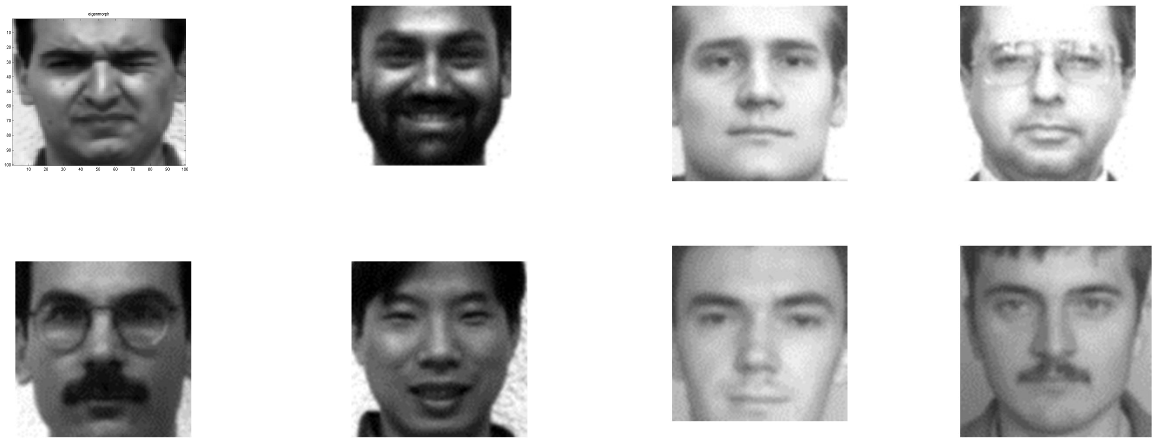
Machine Learning for Signal Processing

ADABOOST

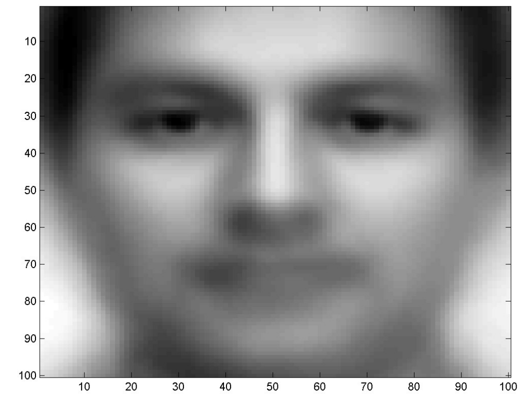
And an application to detecting faces (& other objects) in images

Bhiksha Raj

Last Lecture: How to describe a face

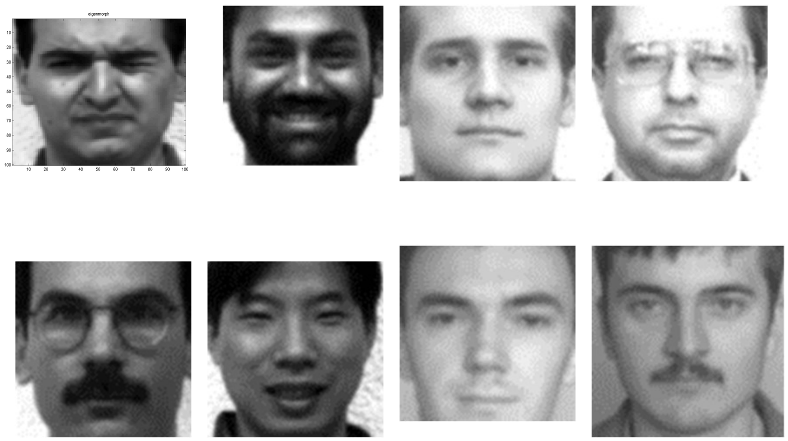


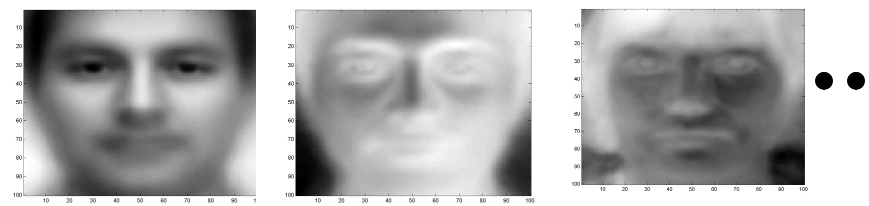
The typical face



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

A collection of least squares typical faces



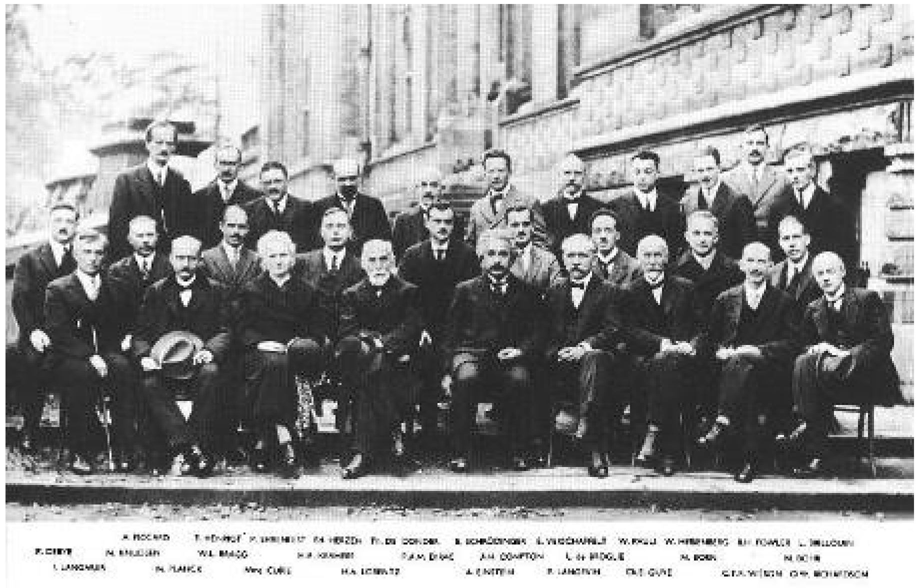


- Extension: Many Eigenfaces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + .. + w_{f,k} V_k$
  - V_2 is used to "correct" errors resulting from using only V_1
  - V_3 corrects errors remaining after correction with V_2
  - And so on..

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

Detecting Faces in Images

Detecting Faces in Images



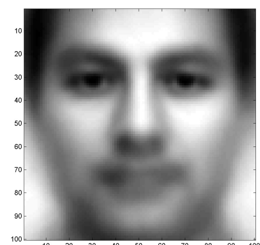
- Finding face like patterns
  - How do we find if a picture has faces in it
  - Where are the faces?

- 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

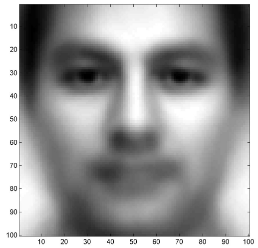
+

?

Finding faces in an image

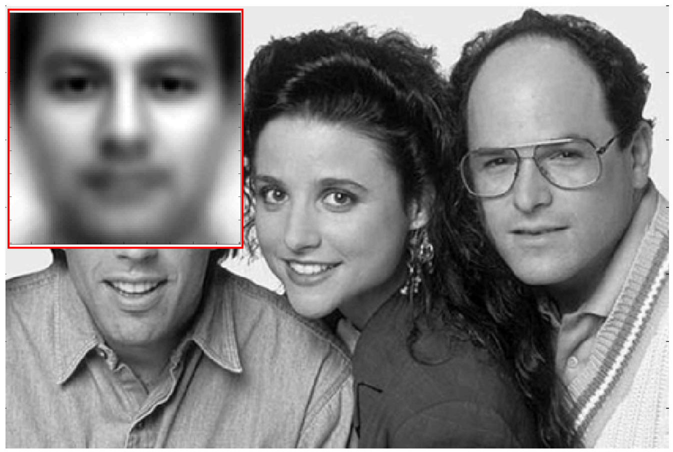
- Picture is larger than the “typical face”
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - R + G + B
  - Not very useful to work in color

Finding faces in an image



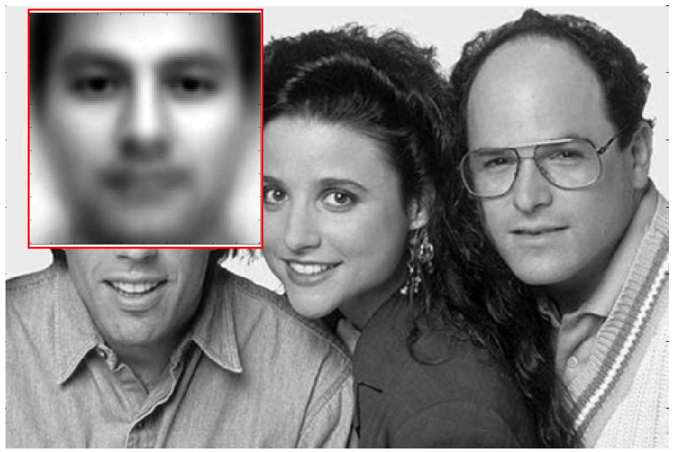
- Goal .. To find out if and where images that look like the "typical" face occur in the picture

Finding faces in an image



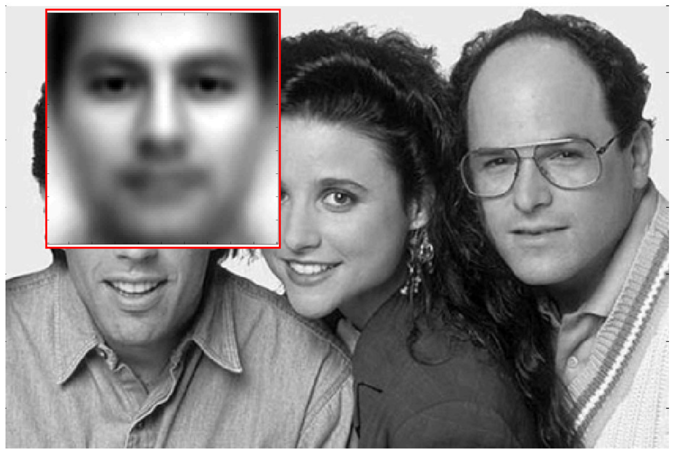
- Try to "match" the typical face to each location in the picture

Finding faces in an image



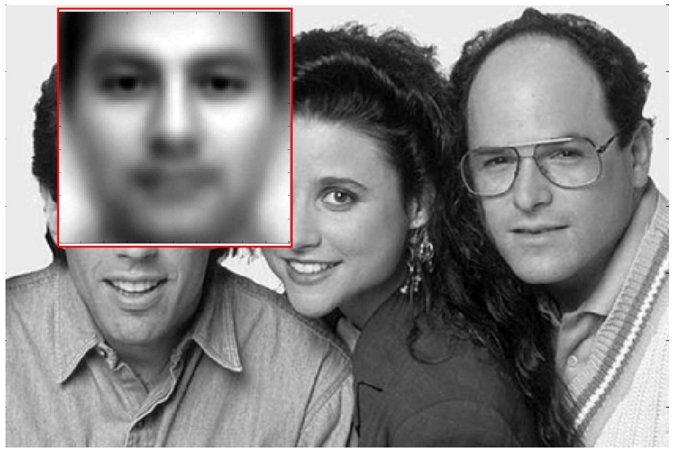
- Try to “match” the typical face to each location in the picture

Finding faces in an image



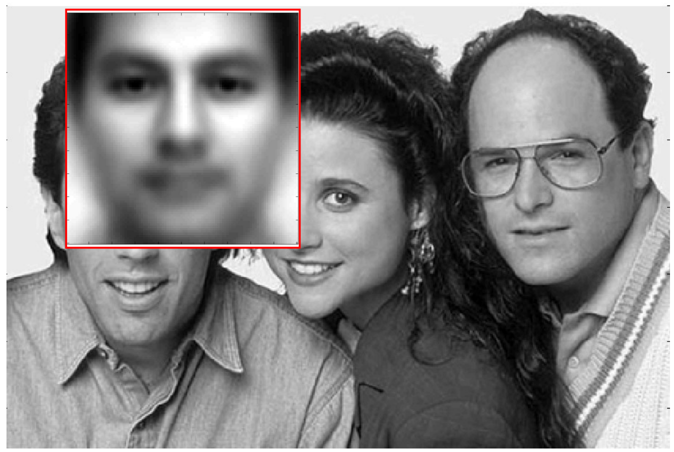
- Try to "match" the typical face to each location in the picture

Finding faces in an image



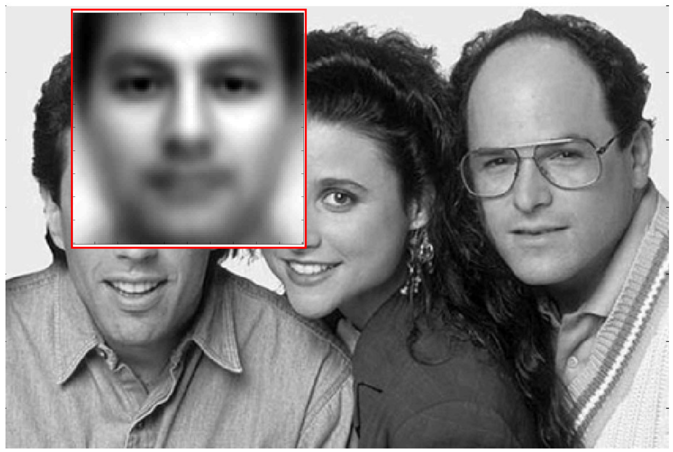
- Try to “match” the typical face to each location in the picture

Finding faces in an image



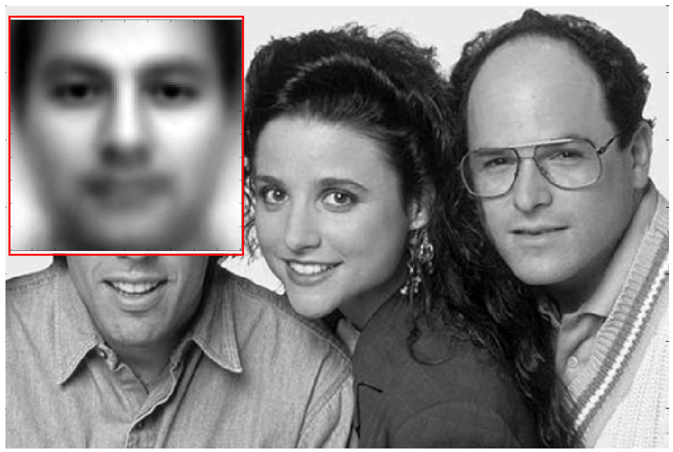
- Try to "match" the typical face to each location in the picture

Finding faces in an image



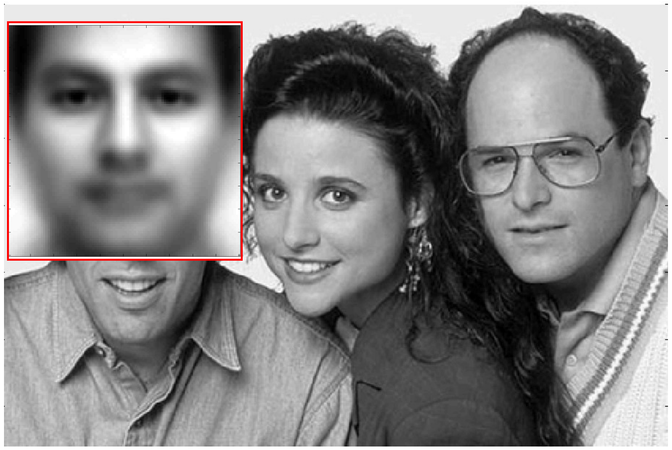
- Try to “match” the typical face to each location in the picture

Finding faces in an image



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

Finding faces in an image

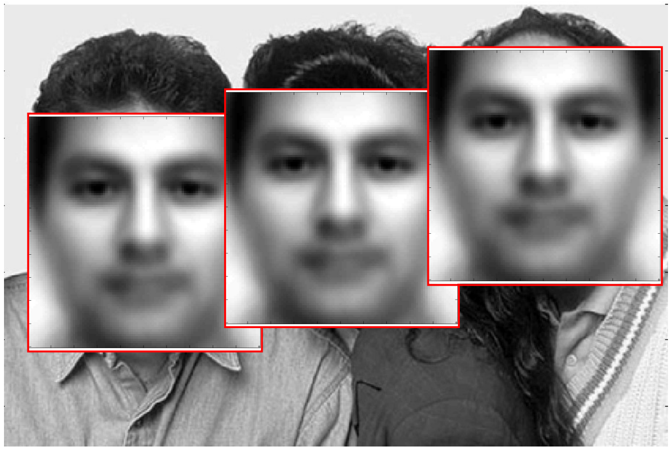


- Try to “match” the typical face to each location in the picture

Finding faces in an image

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

Finding faces in an image

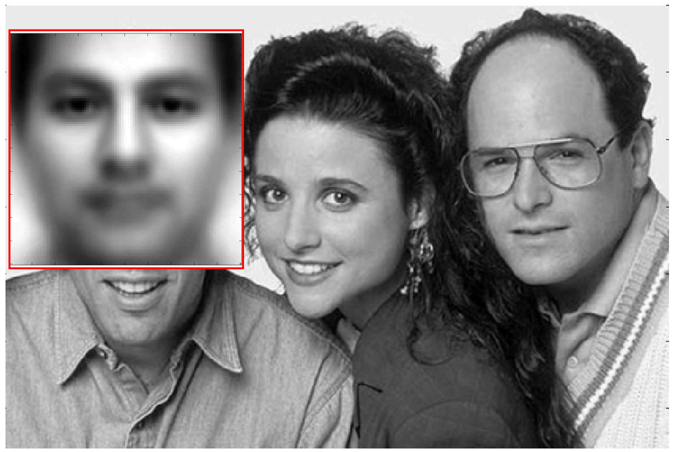


- Try to "match" the typical face to each location in the picture
- The "typical face" will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!

How to “match”

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

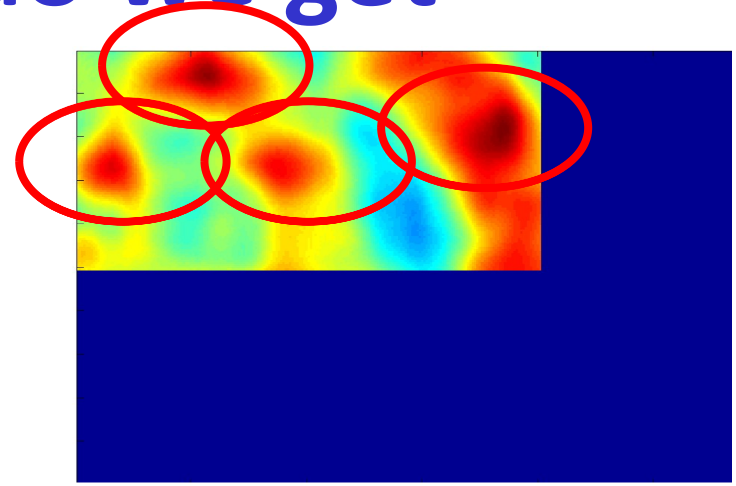
How to "match"



- What exactly is the "match"
  - What is the match "score"

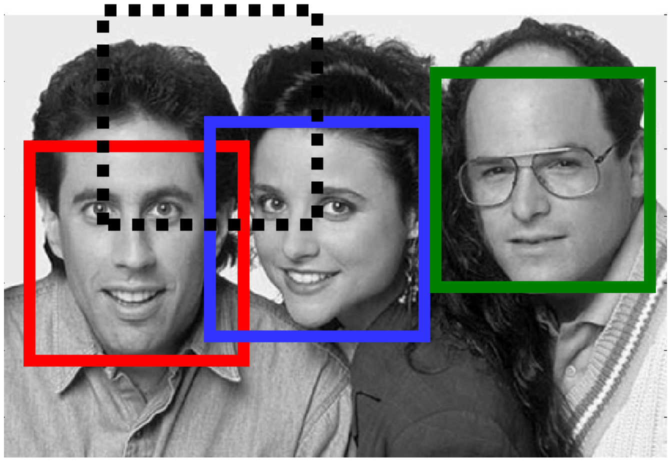
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
  - Compute the dot product of the typical face vector and the "region" vector

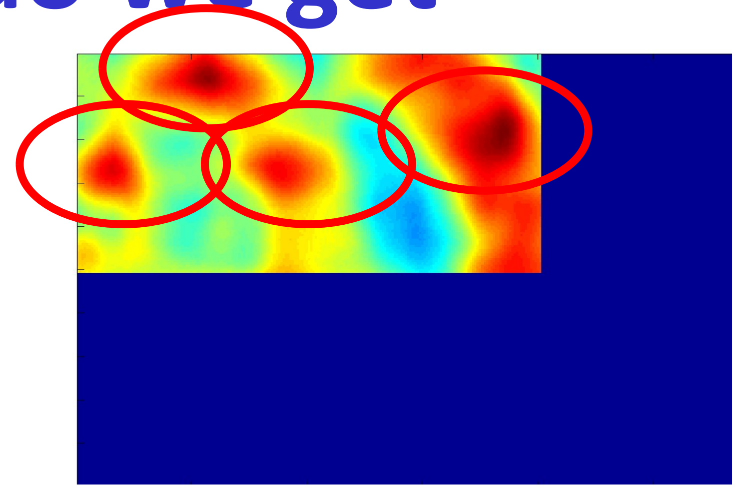
What do we get



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

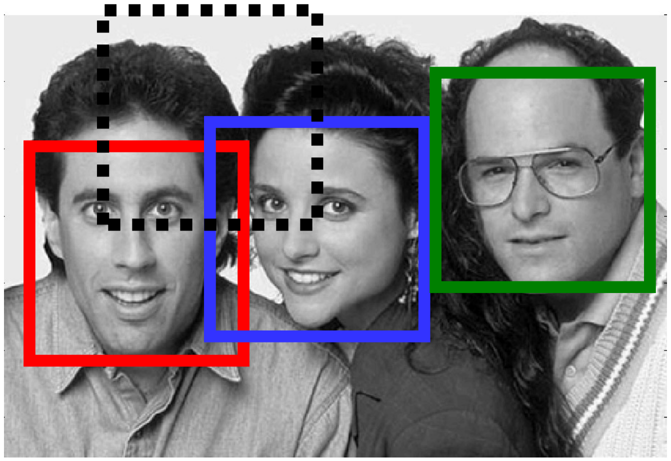
What do we get

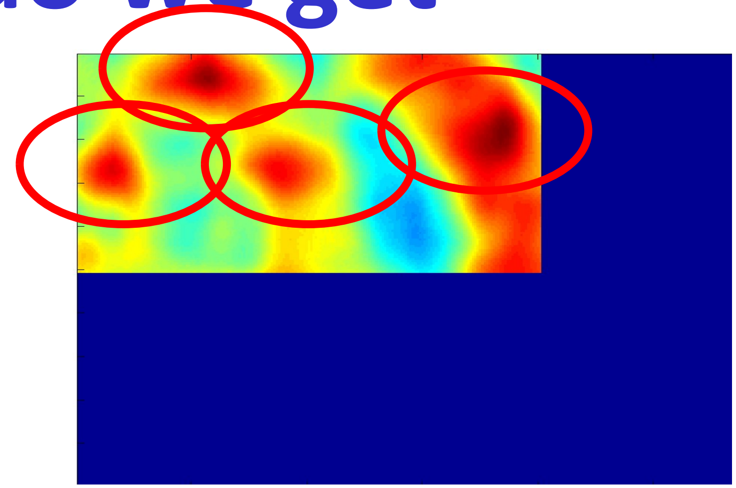




- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George's face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm

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

Poll 1

Q1. For a normalized typical face, the scanning method of face detection described just now is checking the length of the projection on the typical face of every same-sized patch in the image.

1. True
2. False

Q2. False positives occur since the dot product of a patch that looks nothing like a face with the typical face can be large.

1. True
2. False

Poll 1

Q1. For a normalized typical face, the scanning method of face detection described just now is checking the length of the projection on the typical face of every same-sized patch in the image.

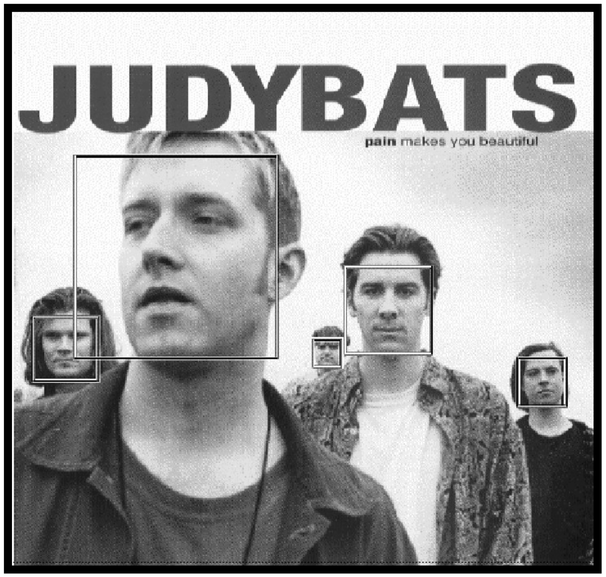
1. **True**
2. False

Q2. False positives occur since the dot product of a patch that looks nothing like a face with the typical face can be large.

1. **True**
2. False

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

Scale the image (but keep your typical face template fixed)

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.

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

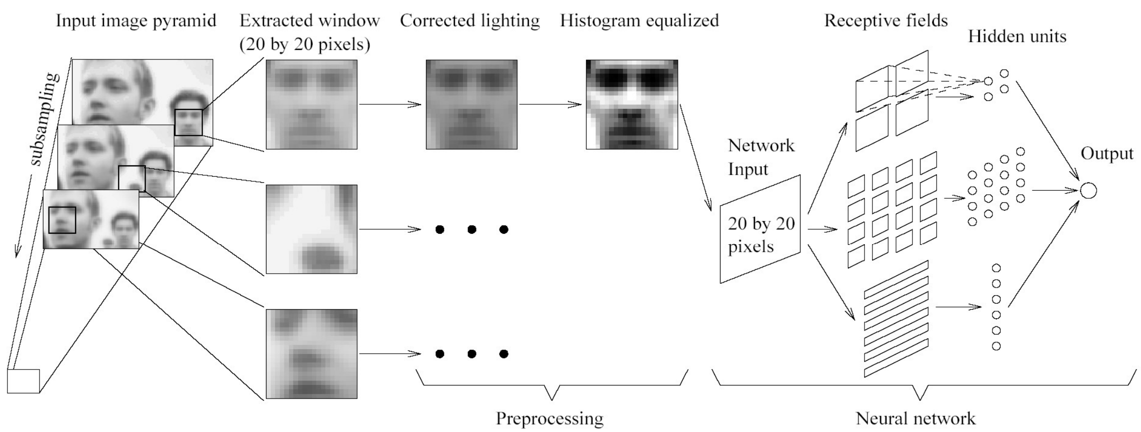


Figure 1: The basic algorithm used for face detection.

- 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

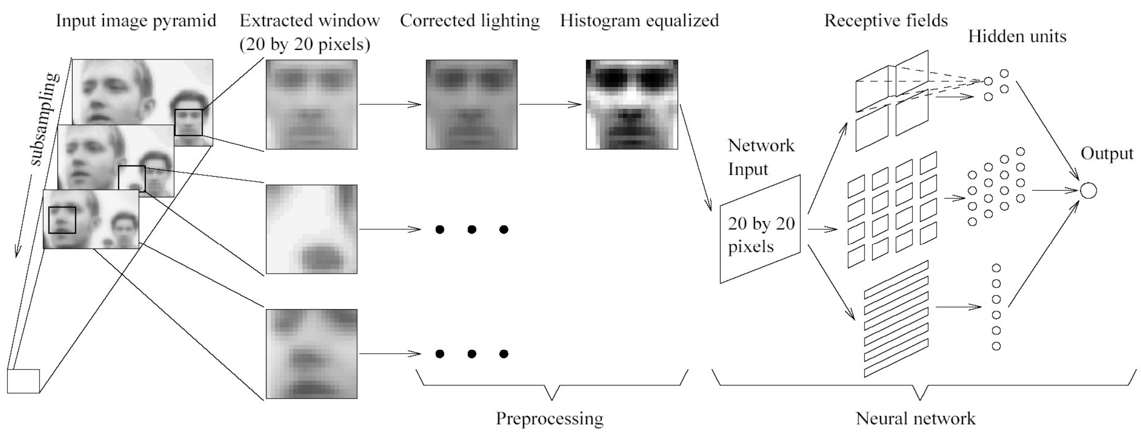


Figure 1: The basic algorithm used for face detection.

- 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 (45K+ 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..”

- 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?

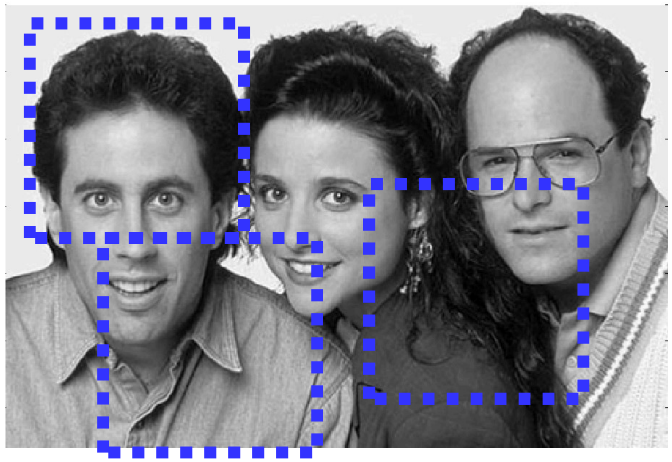
Detection vs Classification

- 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 *not*-X

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

- Restrictions for weak learners
  - Better than 50% correct

- Final classifier is *a combination of the decisions* of the weak classifiers

  - That somehow results in better classification that what is obtained with a single classifier

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 $h_i(x)$

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? $\alpha_{\text{armlength}}$ → **human**

Height? α_{height} → **human**

Lives in house? α_{house} → **chimp**

Lives in zoo? α_{zoo} → **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

- This is a “meta” algorithm, that poses no constraints on the form of the weak learners themselves

Boosting: A Voting Perspective

- Boosting is 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
    - **Boosting by majority**

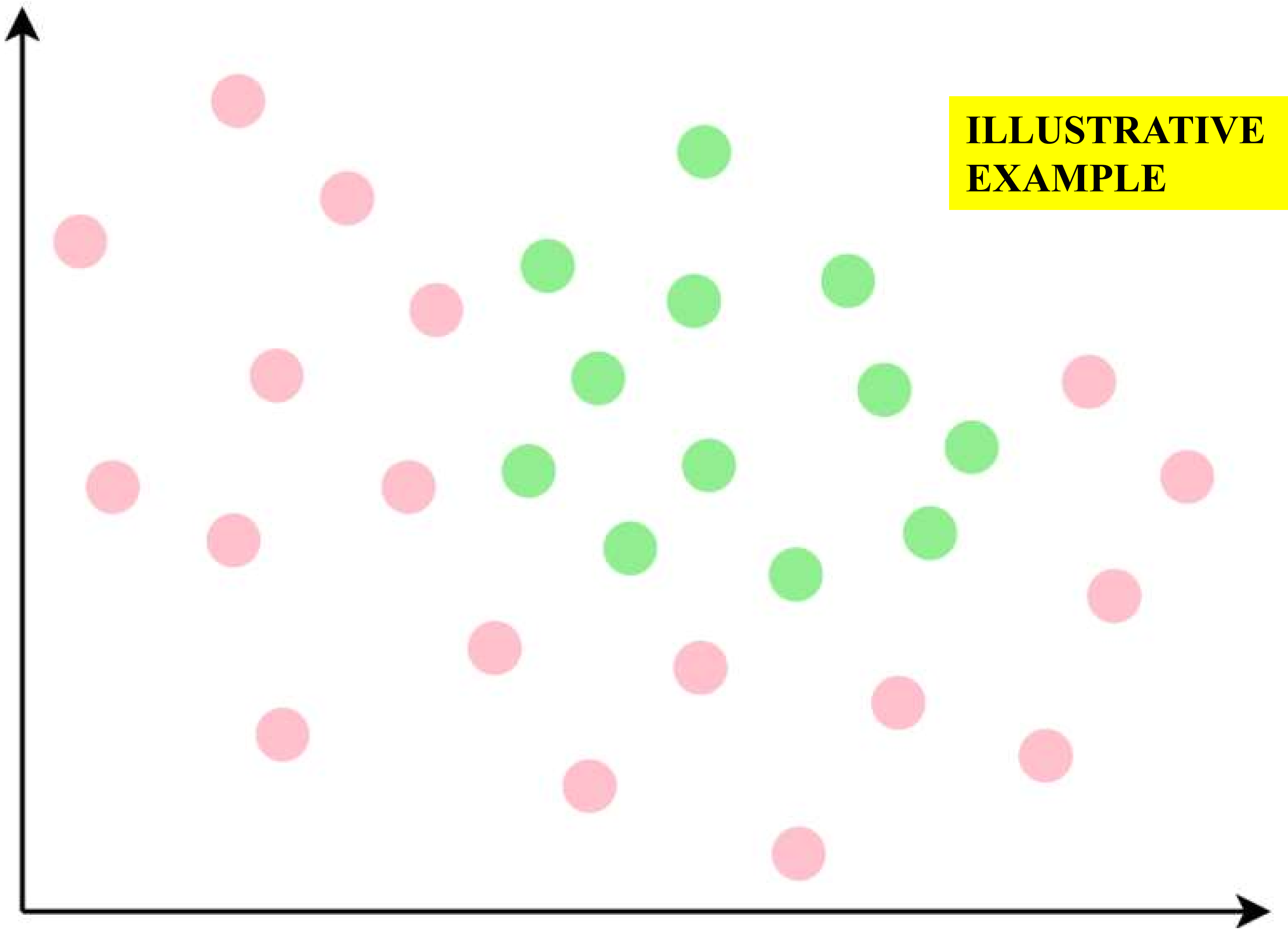
- Boosting by *weighted* majority
  – A (weighted) majority vote taken over all the classifiers
  – How do we compute weights for the classifiers?
  – How do we actually train the classifiers

ADA Boost

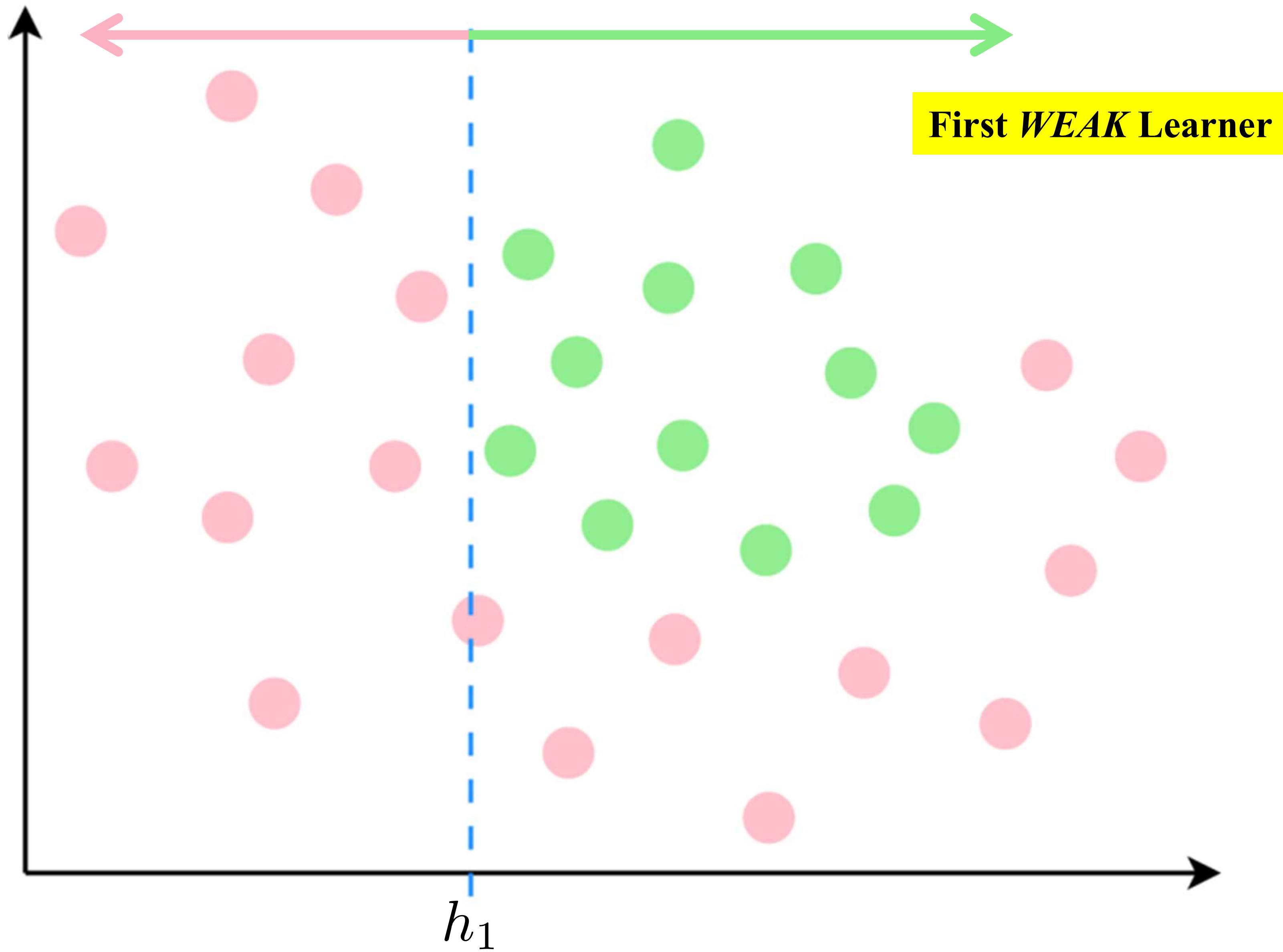
- Challenge: how to optimize the classifiers and their weights?
  - Trivial solution: Train all classifiers independently
  - Optimal: Each classifier focuses on what others missed
  - But joint optimization becomes impossible

- **Ada**ptive **Boost**ing: Greedy incremental optimization of classifiers
  - Keep adding classifiers incrementally, to fix what others missed

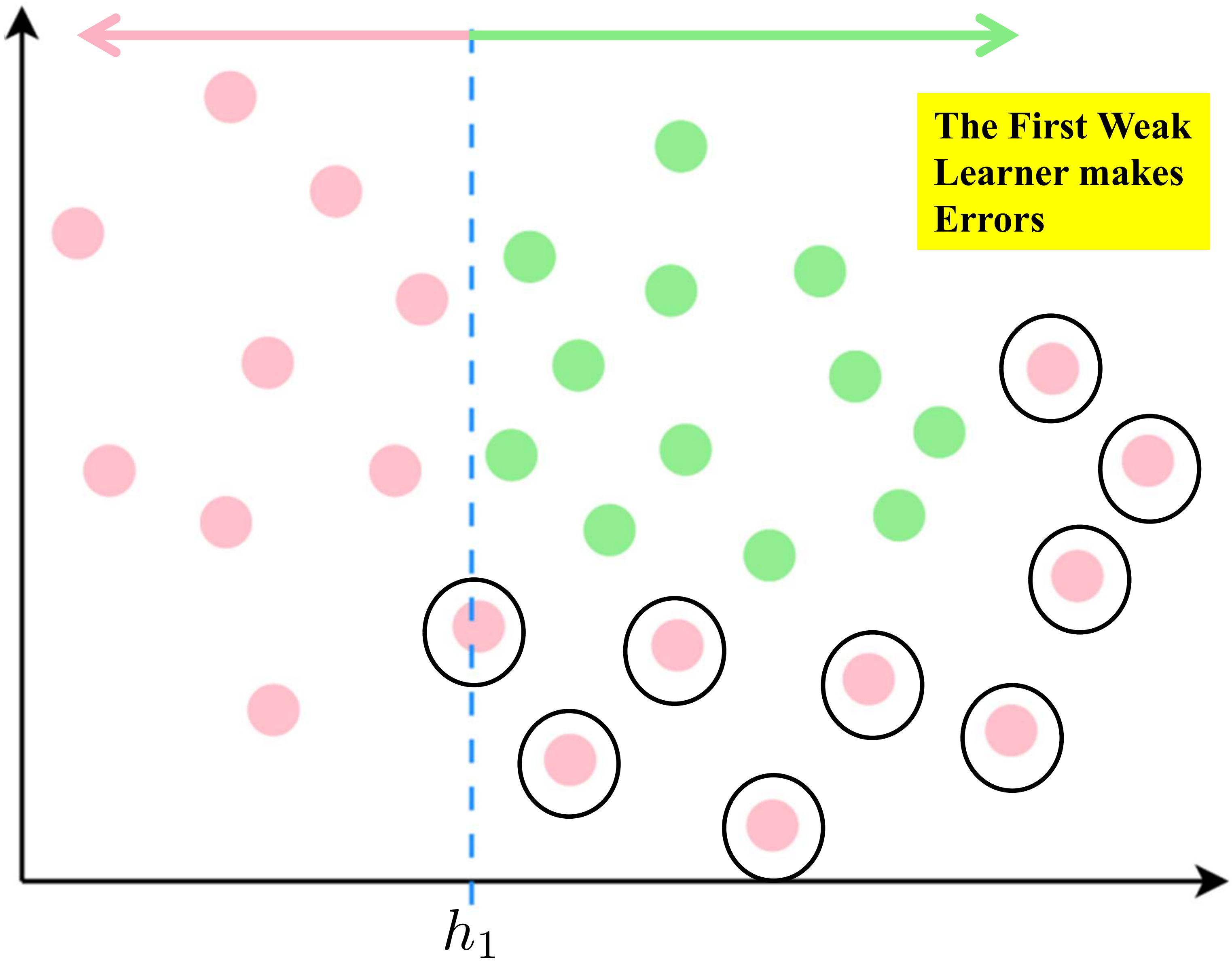
AdaBoost



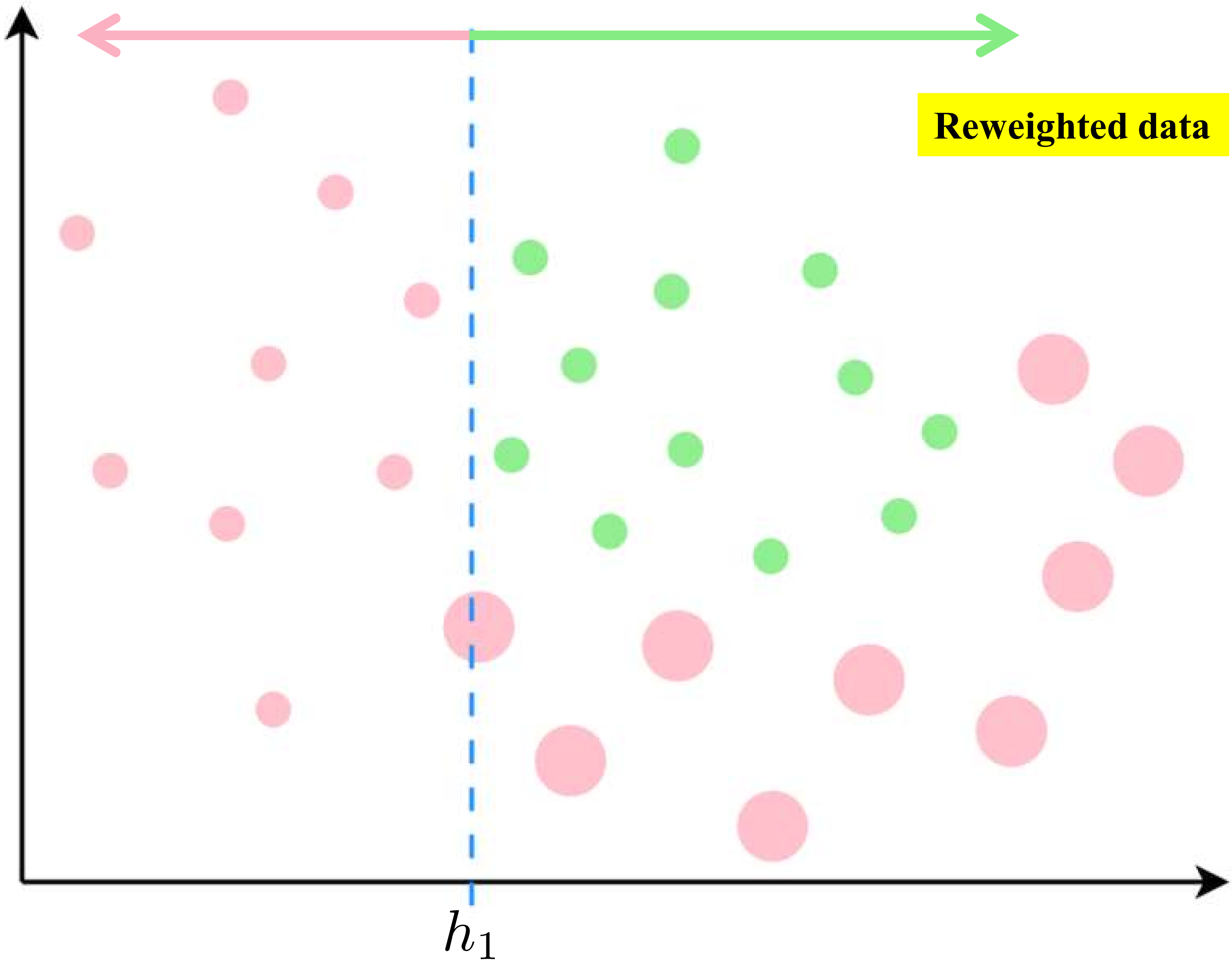
AdaBoost



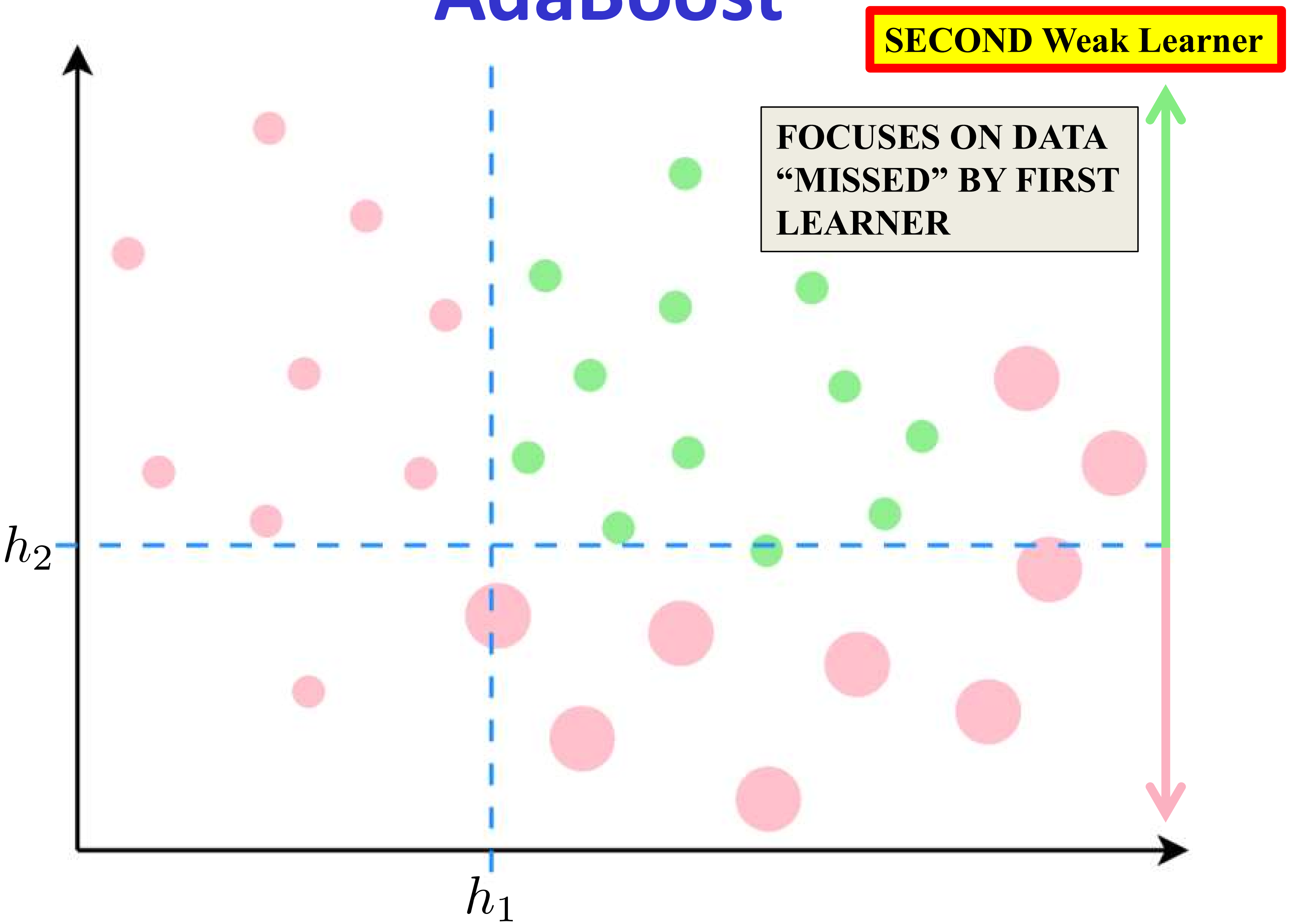
AdaBoost



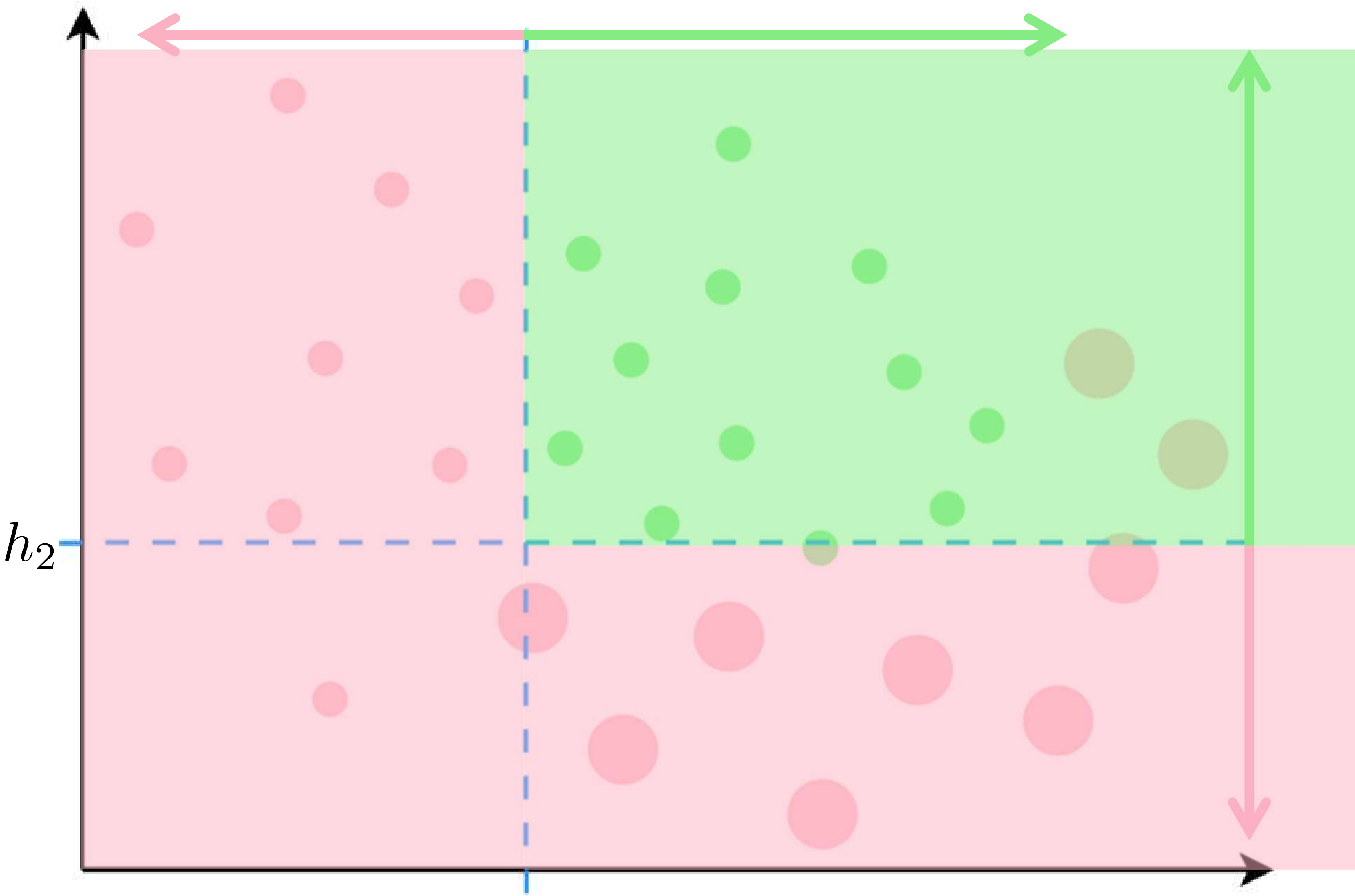
AdaBoost



AdaBoost

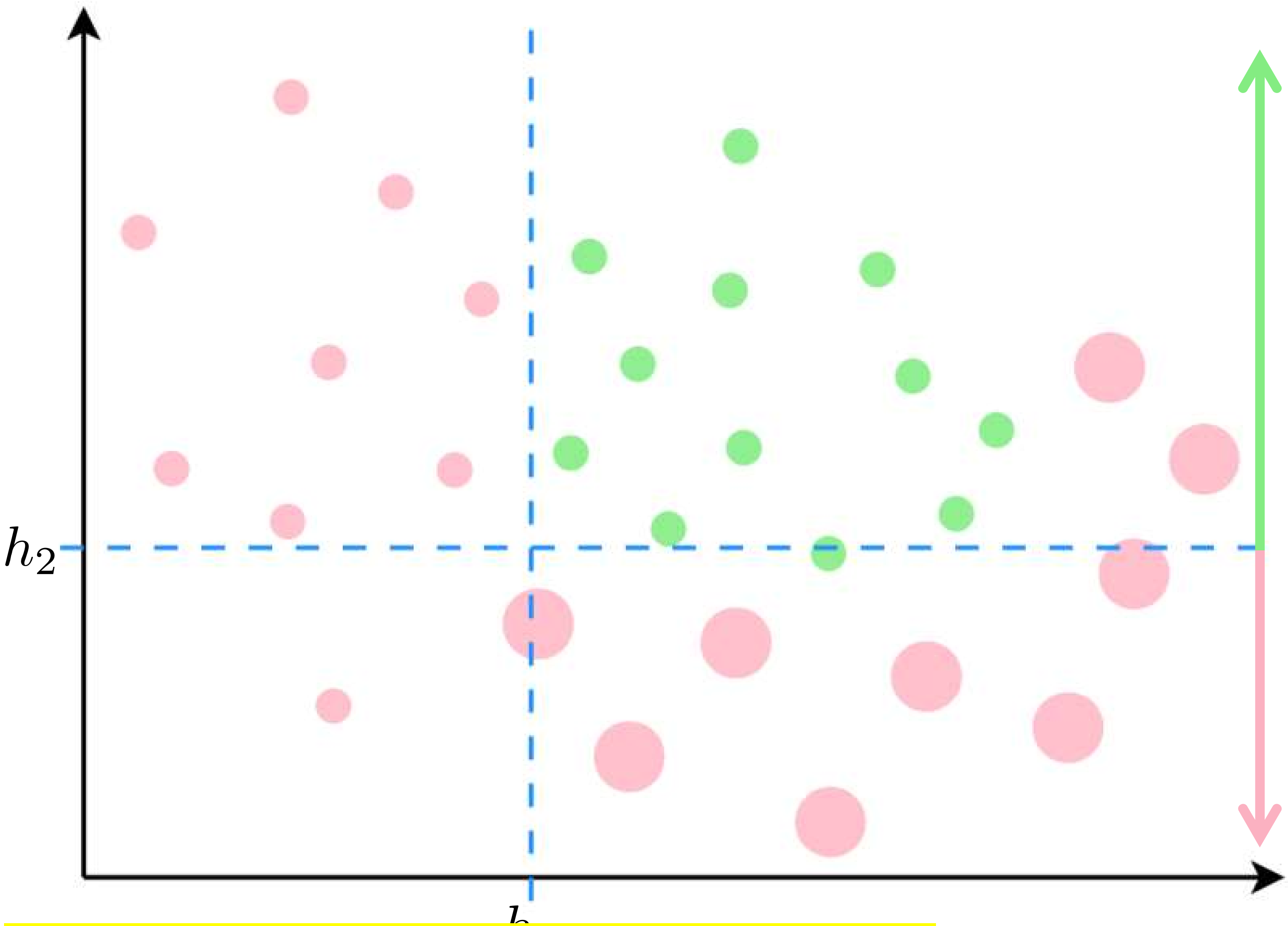

SECOND Weak Learner
FOCUSES ON DATA "MISSED" BY FIRST LEARNER
h_2
h_1

AdaBoost



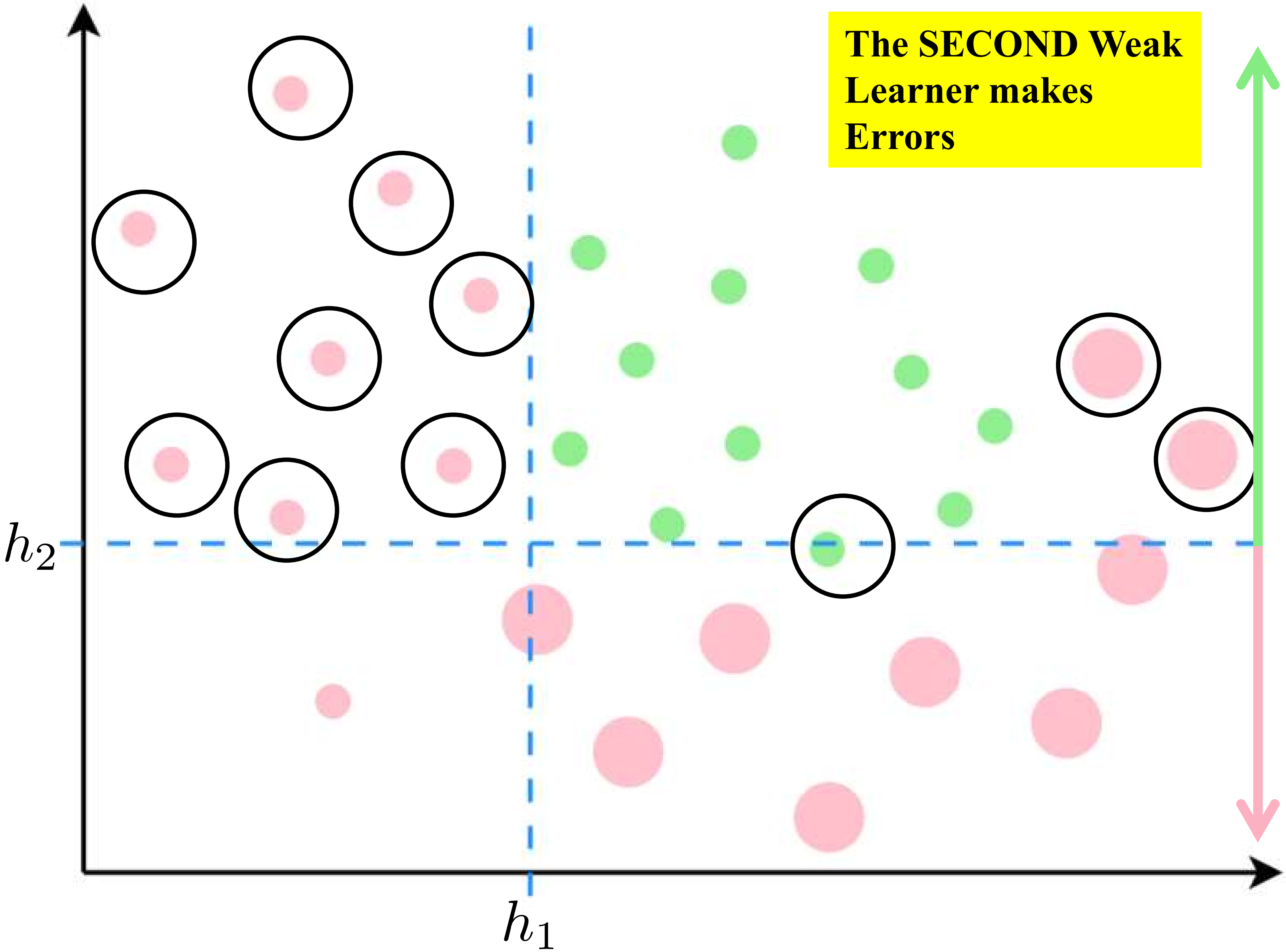
SECOND *STRONG* Learner Combines both Weak Learners

AdaBoost



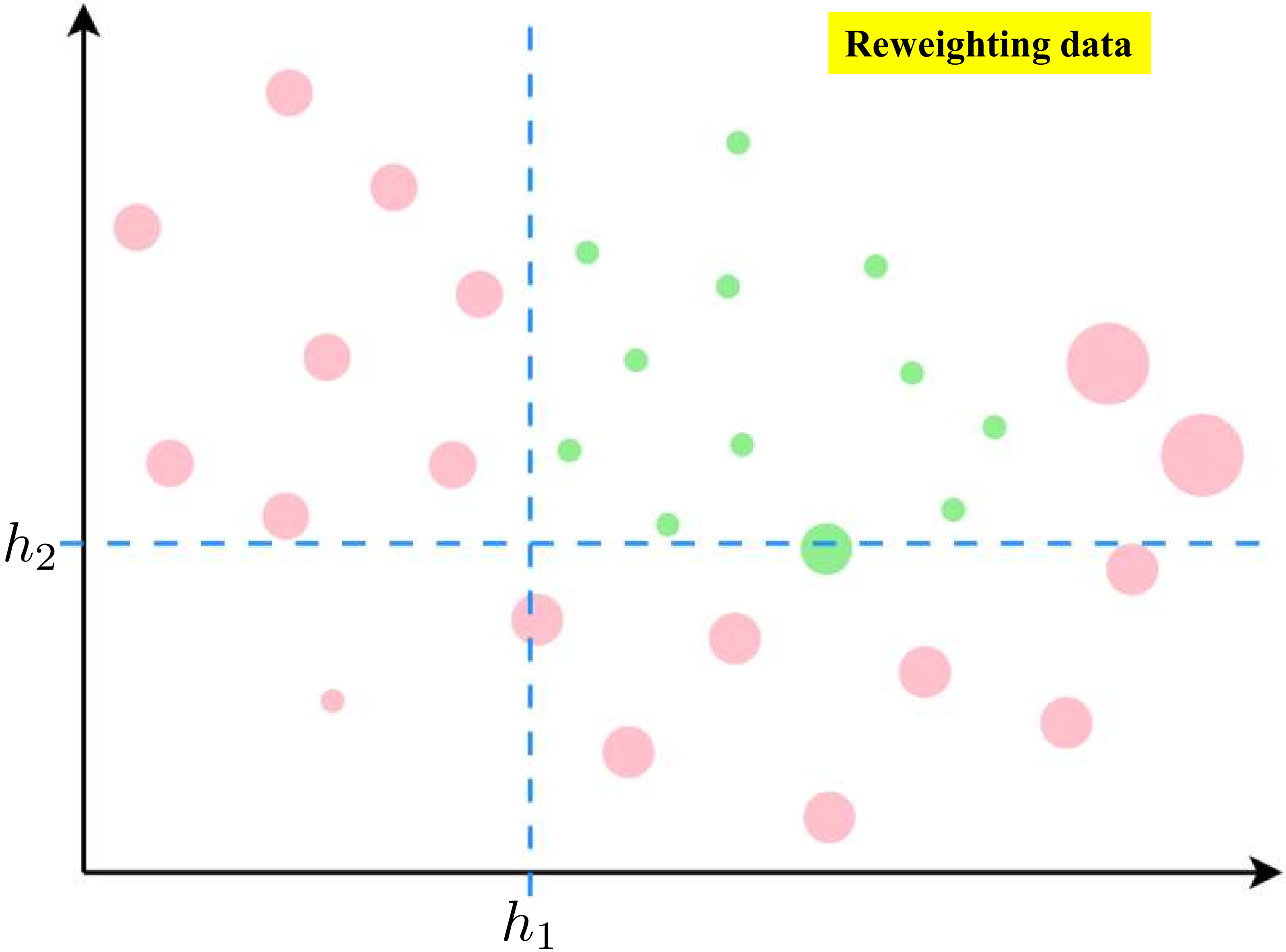
RETURNING TO THE SECOND WEAK LEARNER

AdaBoost



AdaBoost

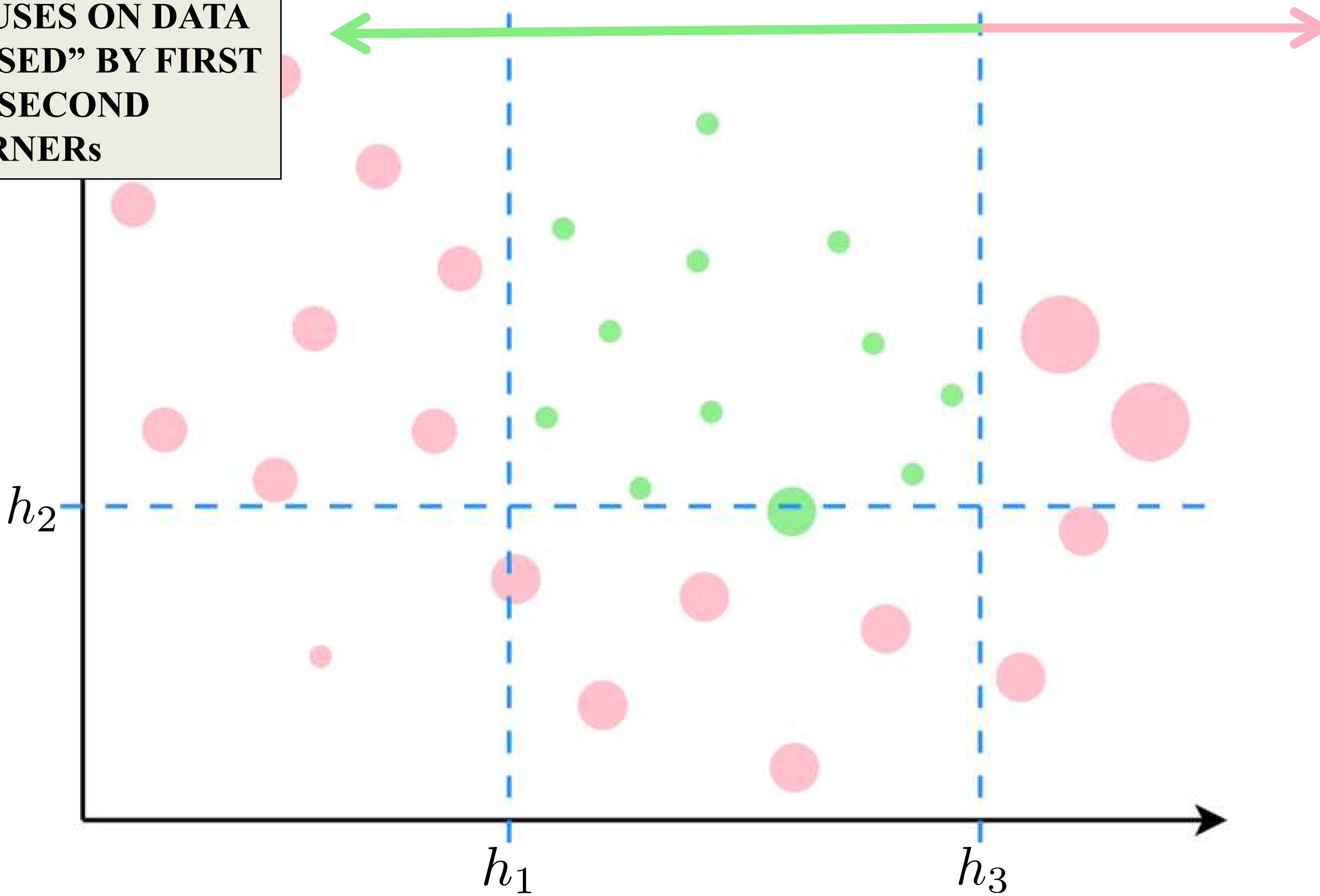
Reweighting data



THIRD Weak Learner

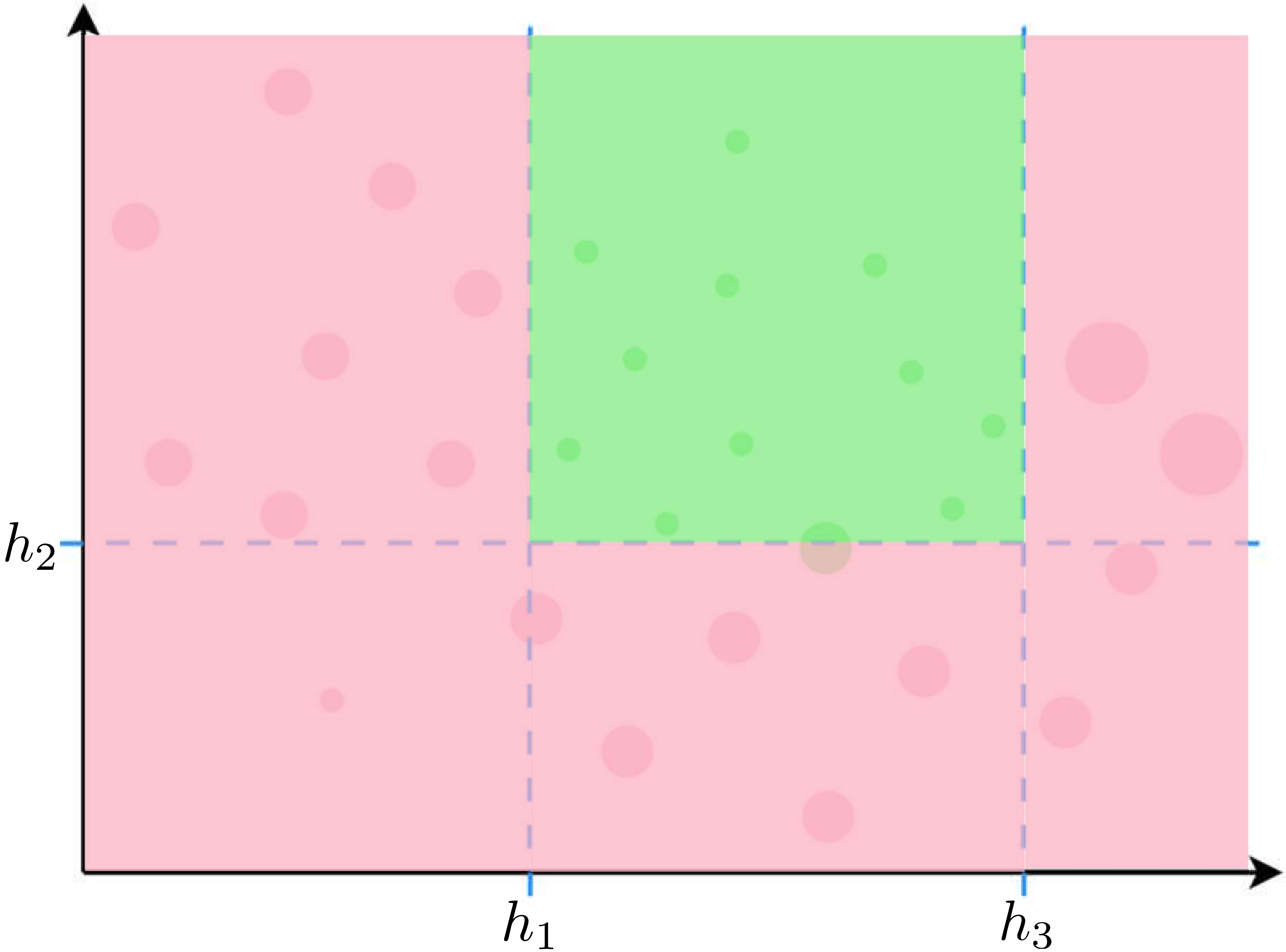
AdaBoost

FOCUSES ON DATA "MISSED" BY FIRST AND SECOND LEARNERs

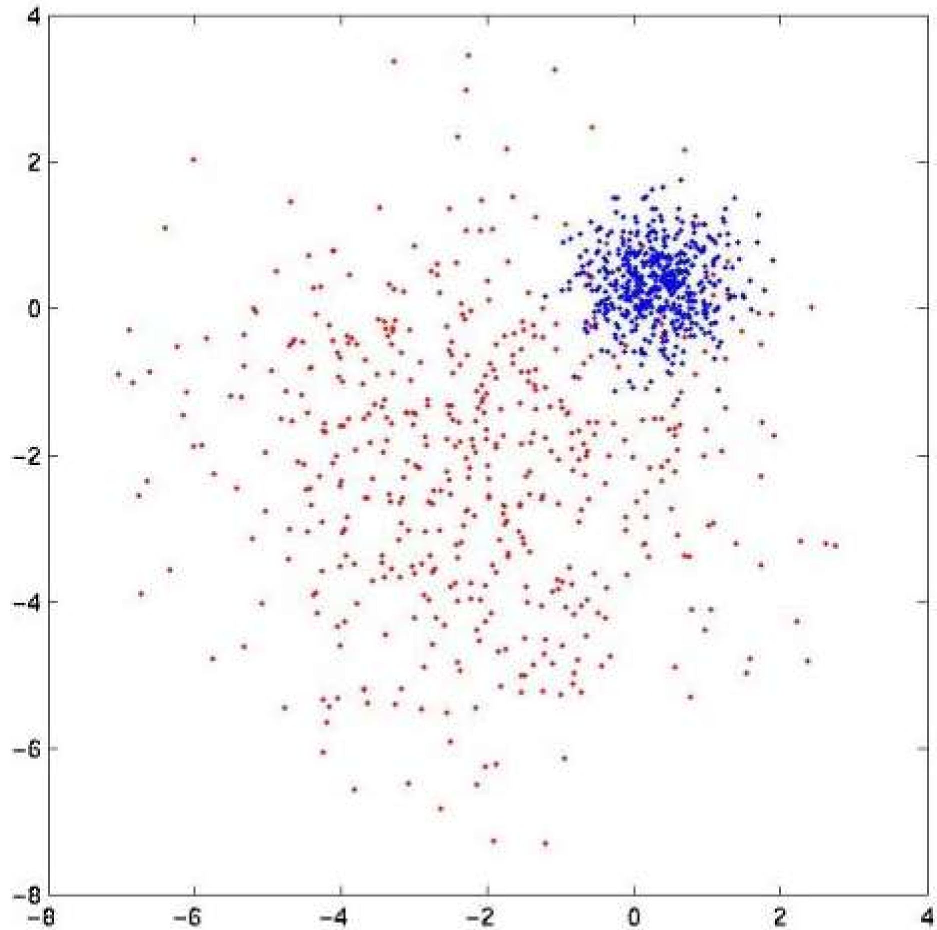


AdaBoost

THIRD STRONG Learner

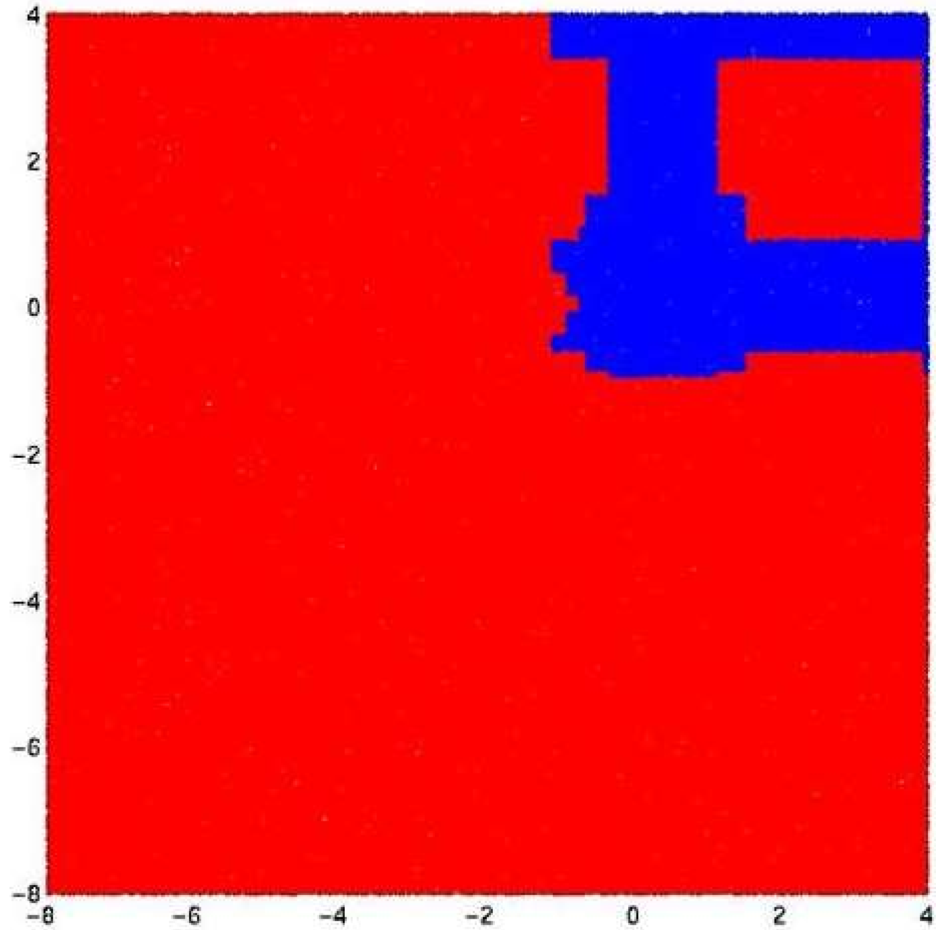


Boosting: An Example



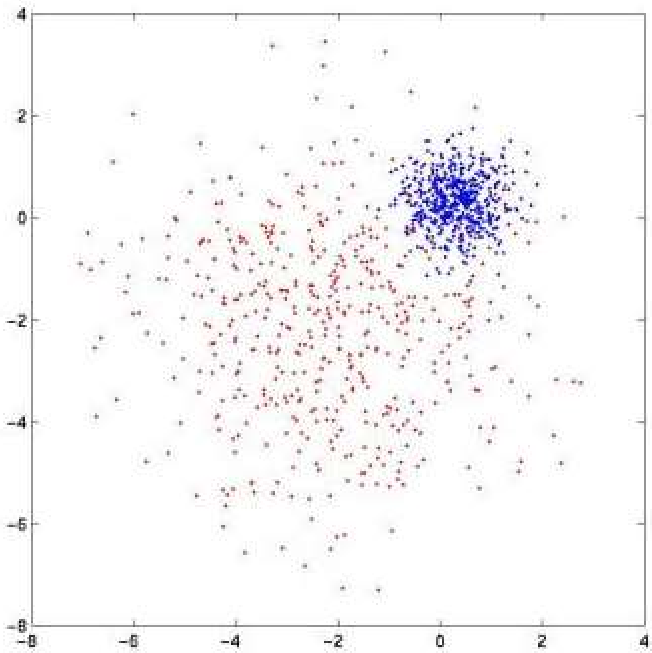
- Red dots represent training data from Red class
- Blue dots represent training data from Blue class

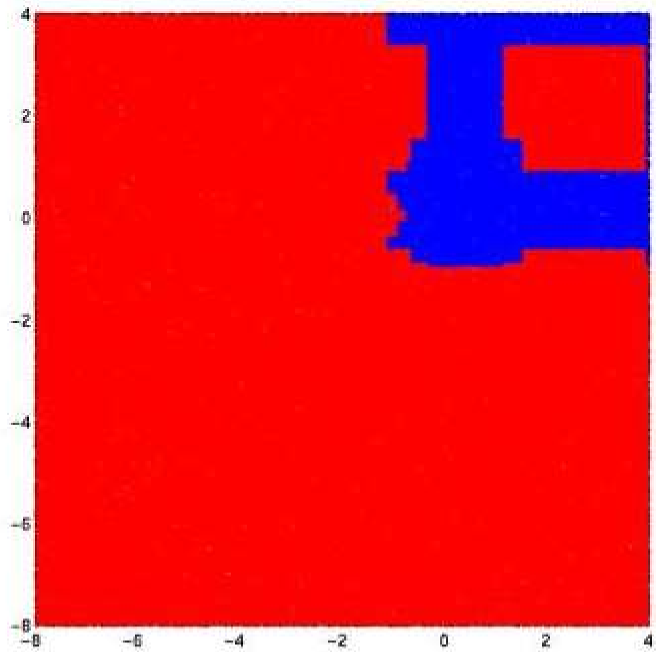
Boosting: An Example



- The final strong learner has learnt a complicated decision boundary

Boosting: An Example

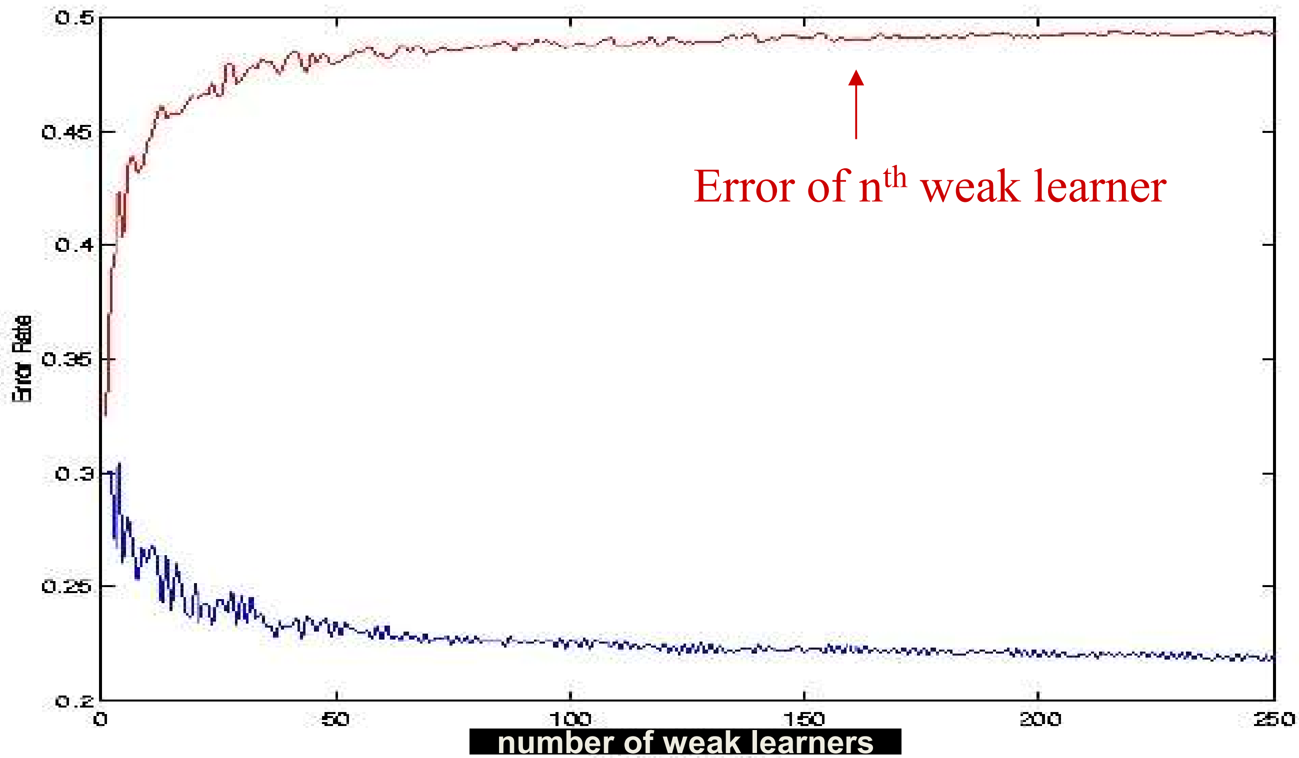




- The final strong learner has learnt a complicated decision boundary
- Decision boundaries in areas with low density of training points assumed inconsequential

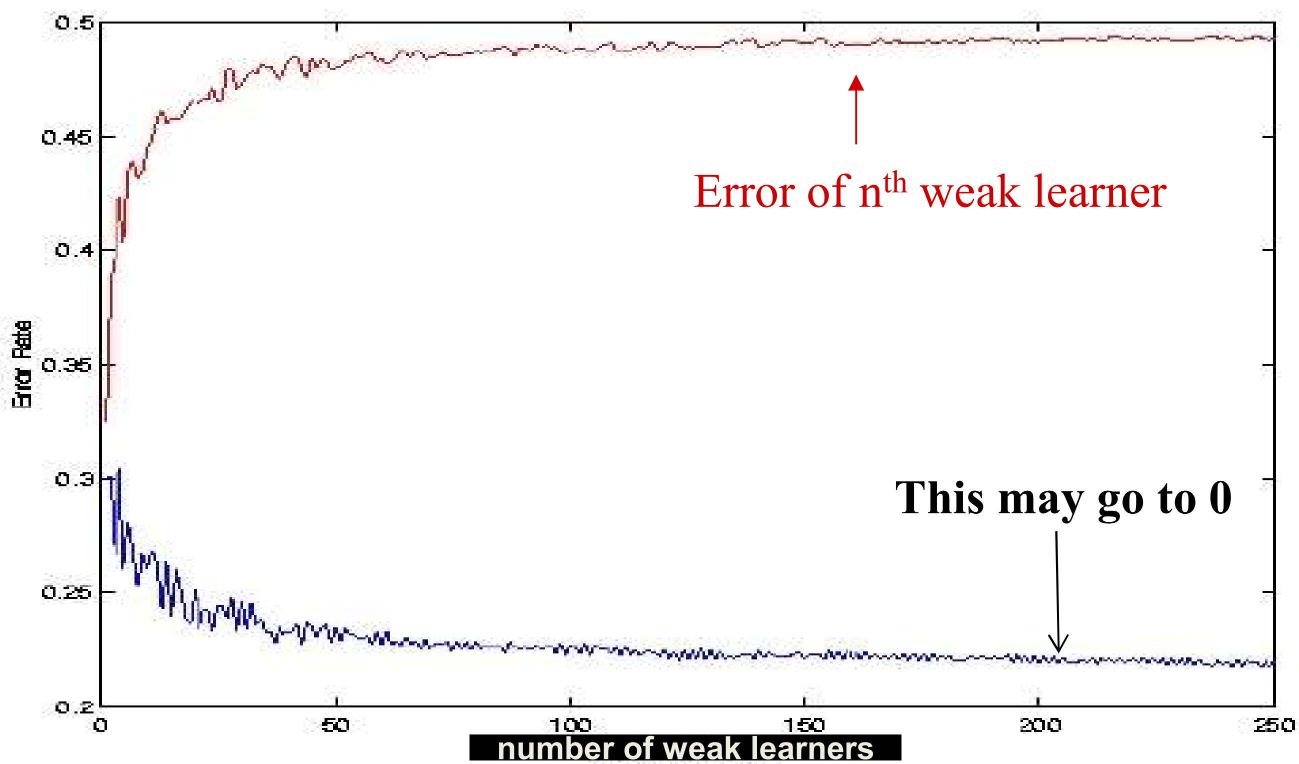
Overall Learning Pattern

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



Overfitting

- Note: Can continue to add weak learners EVEN after strong learner error goes to 0!
  - Shown to IMPROVE generalization!



AdaBoost: Summary

- No relation to Ada Lovelace
- Adaptive Boosting
- Adaptively Selects Weak Learners
- ~17.5K citations of just one paper by Freund and Schapire

Poll 2

Q1. Select the True statements

1. AdaBoost allows for the use of several simple, weak classifiers to build a strong classifier that is a weighted majority vote of the weak classifier.
2. AdaBoost is very prone to overfitting.
3. AdaBoost is a greedy algorithm.

4. AdaBoost trains all classifiers independently

Poll 2

Q1. Select the True statements

1. **AdaBoost allows for the use of several simple, weak classifiers to build a strong classifier that is a weighted majority vote of the weak classifier.**
2. AdaBoost is very prone to overfitting.
3. **AdaBoost is a greedy algorithm.**

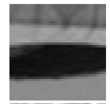
4. AdaBoost trains all classifiers independently

The ADABoost Algorithm

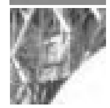
- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier h_t using distribution D_t
  - Compute total error on training data
    - $\varepsilon_t = \text{Sum}\{D_t(x_i)\, ½(1 - y_i\, h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln((1 - \varepsilon_t) / \varepsilon_t)$
  - For $i = 1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

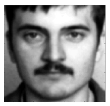
First, some example data

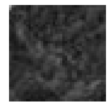
 = 0.3 E1 - 0.6 E2

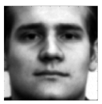
 = 0.2 E1 + 0.4 E2

 = 0.5 E1 - 0.5 E2

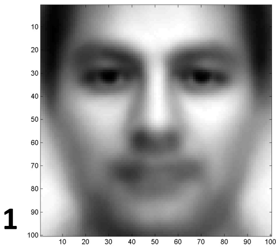
 = -0.8 E1 - 0.1 E2

 = 0.7 E1 - 0.1 E2

 = 0.4 E1 - 0.9 E2

 = 0.6 E1 - 0.4 E2

 = 0.2 E1 + 0.5 E2

E_1 

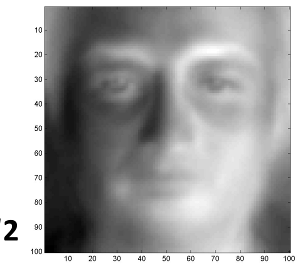
E_2 

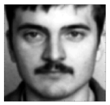
Image = a*E1 + b*E2 → a = Image.E1

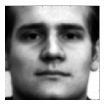
- Face detection with multiple Eigen faces
- Step 0: Derived top 2 Eigen faces from Eigen face training data
- Step 1: On a (different) set of examples, express each image as a linear combination of Eigen faces
  - Examples include both faces and non faces
  - Even the non-face images are explained in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

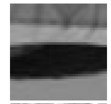
Training Data

A  = 0.3 E1 - 0.6 E2

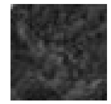
B  = 0.5 E1 - 0.5 E2

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

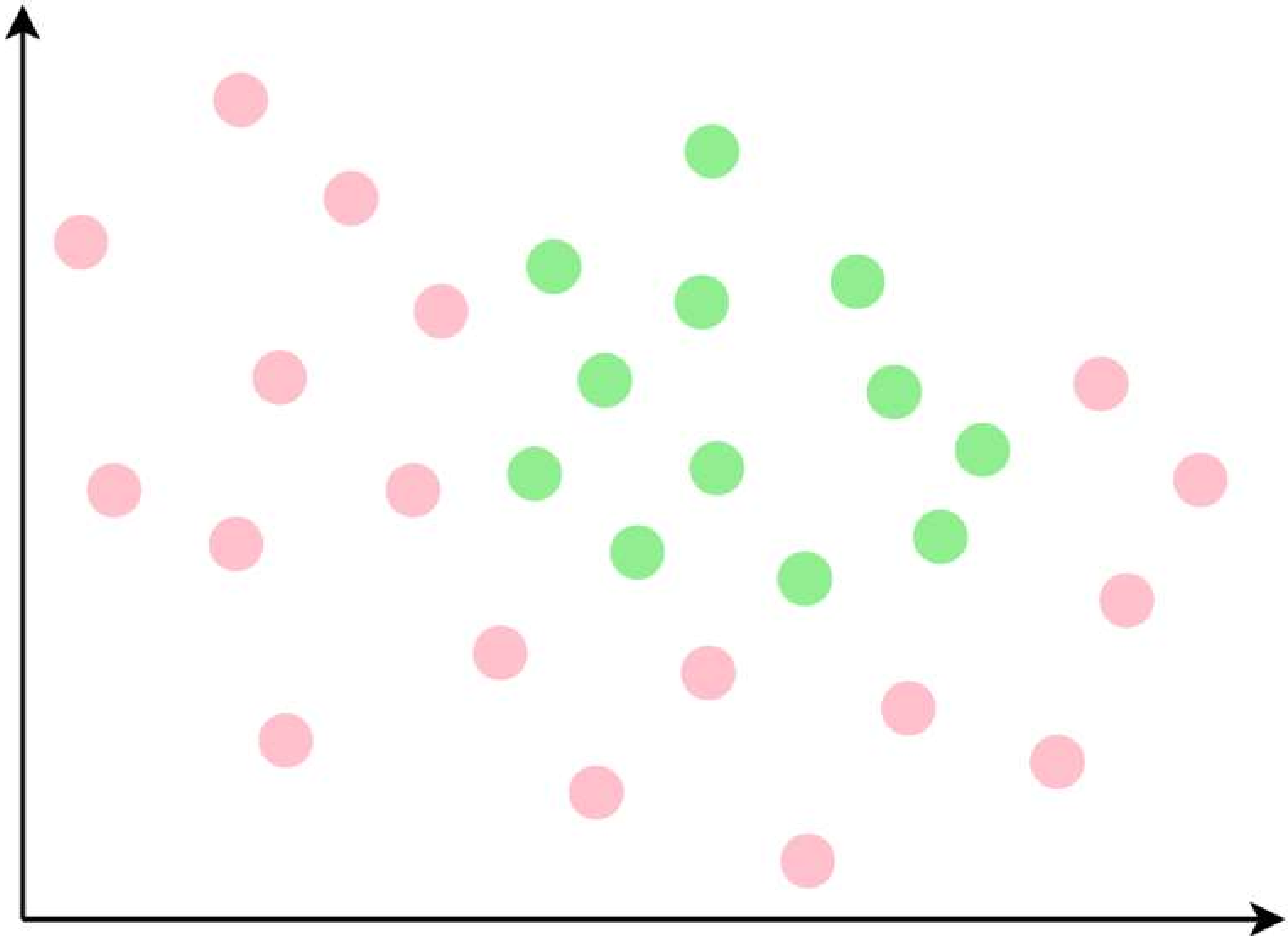
ID	E1	E2.	Class
A	0.3	-0.6	+1
B	0.5	-0.5	+1
C	0.7	-0.1	+1
D	0.6	-0.4	+1
E	0.2	0.4	-1
F	-0.8	-0.1	-1
G	0.4	-0.9	-1
H	0.2	0.5	-1

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 $\alpha_t = ½ \ln((1 - \varepsilon_t) / \varepsilon_t)$
  - 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
  - $H(x) = \text{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

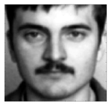
Initialize $D_1(x_i) = 1/N$

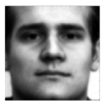


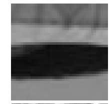
Training Data

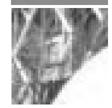

$= 0.3\,E1 - 0.6\,E2$

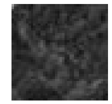

$= 0.5\,E1 - 0.5\,E2$


$= 0.7\,E1 - 0.1\,E2$


$= 0.6\,E1 - 0.4\,E2$


$= 0.2\,E1 + 0.4\,E2$


$= -0.8\,E1 - 0.1\,E2$


$= 0.4\,E1 - 0.9\,E2$


$= 0.2\,E1 + 0.5\,E2$

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"

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

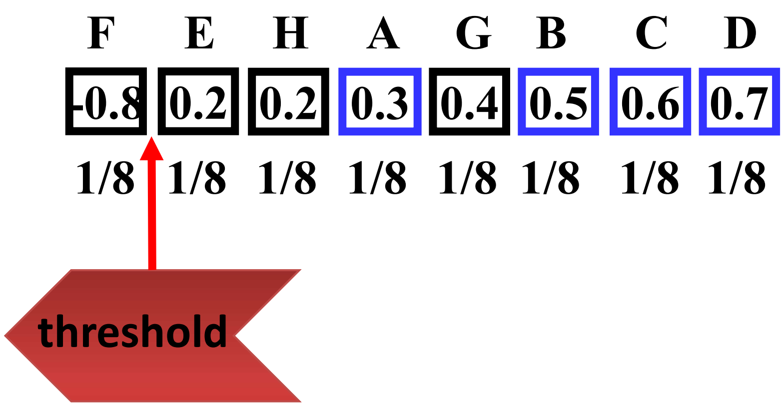
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

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"

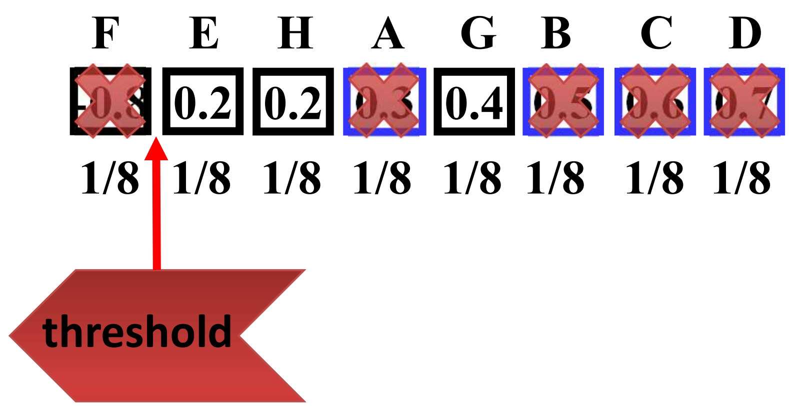


Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
    face = true

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"



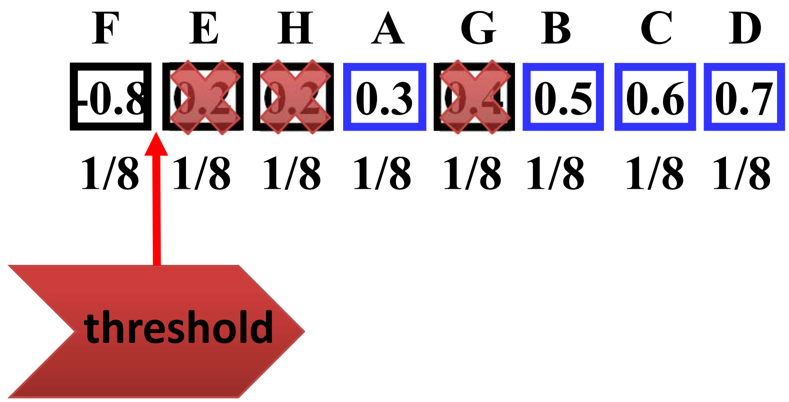
Classifier based on E1:
if (sign*wt(E1) > thresh) > 0)
face = true

sign = +1 or -1

Sign = -1, error = 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 = 3/8

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"

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

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"

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
1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

threshold

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

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 Best 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

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

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"

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

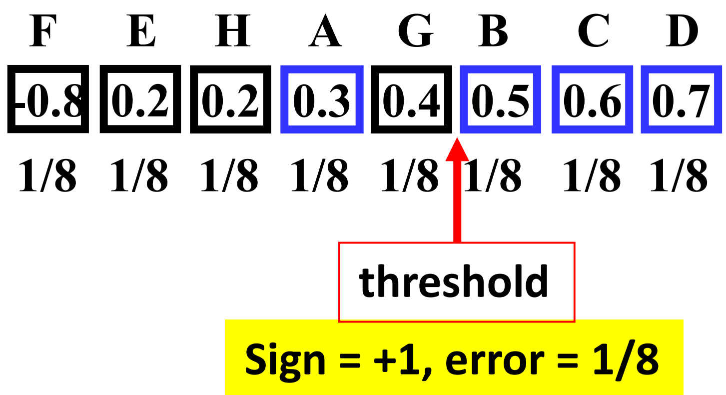
Sign = -1, error = 2/8

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"

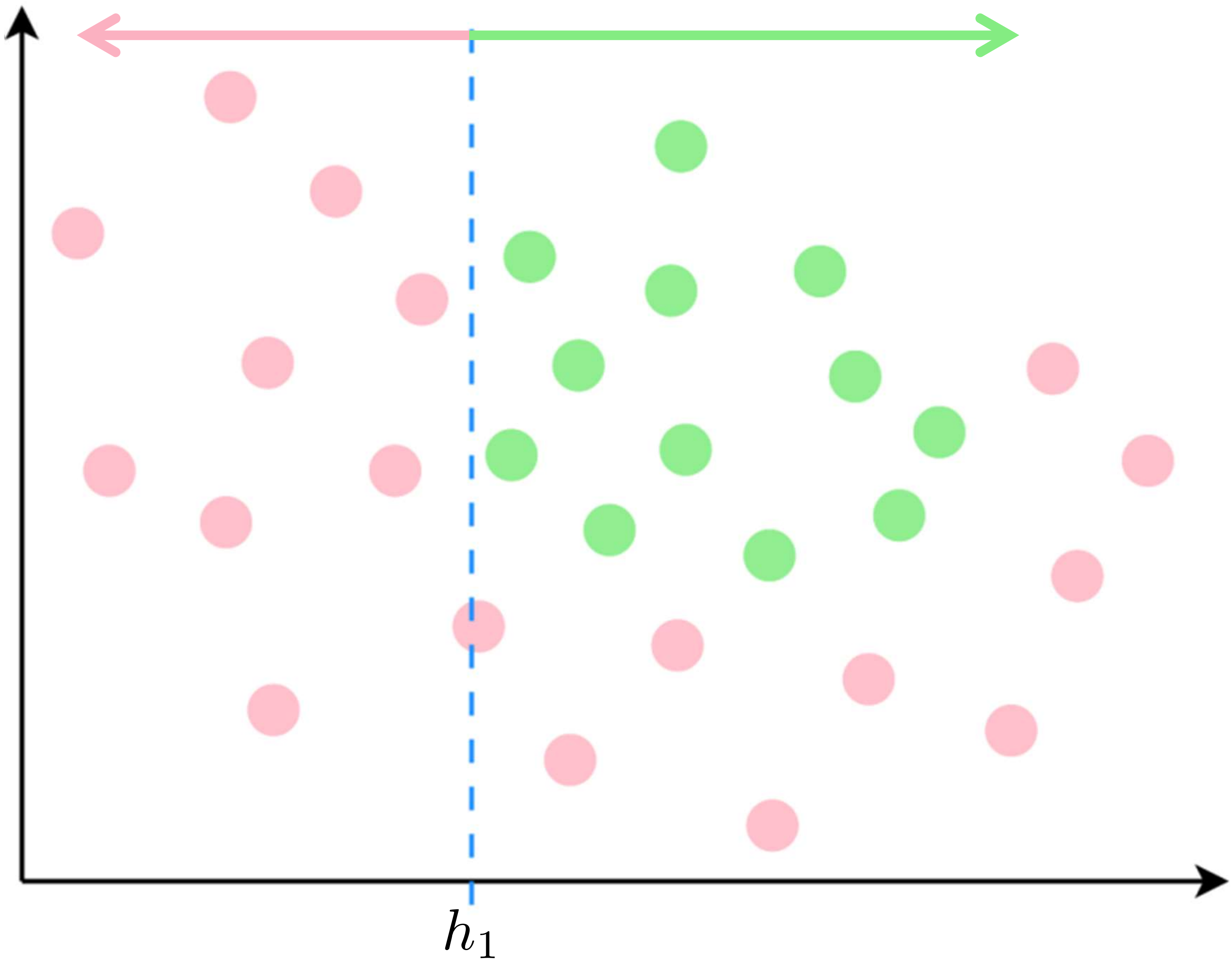


The Best overall classifier based on a single feature is based on E1

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

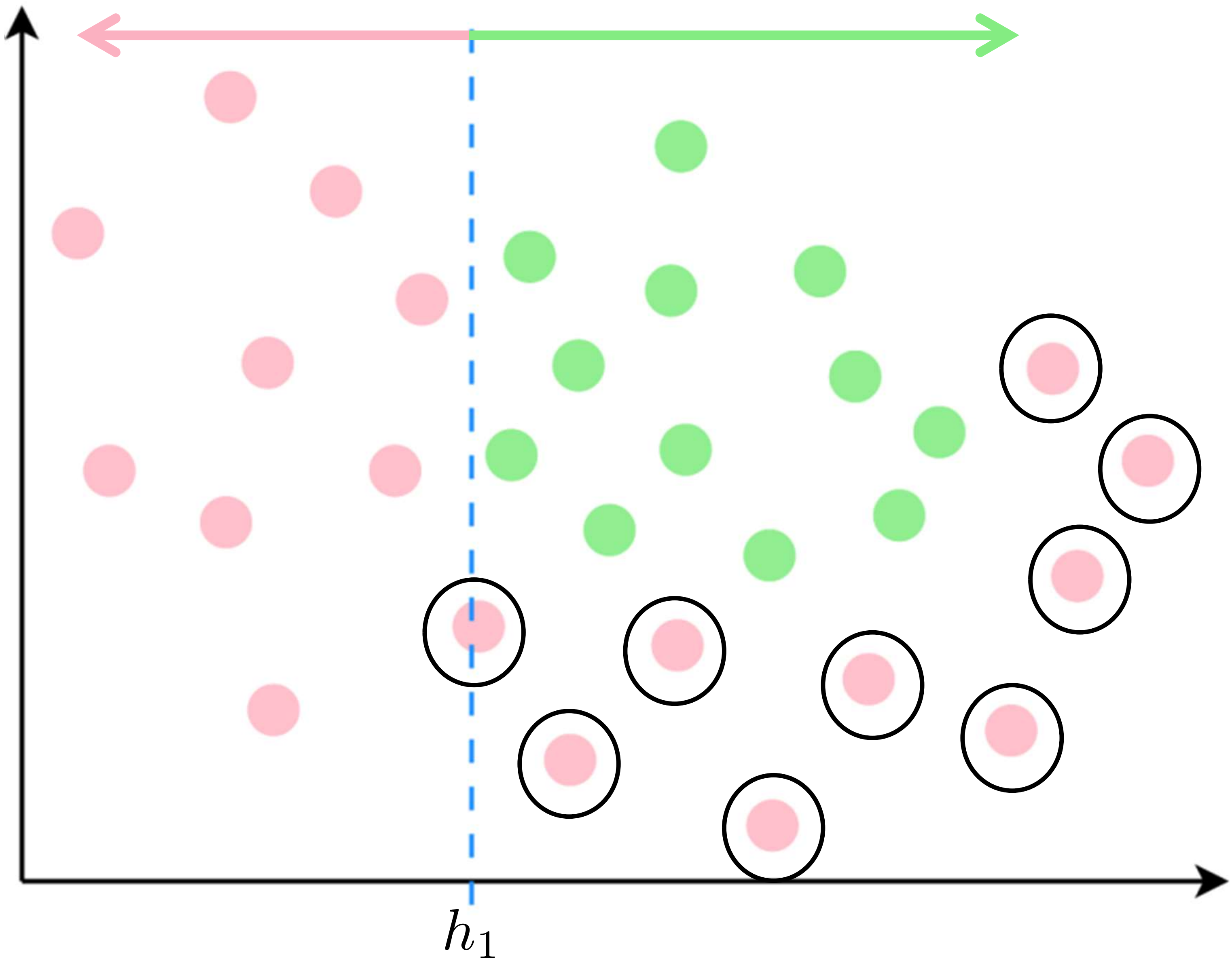
The Best "Stump"



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 $\alpha_t = ½ \ln\left((1 - \varepsilon_t) / \varepsilon_t\right)$
  - 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
  - $H(x) = \text{sign}(\Sigma_t\, \alpha_t\, h_t(x))$

Poll 3

- The classifier weight assigns 0 weight to a perfectly random classifier (T/F)
  - T
  - F
- We assign infinite weight for a perfectly correct classifier (T/F)
  - T
  - F
- We assign 0 weight for a classifier that is always wrong (T/F)
  - T
  - F (We assign $-\infty$ weight)

Poll 3

- The classifier weight assigns 0 weight to a perfectly random classifier (T/F)
  - **T**
  - F
- We assign infinite weight for a perfectly correct classifier (T/F)
  - **T**
  - F
- We assign 0 weight for a classifier that is always wrong (T/F)
  - T
  - **F (We assign $-\infty$ weight)**

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
    - $\varepsilon_t = \text{Average}\,\{½\,(1 - y_i\, h_t(x_i))\}$
  - 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

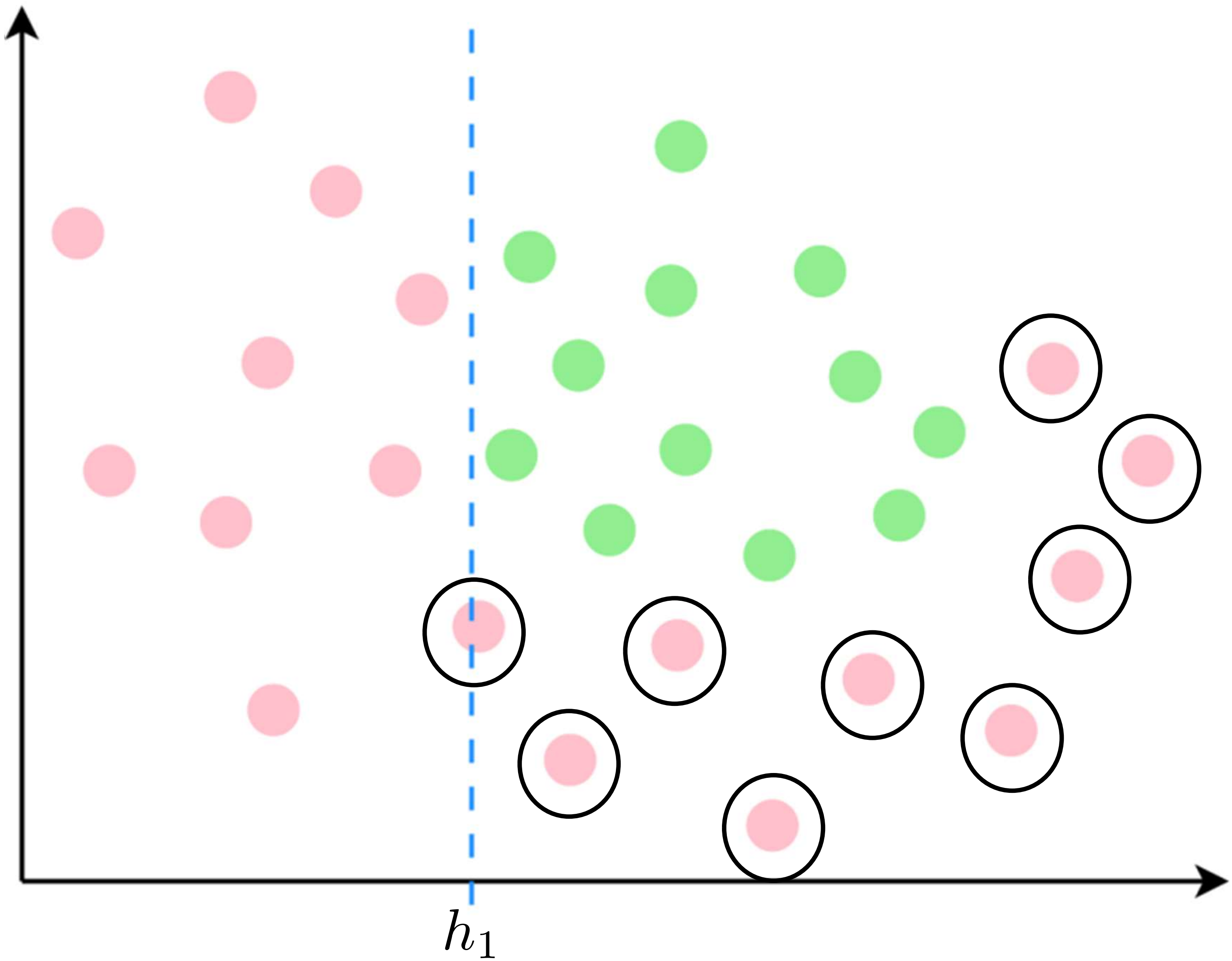
$$D_{t+1}(x_i) = D_t(x_i)\ \exp(-\ \alpha_t\, y_i\, h_t\,(x_i))$$

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

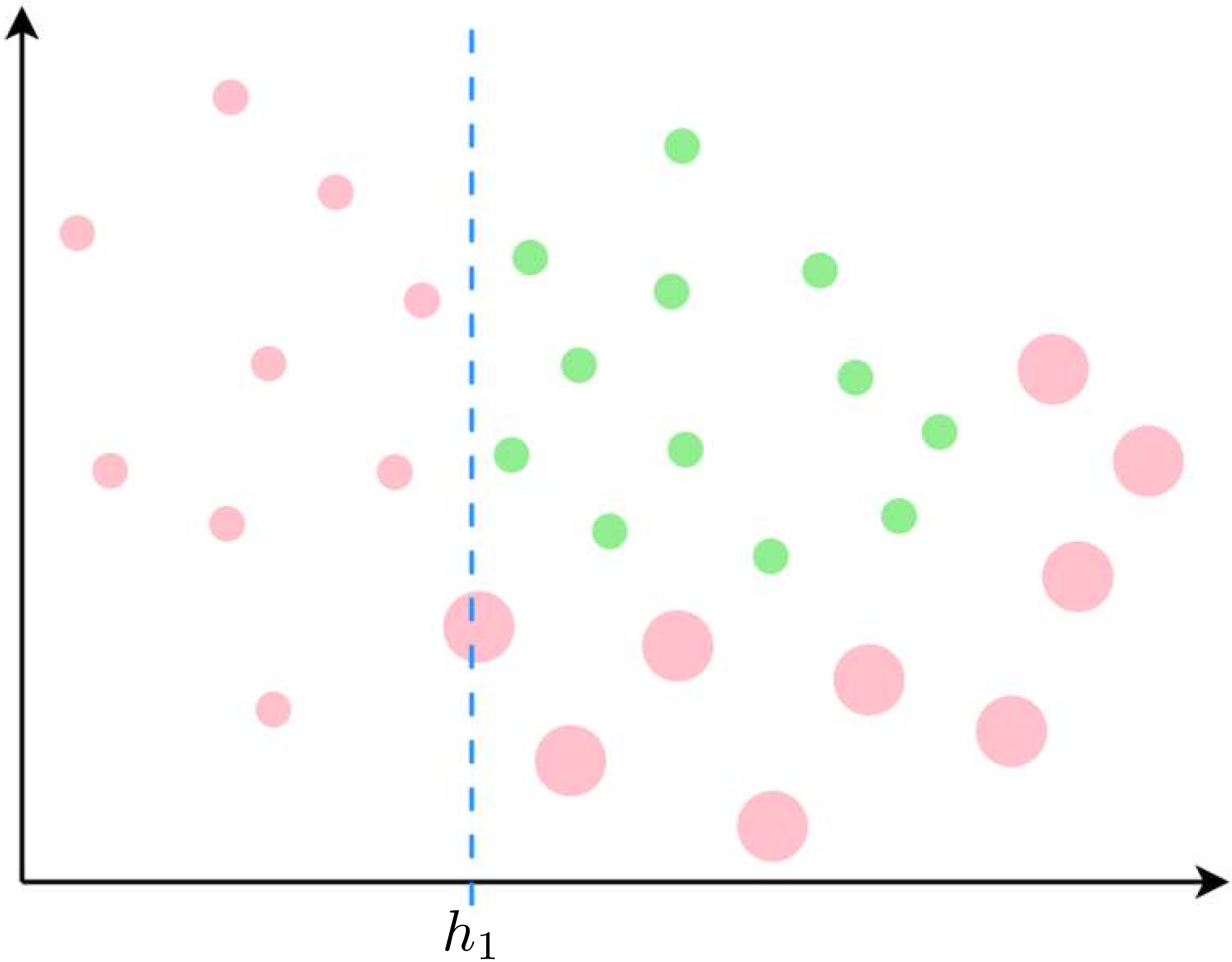
ID	E1	E2.	Class	Weight	Weight
A	0.3	-0.6	+1	1/8 * 2.63	0.33
B	0.5	-0.5	+1	1/8 * 0.38	0.05
C	0.7	-0.1	+1	1/8 * 0.38	0.05
D	0.6	-0.4	+1	1/8 * 0.38	0.05
E	0.2	0.4	-1	1/8 * 0.38	0.05
F	-0.8	0.1	-1	1/8 * 0.38	0.05
G	0.4	-0.9	-1	1/8 * 0.38	0.05
H	0.2	0.5	-1	1/8 * 0.38	0.05

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

AdaBoost



AdaBoost



Poll 4

- If the classifier is perfectly random, we do not change the weights of the instances (T/F)
  – T
  – F

- If the classifier is perfectly correct, we scale down the importance of correctly classified instances to 0 (T/F)
  – T
  – F

- For a (nearly) perfect classifier we scale down the importance of incorrectly classified instances to 0
  – T
  – F (We scale them *up* to infinity – they are inifintely hard to fix)

Poll 4

- If the classifier is perfectly random, we do not change the weights of the instances (T/F)
  - **T**
  - F

- If the classifier is perfectly correct, we scale down the importance of correctly classified instances to 0 (T/F)
  - **T**
  - F

- For a (nearly) perfect classifier we scale down the importance of incorrectly classified instances to 0
  - T
  - **F (We scale them *up* to infinity – they are inifintely hard to fix)**

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

$$D' = D \,/\, sum(D)$$

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

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

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

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

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

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

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

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

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

threshold

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

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

threshold

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

threshold

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

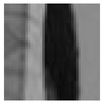
- In this example both of our first two classifiers were based on E1
  - Additional classifiers may switch to E2
- In general, the reweighting of the data will result in a different feature being picked for each classifier
- This also automatically gives us a *feature selection* strategy
  - In this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

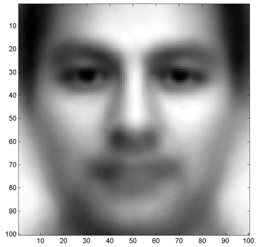
- We can *continue* to add classifiers even after we get 100% classification of the training data
  - Because the weights of the data keep changing
  - Adding new classifiers beyond this point is often a good thing to do

ADA Boost

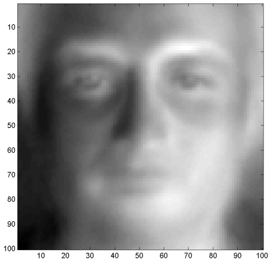


= 0.4 E1 - 0.4 E2

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

- Next class…