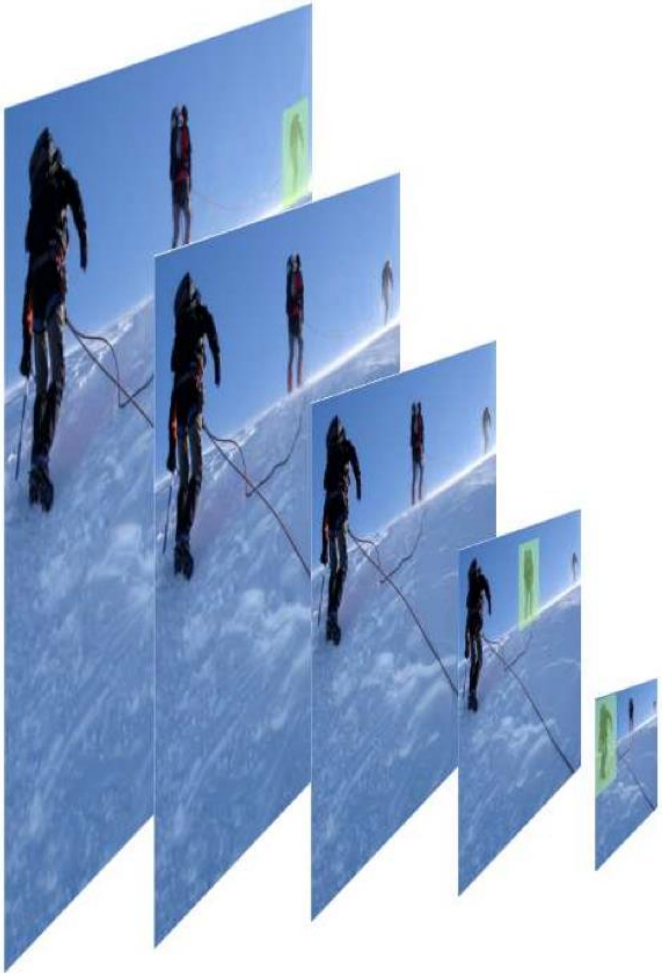
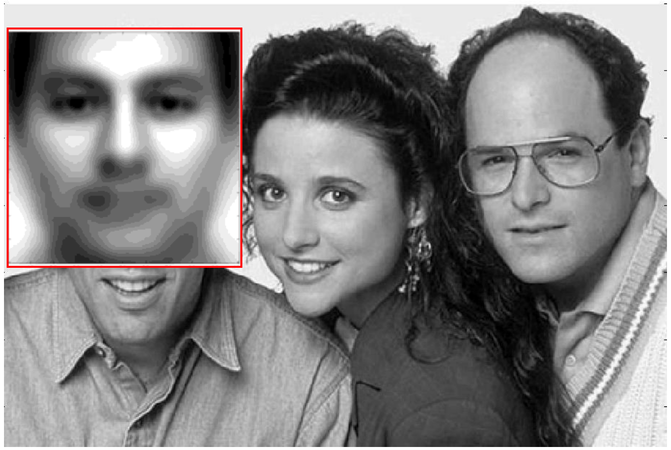


Figure 1.4: The Scale-Space Pyramid. The detector is run using the sliding windows approach over the input image at various scales. When the scale of the person matches the detector scale the classifier will (hopefully) fire yielding an accurate detection.

Speed concerns

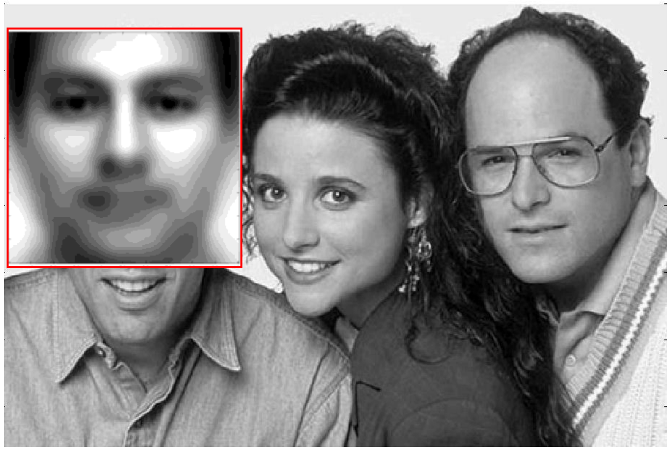
- Sliding windows AND Scale-space pyramid may yield million’s of ‘windows’ to investigate!
- Especially for small objects in large images

How to “match”



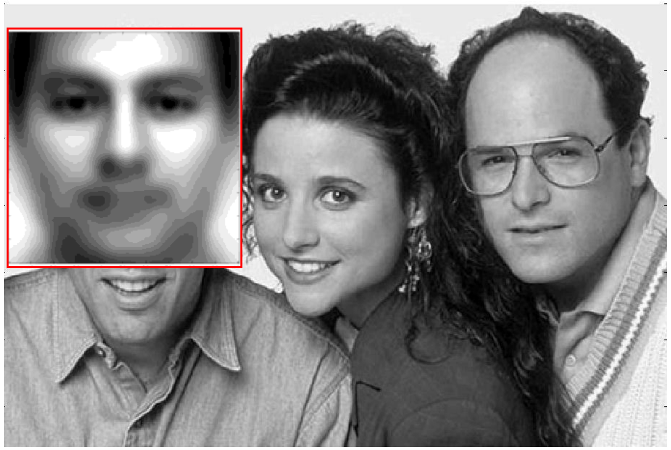
- What exactly is the “match”
  - What is the match “score”

How to “match”



- What exactly is the “match”
  - What is the match “score”
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of the image
  - Compute the dot product of the typical face vector and the “region” vector

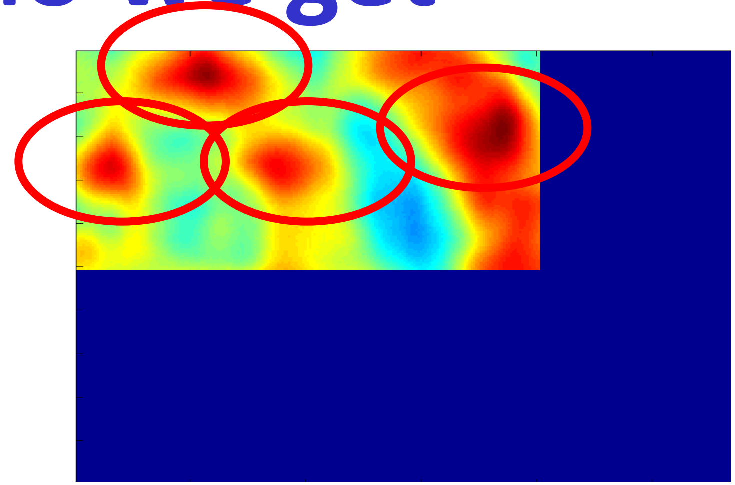
How to “match”



- What exactly is the “match”
  - What is the match “score”
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of the image
  - Compute the dot product of the typical face vector and the “region” vector

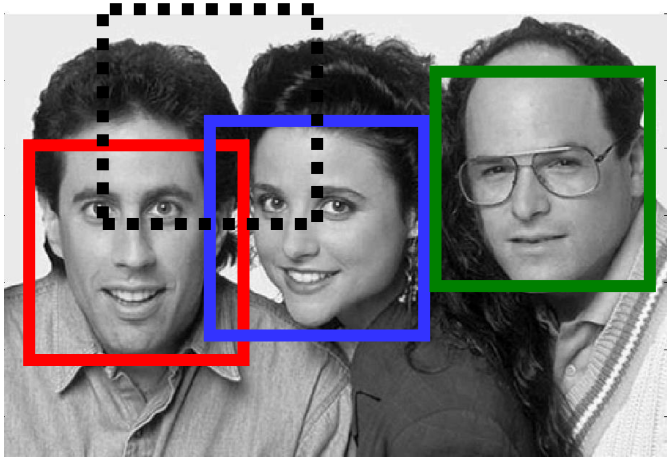
I hate this! ☺

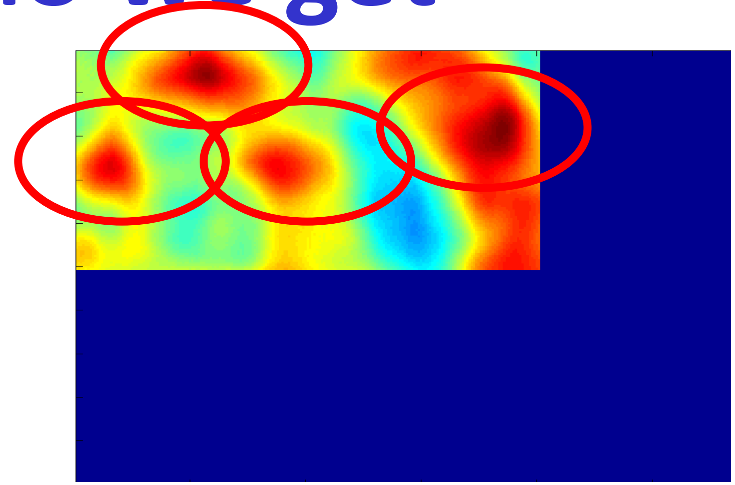
What do we get



- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!

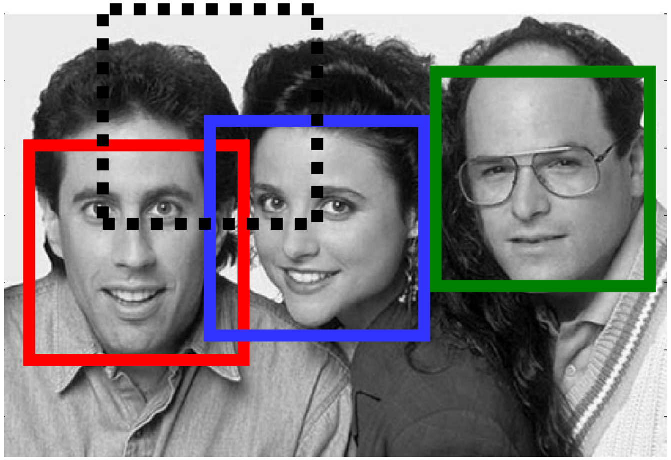
What do we get

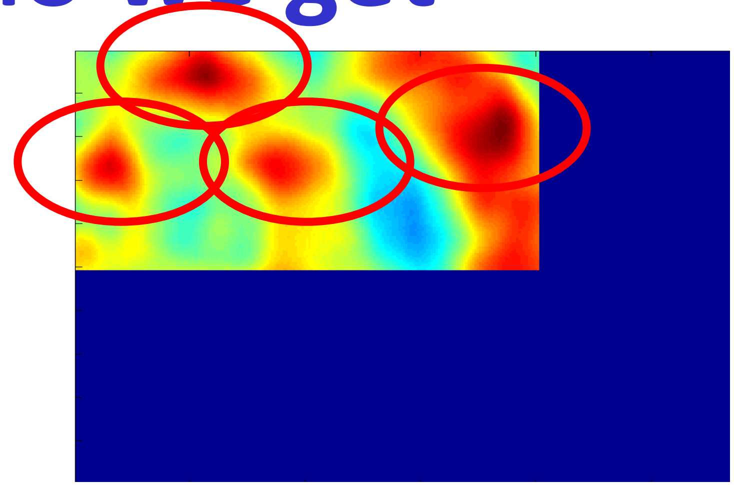




- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George’s face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm

What do we get

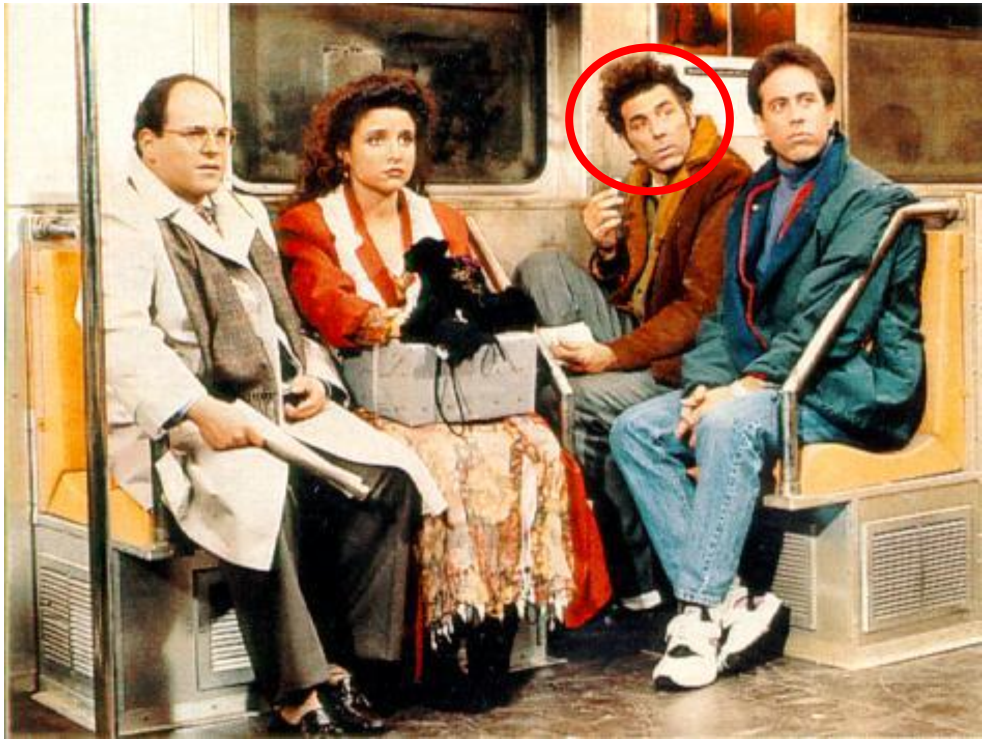




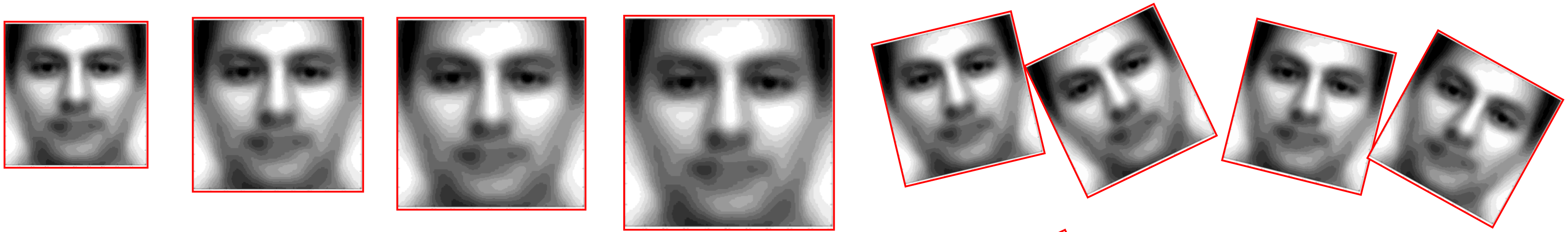
- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George's face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm

Location – Scale – What about Rotation?

- The head need not always be upright!
    - Our typical face image was upright



Solution



- Create many “typical faces”
  - One for each scaling factor
  - One for each rotation
    - How will we do this?
- Match them all
- Does this work
  - Kind of .. Not well enough at all
  - We need more sophisticated models

Face Detection: A Quick Historical Perspective

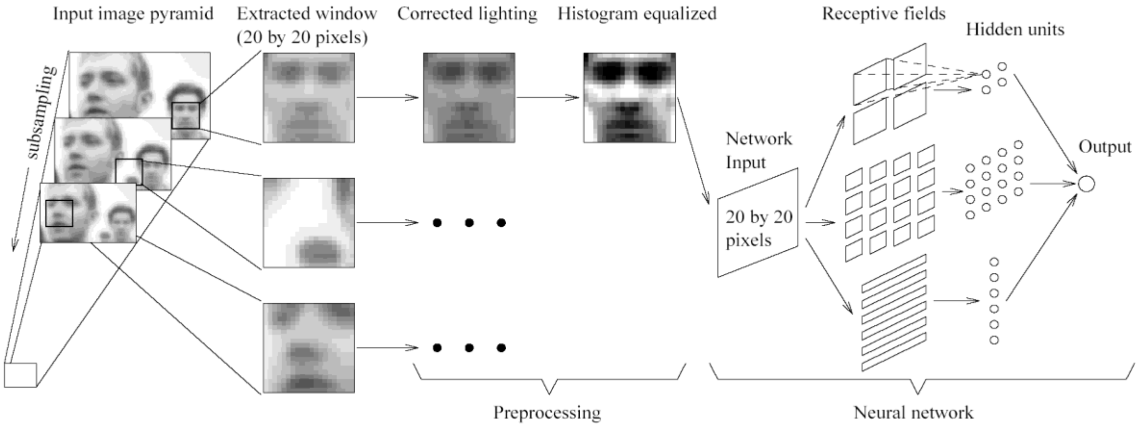


Figure 1: The basic algorithm used for face detection.

- Many more complex methods
  - Use edge detectors and search for face like patterns
  - Find "feature" detectors (noses, ears..) and employ them in complex neural networks..

- The Viola Jones method
  - Boosted cascaded classifiers

Face Detection: A Quick Historical Perspective

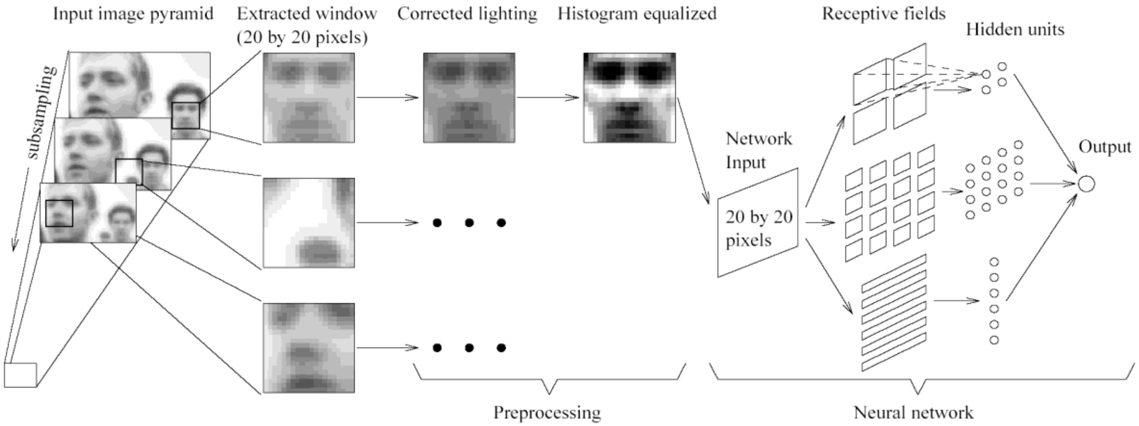


Figure 1: The basic algorithm used for face detection.

- Many more complex methods
  - Use edge detectors and search for face like patterns
  - Find "feature" detectors (noses, ears..) and employ them in complex neural networks..

- **The Viola Jones method (20K+ Citations!)**
  - **Boosted cascaded classifiers**

And even before that – what is classification?

- Given “features” describing an entity, determine the category it belongs to
  - Walks on two legs, has no hair. Is this
    - A Chimpanizee
    - A Human
  - Has long hair, is 5’6” tall, is this
    - A man
    - A woman
  - Matches “eye” pattern with score 0.5, “mouth pattern” with score 0.25, “nose” pattern with score 0.1. Are we looking at
    - A face
    - Not a face?

Classification

- Multi-class classification
  - Many possible categories
    - E.g. Sounds “AH, IY, UW, EY..”
    - E.g. Images “Tree, dog, house, person..”

- Classes may overlap
  - Phoneme Classification vs Emotive State Classification vs Accents etc.

- Binary classification
  - Only two categories
    - Man vs. Woman
    - Face vs. not a face…

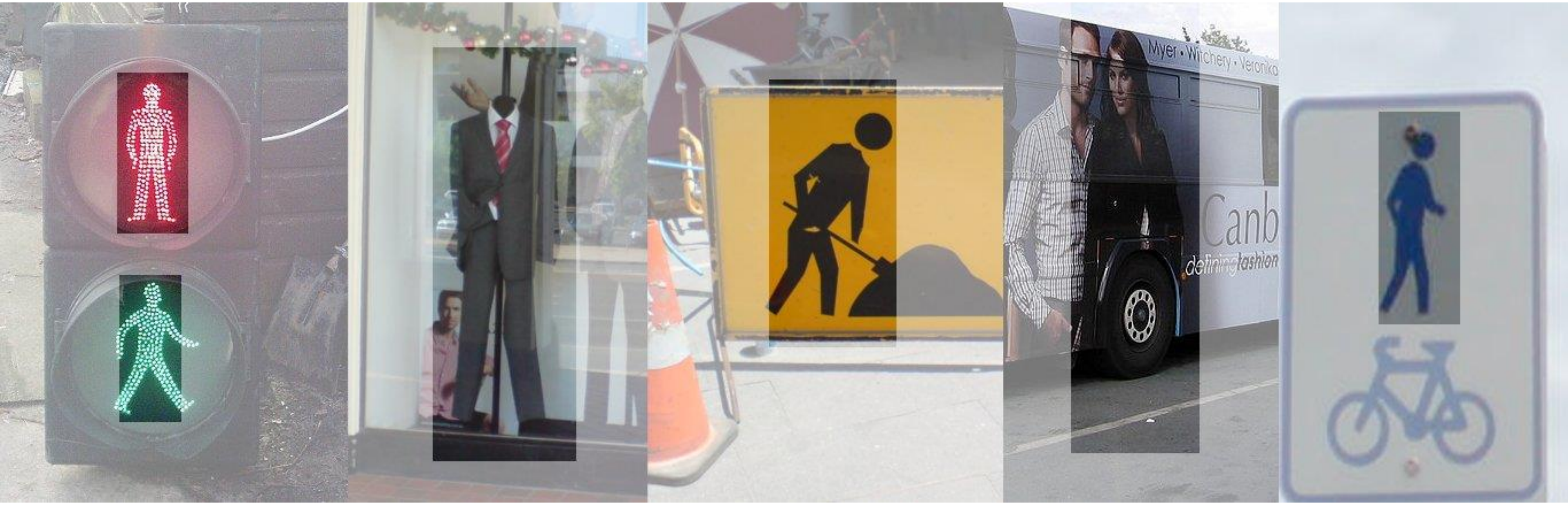
Classification

- Multi-class classification
  - Many possible categories
    - E.g. Sounds “AH, IY, UW, EY..”
    - E.g. Images “Tree, dog, house, person..”

- Classes may overlap
  - Phoneme Classification vs Emotive State Classification vs Accents etc.

- Binary classification
  - Only two categories
    - Man vs. Woman
    - Face vs. **not a face…**

Negative Classes (Not an X)



Which of these IS NOT a Person/Pedestrian?

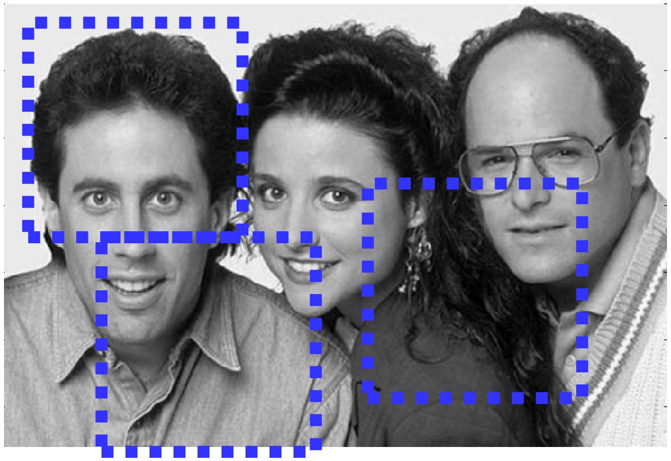
Detection vs Classification

- Detection: Find an X
- Classification: Find the correct label X,Y,Z etc.

Detection vs Classification

- Detection: Find an X
- Classification: Find the correct label X,Y,Z etc.
- Binary Classification as Detection: Find the correct label X or notX

Face Detection as Classification



For each square, run a classifier to find out if it is a face or not

- Faces can be many sizes
- They can happen anywhere in the image
- For each face size
  - For each location
    - Classify a rectangular region of the face size, at that location, as a face or not a face
- This is a series of ***binary*** classification problems

Binary classification

- Classification can be abstracted as follows
- H: X → (+1,-1)
- A function H that takes as input some X and outputs a +1 or -1
  - X is the set of "features"
  - +1/-1 represent the two classes
- Many mechanisms (may types of "H")
  - Any many ways of characterizing "X"

- We' ll look at a specific method based on voting with simple rules
  - A "META" method

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

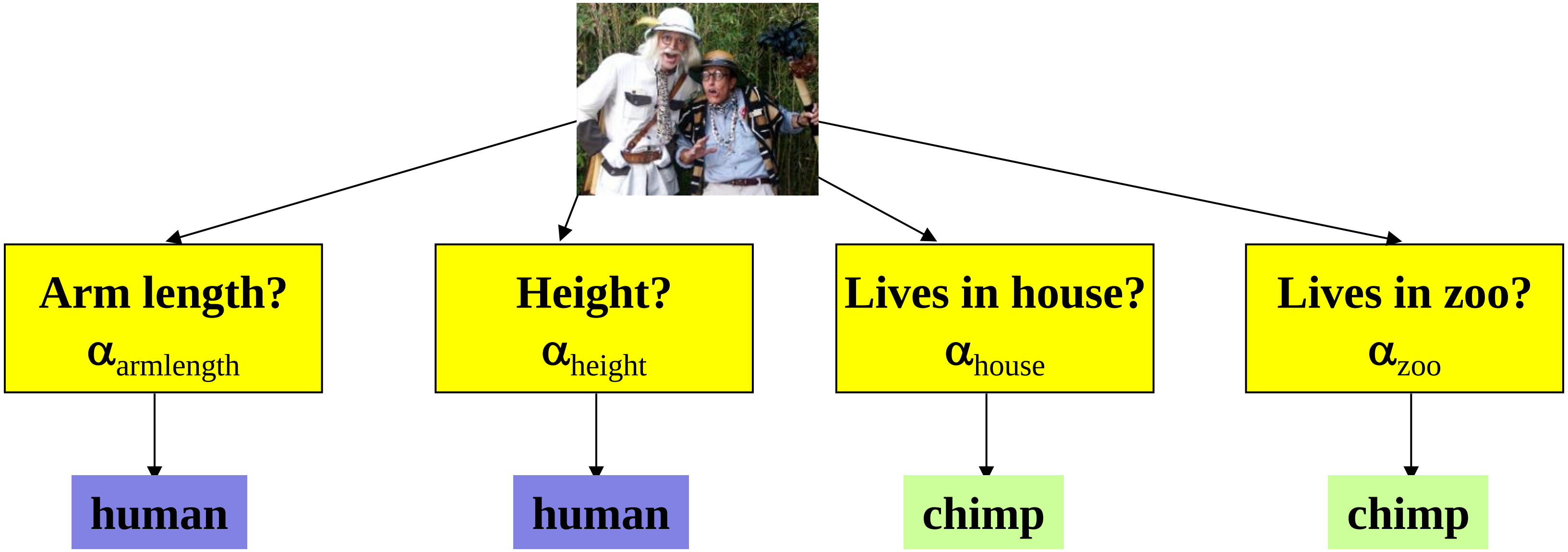
Boosting: A very simple idea

- 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

Boosting and the Chimpanzee Problem



Arm length?

$\alpha_{\text{armlength}}$

Height?

α_{height}

Lives in house?

α_{house}

Lives in zoo?

α_{zoo}

human

human

chimp

chimp

- 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 as defined by Freund

- A gambler wants to write a program to predict winning horses. His program must encode the expertise of his brilliant winner friend
- The friend has no single, encodable algorithm. Instead he has many rules of thumb
  - He uses a different rule of thumb for each set of races
    - E.g. “in this set, go with races that have black horses with stars on their foreheads”
  - But cannot really enumerate what rules of thumbs go with what sets of races: he simply “knows” when he encounters a set
    - A common problem that faces us in many situations
- Problem:
  - How best to combine all of the friend’s rules of thumb
  - What is the best set of races to present to the friend, to extract the various rules of thumb

Boosting

- The basic idea: Can a “weak” learning algorithm that performs just slightly better than a random guess be *boosted* into an arbitrarily accurate “strong” learner
  - Each of the gambler’s rules may be just better than random guessing

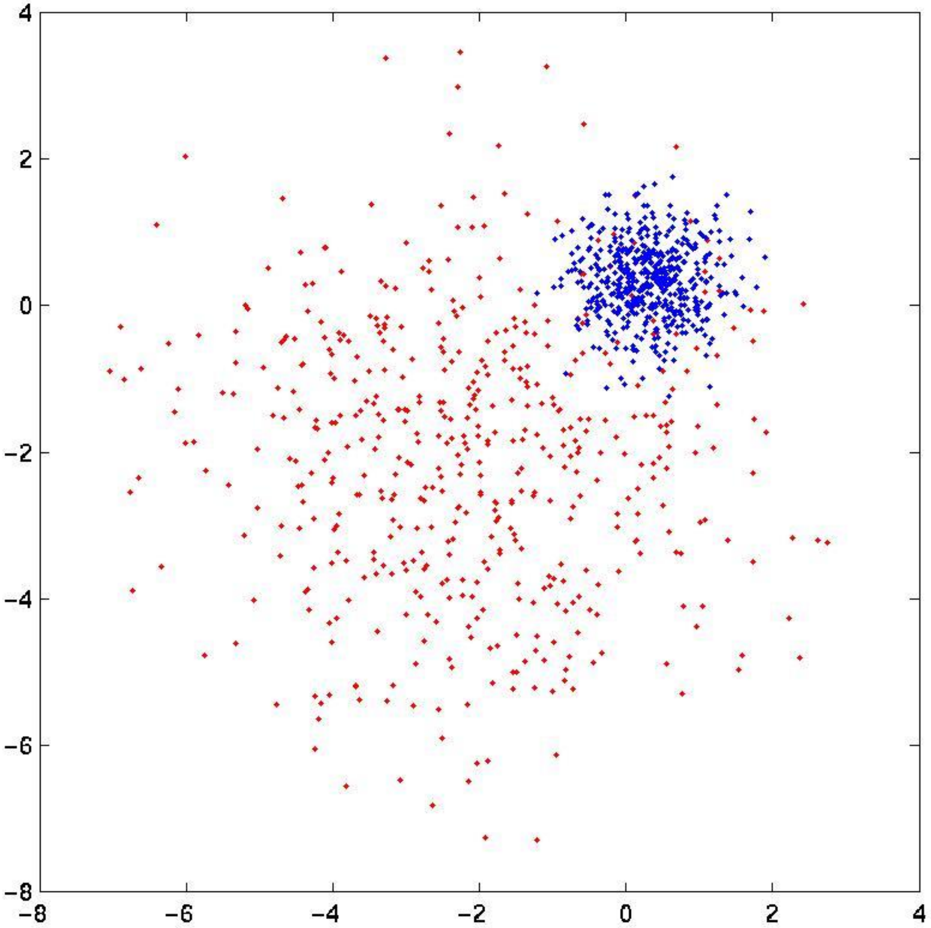
- This is a “meta” algorithm, that poses no constraints on the form of the weak learners themselves
  - The gambler’s rules of thumb can be anything

Boosting: A Voting Perspective

- Boosting can be considered a form of voting
  - Let a number of different classifiers classify the data
  - Go with the majority
  - Intuition says that as the number of classifiers increases, the dependability of the majority vote increases

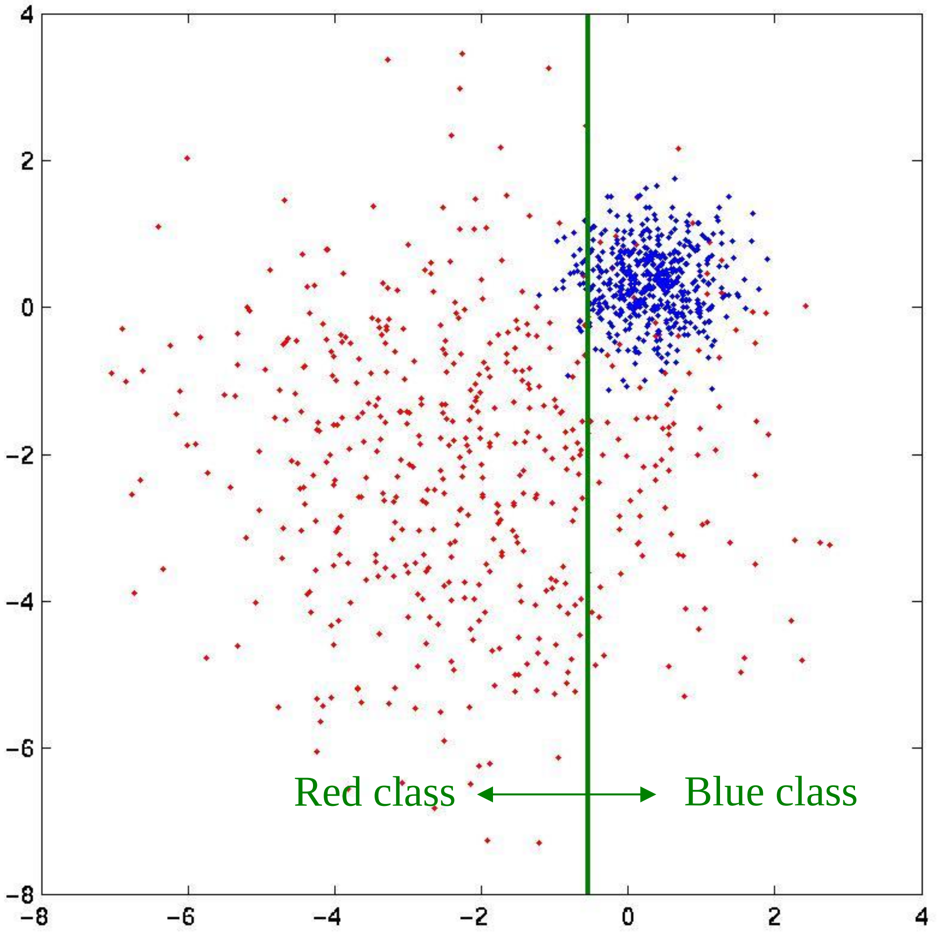
- The corresponding algorithms were called Boosting by majority
  - A (weighted) majority vote taken over all the classifiers
  - How do we compute weights for the classifiers?
  - How do we actually train the classifiers

Boosting: An Example



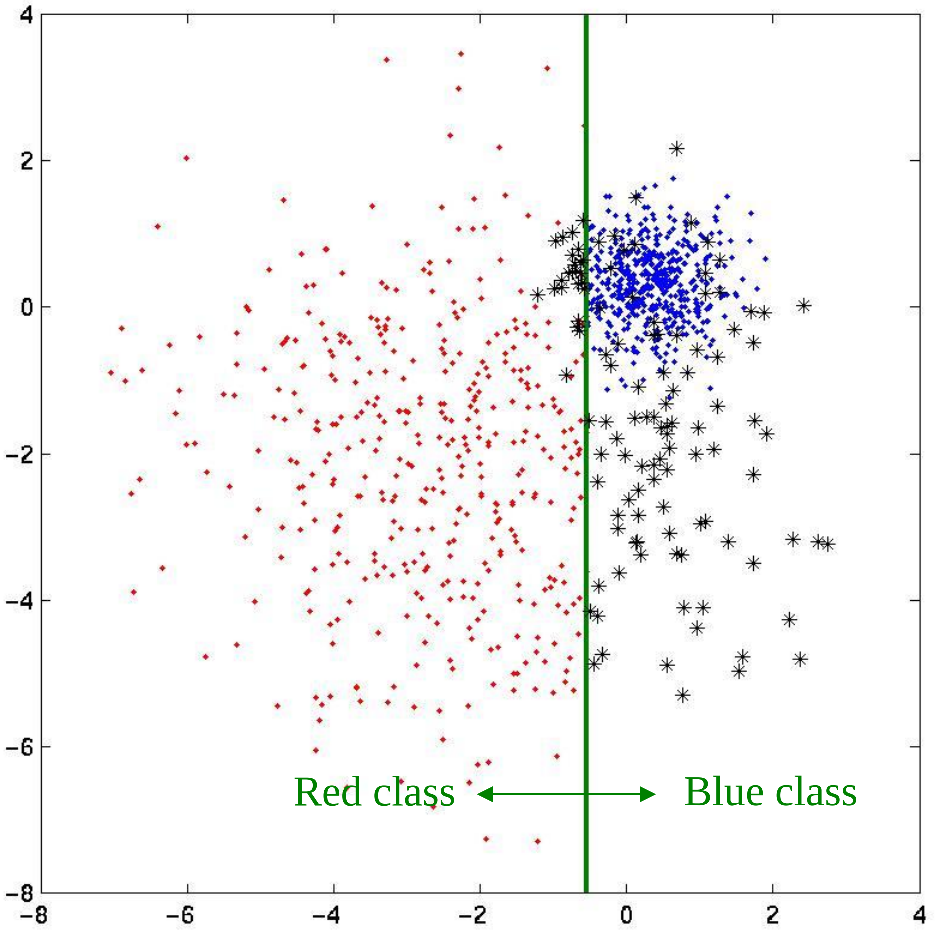
- Red dots represent training data from Red class
- Blue dots represent training data from Blue class

Boosting: An Example



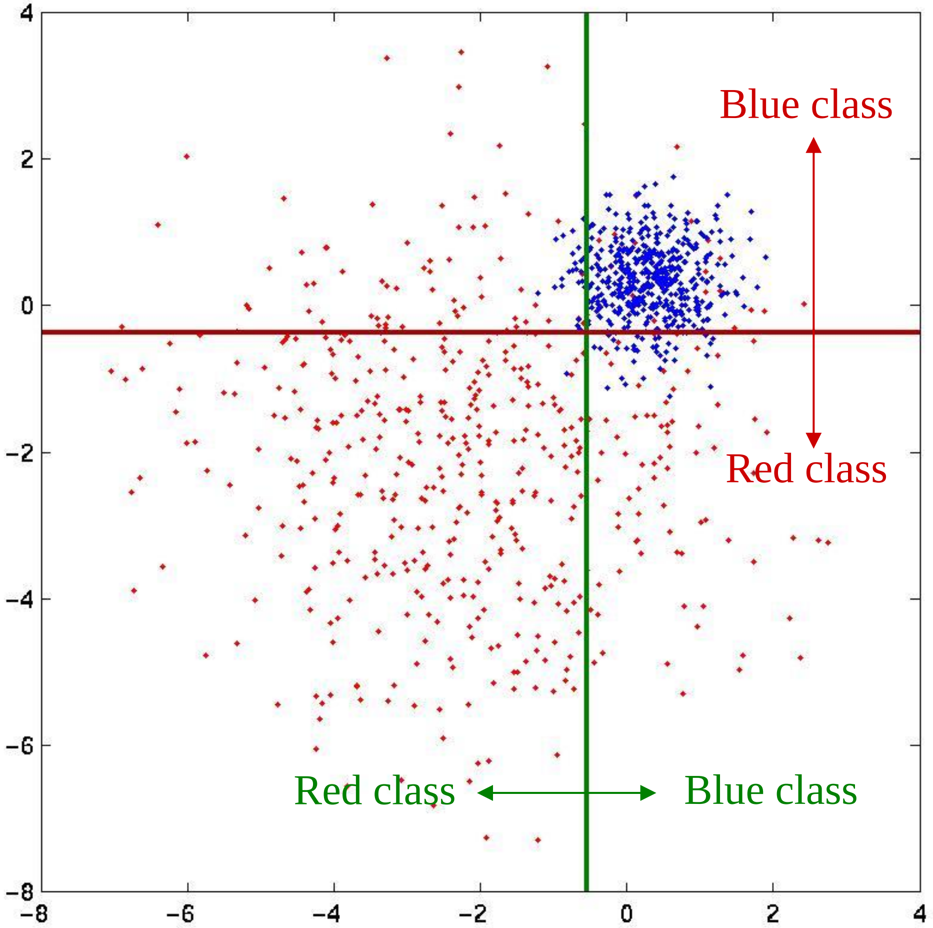
- Very simple weak learner
  - A line that is parallel to one of the two axes

Boosting: An Example



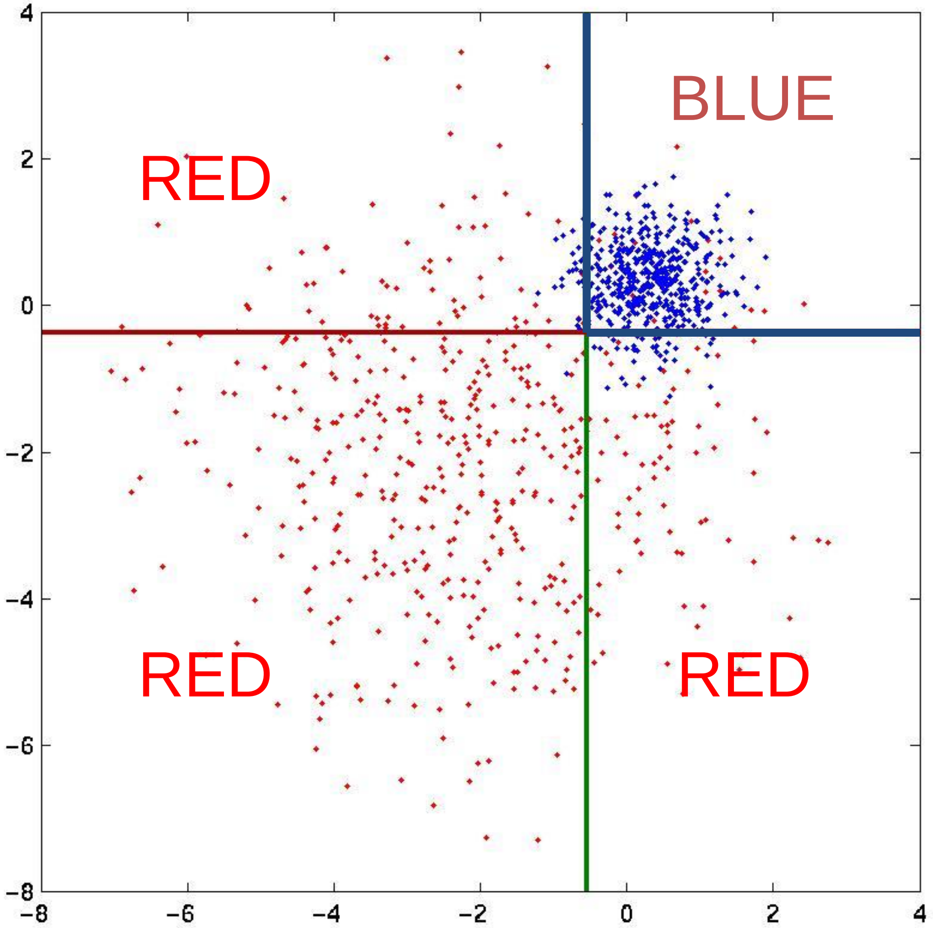
- First weak learner makes many mistakes
  - Errors coloured black

Boosting: An Example



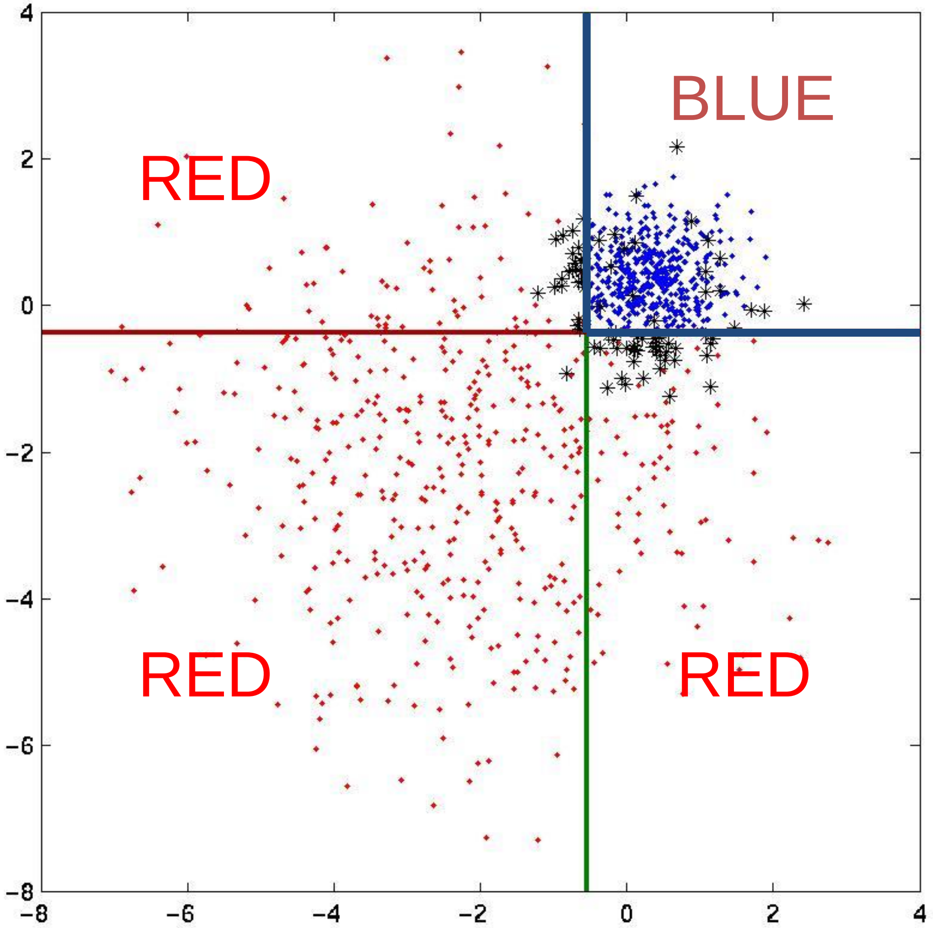
- Second weak learner focuses on errors made by first learner

Boosting: An Example



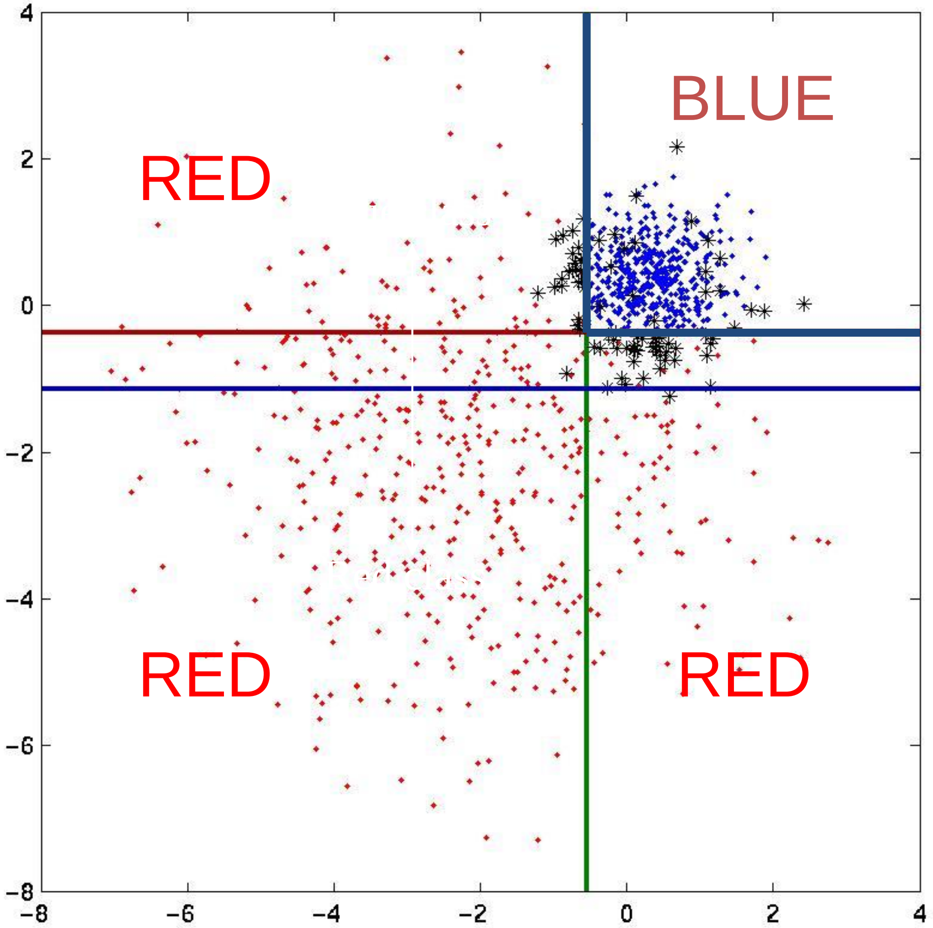
- Second strong learner: weighted combination of first and second weak learners
  - Decision boundary shown by black lines

Boosting: An Example



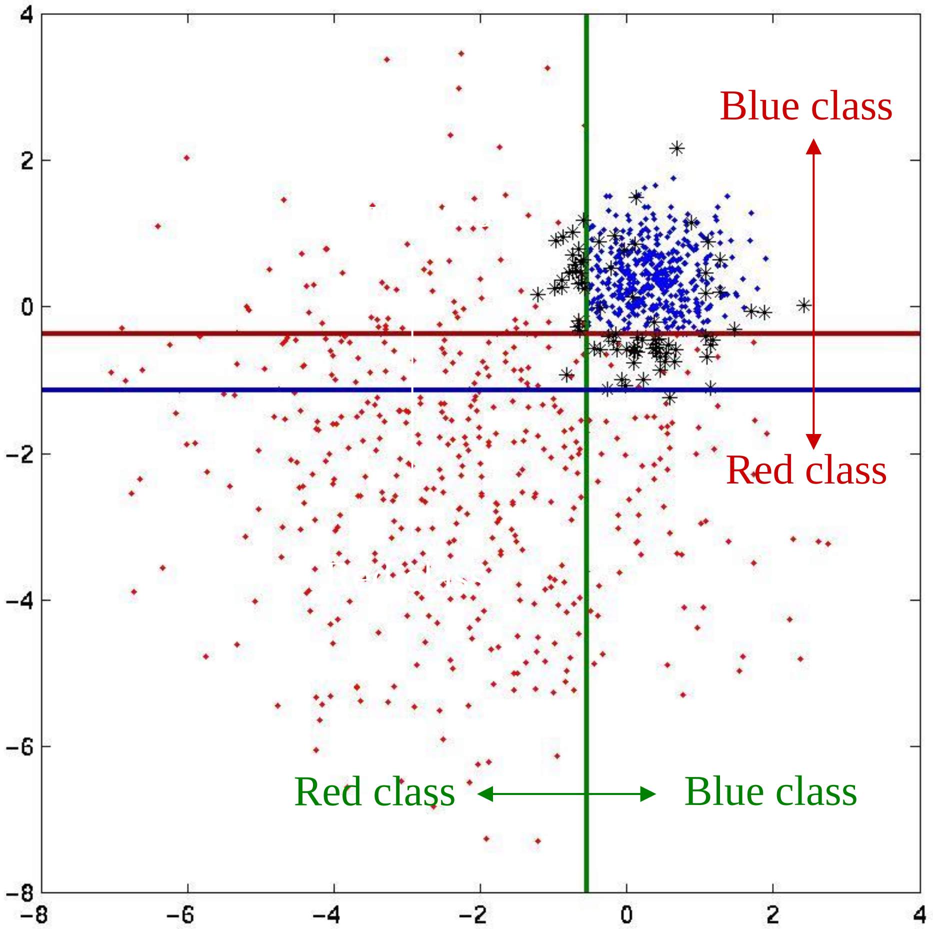
- The second strong learner also makes mistakes
  - Errors colored black

Boosting: An Example



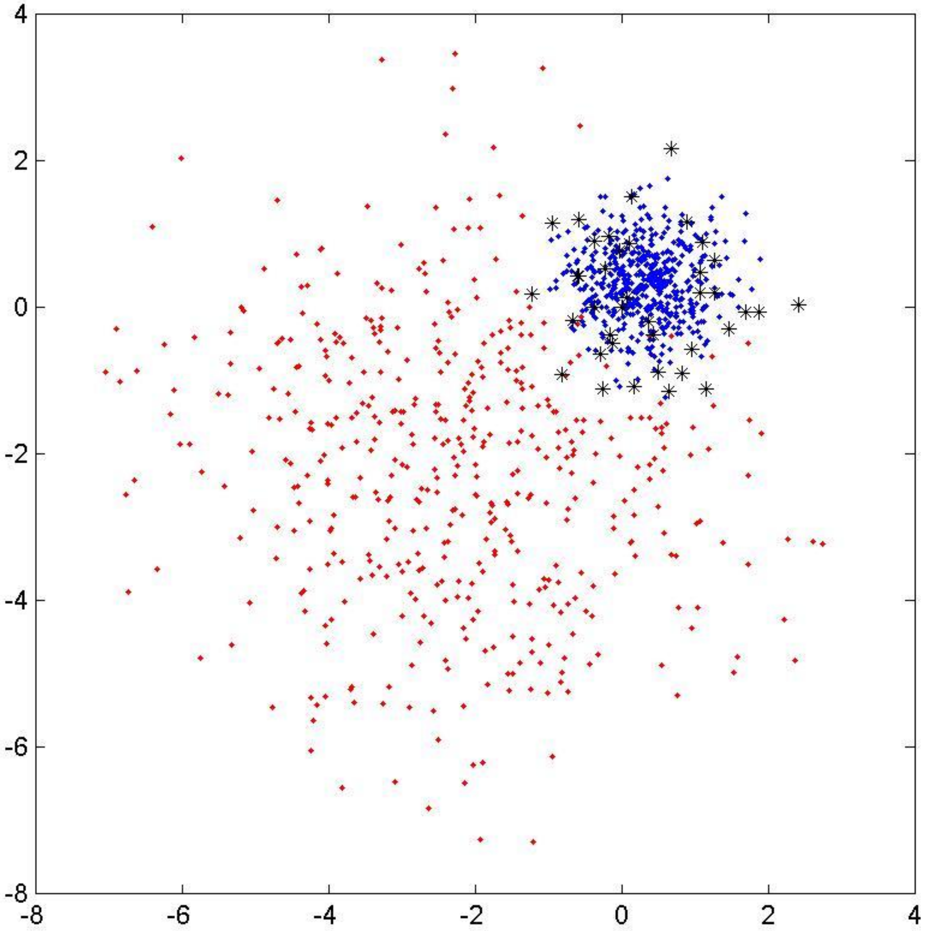
- Third weak learner concentrates on errors made by second strong learner

Boosting: An Example



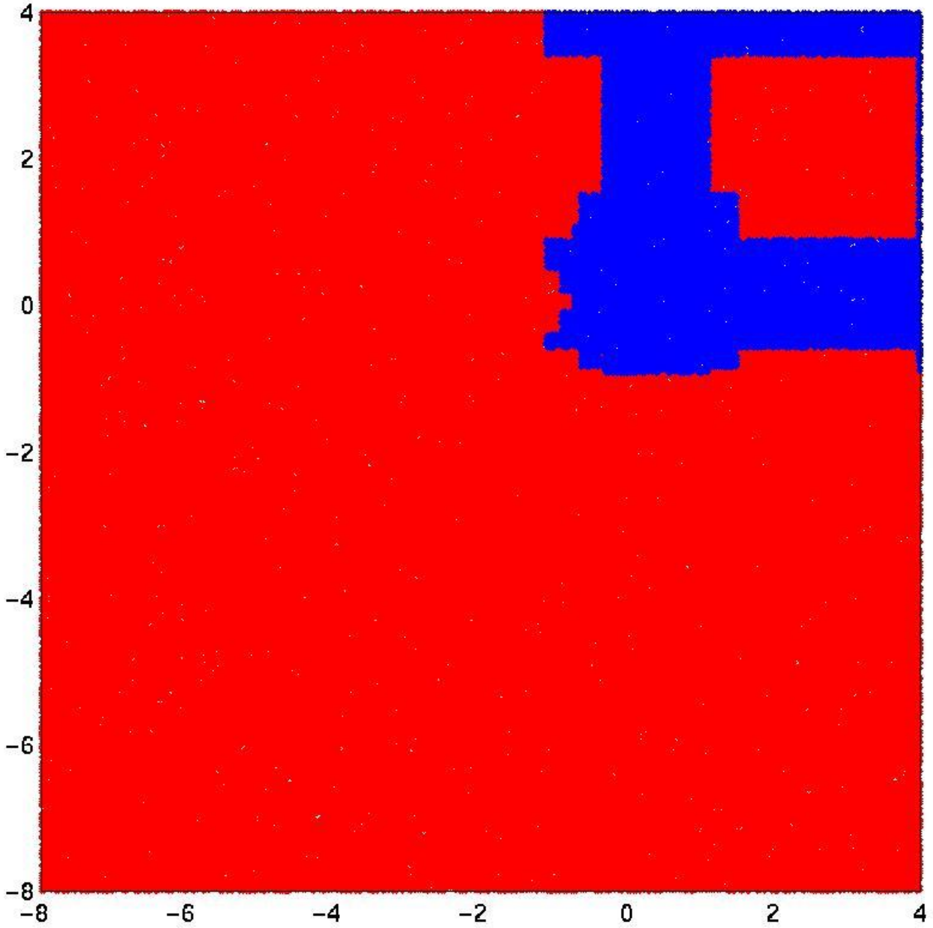
- Third weak learner concentrates on errors made by combination of previous weak learners
- Continue adding weak learners until….

Boosting: An Example



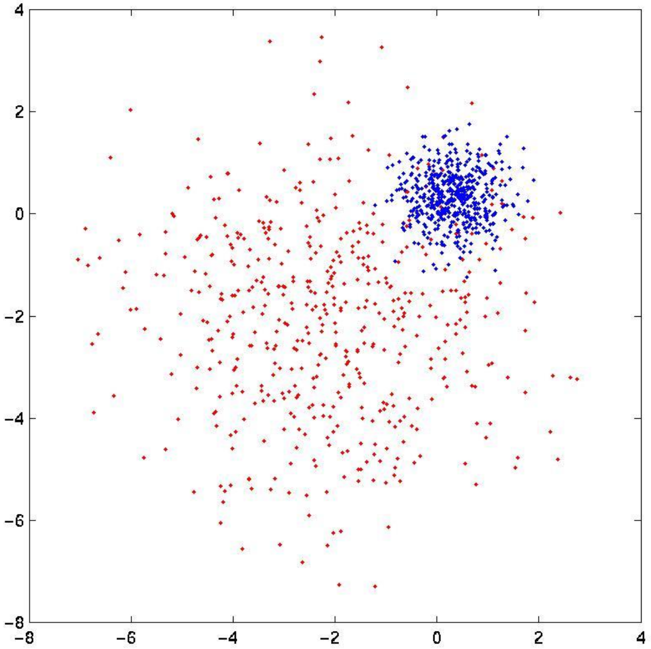
- Voila! Final strong learner: very few errors on the training data

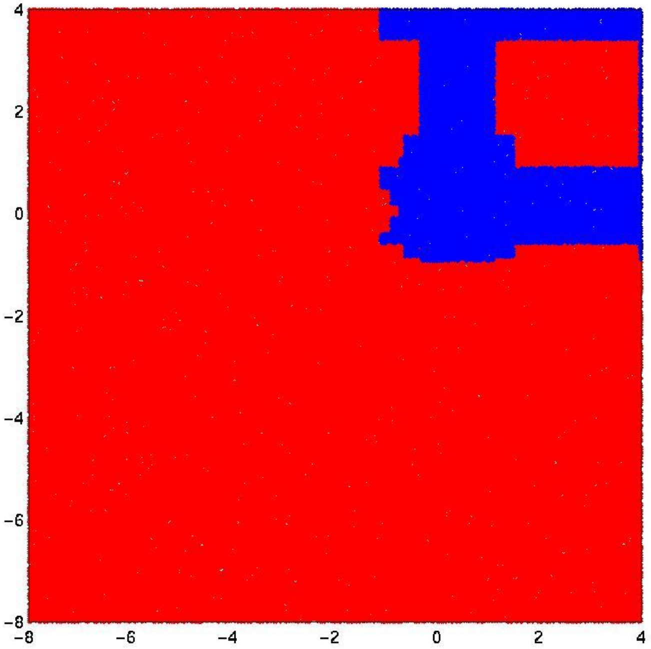
Boosting: An Example



- The final strong learner has learnt a complicated decision boundary

Boosting: An Example



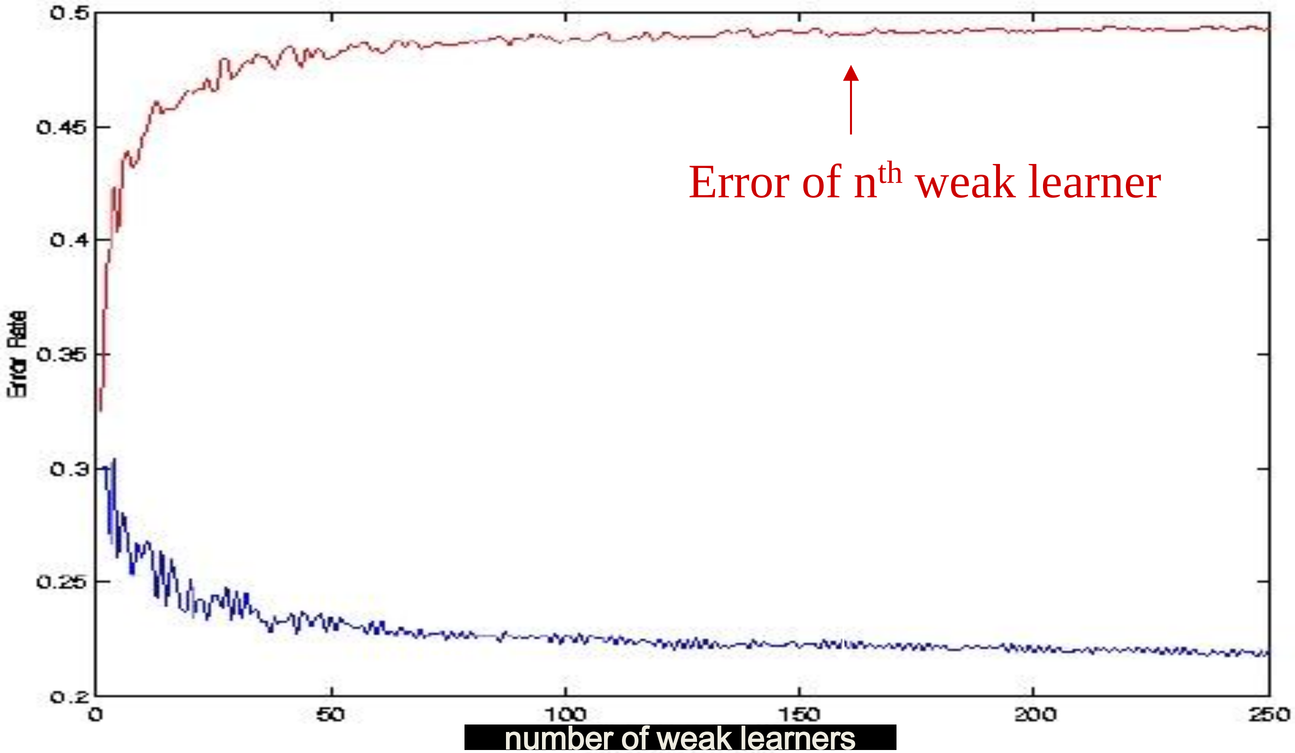


- The final strong learner has learnt a complicated decision boundary

- Decision boundaries in areas with low density of training points assumed inconsequential

Overall Learning Pattern

- Strong learner increasingly accurate with increasing number of weak learners
- Residual errors increasingly difficult to correct
  - Additional weak learners less and less effective



Error of n^{th} weak learner

AdaBoost

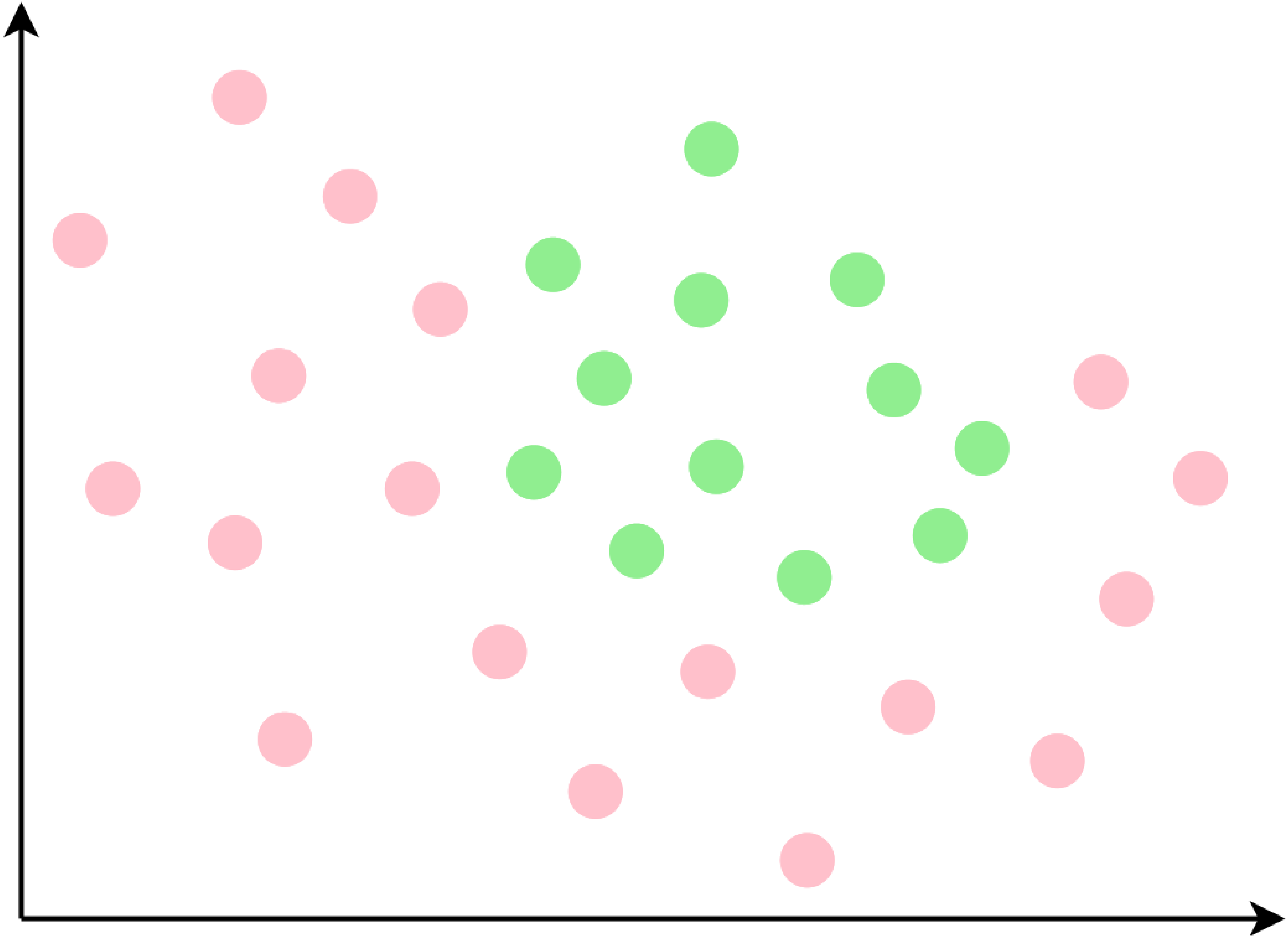
- No relation to Ada Lovelace
- Adaptive Boosting
- Adaptively Selects Weak Learners
- ~10K citations Freund and Schapire

h_2

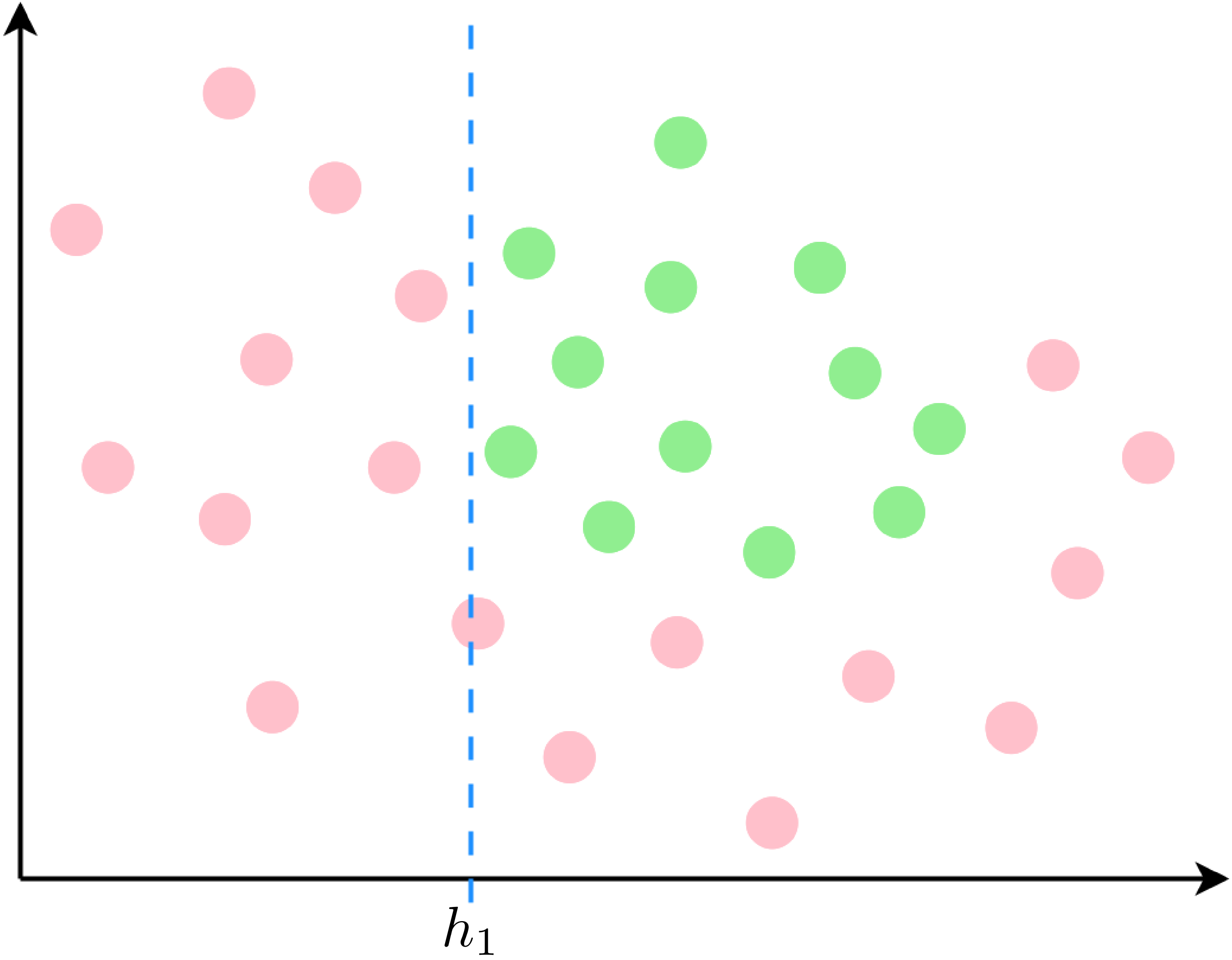
The ADABoost Algorithm

- 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
  - $H(x) = \text{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

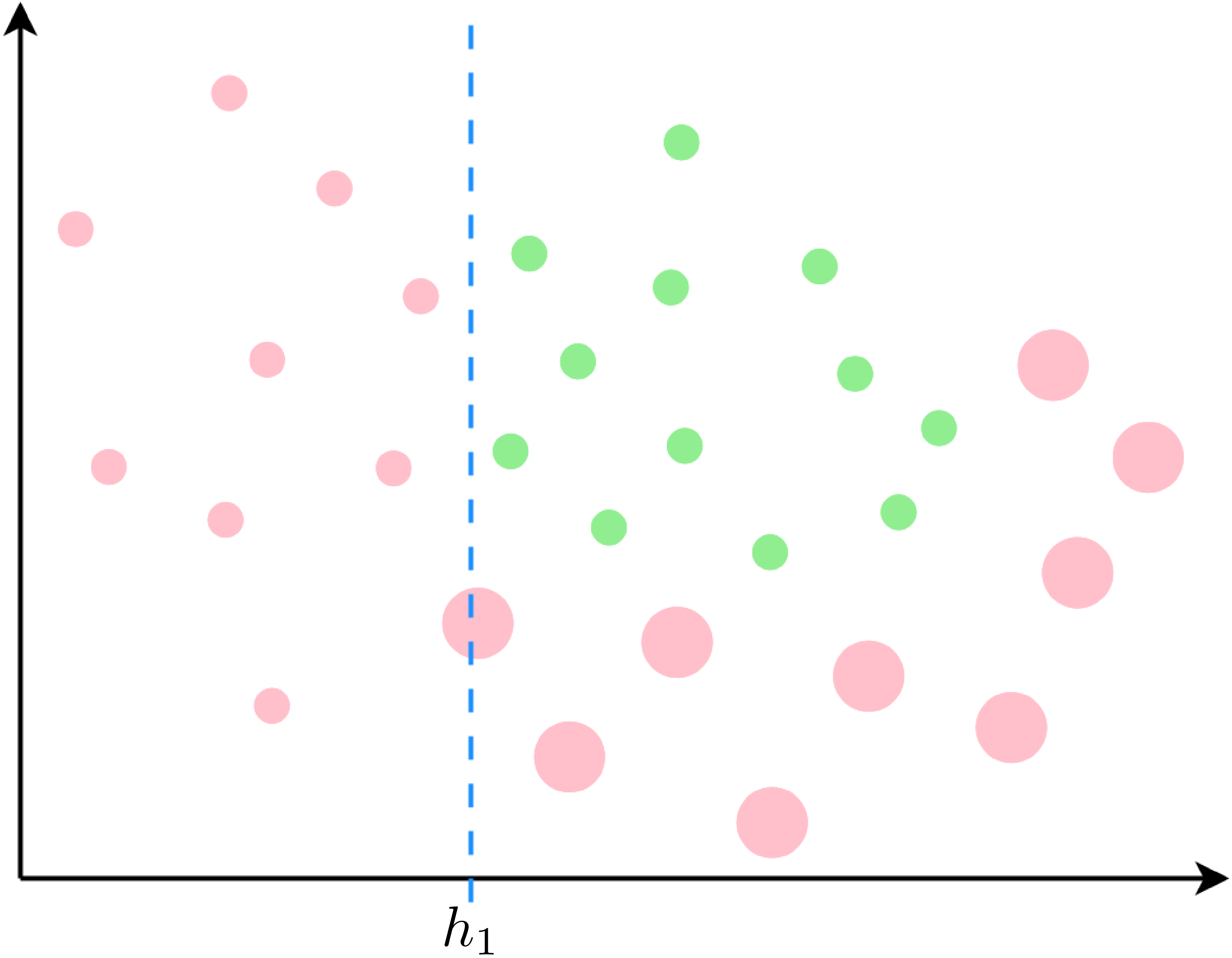
AdaBoost



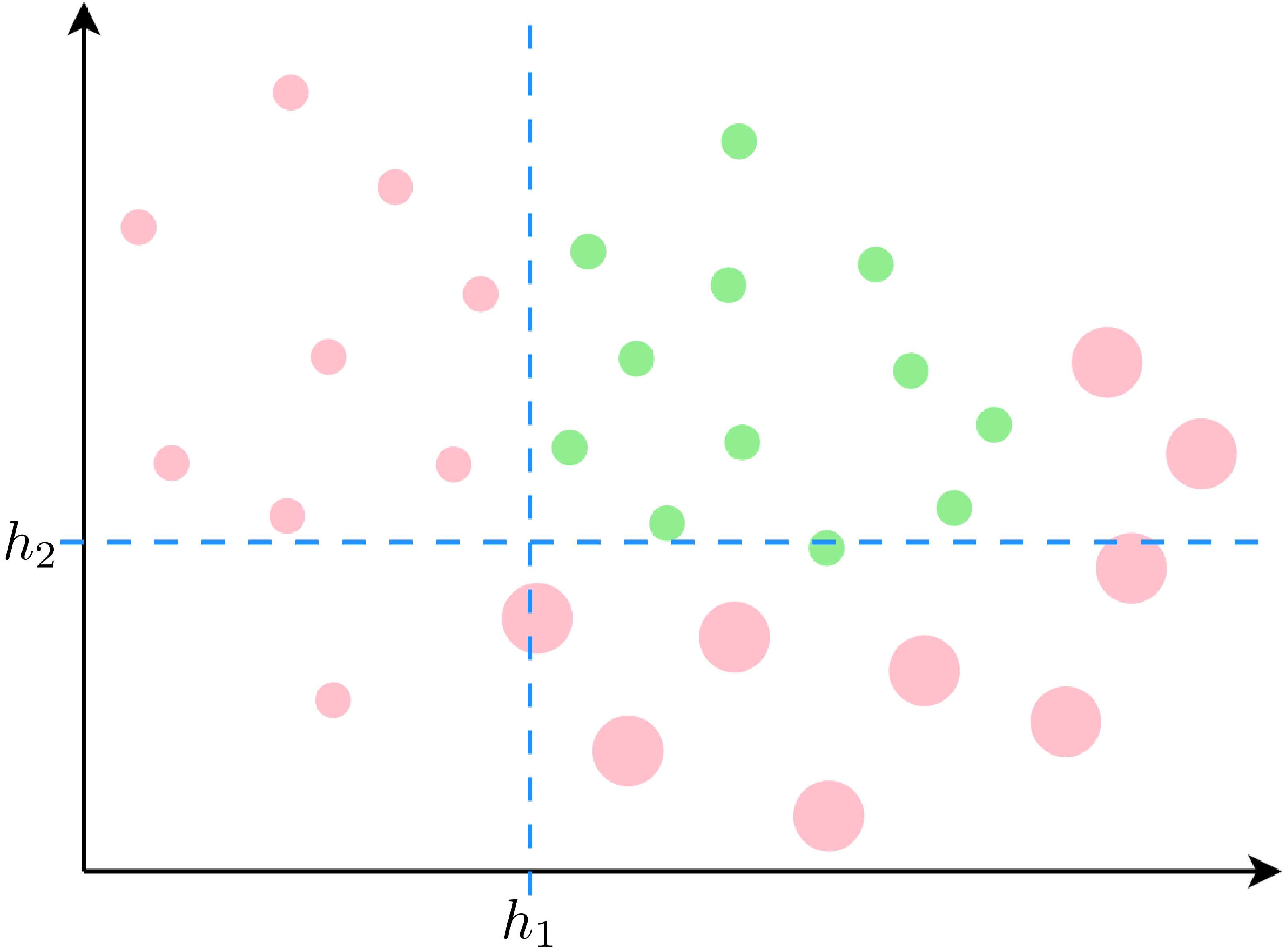
AdaBoost



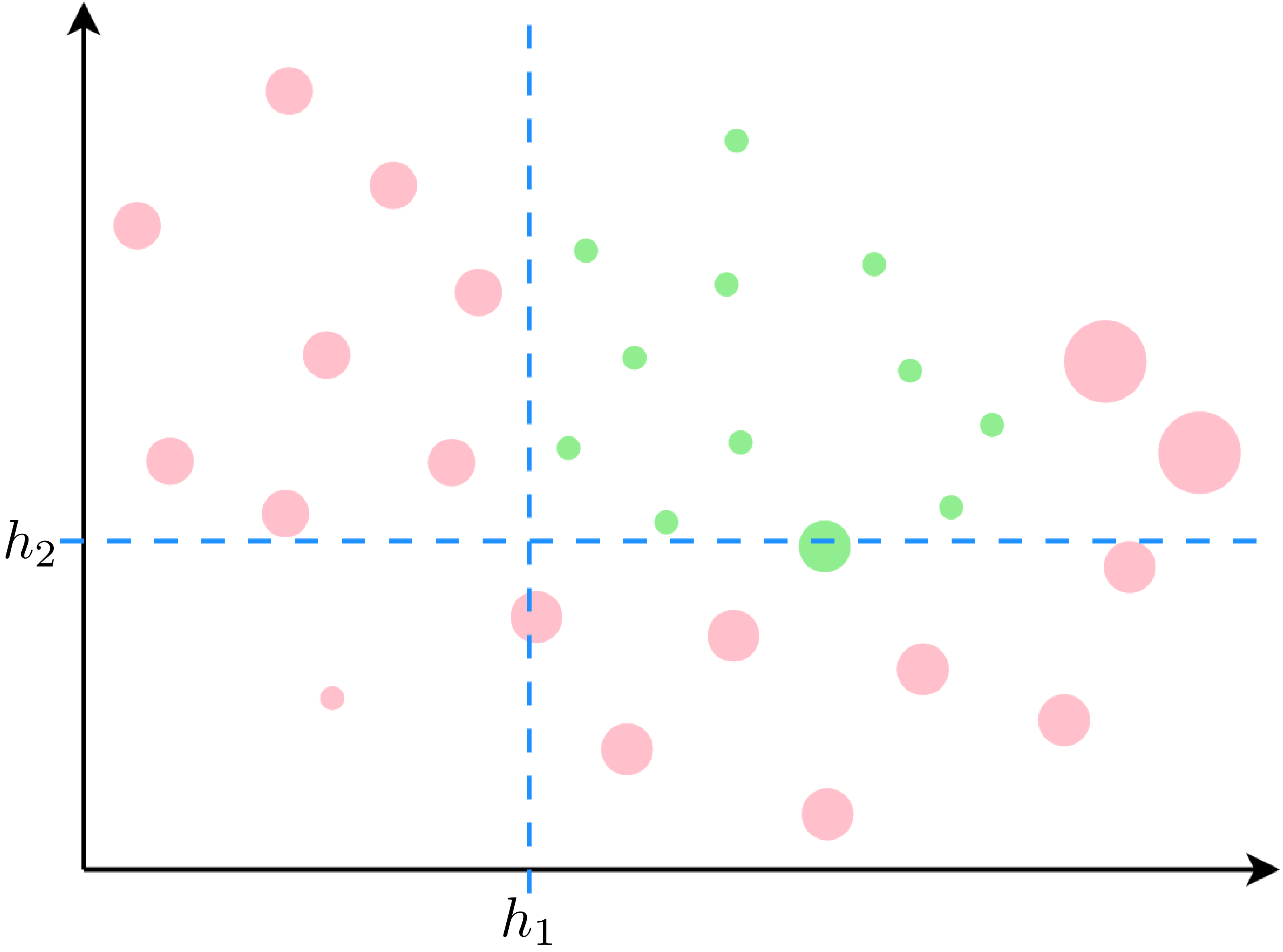
AdaBoost



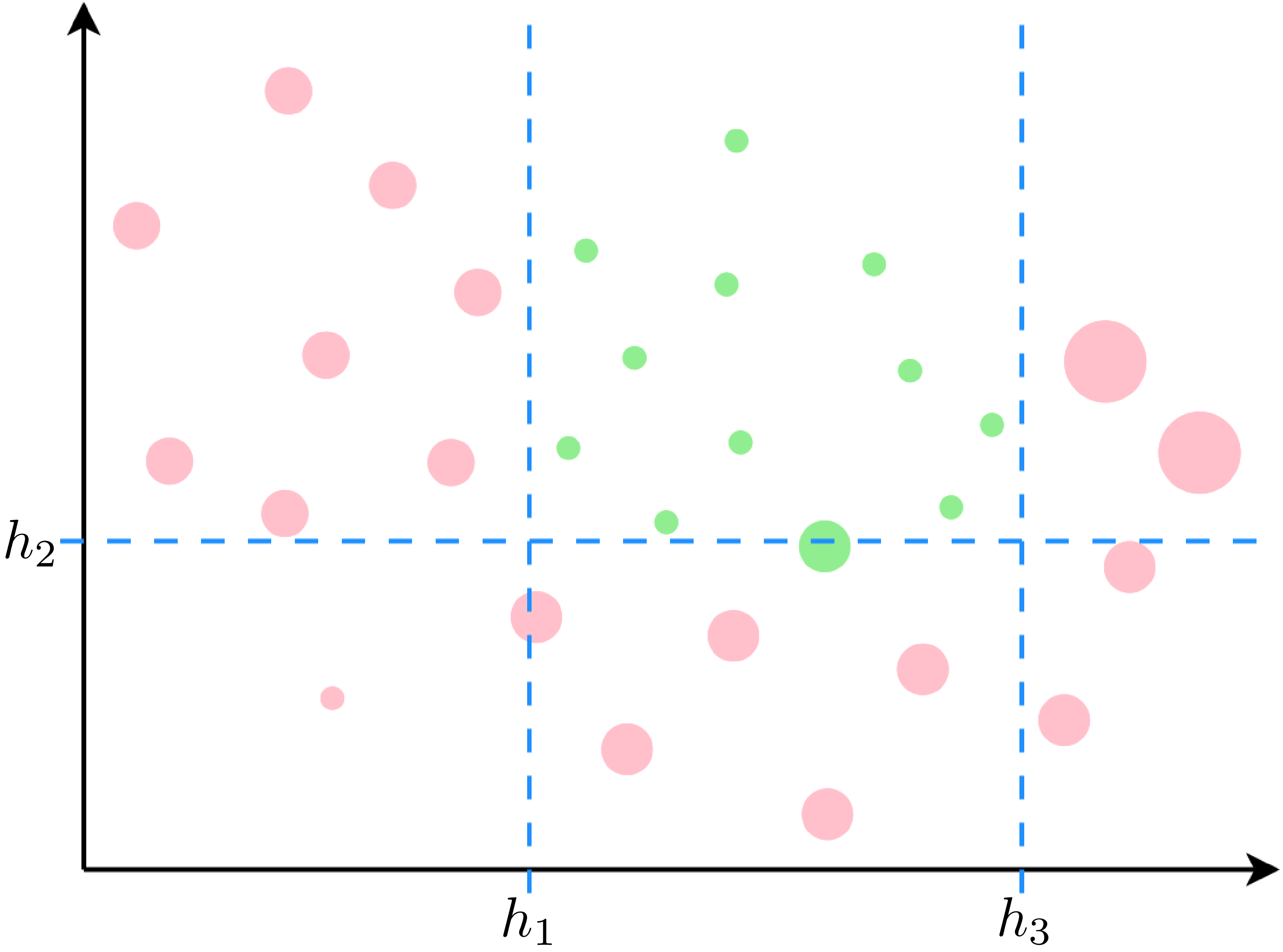
AdaBoost



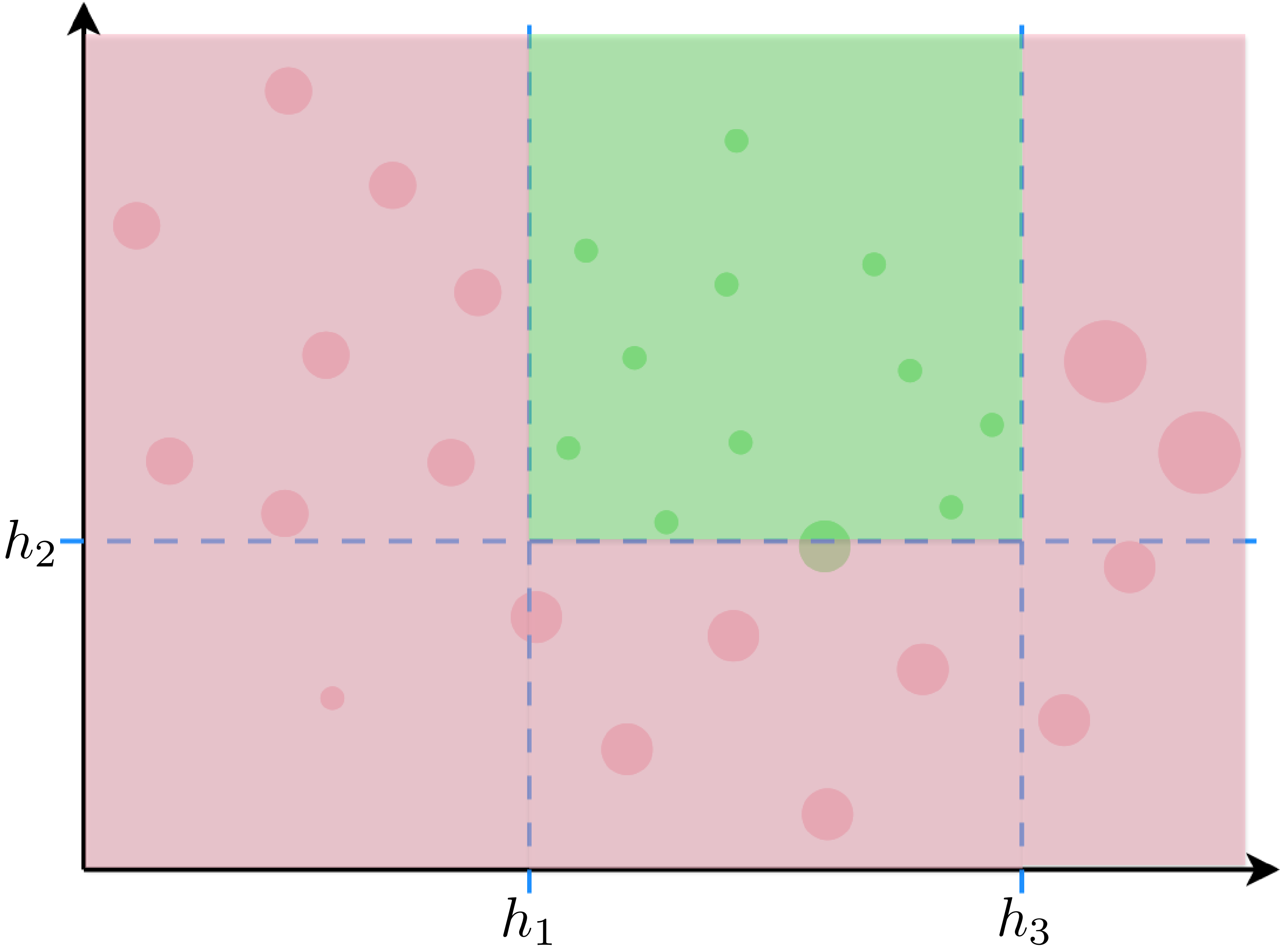
AdaBoost



AdaBoost



AdaBoost



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

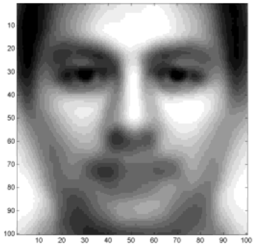
= -0.8 E1 - 0.1 E2

= 0.4 E1 - 0.9 E2

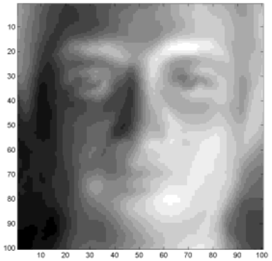
= 0.2 E1 + 0.5 E2

Image = a*E1 + b*E2 → a = Image.E1

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 are explained in terms of the Eigen faces

Training Data

A = 0.3 E1 - 0.6 E2

B = 0.5 E1 - 0.5 E2

C = 0.7 E1 - 0.1 E2

D = 0.6 E1 - 0.4 E2

D = 0.2 E1 + 0.4 E2

E = -0.8 E1 - 0.1 E2

F = 0.4 E1 - 0.9 E2

G = 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

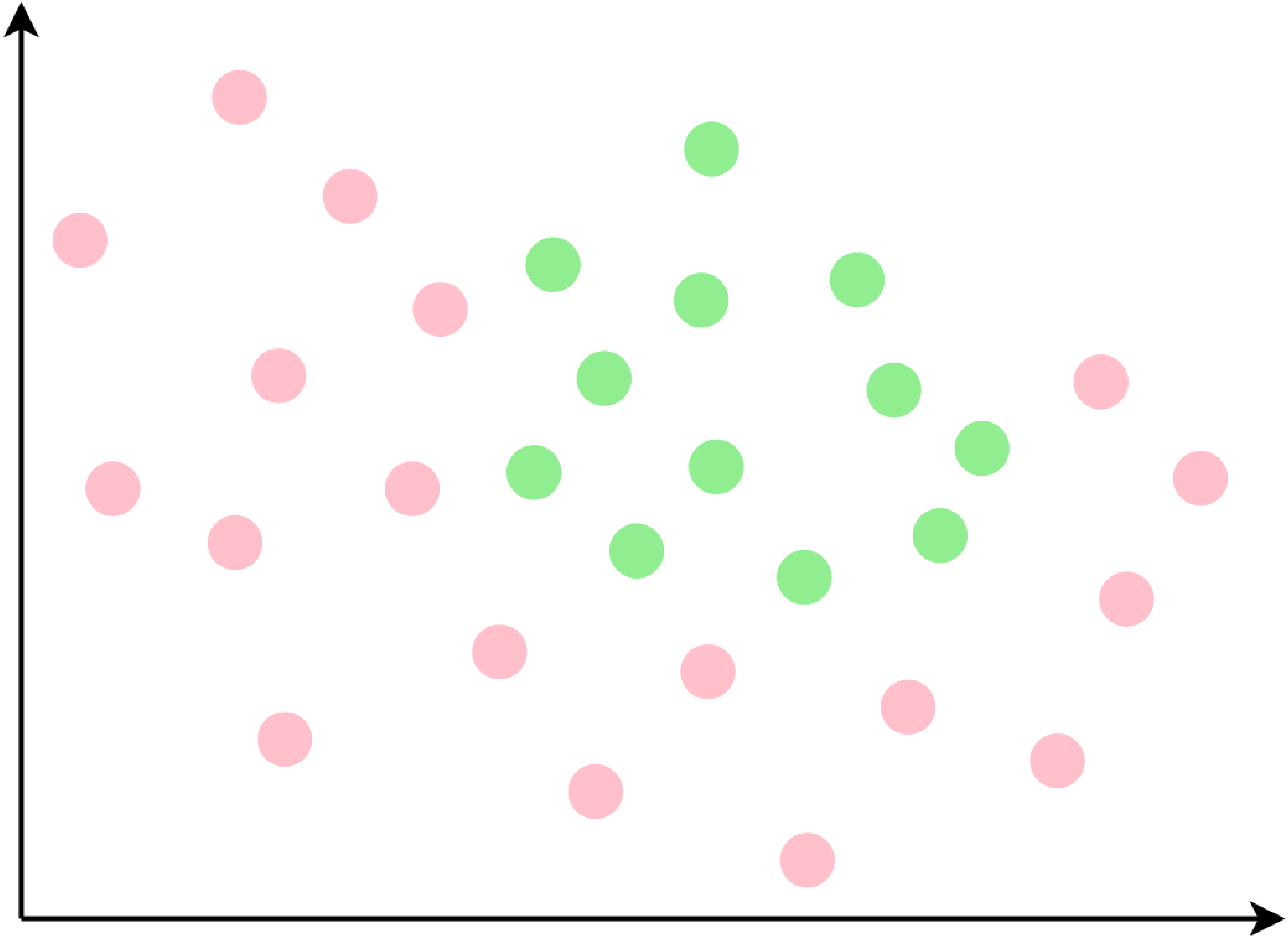
Face = +1

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
    - $\varepsilon_t = \text{Sum}\,\{D_t\,(x_i)\,½(1 - y_i\,h_t(x_i))\}$
  - – Set $\alpha_t = ½ \ln\left((1 - \varepsilon_t) / \varepsilon_t\right)$
  - – 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
  - – $H(x) = \text{sign}(\Sigma_t\,\alpha_t\,h_t(x))$

Initialize $D_1(x_i) = 1/N$



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

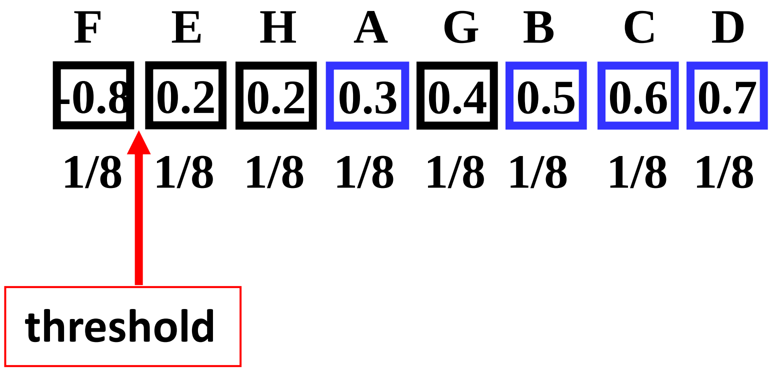
= 0.2 E1 + 0.5 E2

ID	E1	E2.	Class	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"



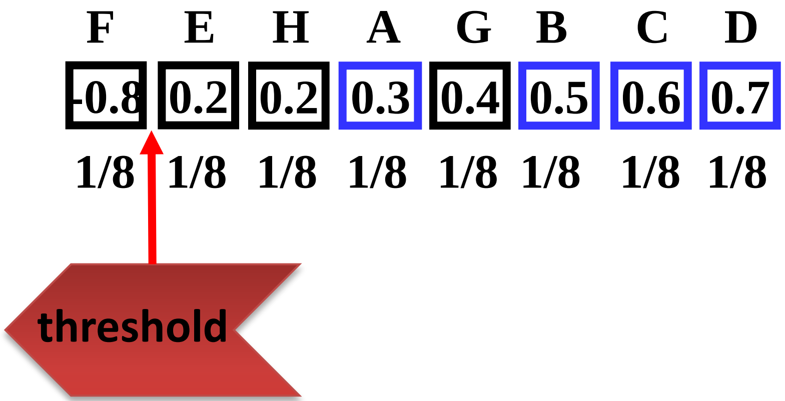
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

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”



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

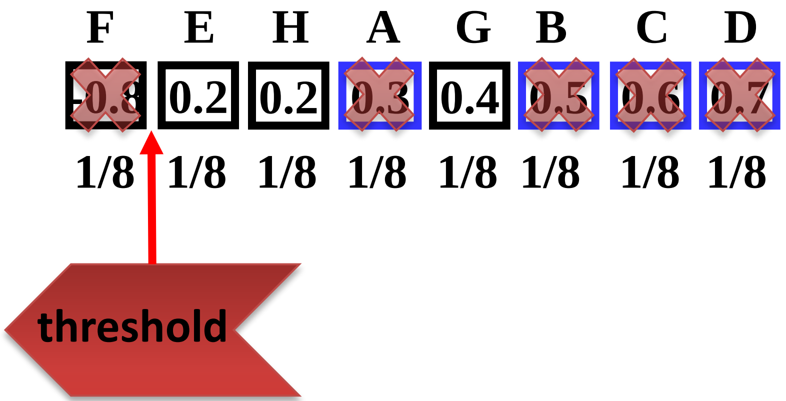
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

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”



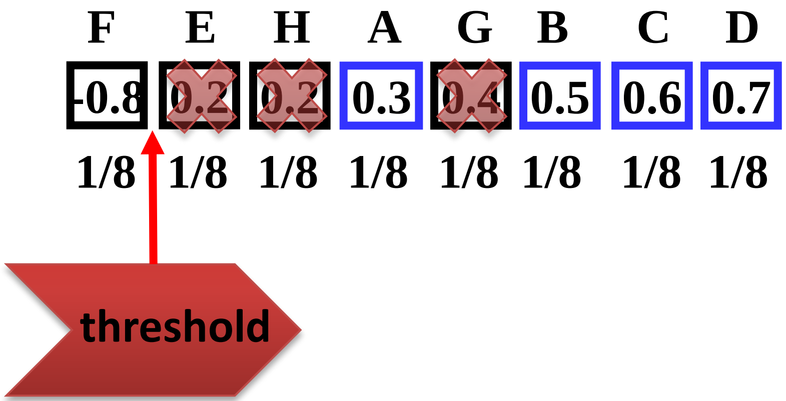
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

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”



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

threshold

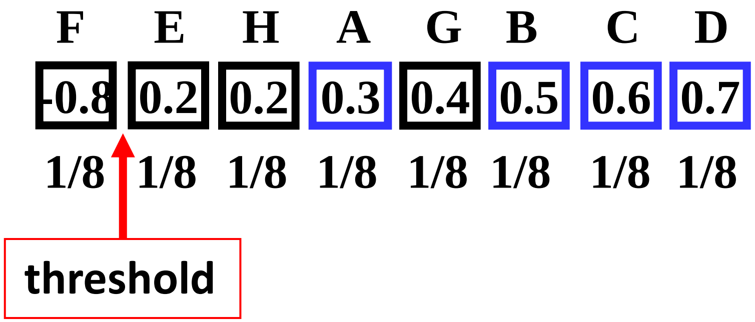
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

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"



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

threshold

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

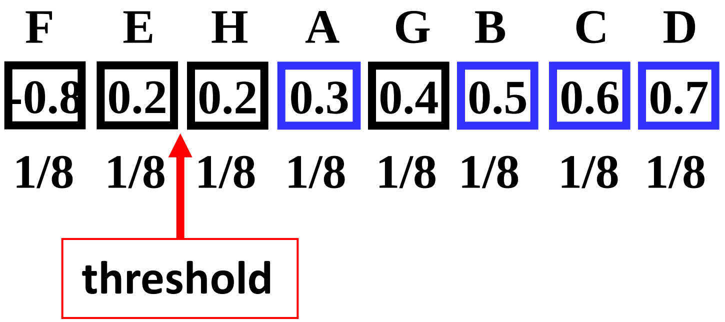
sign = +1 or -1

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"



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

threshold

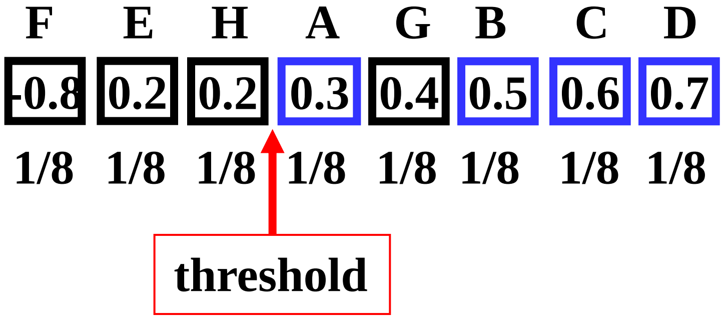
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

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”



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

threshold

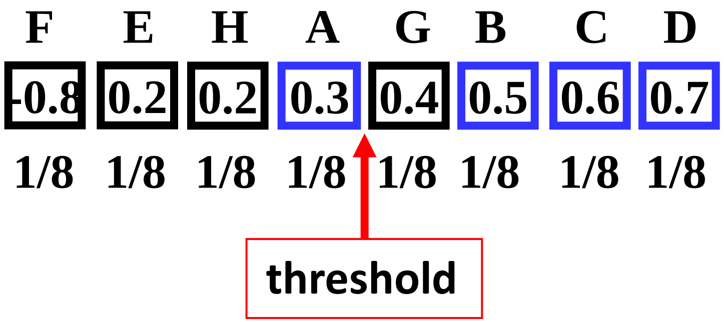
```
Classifier based on E1:
if ( sign*wt(E1) > thresh) > 0)
    face = true

sign = +1 or -1
```

Sign = +1, error = 1/8

Sign = -1, error = 7/8

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"



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

threshold

Classifier based on E1:

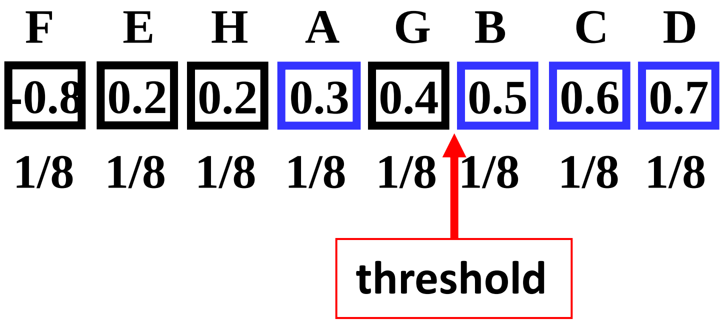
```
if ( sign*wt(E1) > thresh) > 0)
    face = true

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"



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

threshold

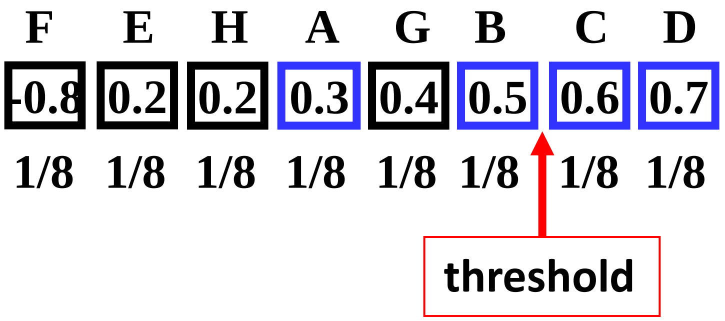
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

sign = +1 or -1

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

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”



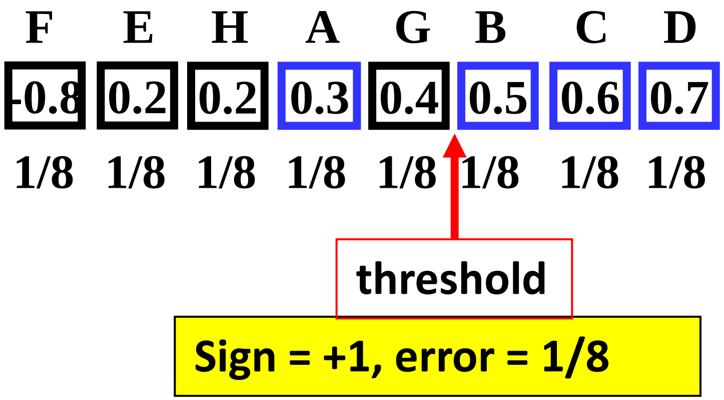
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

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"



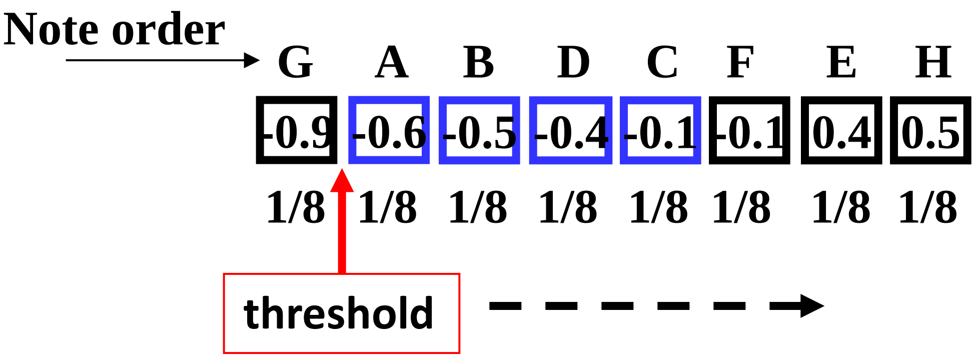
Sign = +1, error = 1/8

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"



Note order

G	A	B	D	C	F	E	H
-0.9	-0.6	-0.5	-0.4	-0.1	-0.1	0.4	0.5
1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

threshold

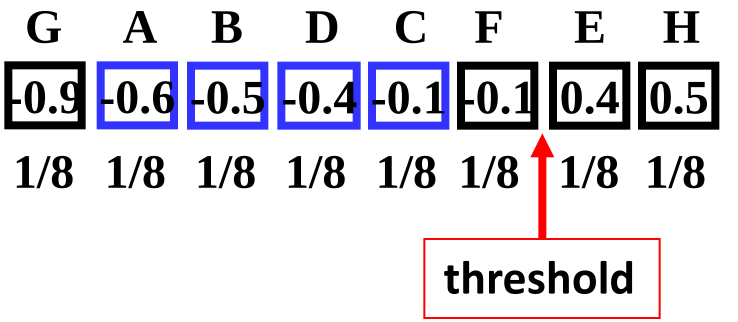
Classifier based on E2:
if (sign*wt(E2) > thresh) > 0)
    face = true

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 Best E2“Stump”



G	A	B	D	C	F	E	H
-0.9	-0.6	-0.5	-0.4	-0.1	-0.1	0.4	0.5
1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

threshold

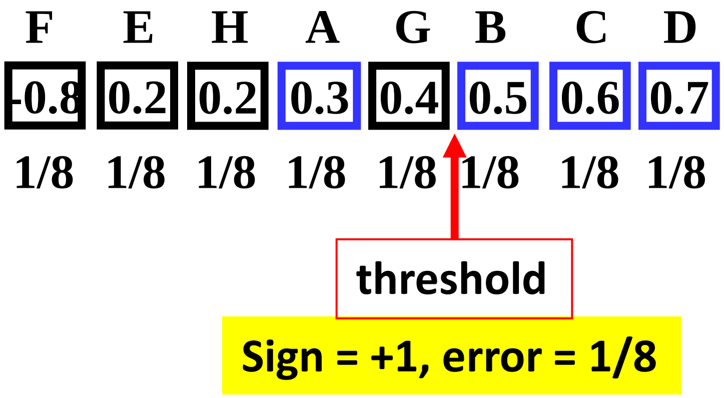
Sign = -1, error = 2/8

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"



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

threshold

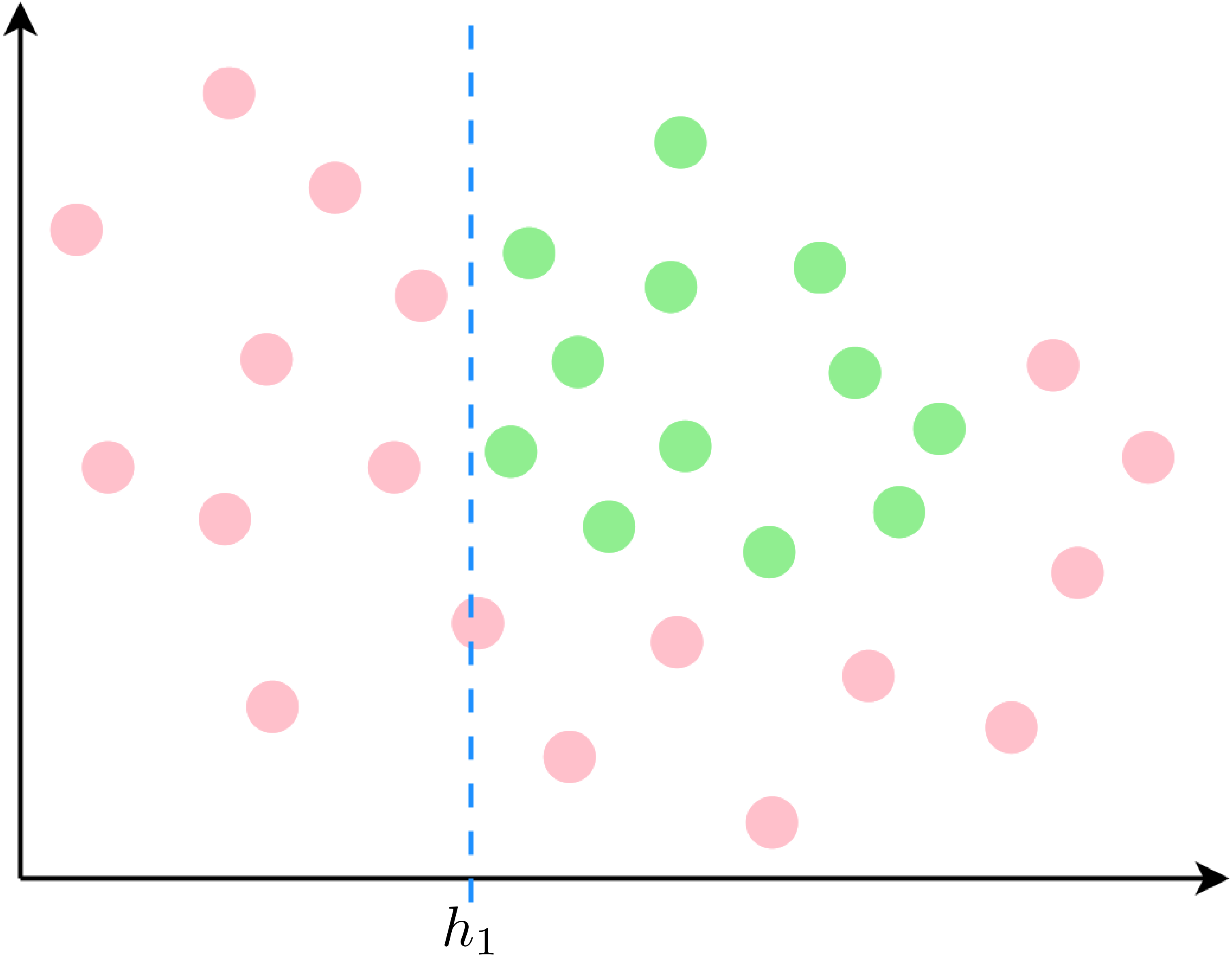
Sign = +1, error = 1/8

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

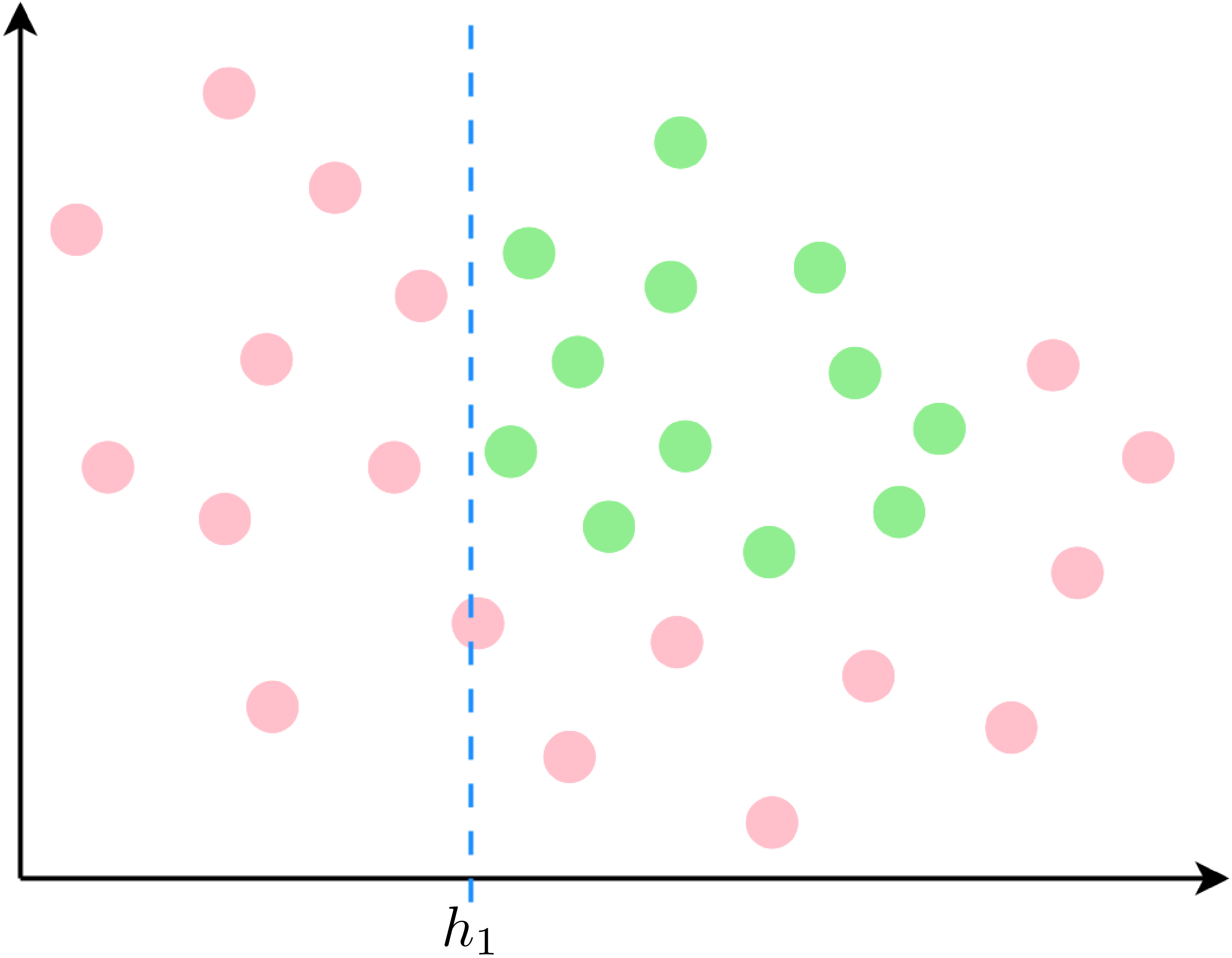
The Best “Stump”



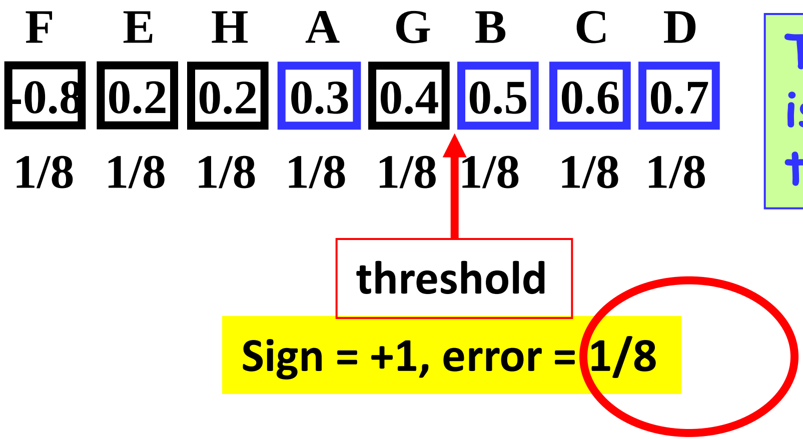
The ADABoost Algorithm

- 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 “Stump”



The Best Error



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

threshold

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

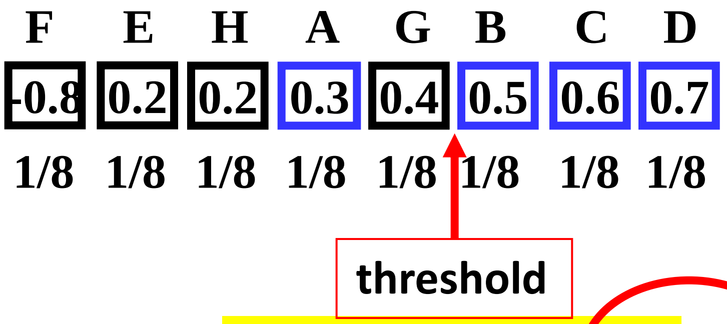
threshold

Sign = +1, error = 1/8

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

= 0.5 ln(7) = 0.97

The Boosted Classifier Thus Far



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

threshold

Sign = +1, error = 1/8

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

= 0.5 ln(7) = 0.97

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

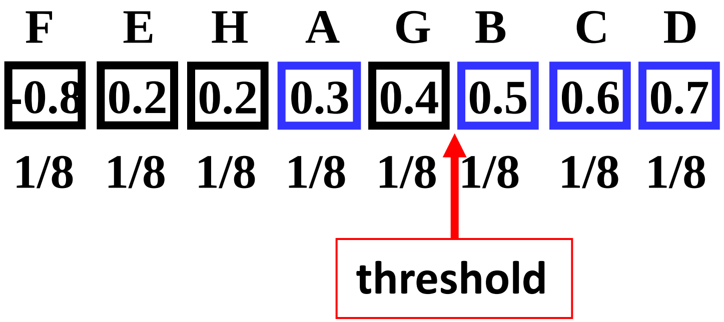
H(X) = sign(0.97 * h1(X))

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
    - $\varepsilon_t = \text{Average}\{\frac{1}{2}(1 - y_i h_t(x_i))\}$
  - Set $\alpha_t = \frac{1}{2} \ln((1 - \varepsilon_t) / \varepsilon_t)$
  - 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
  - $H(x) = \text{sign}(\Sigma_t \alpha_t h_t(x))$

The Best Error



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

threshold

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

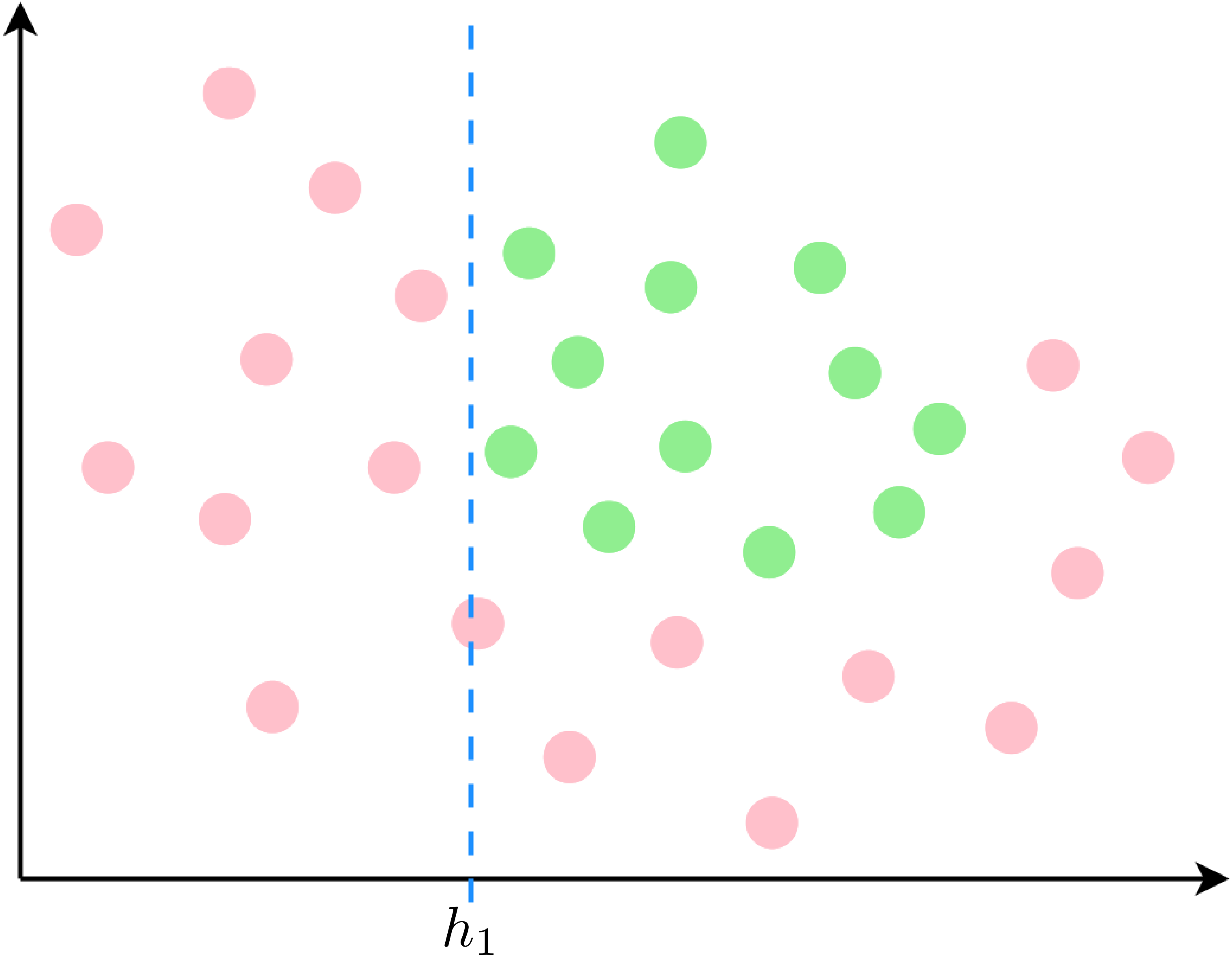
$$\exp(\alpha_t) = \exp(0.97) = 2.63$$

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

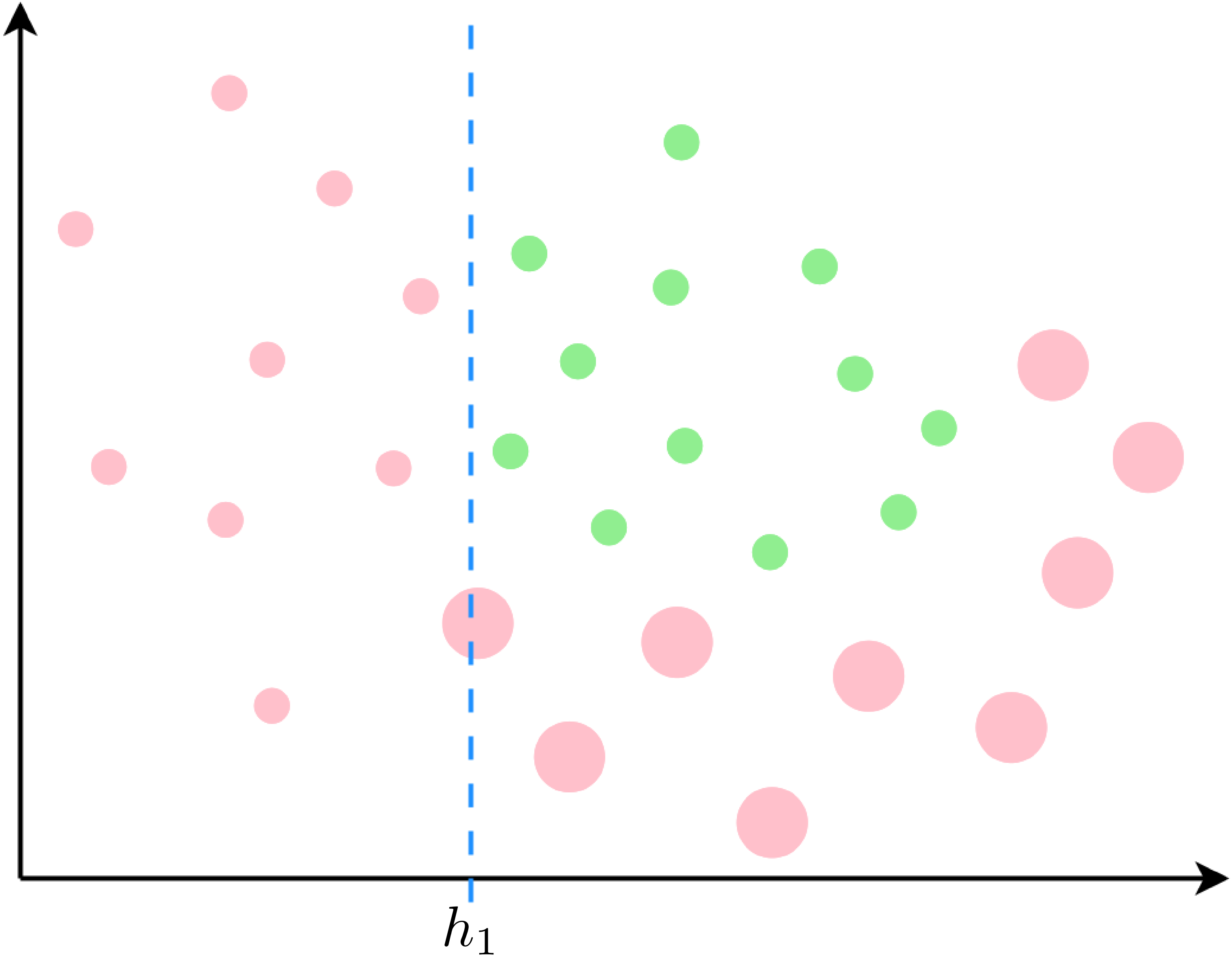
ID	E1	E2.	Class	Weight	Weight
A	0.3	-0.6	+1	1/8 * 2.63	0.33
B	0.5	-0.5	+1	1/8 * 0.38	0.05
C	0.7	-0.1	+1	1/8 * 0.38	0.05
D	0.6	-0.4	+1	1/8 * 0.38	0.05
E	0.2	0.4	-1	1/8 * 0.38	0.05
F	-0.8	0.1	-1	1/8 * 0.38	0.05
G	0.4	-0.9	-1	1/8 * 0.38	0.05
H	0.2	0.5	-1	1/8 * 0.38	0.05

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

AdaBoost



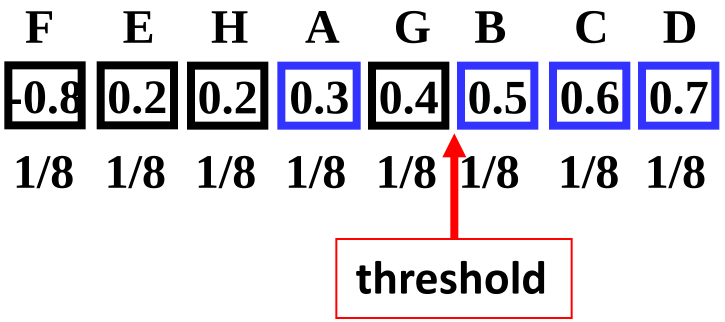
AdaBoost



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



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

threshold

$D' = D / sum(D)$

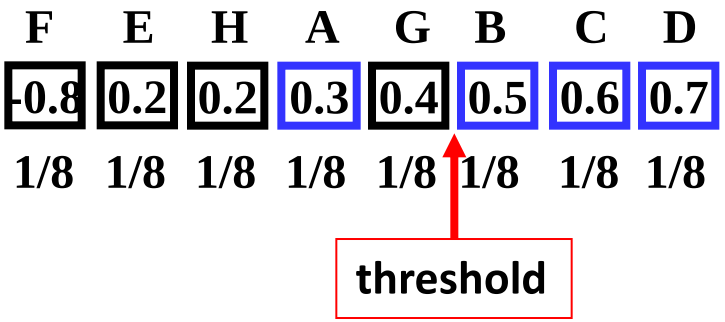
ID	E1	E2.	Class	Weight	Weight	Weight
A	0.3	-0.6	+1	1/8 * 2.63	0.33	0.48
B	0.5	-0.5	+1	1/8 * 0.38	0.05	0.074
C	0.7	-0.1	+1	1/8 * 0.38	0.05	0.074
D	0.6	-0.4	+1	1/8 * 0.38	0.05	0.074
E	0.2	0.4	-1	1/8 * 0.38	0.05	0.074
F	-0.8	0.1	-1	1/8 * 0.38	0.05	0.074
G	0.4	-0.9	-1	1/8 * 0.38	0.05	0.074
H	0.2	0.5	-1	1/8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



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

threshold

D' = D / sum(D)

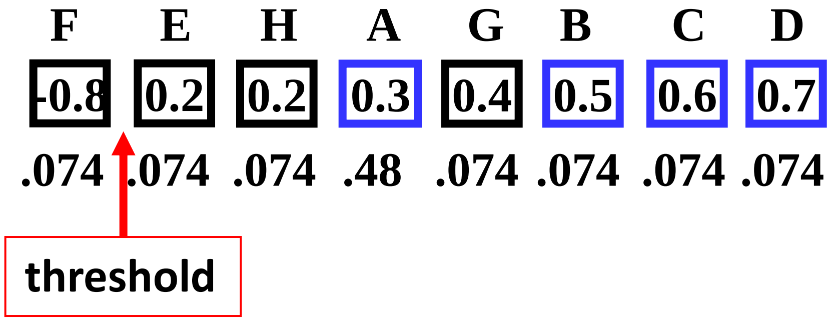
ID	E1	E2.	Class	Weight
A	0.3	-0.6	+1	0.48
B	0.5	-0.5	+1	0.074
C	0.7	-0.1	+1	0.074
D	0.6	-0.4	+1	0.074
E	0.2	0.4	-1	0.074
F	-0.8	0.1	-1	0.074
G	0.4	-0.9	-1	0.074
H	0.2	0.5	-1	0.074

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



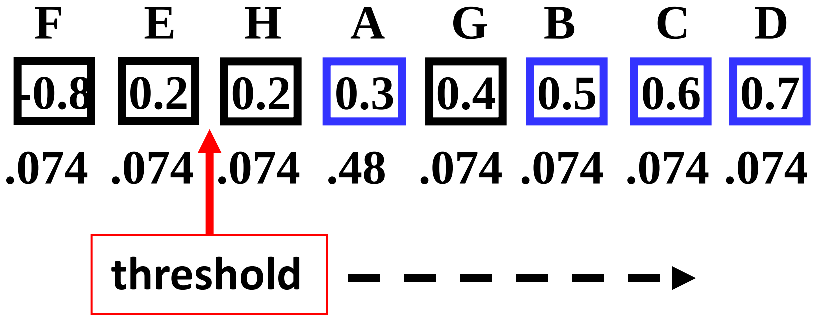
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

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

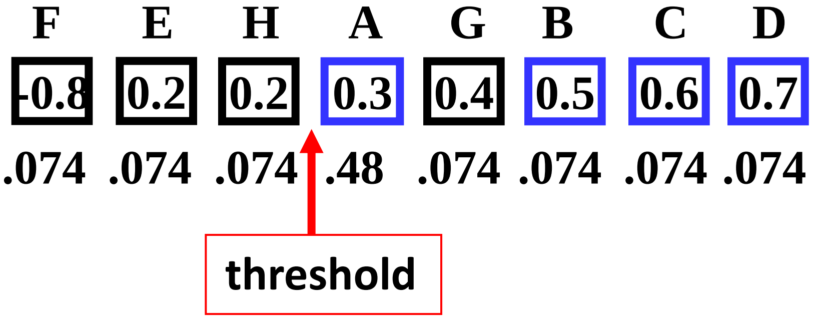
Sign = -1, error = 0.852

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



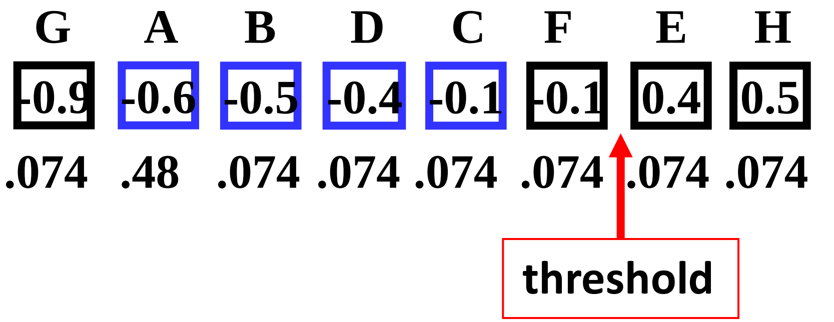
F	E	H	A	G	B	C	D
-0.8	0.2	0.2	0.3	0.4	0.5	0.6	0.7
.074	.074	.074	.48	.074	.074	.074	.074

threshold

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



G	A	B	D	C	F	E	H
-0.9	-0.6	-0.5	-0.4	-0.1	-0.1	0.4	0.5
.074	.48	.074	.074	.074	.074	.074	.074

threshold

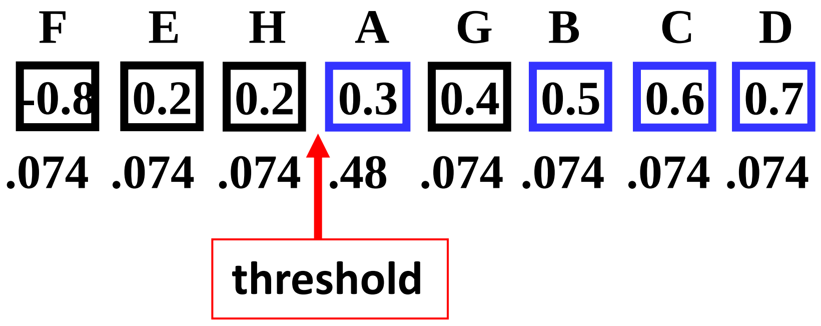
Classifier based on E2:
if (sign*wt(E2) > thresh) > 0)
    face = true

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



F	E	H	A	G	B	C	D
-0.8	0.2	0.2	0.3	0.4	0.5	0.6	0.7
.074	.074	.074	.48	.074	.074	.074	.074

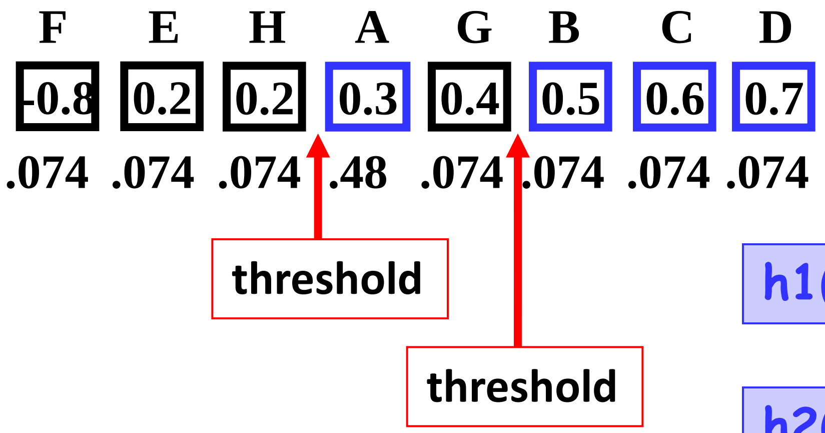
Sign = +1, error = 0.074

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

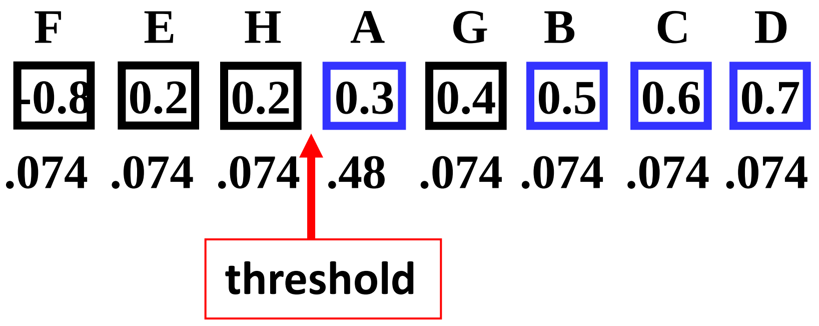


h1(X) = wt(E1) > 0.45 ? +1 : -1

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

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

Reweighting the Data



F	E	H	A	G	B	C	D
-0.8	0.2	0.2	0.3	0.4	0.5	0.6	0.7
.074	.074	.074	.48	.074	.074	.074	.074

threshold

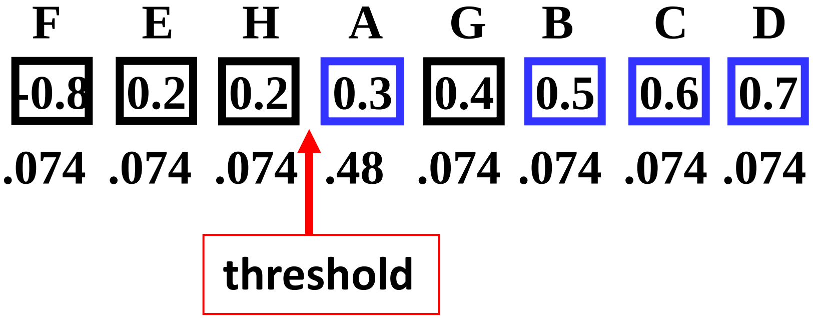
Sign = +1, error = 0.074

Exp(alpha) = exp(1.26) = 3.5
Exp(-alpha) = exp(-1.26) = 0.28

ID	E1	E2.	Class	Weight	
A	0.3	-0.6	+1	0.48*0.28	0.32
B	0.5	-0.5	+1	0.074*0.28	0.05
C	0.7	-0.1	+1	0.074*0.28	0.05
D	0.6	-0.4	+1	0.074*0.28	0.05
E	0.2	0.4	-1	0.074*0.28	0.05
F	-0.8	0.1	-1	0.074*0.28	0.05
G	0.4	-0.9	-1	0.074*3.5	0.38
H	0.2	0.5	-1	0.074*0.28	0.05

RENORMALIZE

Reweighting the Data



F	E	H	A	G	B	C	D
-0.8	0.2	0.2	0.3	0.4	0.5	0.6	0.7
.074	.074	.074	.48	.074	.074	.074	.074

Sign = +1, error = 0.074

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

ID	E1	E2.	Class	Weight	
A	0.3	-0.6	+1	0.48*0.28	0.32
B	0.5	-0.5	+1	0.074*0.28	0.05
C	0.7	-0.1	+1	0.074*0.28	0.05
D	0.6	-0.4	+1	0.074*0.28	0.05
E	0.2	0.4	-1	0.074*0.28	0.05
F	-0.8	0.1	-1	0.074*0.28	0.05
G	0.4	-0.9	-1	0.074*3.5	0.38
H	0.2	0.5	-1	0.074*0.28	0.05

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

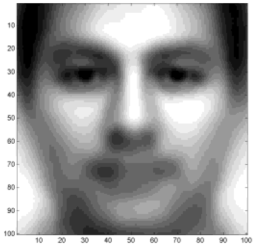
AdaBoost

- 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

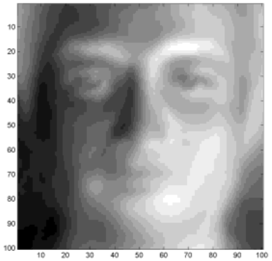
- We can *continue* to add classifiers even after we get 100% classification of the training data
  - Because the weights of the data keep changing
  - Adding new classifiers beyond this point is often a good thing to do

ADA Boost

= 0.4 E1 - 0.4 E2



$\mathbf{E_1}$



$\mathbf{E_2}$

- 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 is the basis of one of the most popular methods for face detection: The Viola-Jones algorithm
  - Current methods use other classifiers like SVMs, but adaboost classifiers remain easy to implement and popular
  - OpenCV implements Viola Jones..

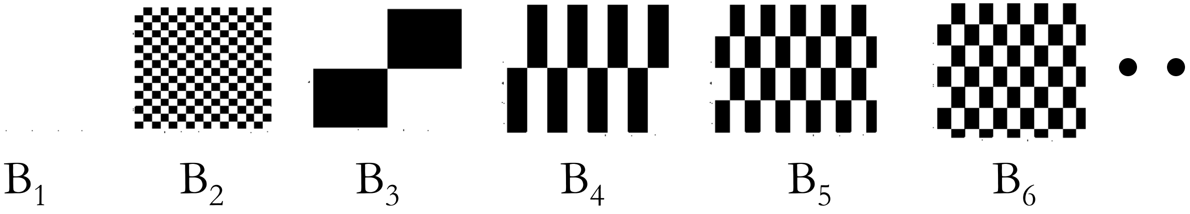
The problem of face detection

- 1. Defining Features
  - Should we be searching for noses, eyes, eyebrows etc.?
    - Nice, but expensive
  - Or something simpler

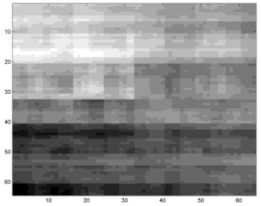
- 2. Selecting Features
  - Of all the possible features we can think of, which ones make sense

- 3. Classification: Combining evidence
  - How does one combine the evidence from the different features?

Features: The Viola Jones Method



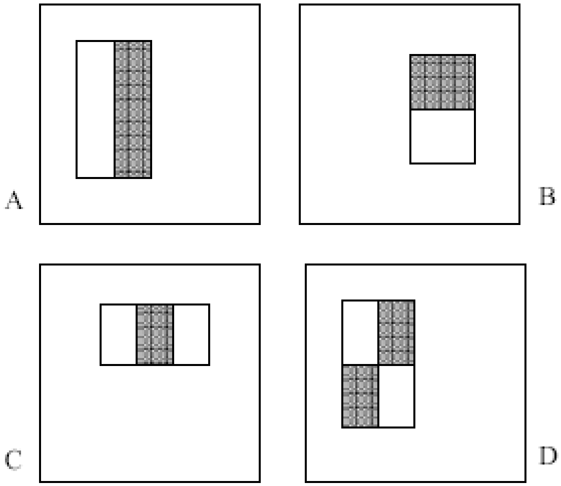
$$\mathrm{Image} \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

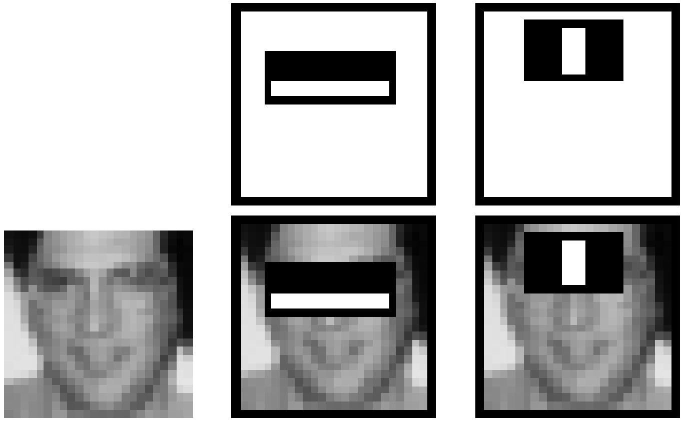


- Integral Features!!
  - Like the Checkerboard
- The same principle as we used to decompose images in terms of checkerboards:
  - The image of any object has changes at various scales
  - These can be represented coarsely by a checkerboard pattern
- The checkerboard patterns must however now be *localized*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

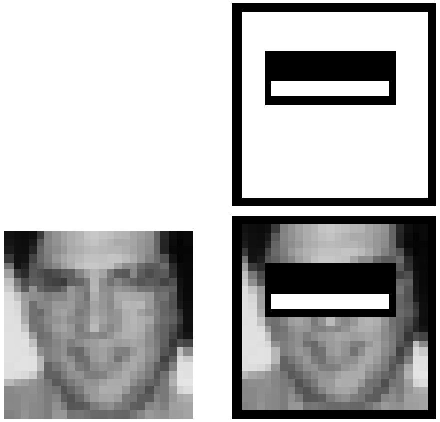
Features

- 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)



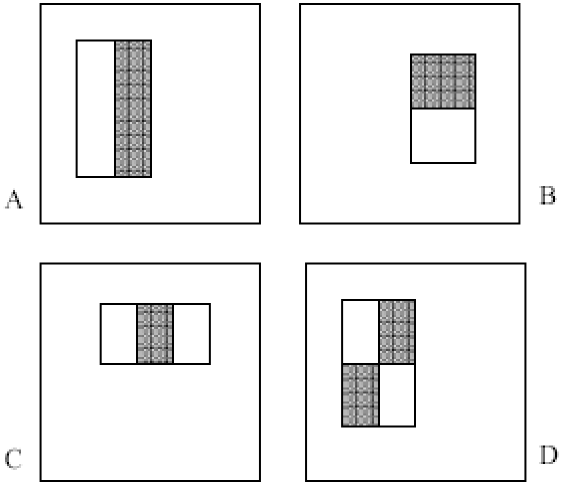


Explaining a portion of the face with a checker..



- 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” features

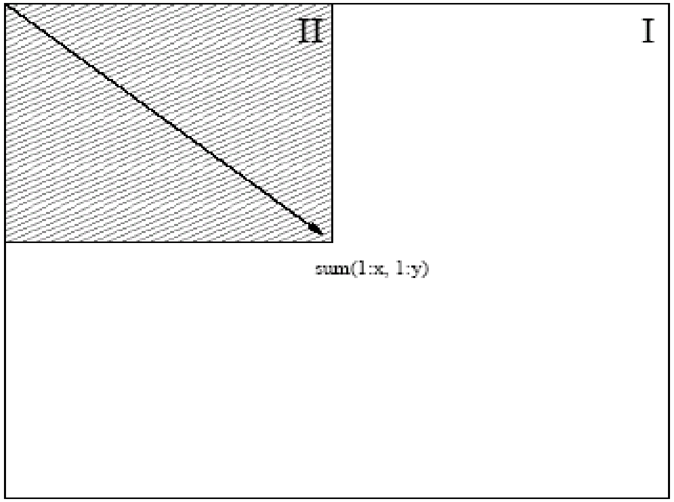


- Each checkerboard has the following characteristics
  - Length
  - Width
  - Type
    - Specifies the number and arrangement of bands

- The four checkerboards above are the four used by Viola and Jones

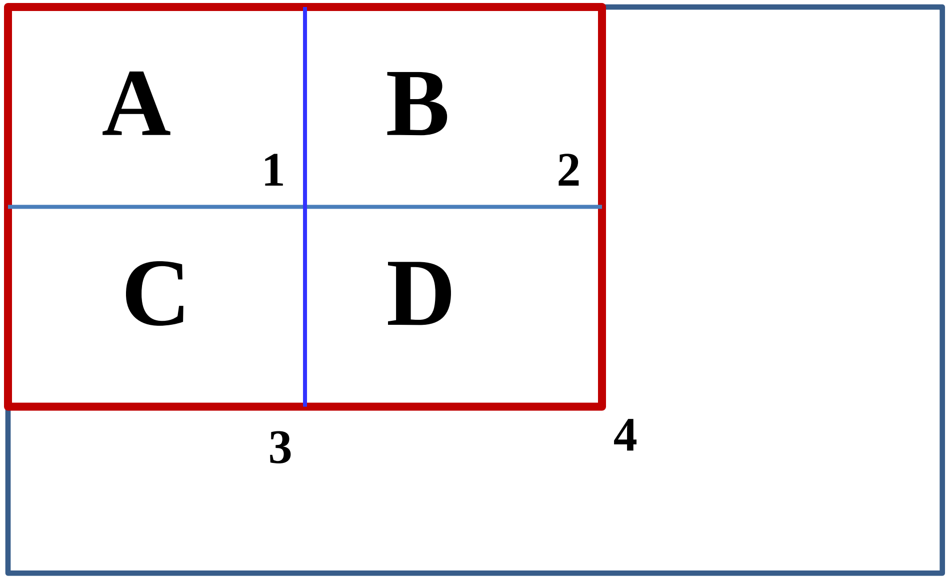
Integral images

- Summed area tables



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

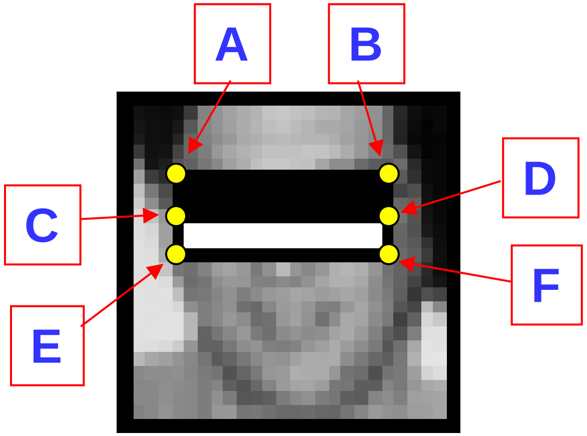
Fast Computation of Pixel Sums



- To compute the sum of the pixels within "D":
  - Pixelsum(1) = Area(A)
  - Pixelsum(2) = Area(A) + Area(B)
  - Pixelsum(3) = Area(A) + Area(C)
  - Pixelsum(4) = Area(A)+Area(B)+Area(C) +Area(D)

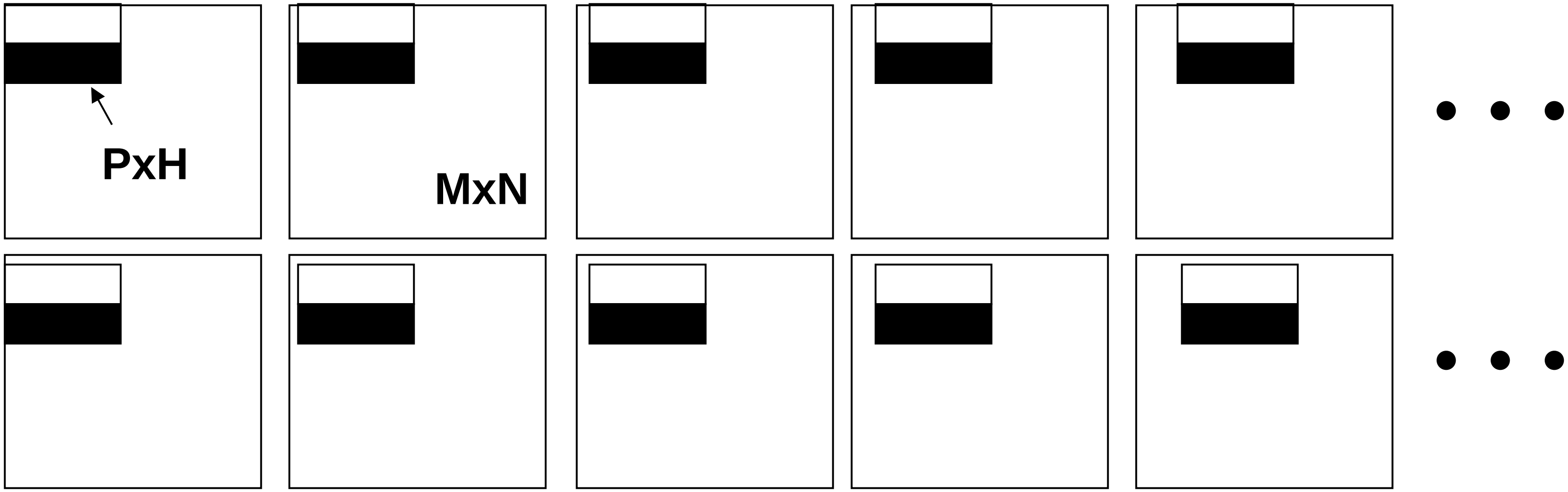
- Area(D) = Pixelsum(4) – Pixelsum(2) – Pixelsum(3) + Pixelsum(1)

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)

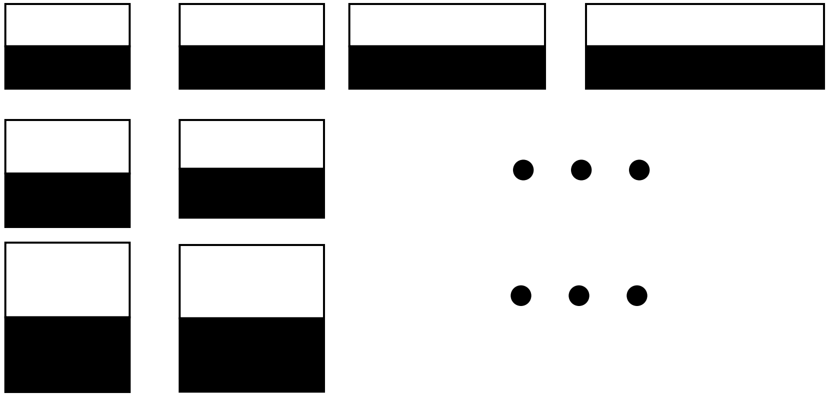
How many features?



- Each checker board of width P and height H can start at any of (N-P)(M-H) pixels

- (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

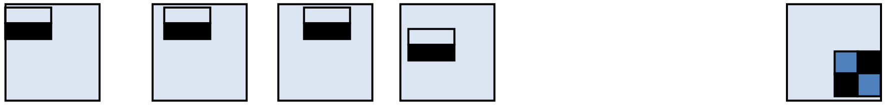


- Each feature can have many sizes
  - Width from (min) to (max) pixels
  - Height from (min ht) to (max ht) pixels
- At each size, there can be many starting locations
  - Total number of possible checkerboards of one type:
    No. of possible sizes x No. of possible locations
- There are four types 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 $\theta(f)$ and polarity *p*(f) (*p*(f) *= -1 or p*(f) *= 1*) such that (f > *p*(f) $\theta(f)$) 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') $\theta(f')$) => face

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

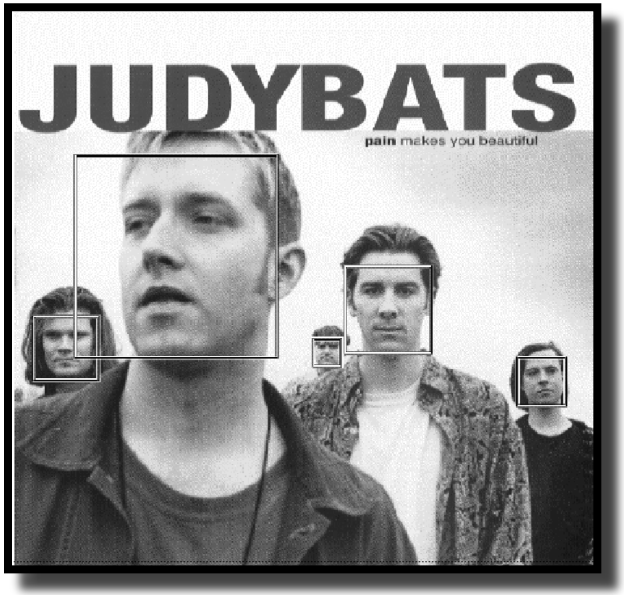
The Viola Jones Classifier

- 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 weighted data and its optimal threshold (weighting from two weak learners)
      - Fourth weak learner: Find the best feature for the weighted data and its optimal threhsold (weighting from three weak learners)
        - » ..

To Train

- Collect a large number of 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
  - These are our "non-face" training set
- Train a boosted classifier

The Viola Jones Classifier



- During tests:
  - Given any new 24x24 image
    - R = $\Sigma_f \alpha_f$ (f > p_f θ(f))
    - Only a small number of features (f < 100) typically used
- Problems:
  - Only classifies 24 x 24 images entirely as faces or non-faces
    - Pictures are typically 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

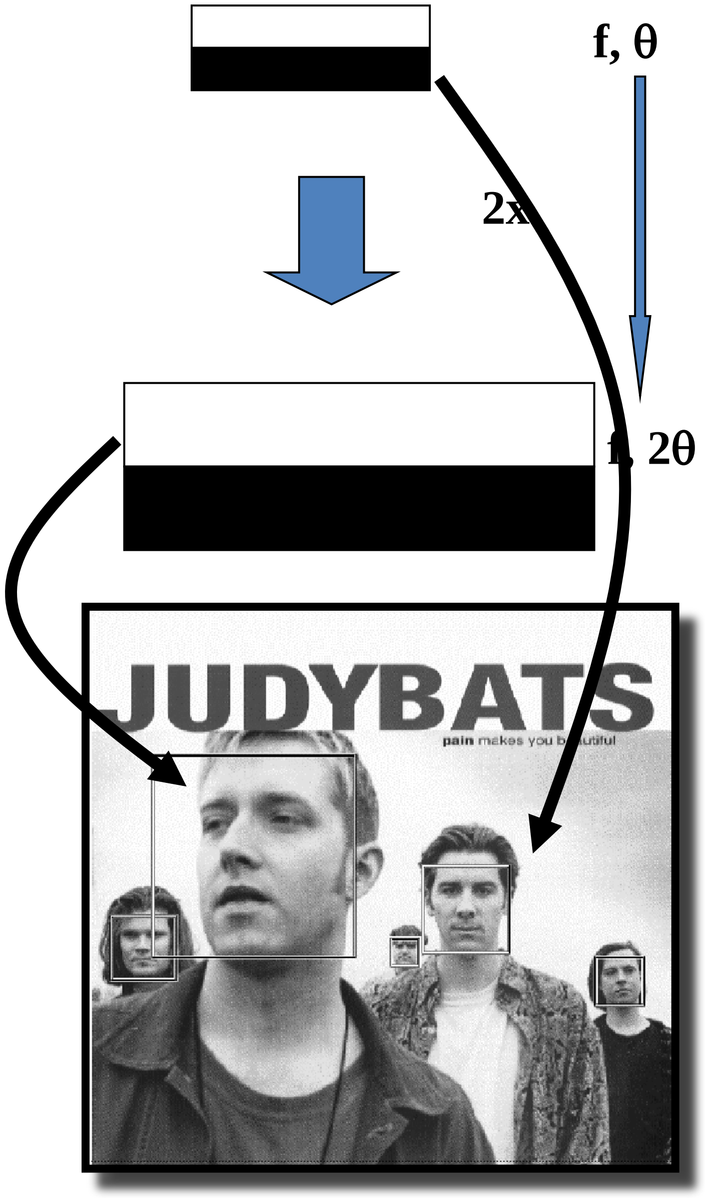
- Scan the image
  - Classify each 24x24 rectangle from the photo
  - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform (N-24)*(M-24) classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

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

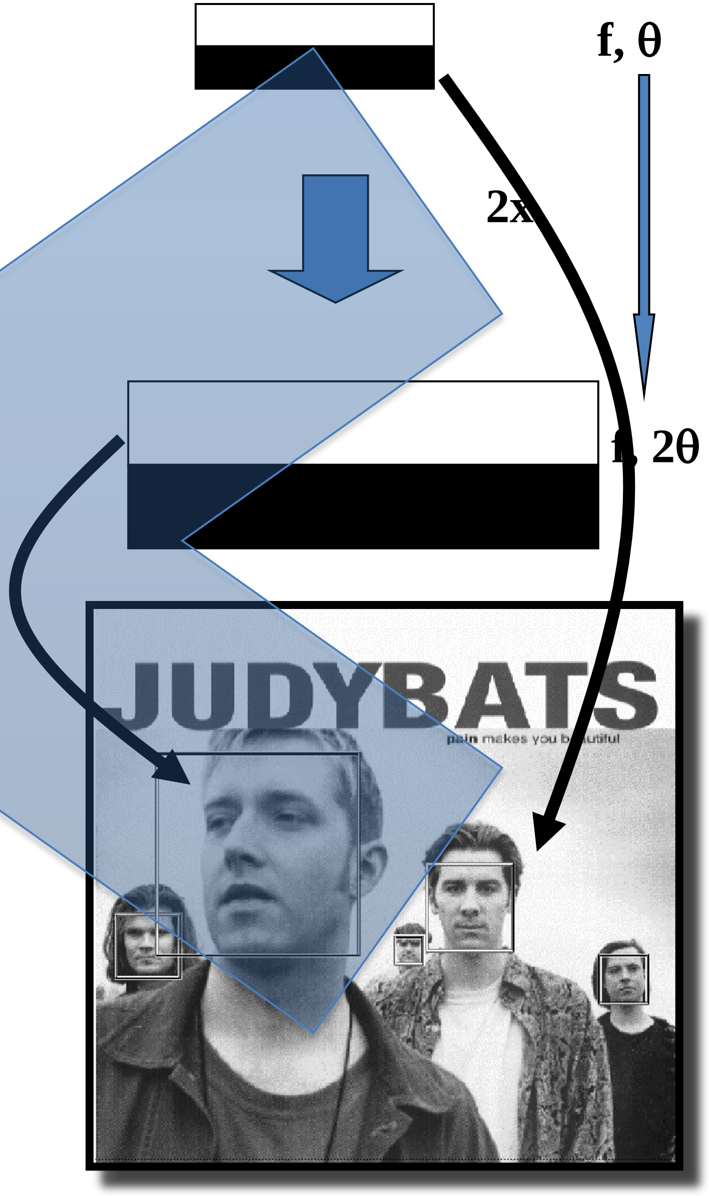
Picture size solution

- 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 $\alpha\theta$.
  - Do this for many scaling factors

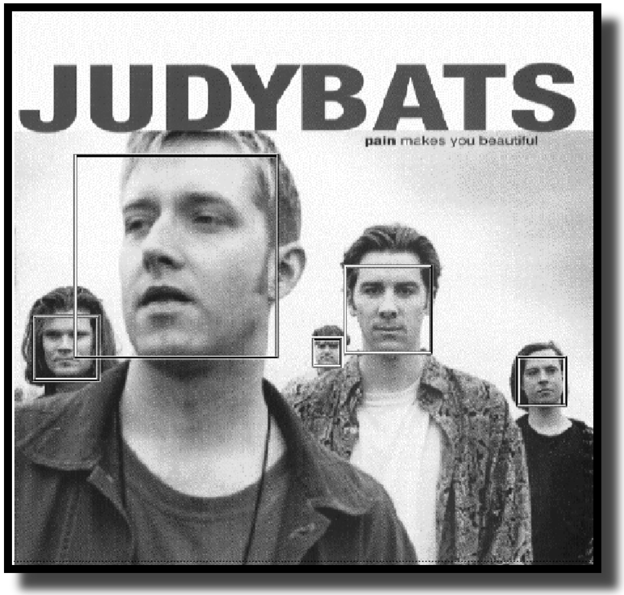


Picture size solution

- 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 $\alpha\theta$.
  - 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.

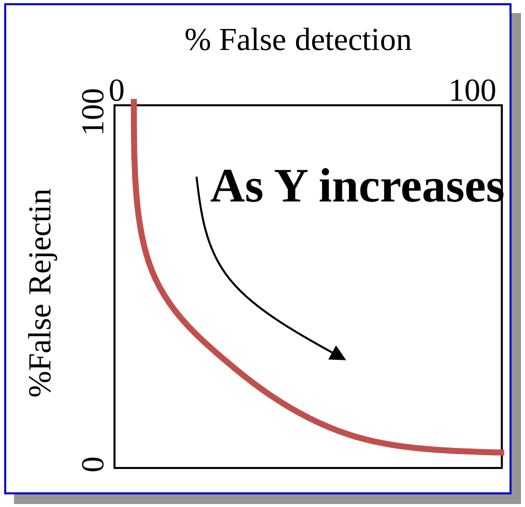
- Faces of different sizes will be found at different scales

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.

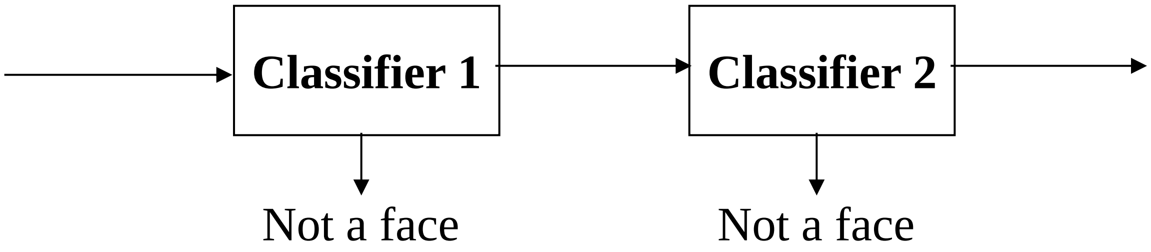
- Classifier:
  - Standard boosted classifier: $H(x) = \text{sign}(\Sigma_t \alpha_t h_t(x))$
  - Modified classifier $H(x) = \text{sign}(\Sigma_t \alpha_t h_t(x) + Y)$
    - $\Sigma_t \alpha_t h_t(x)$ is a measure of certainty
      - The higher it is, the more certain we are that we found a face
    - 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

ROC



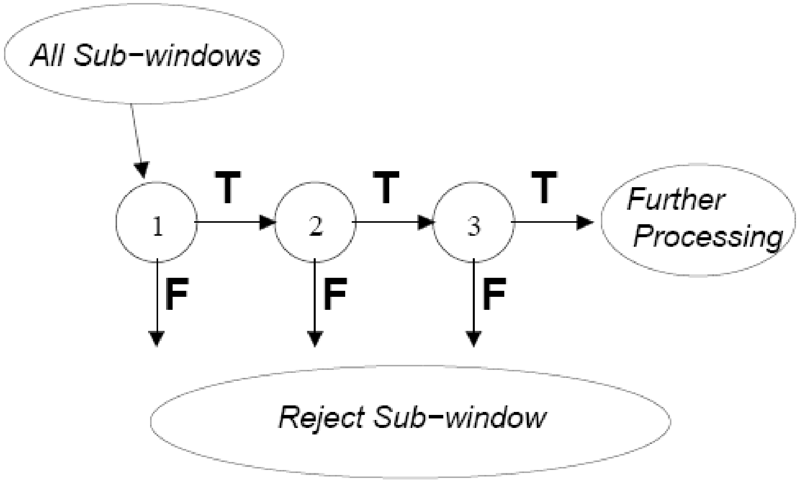
- Ideally false rejection will be 0%, false detection will also be 0%
- As Y increaases, we reject faces less and less
  - But accept increasing amounts of garbage as faces
- Can set Y so that we rarely miss a face

Problem: Not accurate enough, too slow



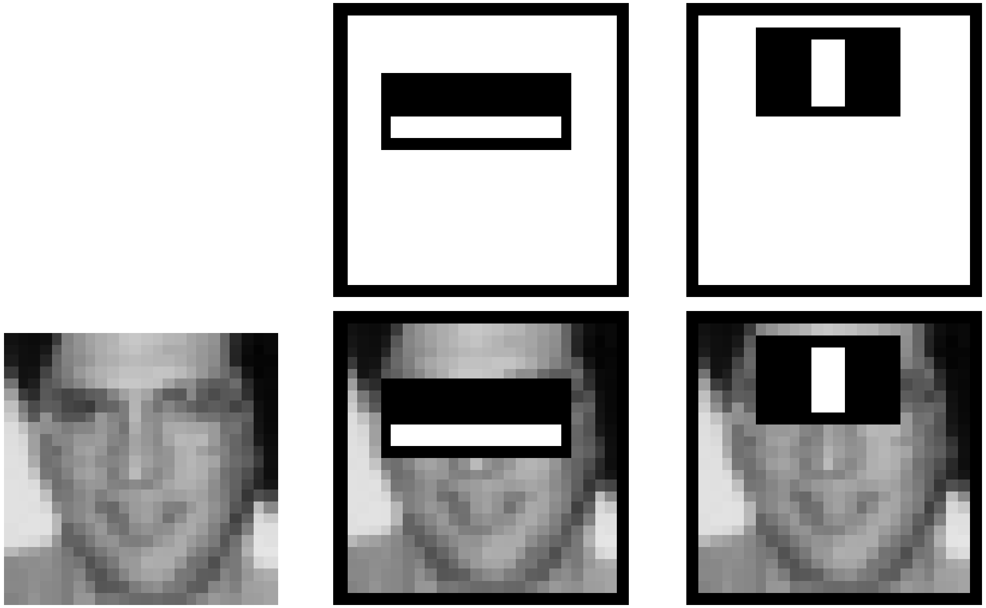
- If we set Y high enough, we will never miss a face
  - – But will classify a lot of junk as faces
- Solution: Classify the output of the first classifier with a second classifier
  - – And so on.

Problem: Not accurate enough, too slow

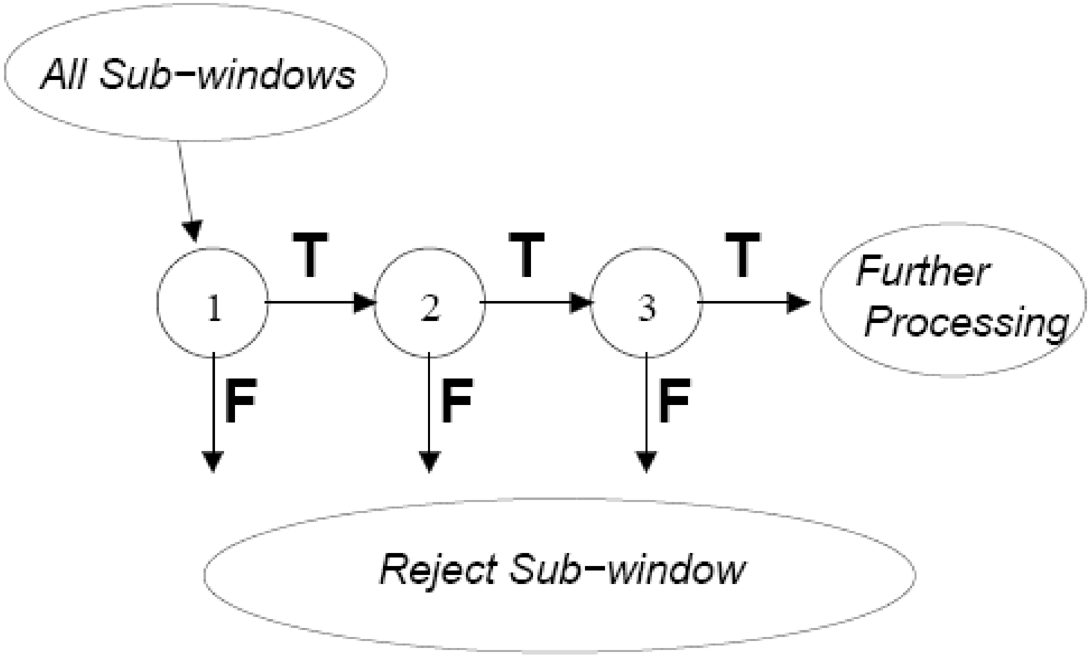


- If we set Y high enough, we will never miss a face
  – But will classify a lot of junk as faces
- Solution: Classify the output of the first classifier with a second classifier
  – And so on.

Useful Features Learned by Boosting



A Cascade of Classifiers



Detection in Real Images

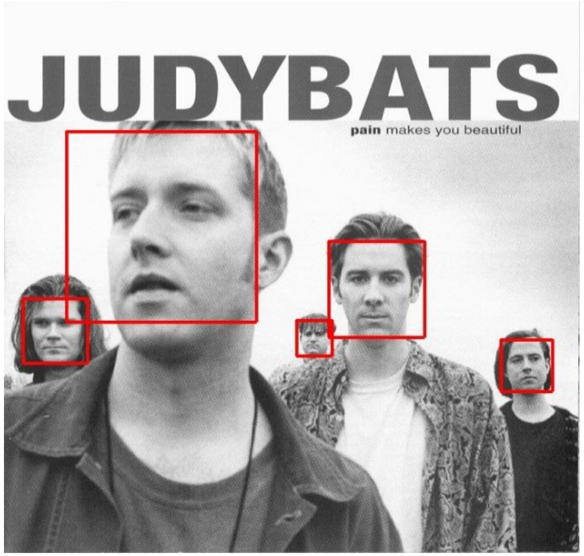
- Basic classifier operates on 24 x 24 subwindows
- Scaling:
  - Scale the detector (rather than the images)
  - Features can easily be evaluated at any scale
  - Scale by factors of 1.25
- Location:
  - Move detector around the image (e.g., 1 pixel increments)
- Final Detections
  - A real face may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

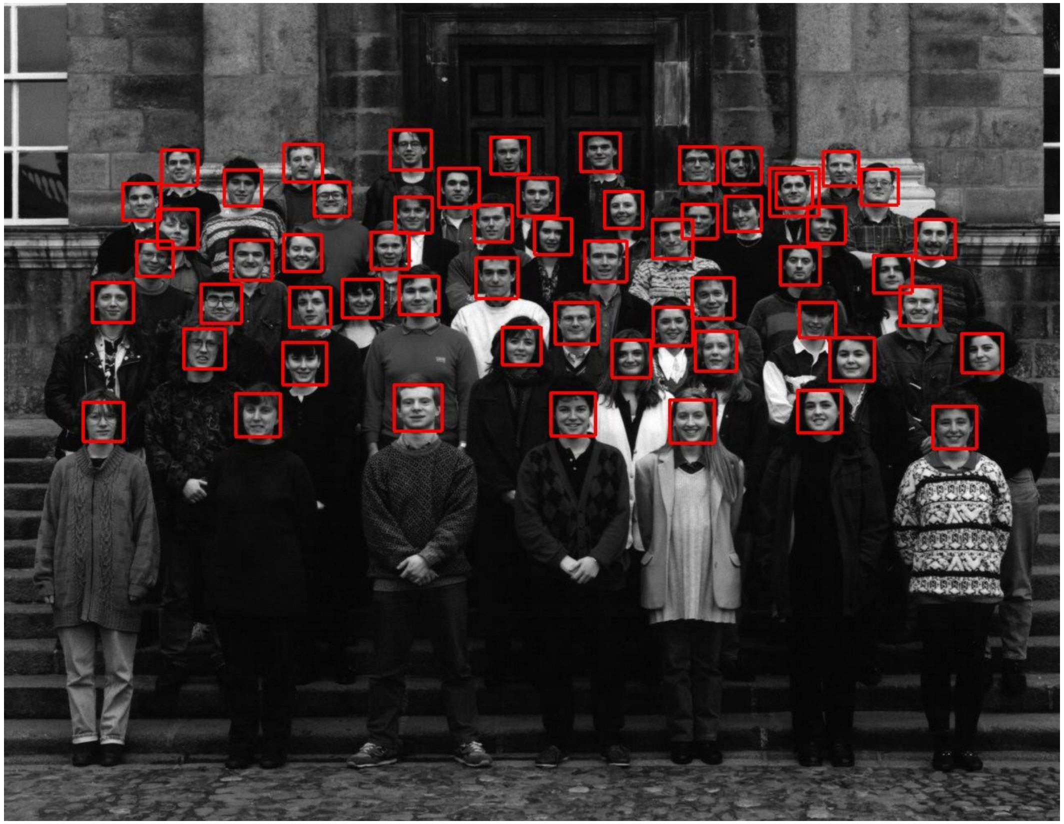
Training

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

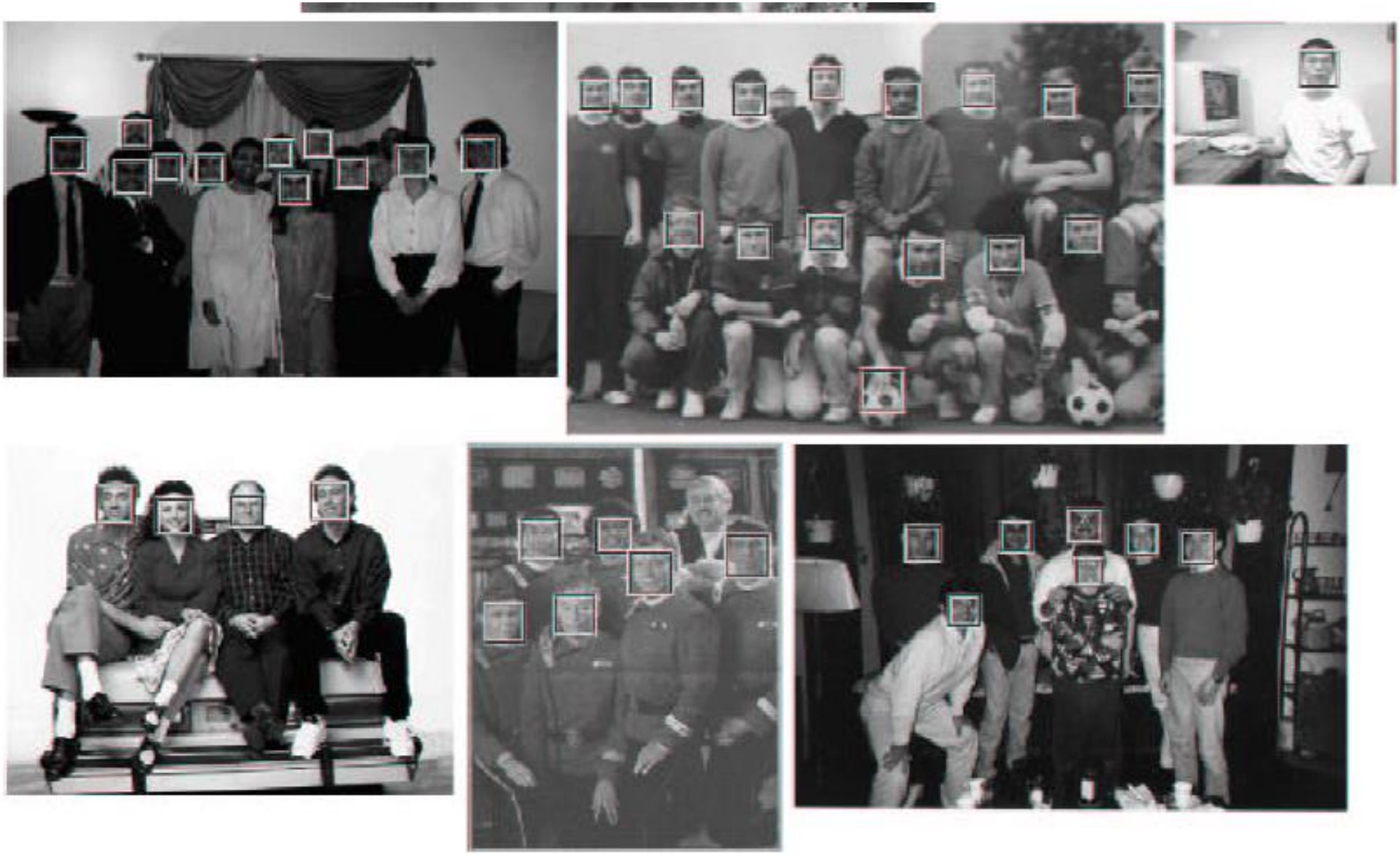
Sample results using the Viola-Jones Detector

- Notice detection at multiple scales





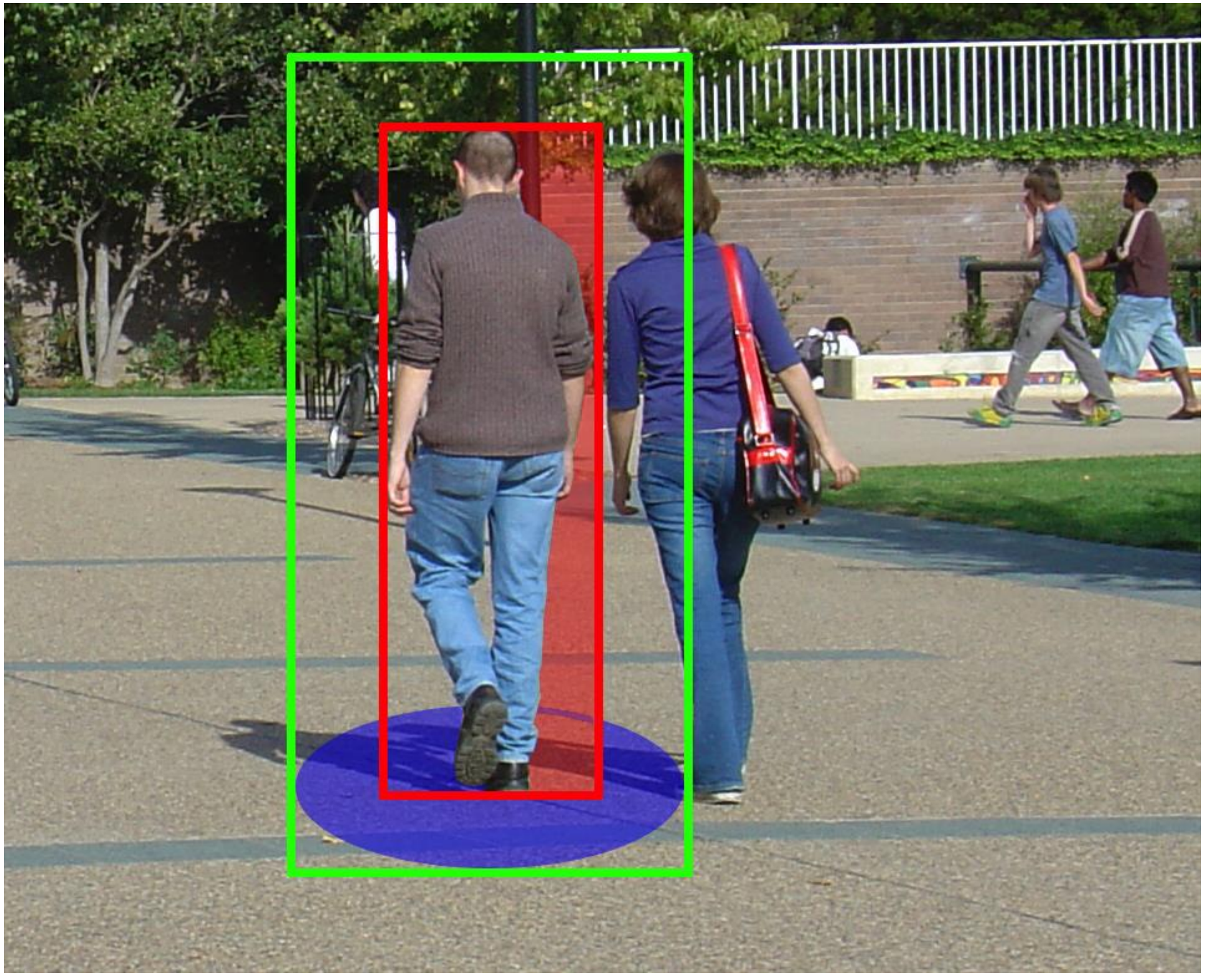
More Detection Examples



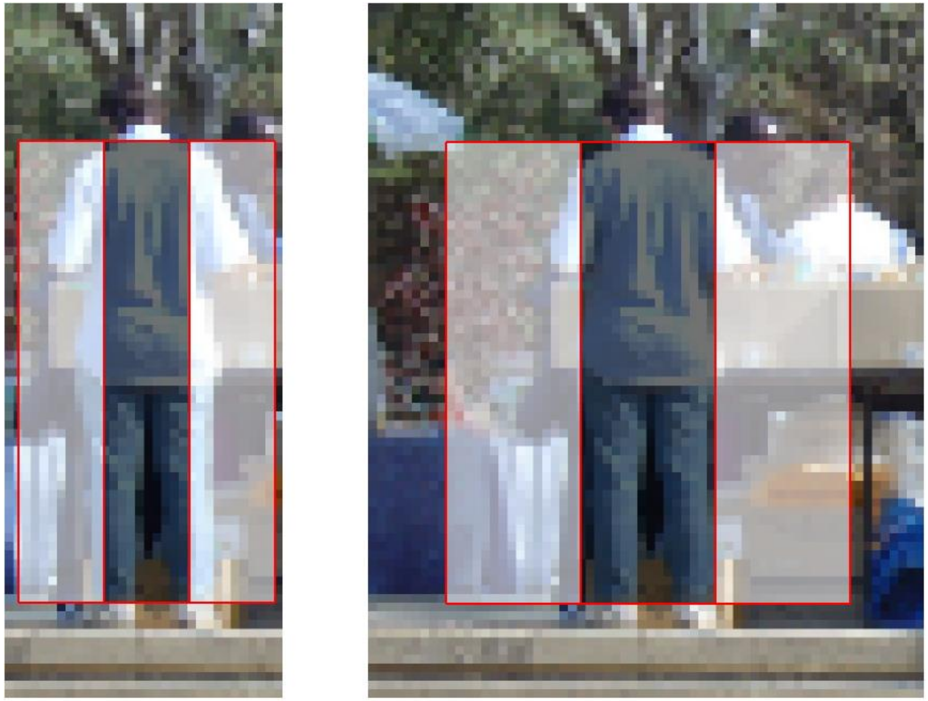
Practical implementation

- 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)

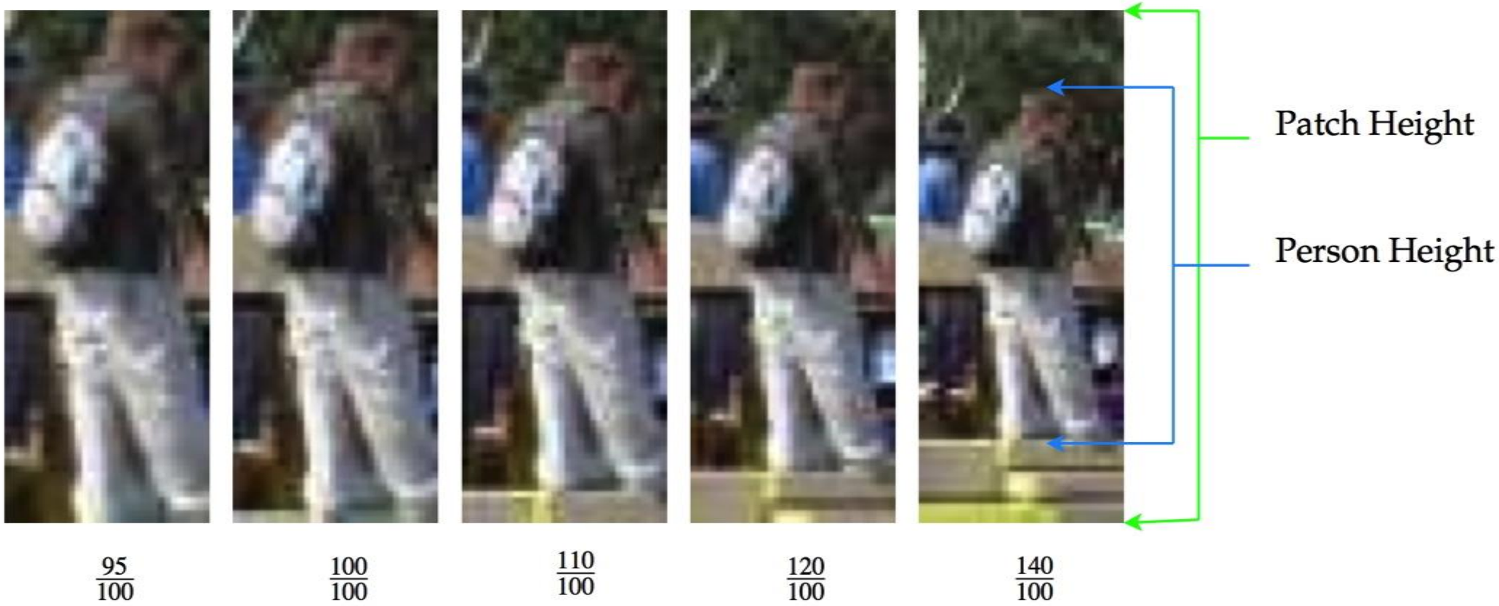
Best Window/Background Issues



Best Window/Background Issues



Best Window/Background Issues



Key Ideas

- EigenFace feature
- Sliding windows & scale-space pyramid
- Boosting an ensemble of weak classifiers
- Integral Image / Haar Features
- Cascaded Strong Classifiers