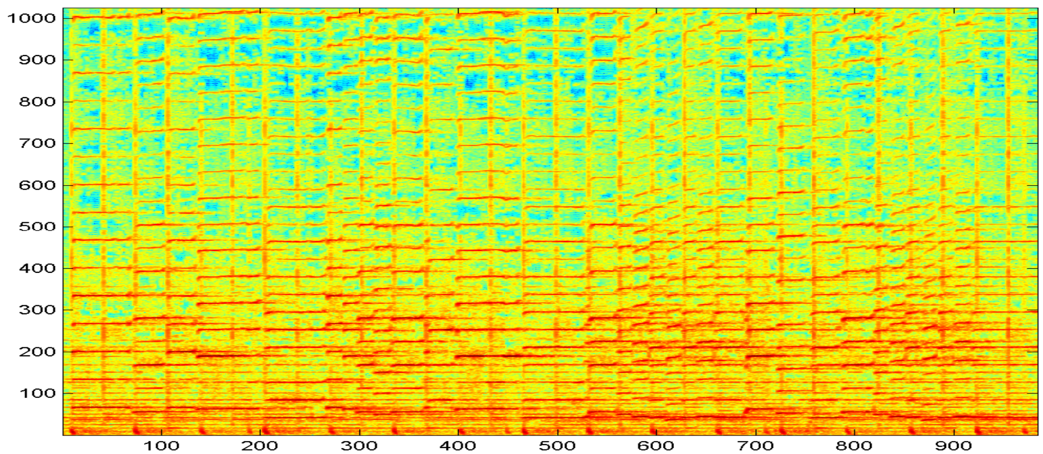
Representing Images
Detecting faces in images

Class 5. 15 Sep 2009

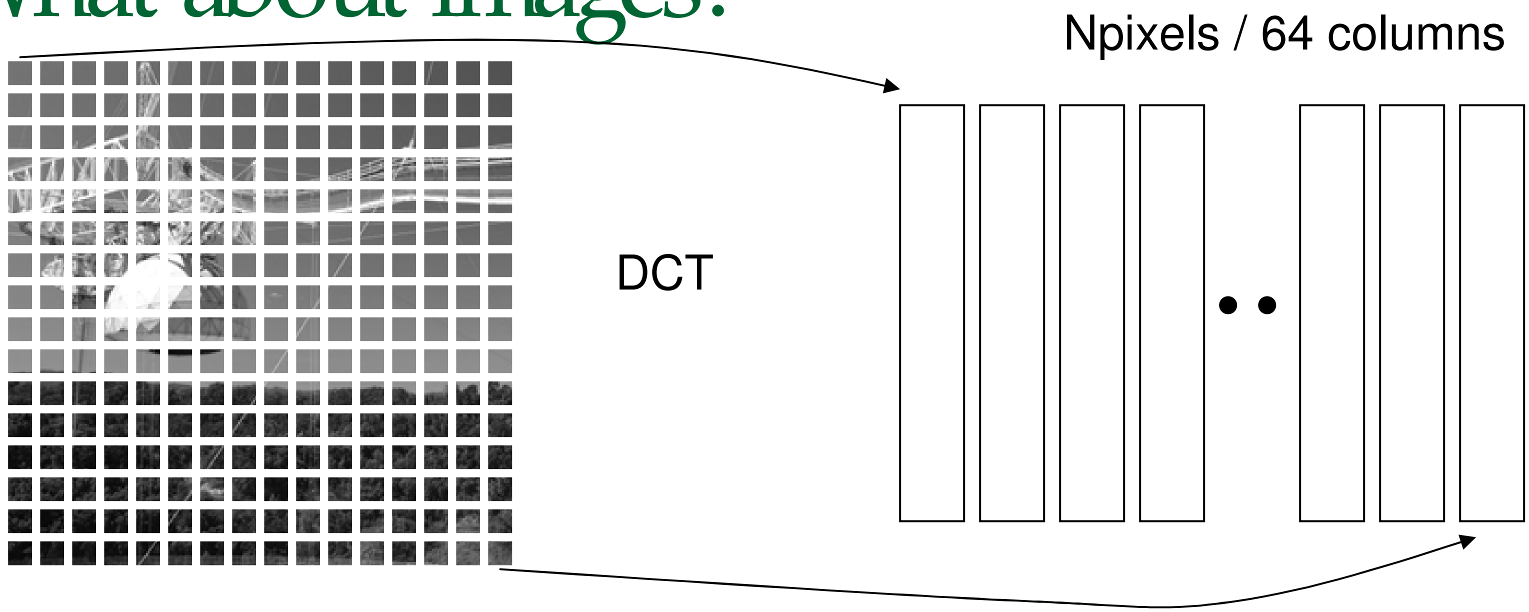
Instructor: Bhiksha Raj

Last Class: Representing Audio



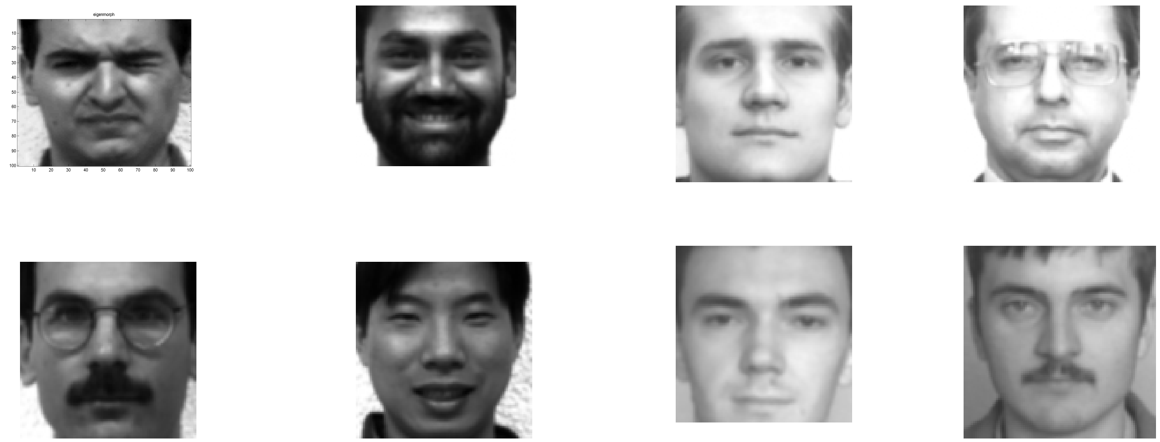
- Basic DFT
- Computing a Spectrogram
- Computing additional features from a spectrogram

What about images?



- DCT of small segments
  - 8x8
  - Each image becomes a matrix of DCT vectors
- DCT of the image
- Haar transform (checkerboard)
- ***Or data-driven representations..***

Returning to Eigen Computation



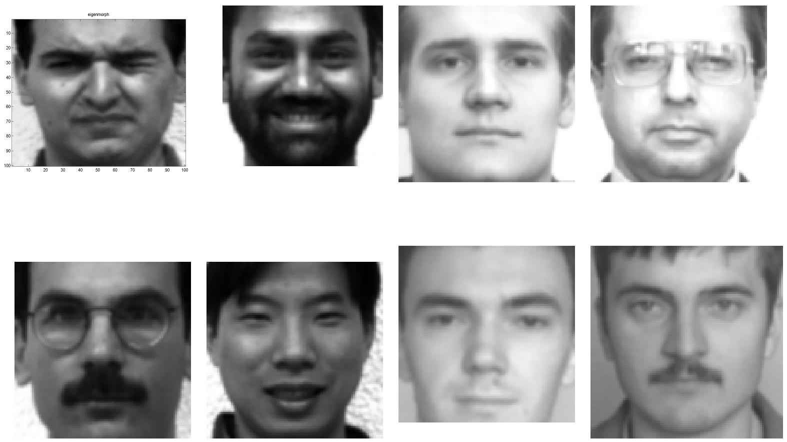
- A collection of faces
  - All normalized to 100x100 pixels
- What is common among all of them?
  - Do we have a common descriptor?

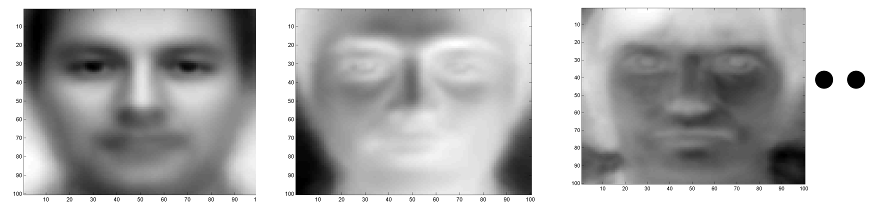
A least squares typical face

The typical face

- Can we do better than a blank screen to find the most common portion of faces?
  - The first checkerboard; the zeroth frequency component..
- Assumption: There is a "typical" face that captures most of what is common to all faces
  - Every face can be represented by a scaled version of a typical face
    - What is this face?
- Approximate **every** face f as $f = w_f V$
- Estimate V to minimize the squared error
  - How?
  - What is V?

A collection of least squares typical faces

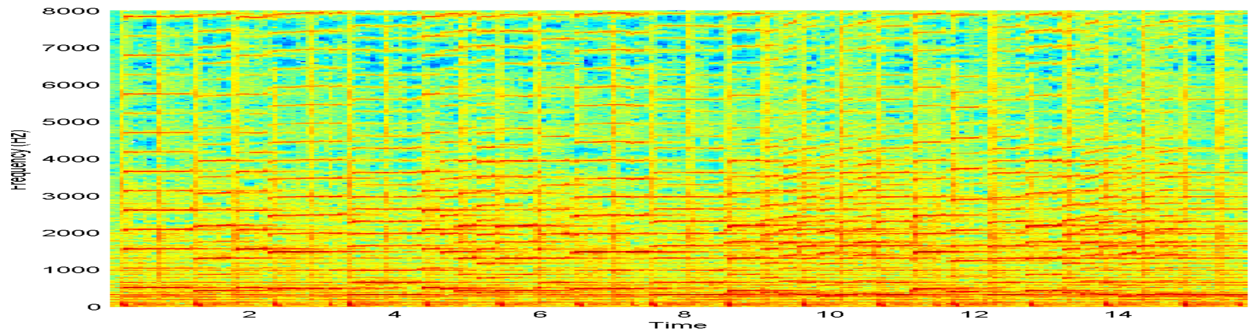




- Assumption: There are a set of K "typical" faces that captures most of all faces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + w_{f,3} V_3 + .. + w_{f,k} V_k$
  - V_2 is used to "correct" errors resulting from using only V_1
    - So the total energy in $w_{f,2}$ ($\Sigma\, w_{f,2}^2$) must be lesser than the total energy in $w_{f,1}$ ($\Sigma\, w_{f,1}^2$)
  - V_3 corrects errors remaining after correction with V_2
    - The total energy in $w_{f,3}$ must be lesser than that even in $w_{f,2}$
  - And so on..
  - $V = [V_1\ V_2\ V_3]$
- Estimate V to minimize the squared error
  - How?
  - What is V?

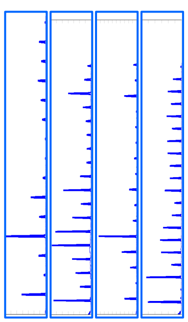
A recollection

M =



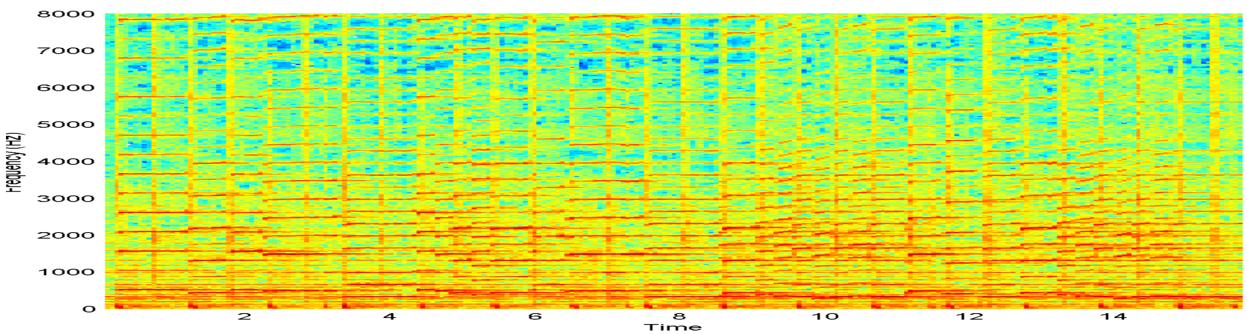
V=PINV(W)*M

W =

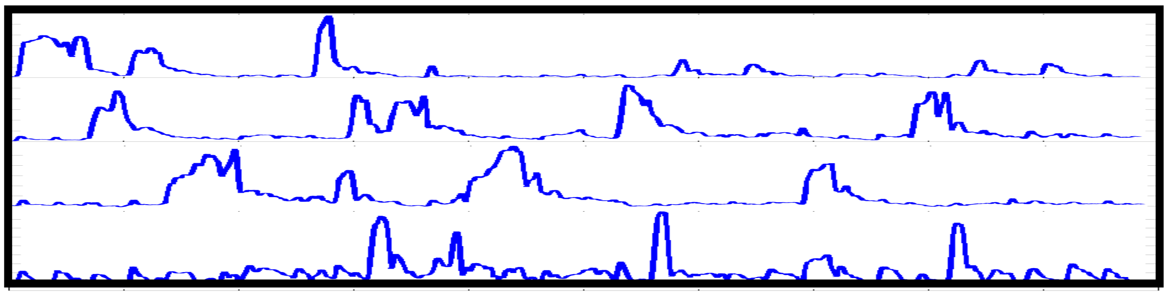


How about the other way?

M =



V =



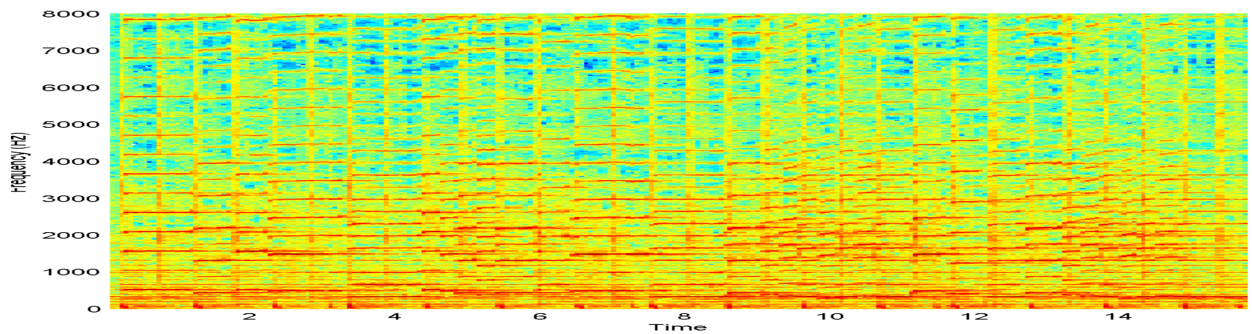
W = ?

U = ?

- W = M * Pinv(V)

How about the other way?

M =



V = ?

W = ?

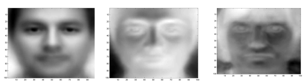
U = ?

- W V \approx = M

Eigen Faces!

M = Data Matrix

V

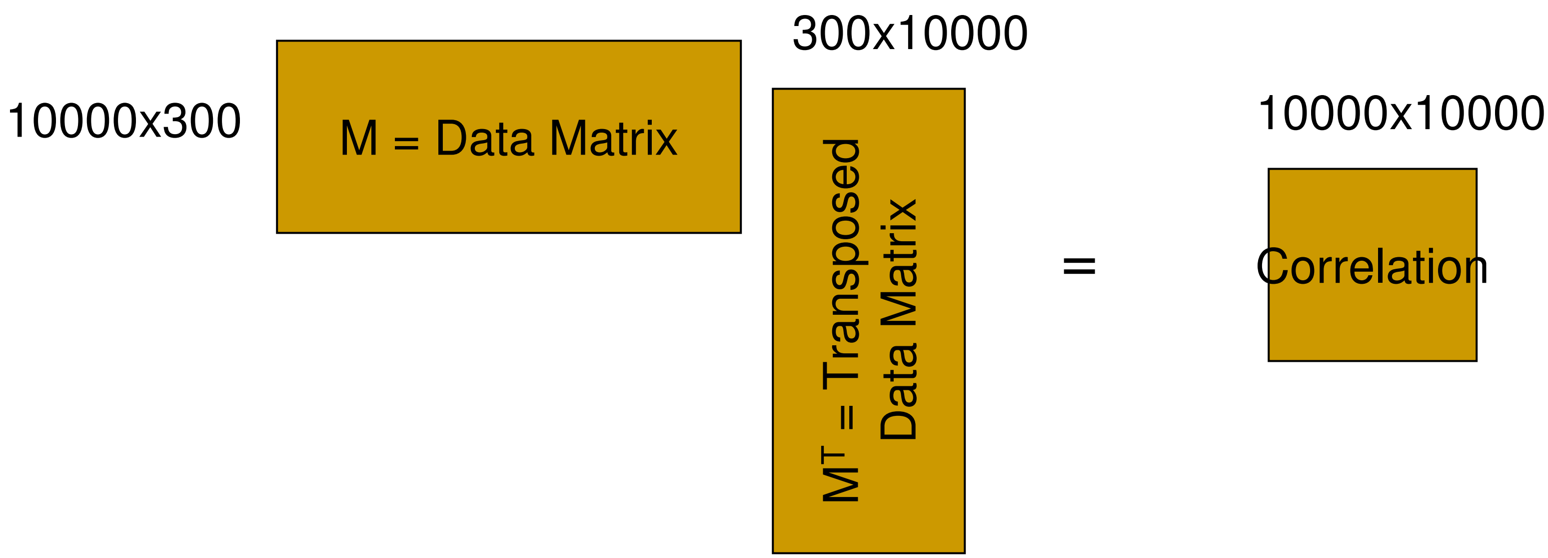


W

U = Approximation

- Here W, V and U are ALL unknown and must be determined
  - Such that the squared error between U and M is minimum
- Eigen analysis allows you to find W and V such that U = WV has the least squared error with respect to the original data M
- If the original data are a collection of faces, the columns of W are *eigen faces.*

Eigen faces



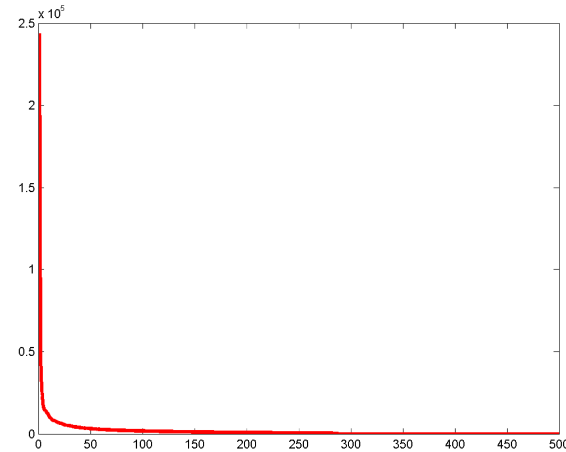
- Lay all faces side by side in vector form to form a matrix
  - In my example: 300 faces. So the matrix is 10000 x 300
- Multiply the matrix by its transpose
  - The correlation matrix is 10000x10000

Eigen faces

[U,S] = eig(correlation)

$$S = \begin{bmatrix} \lambda_1 & . & 0 & . & 0 \\ 0 & \lambda_2 & 0 & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & 0 & . & \lambda_{10000} \end{bmatrix}$$

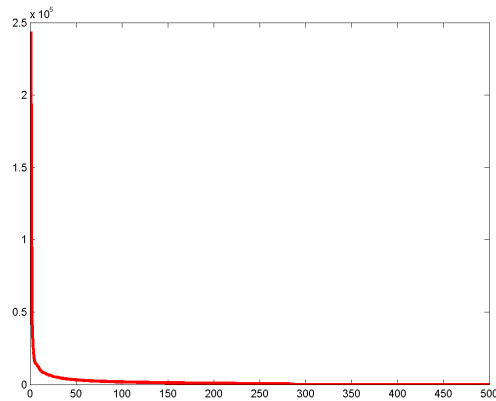
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{bmatrix}$$



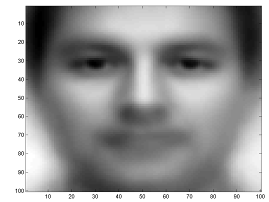
- Compute the eigen vectors
  - Only 300 of the 10000 eigen values are non-zero
    - Why?
- Retain eigen vectors with high eigen values (>0)
  - Could use a higher threshold

Eigen Faces

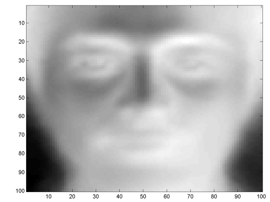
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{bmatrix}$$

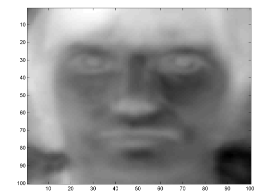


eigenface1



eigenface2

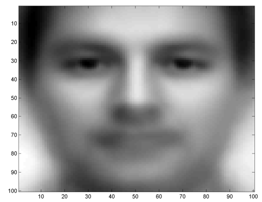
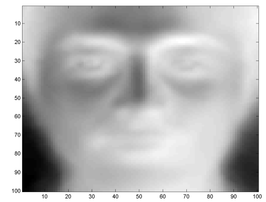
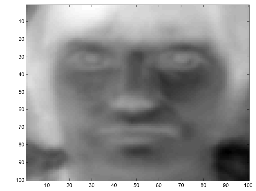


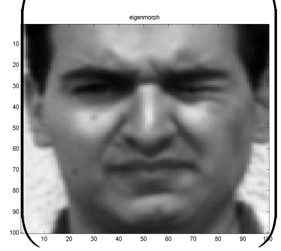


eigenface3

- The eigen vector with the highest eigen value is the first typical face
- The vector with the second highest eigen value is the second typical face.
- Etc.

Representing a face

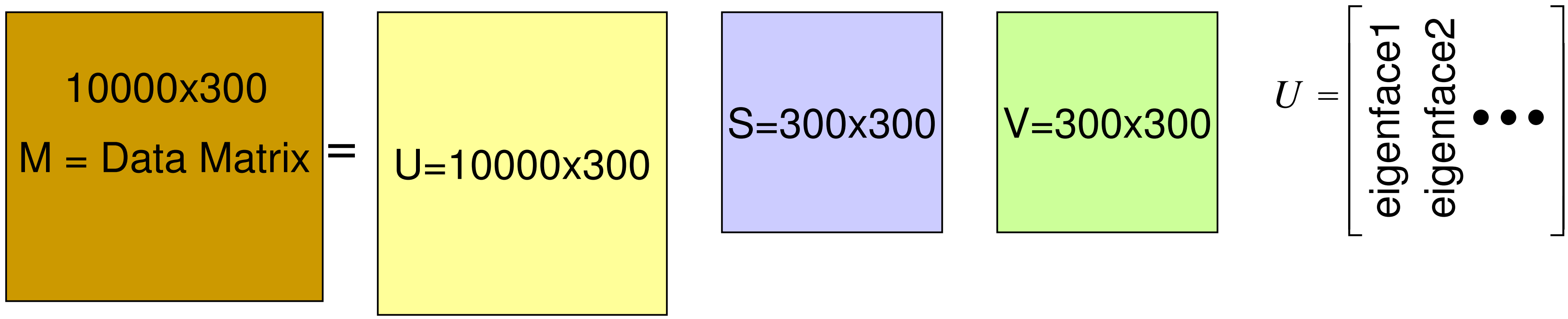
$=$ w_1  $+$ w_2  $+$ w_3  $\bullet\bullet\bullet$

Representation $=$ 

$= [w_1\ w_2\ w_3\ \ldots.]^T$

- The weights with which the eigen faces must be combined to compose the face are used to represent the face!

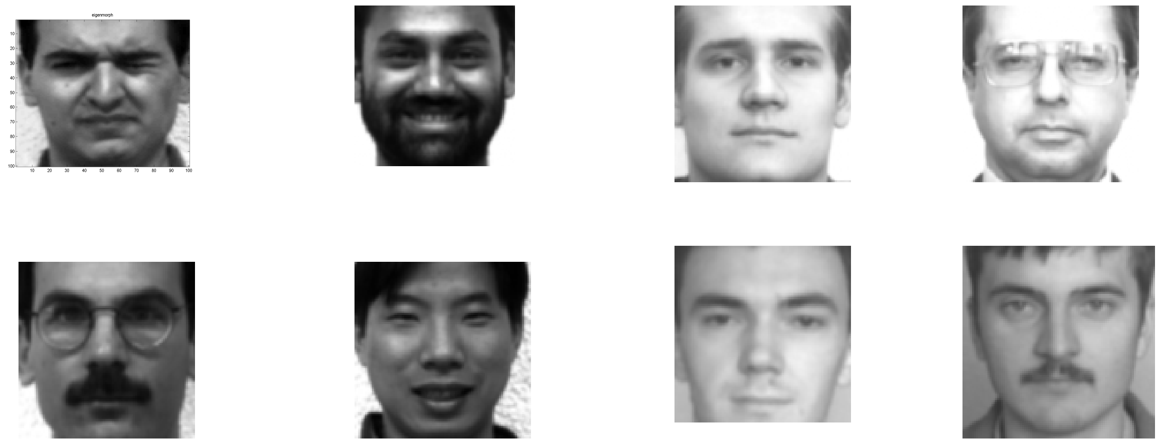
SVD instead of Eigen



- Do we need to compute a 10000 x 10000 correlation matrix and then perform Eigen analysis?
  - Will take a very long time on your laptop
- SVD
  - Only need to perform “Thin” SVD. Very fast
    - U = 10000 x 300
      - The columns of U are the eigen faces!
      - The Us corresponding to the “zero” eigen values are not computed
    - S = 300 x 300
    - V = 300 x 300

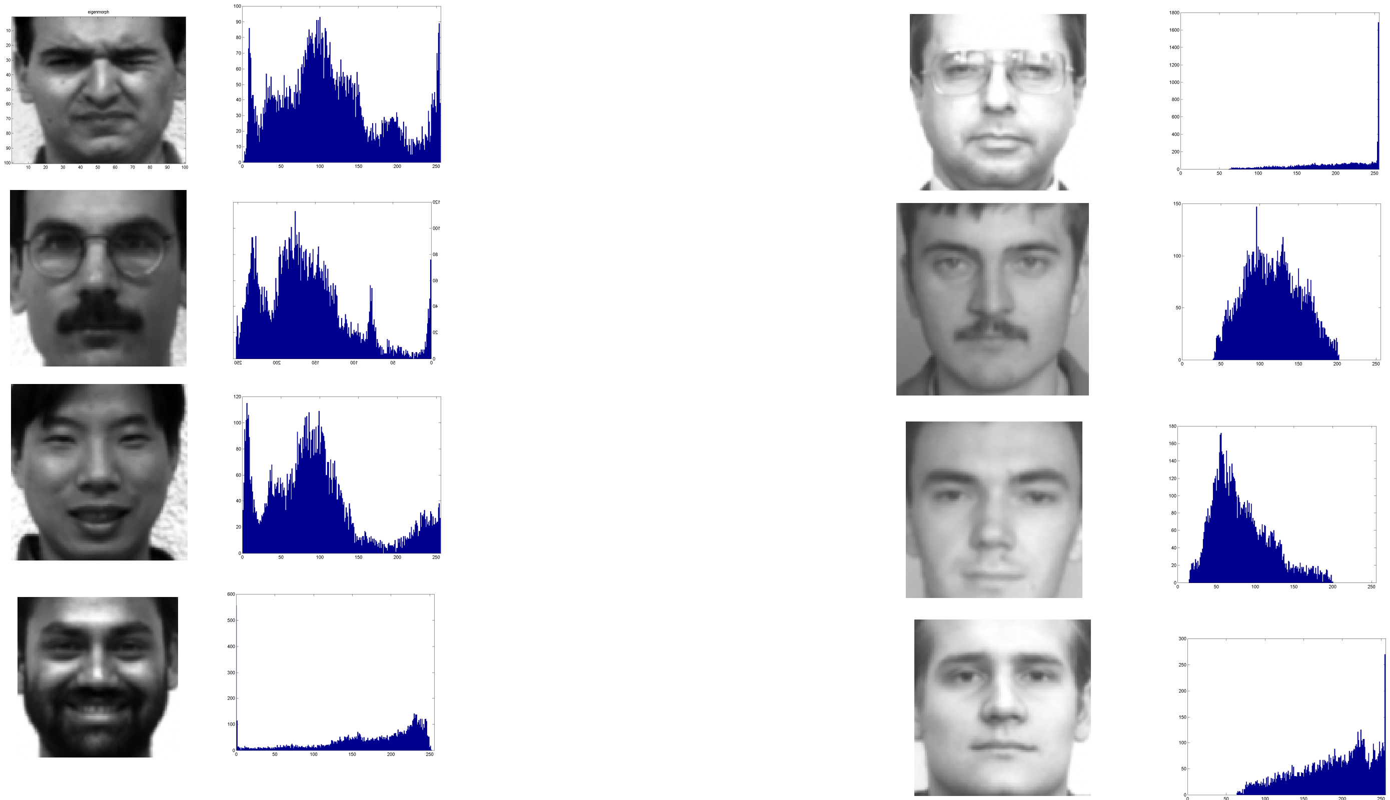
NORMALIZING OUT VARIATIONS

Images: Accounting for variations



- What are the obvious differences in the above images
- How can we capture these differences
  - Hint – image histograms..

Images -- Variations



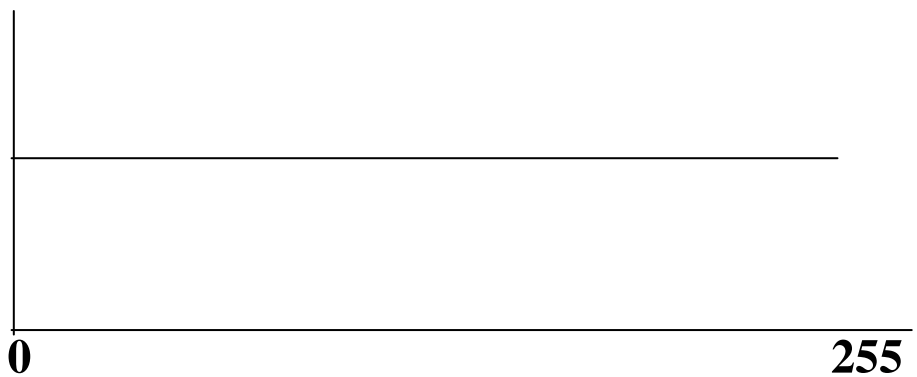
- Pixel histograms: what are the differences

Normalizing Image Characteristics

- Normalize the pictures
  - Eliminate lighting/contrast variations
  - All pictures must have “similar” lighting
    - How?
- Lighting and contrast are represented in the image histograms:

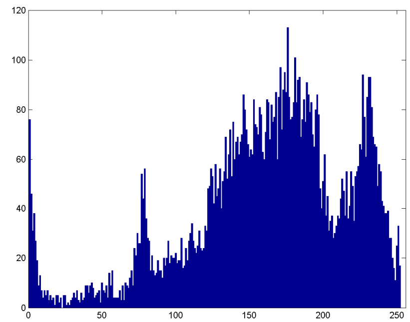
Histogram Equalization

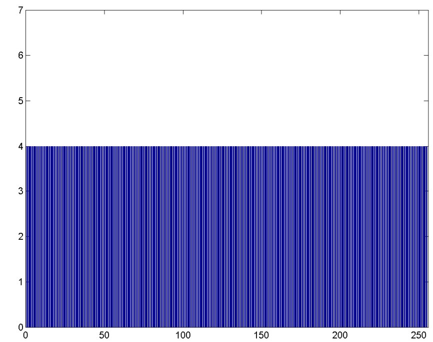
- Normalize histograms of images
  - Maximize the contrast
    - Contrast is defined as the “flatness” of the histogram
    - For maximal contrast, every greyscale must happen as frequently as every other greyscale



- Maximizing the contrast: Flattening the histogram
  - Doing it for every image ensures that every image has the same constrast
    - I.e. exactly the same histogram of pixel values
      - Which should be flat

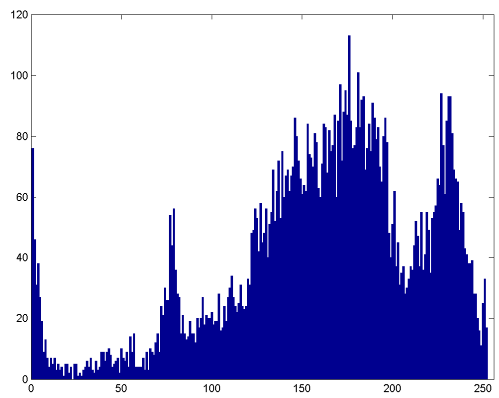
Histogram Equalization

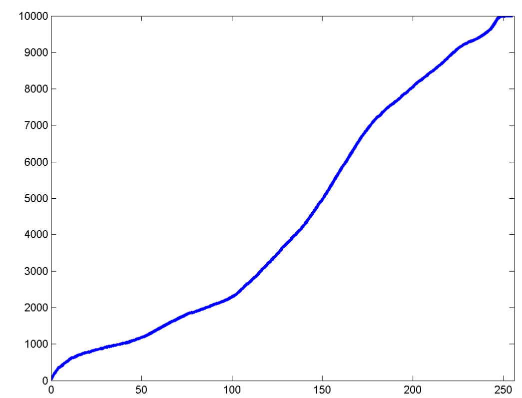




- Modify pixel values such that histogram becomes “flat”.
- For each pixel
  - New pixel value = f(old pixel value)
  - What is f()?
- Easy way to compute this function: map cumulative counts

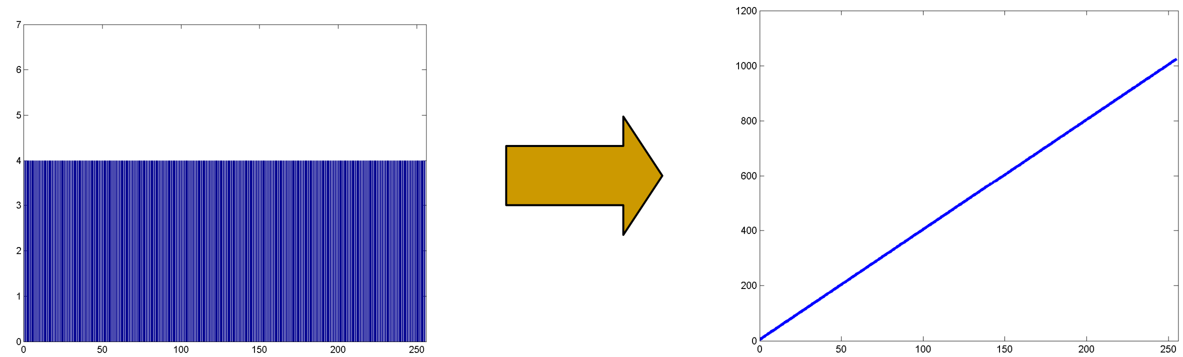
Cumulative Count Function



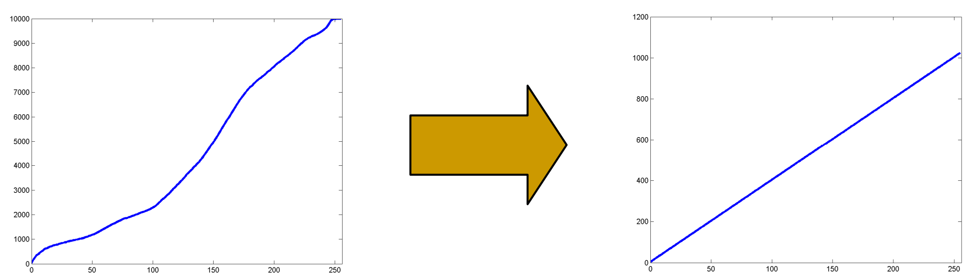


- The *histogram (count)* of a pixel value X is the number of pixels in the image that have value X
  - E.g. in the above image, the count of pixel value 180 is about 110
- The *cumulative count* at pixel value X is the total number of pixels that have values in the range 0 <= x <= X
  - CCF(X) = H(1) + H(2) + .. H(X)

Cumulative Count Function

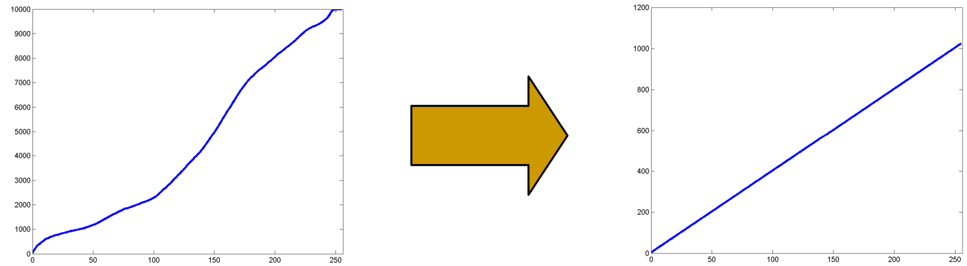


- The cumulative count function of a uniform histogram is a line



- We must modify the pixel values of the image so that its cumulative count is a line

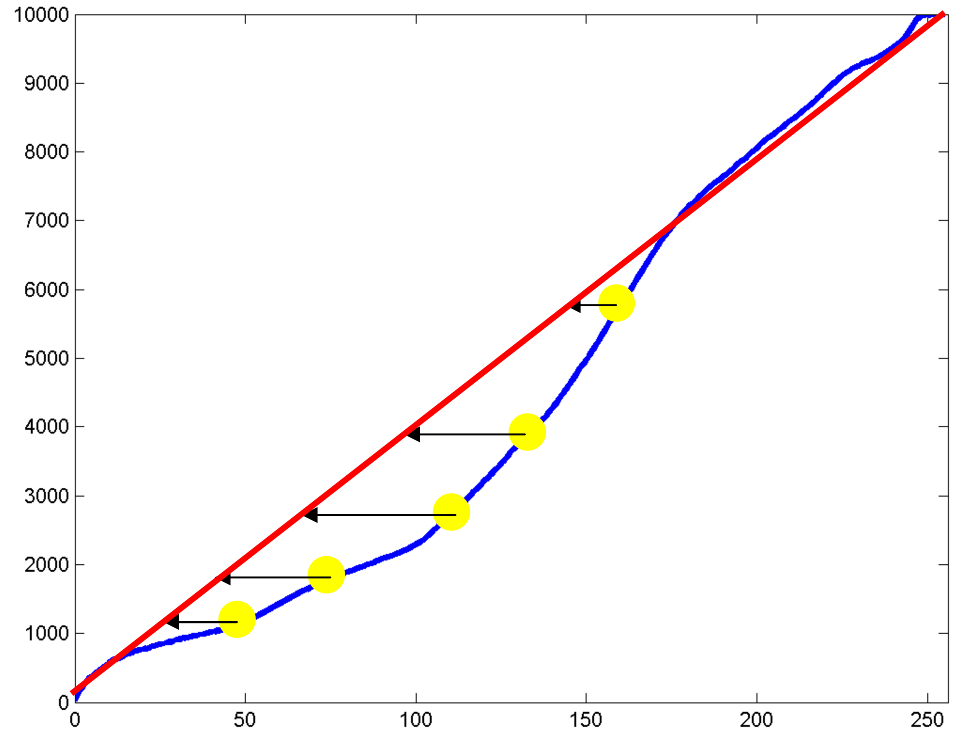
Mapping CCFs



Move x axis levels around until the plot to the left looks like the plot to the right

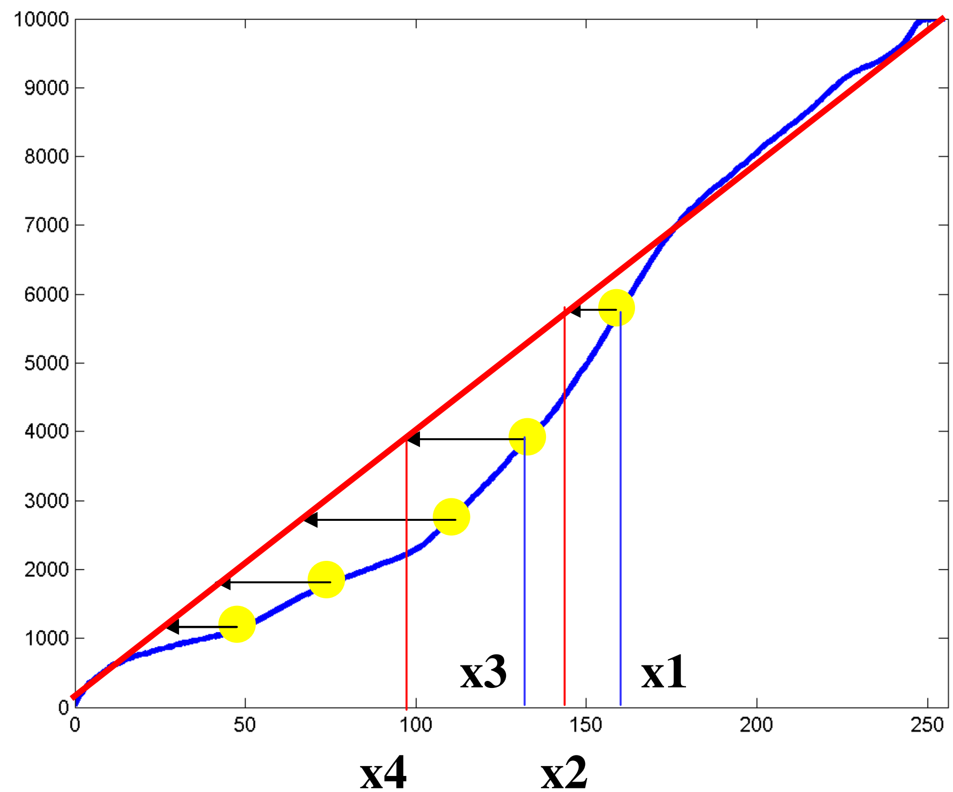
- CCF(f(x)) -> a*f(x)   [of a*(f(x)+1) if pixels can take value 0]
  - x = pixel value
  - f() is the function that converts the old pixel value to a new (normalized) pixel value
  - a = (total no. of pixels in image) / (total no. of pixel levels)
    - The no. of pixel levels is 256 in our examples
    - Total no. of pixels is 10000 in a 100x100 image

Mapping CCFs



- For each pixel value x:
  - Find the location on the red line that has the closet Y value to the observed CCF at x

Mapping CCFs



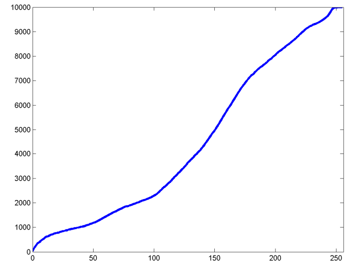
f(x1) = x2

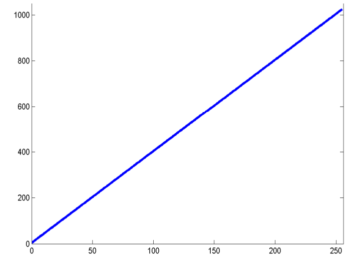
f(x3) = x4

Etc.

- For each pixel value x:
  - Find the location on the red line that has the closet Y value to the observed CCF at x

Mapping CCFs

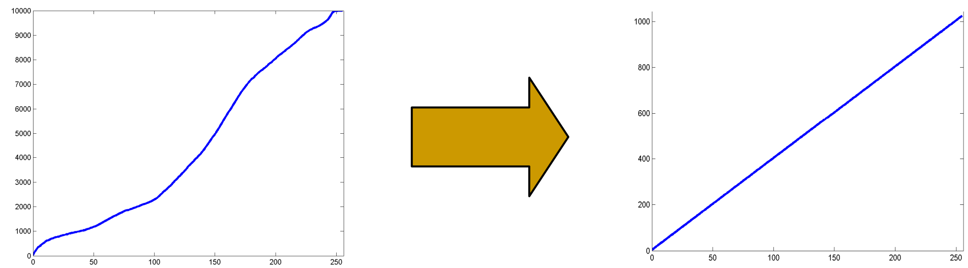




Move x axis levels around until the plot to the left looks like the plot to the right

- For each pixel in the image to the left
  - The pixel has a value x
  - Find the CCF at that pixel value CCF(x)
  - Find x' such that CCF(x') in the function to the right equals CCF(x)
    - x' such that CCF_flat(x') = CCF(x)
  - Modify the pixel value to x'

Doing it Formulaically



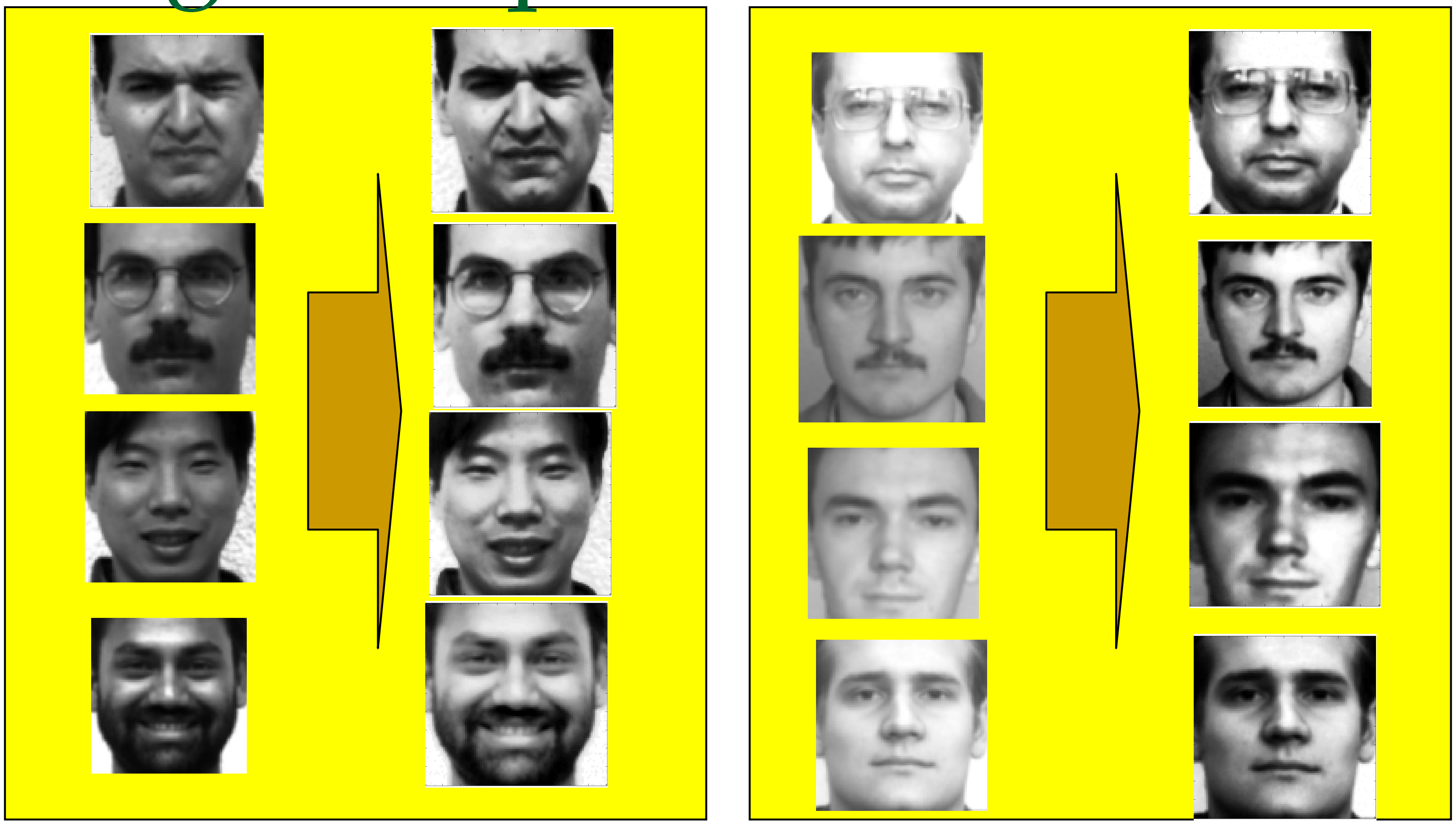
$$f(x) = round\left(\frac{CCF(x) - CCF_{min}}{Npixels - CCF_{min}} Max.pixel.value\right)$$

- CCF_{min} is the smallest non-zero value of CCF(x)
  - The value of the CCF at the smallest observed pixel value
- Npixels is the total no. of pixels in the image
  - 10000 for a 100x100 image
- Max.pixel.value is the highest pixel value
  - 255 for 8-bit pixel representations

Or even simpler

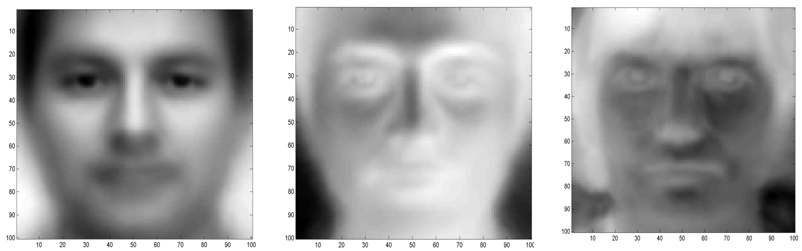
- Matlab:
  - `Newimage = histeq(oldimage)`

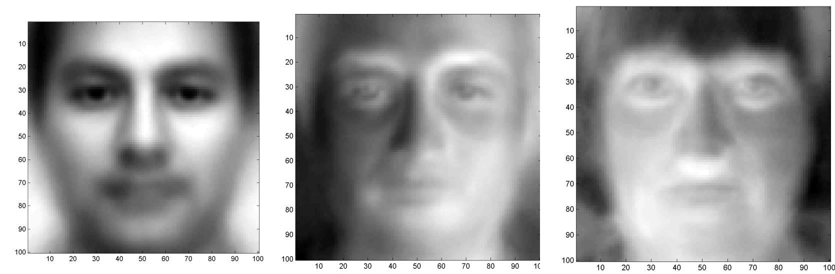
Histogram Equalization



- Left column: Original image
- Right column: Equalized image
- All images now have similar contrast levels

Eigenfaces after Equalization

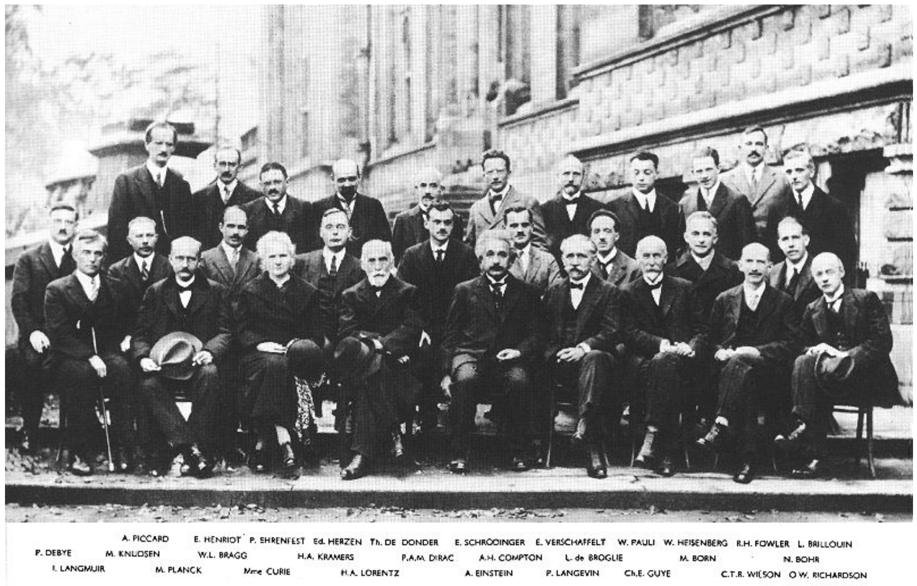




- Left panel : Without HEQ
- Right panel: With HEQ
  - Eigen faces are more face like..
    - Need not always be the case

Detecting Faces in Images

Detecting Faces in Images



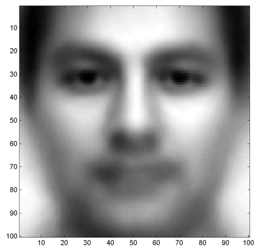
- Finding face like patterns
  - How do we find if a picture has faces in it
  - Where are the faces?

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

Finding faces in an image

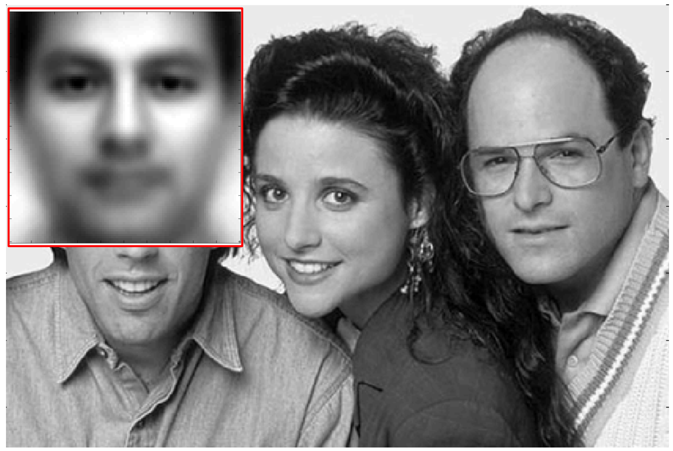
- Picture is larger than the "typical face"
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - R + G + B
  - Not very useful to work in color

Finding faces in an image



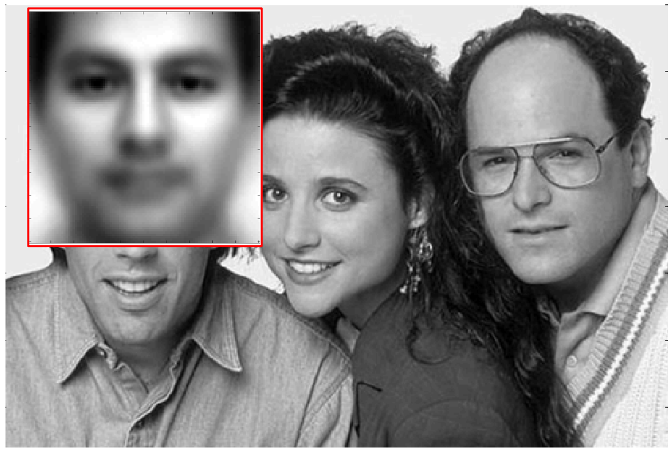
- Goal .. To find out if and where images that look like the “typical” face occur in the picture

Finding faces in an image



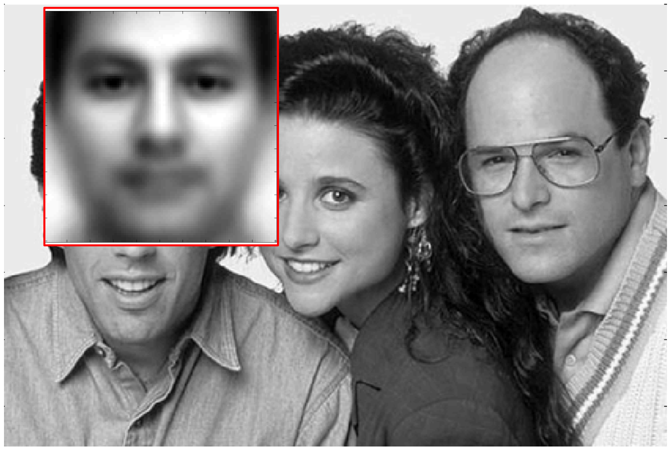
- Try to “match” the typical face to each location in the picture

Finding faces in an image



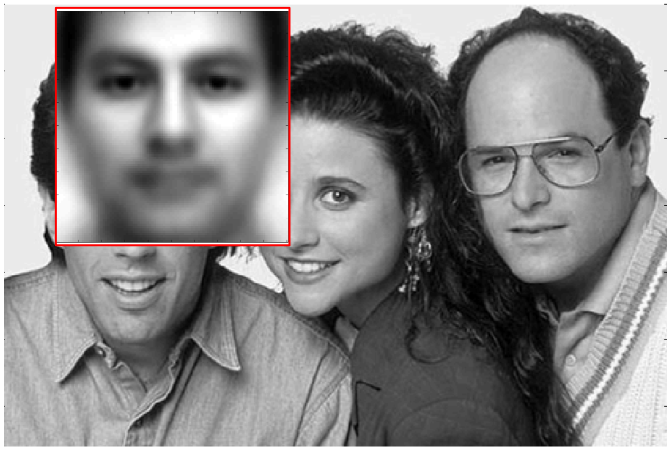
- Try to "match" the typical face to each location in the picture

Finding faces in an image



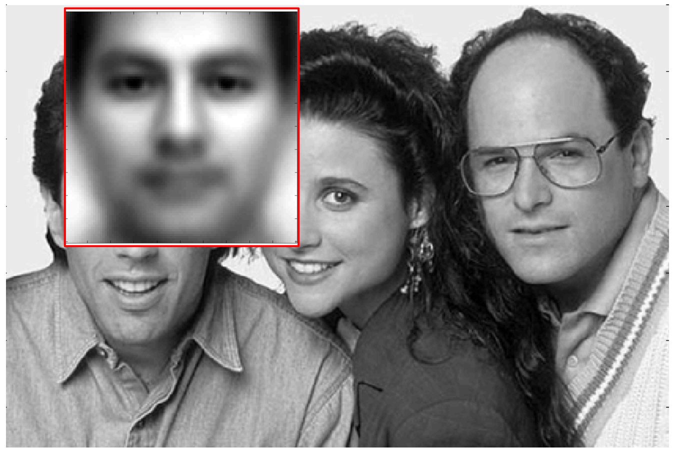
- Try to “match” the typical face to each location in the picture

Finding faces in an image



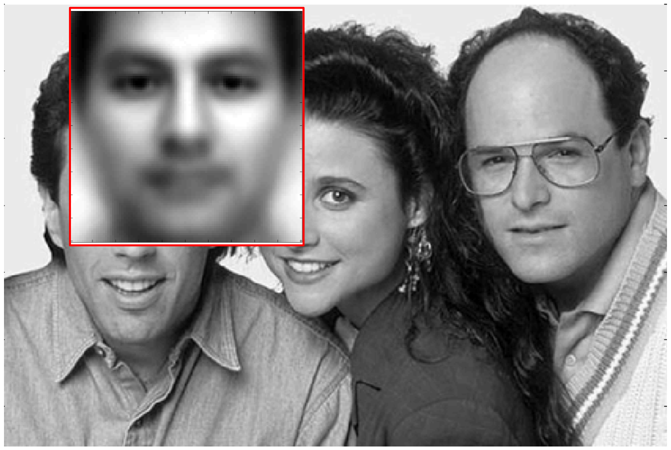
- Try to “match” the typical face to each location in the picture

Finding faces in an image



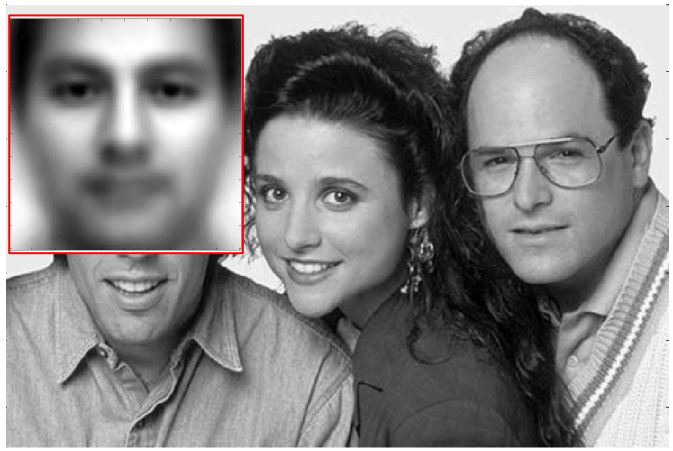
- Try to "match" the typical face to each location in the picture

Finding faces in an image



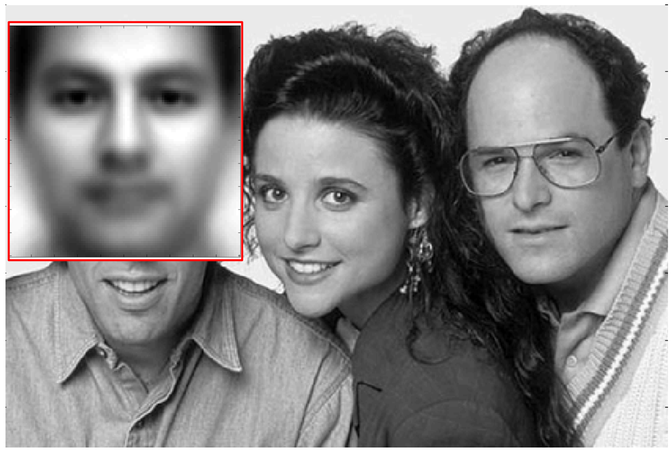
- Try to “match” the typical face to each location in the picture

Finding faces in an image



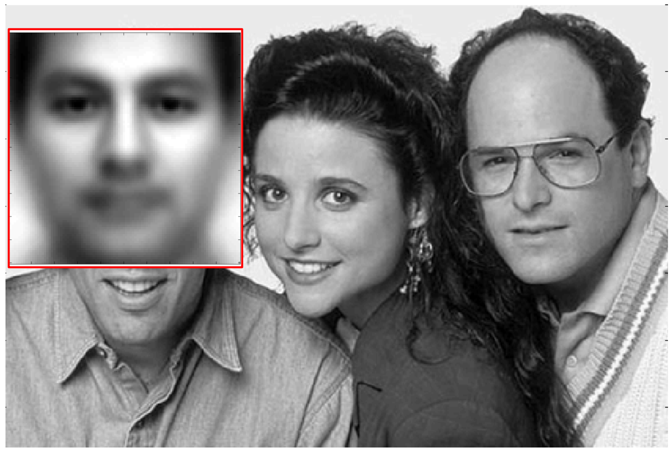
- Try to “match” the typical face to each location in the picture

Finding faces in an image



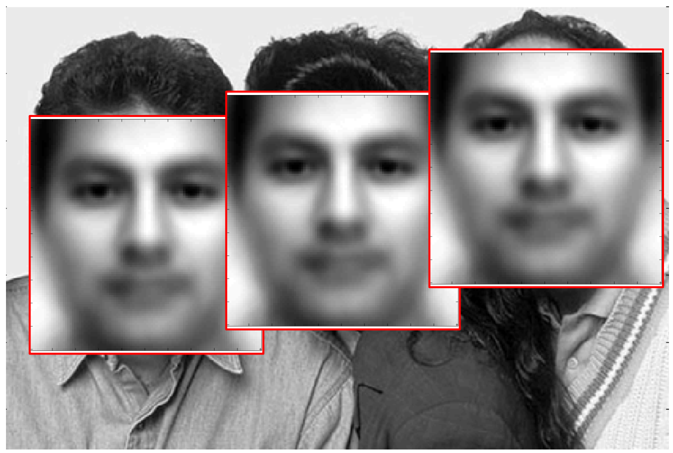
- Try to “match” the typical face to each location in the picture

Finding faces in an image



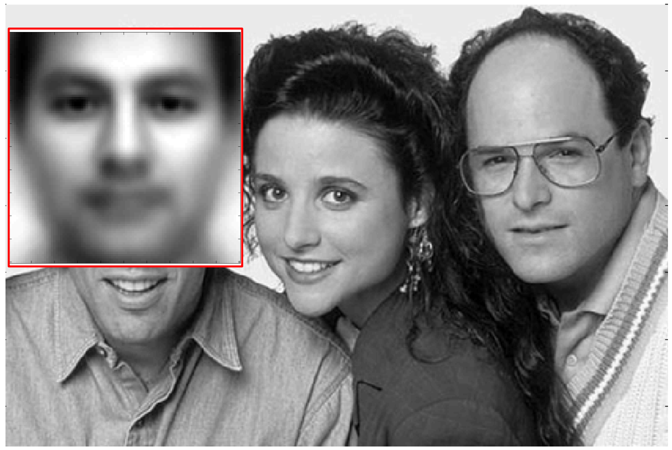
- Try to “match” the typical face to each location in the picture

Finding faces in an image



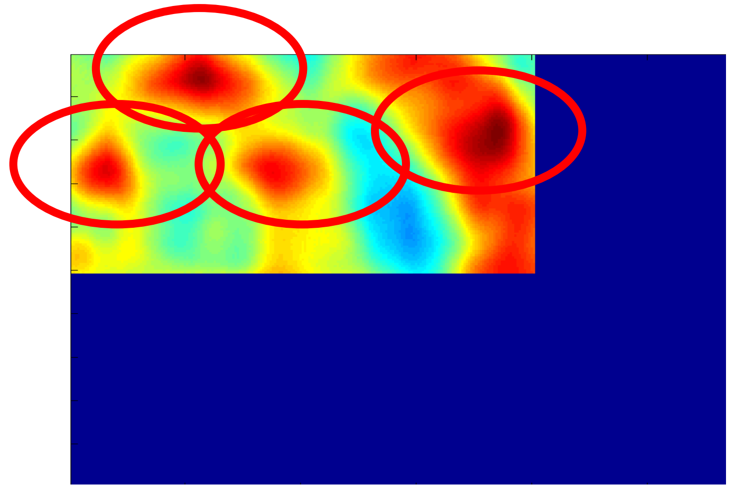
- Try to "match" the typical face to each location in the picture
- The "typical face" will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!

How to "match"



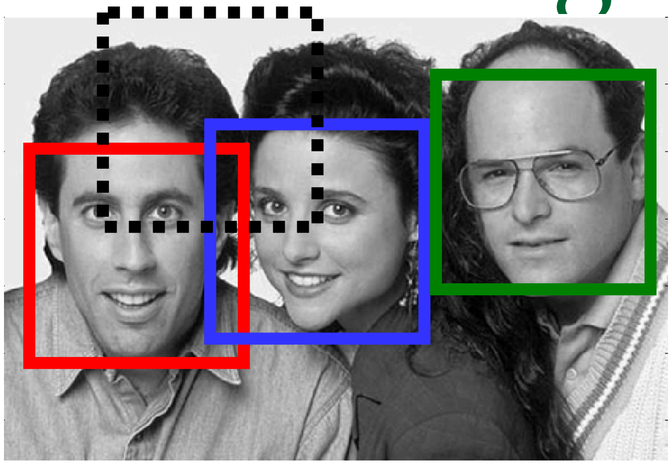
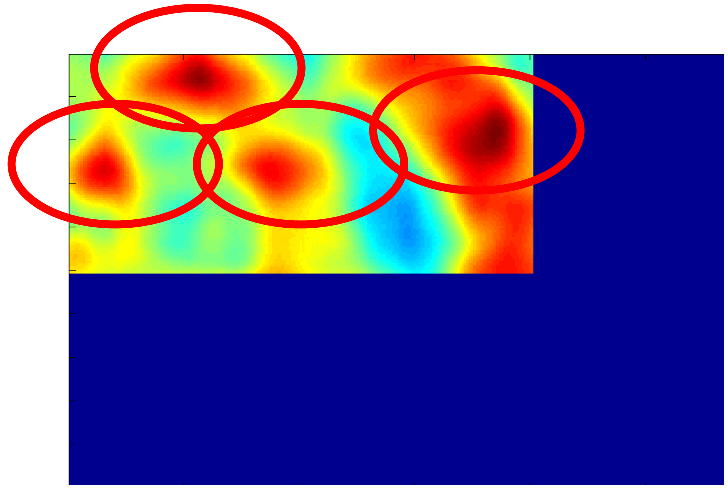
- What exactly is the "match"
  - What is the match "score"
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of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

What do we get



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

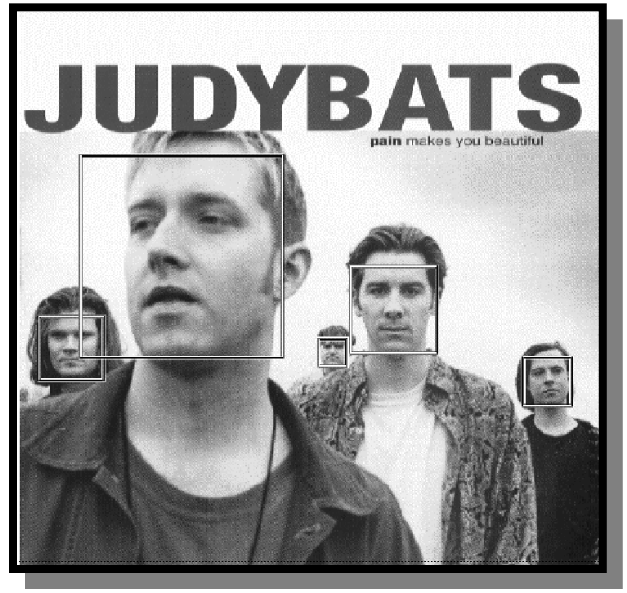
What do we get

- The right panel shows the dot product a various loctions
  - 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

Scaling and Rotation Problems

- Scaling
  - 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



- 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

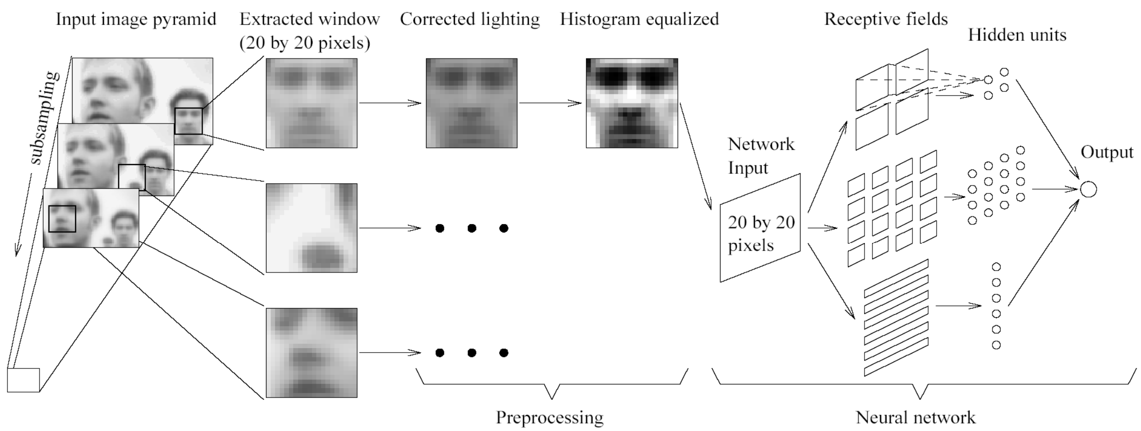


Figure 1: The basic algorithm used for face detection.

- 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
- But first, what is boosting

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 5’4” 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..
- Face detection: Recast as binary face classification
  - For each little square of the image, determine if the square represents a face or not

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

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

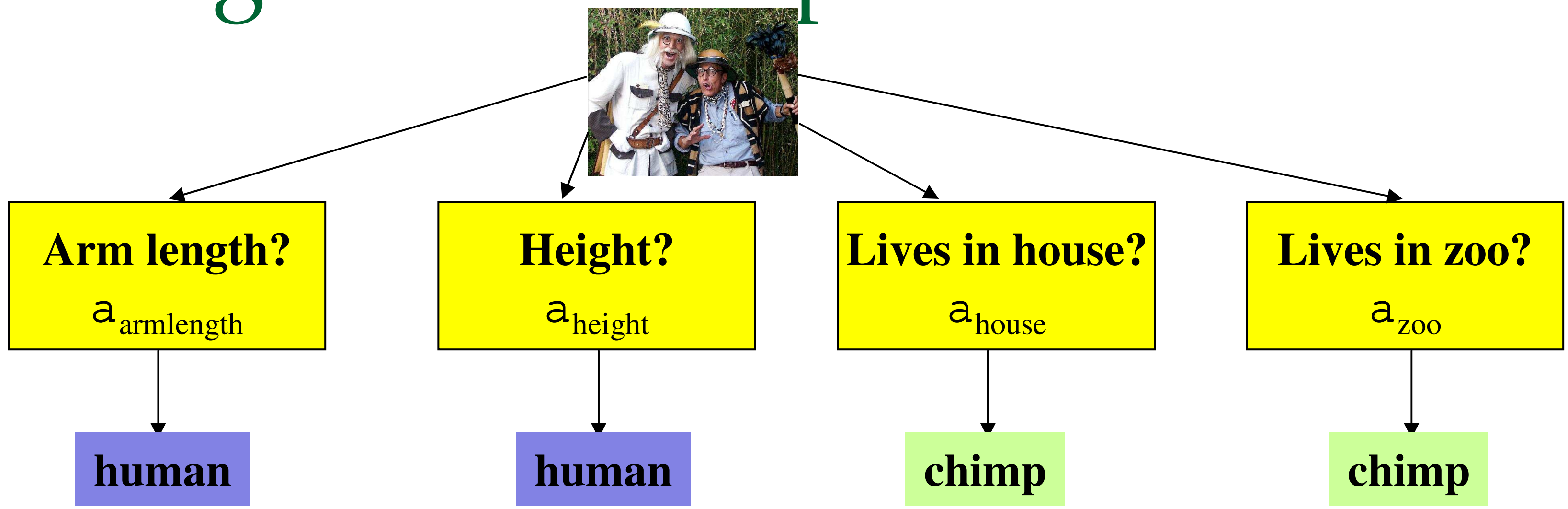
Boosting: A very simple idea

- 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



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

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

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

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

- 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 random guessing 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

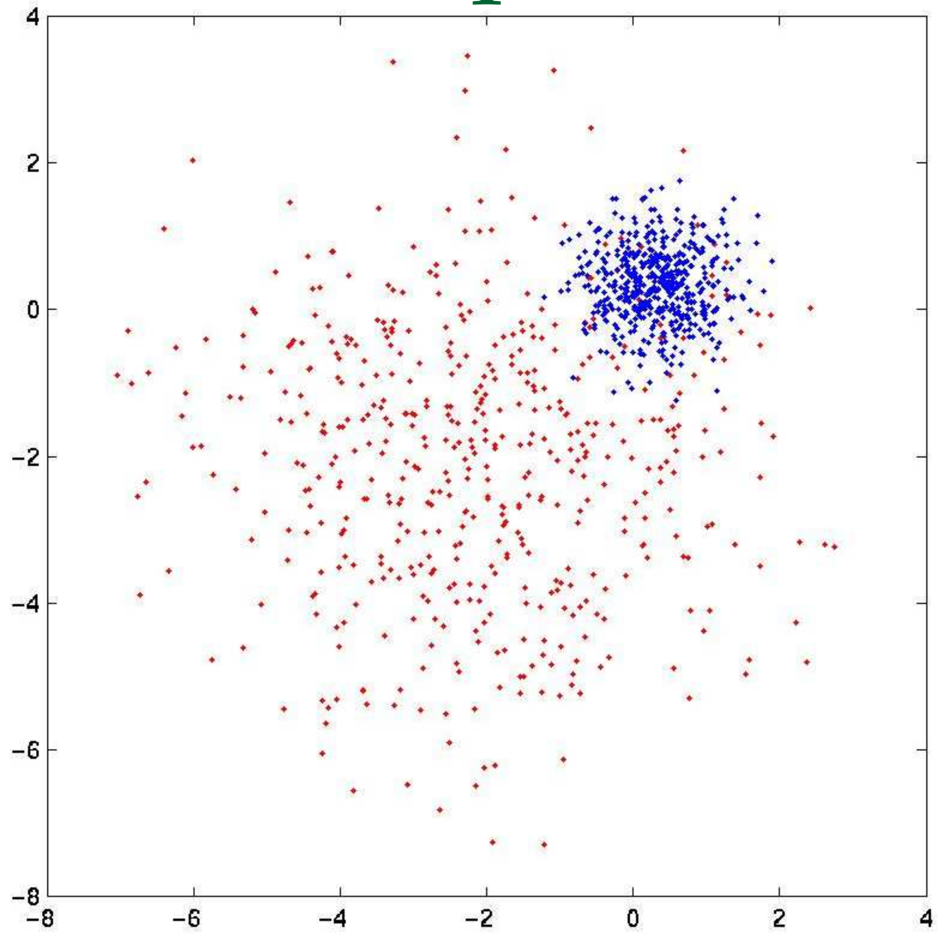
Boosting: A Voting Perspective

- 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

ADA Boost: Adaptive algorithm for learning the weights

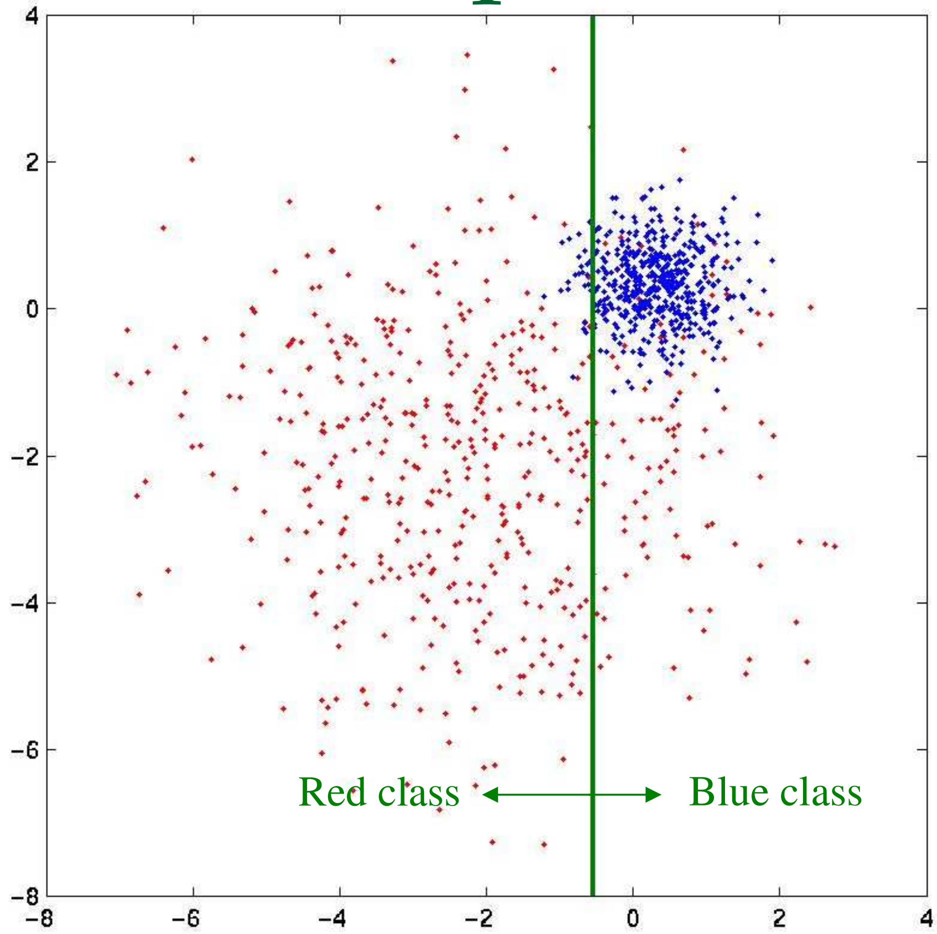
- ADA Boost: Not named of ADA Lovelace
- An *adaptive* algorithm that learns the weights of each classifier sequentially
  - Learning adapts to the current accuracy
- Iteratively:
  - Train a simple classifier from training data
    - It will make errors even on training data
    - Train a new classifier that focuses on the training data points that have been misclassified

Boosting: An Example



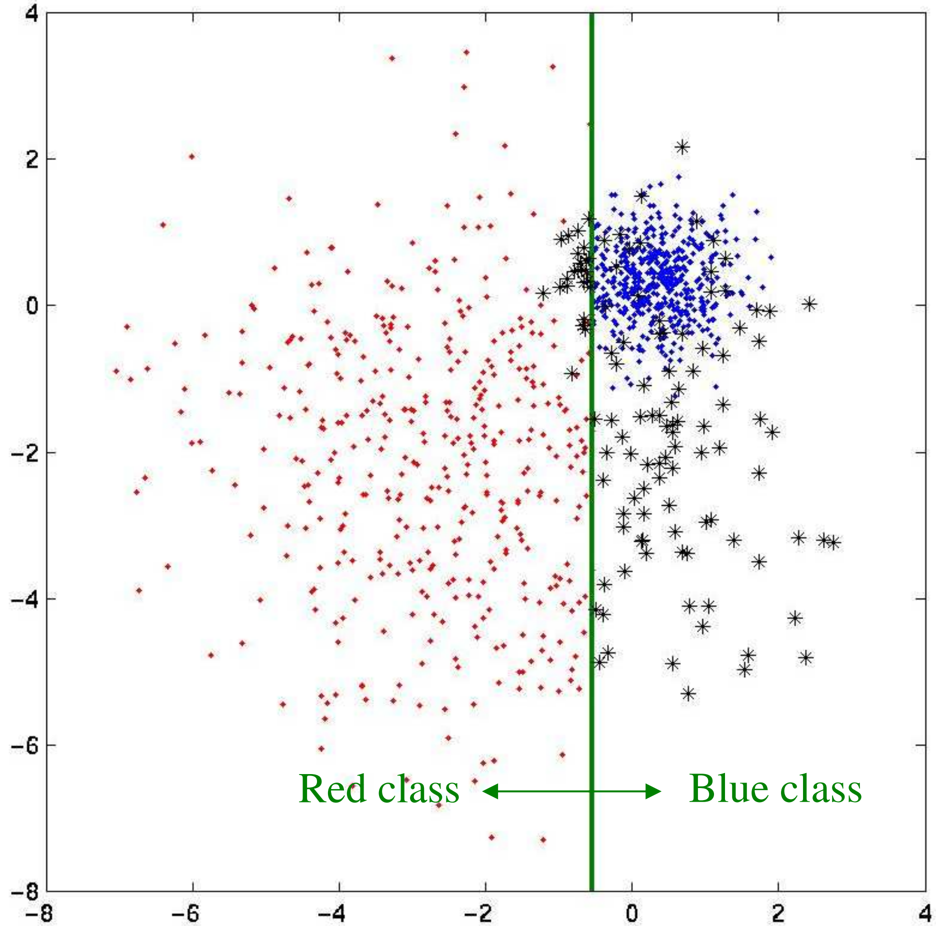
- Red dots represent training data from Red class
- Blue dots represent training data from Blue class

Boosting: An Example



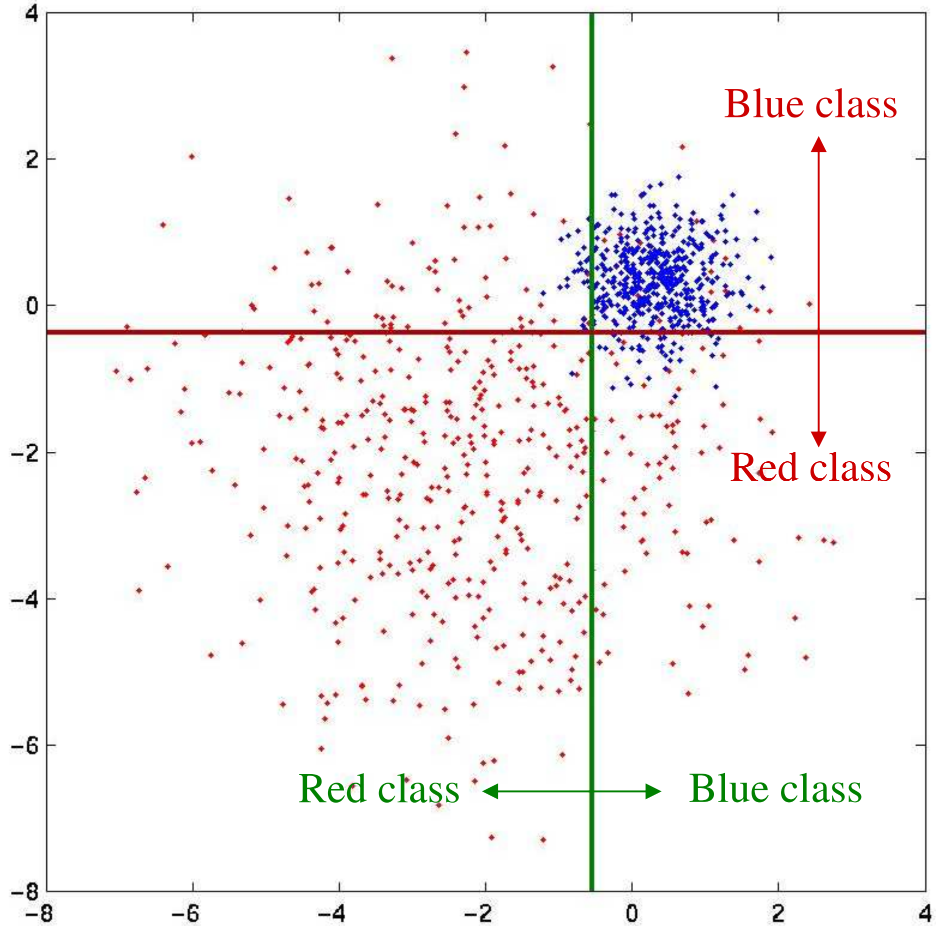
- Very simple weak learner
  - A line that is parallel to one of the two axes

Boosting: An Example



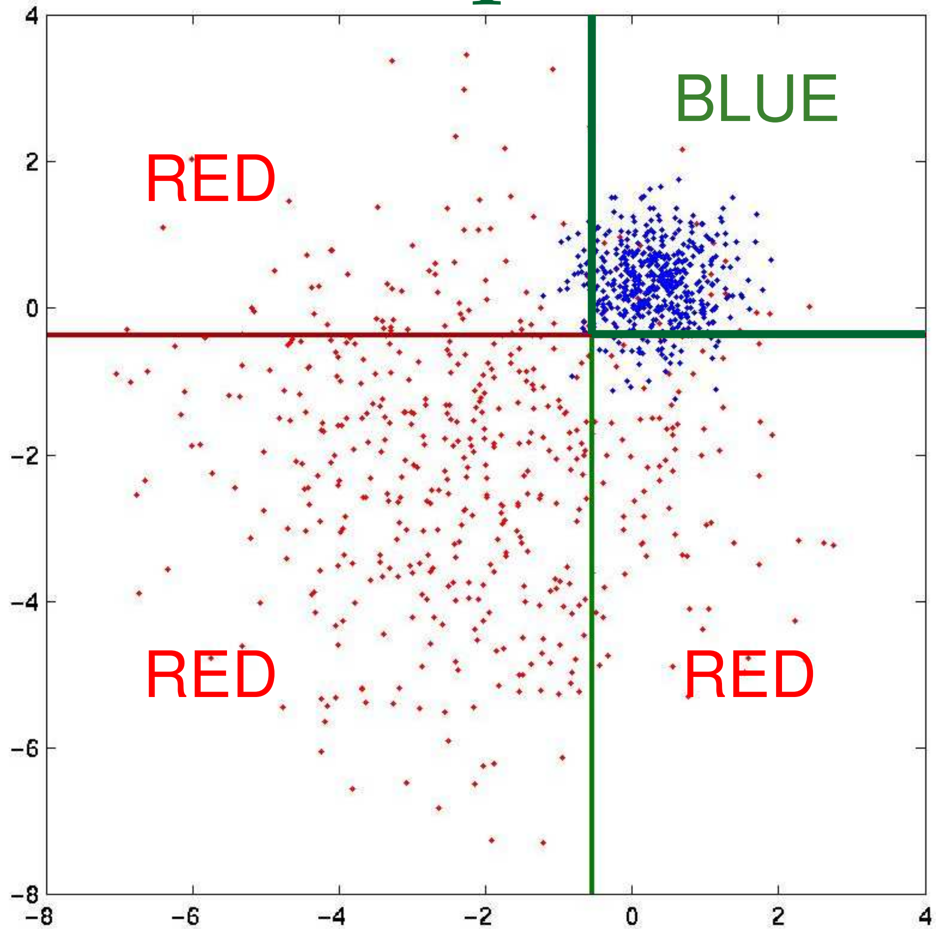
- First weak learner makes many mistakes
  - Errors coloured black

Boosting: An Example



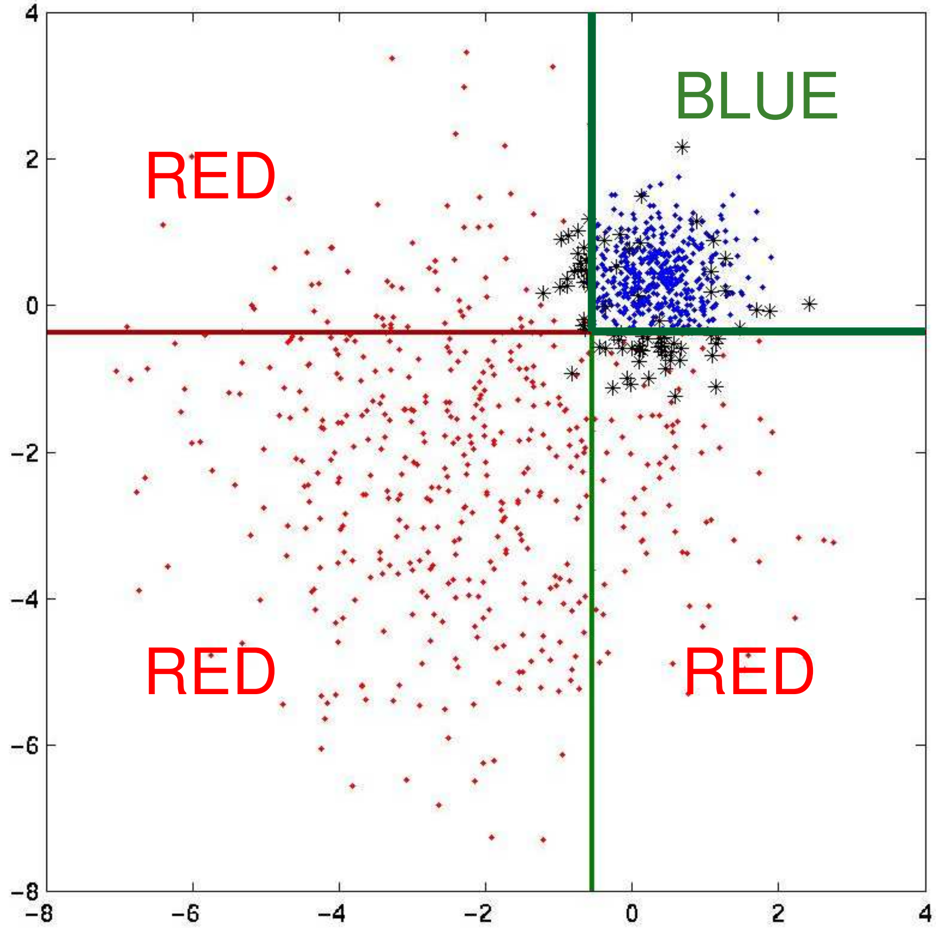
- Second weak learner focuses on errors made by first learner

Boosting: An Example



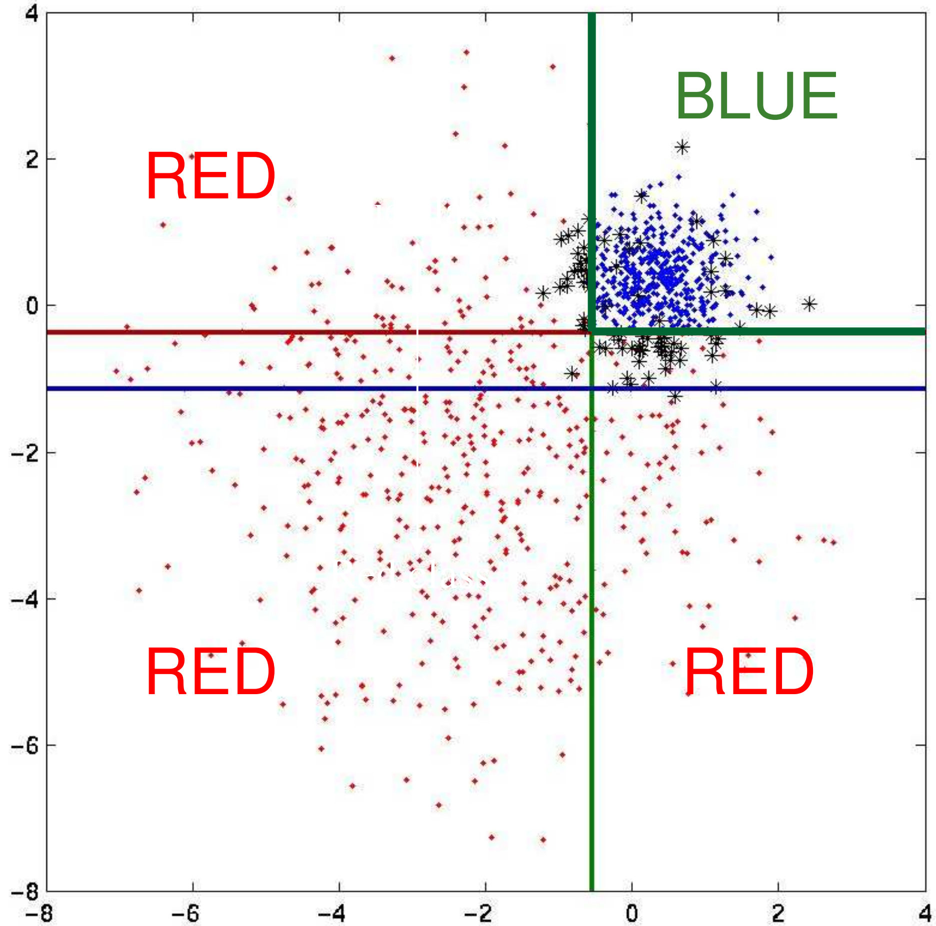
- Second strong learner: weighted combination of first and second weak learners
  - Decision boundary shown by black lines

Boosting: An Example



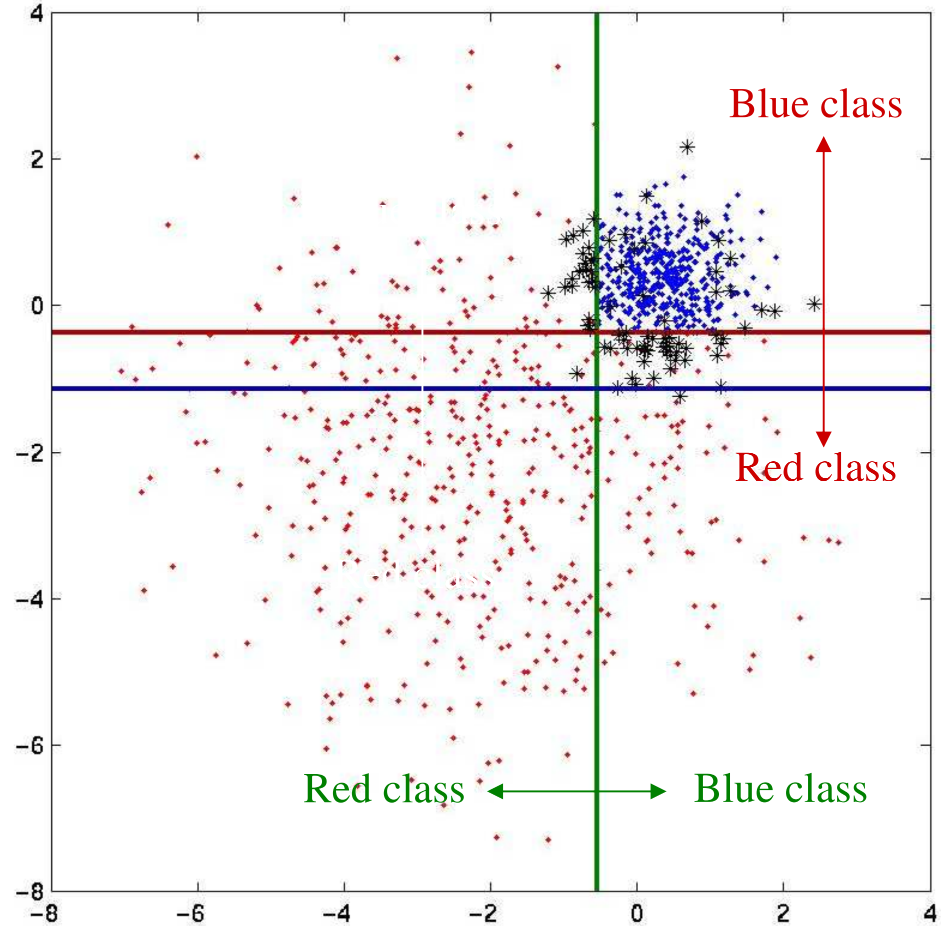
- The second strong learner also makes mistakes
  - Errors colored black

Boosting: An Example



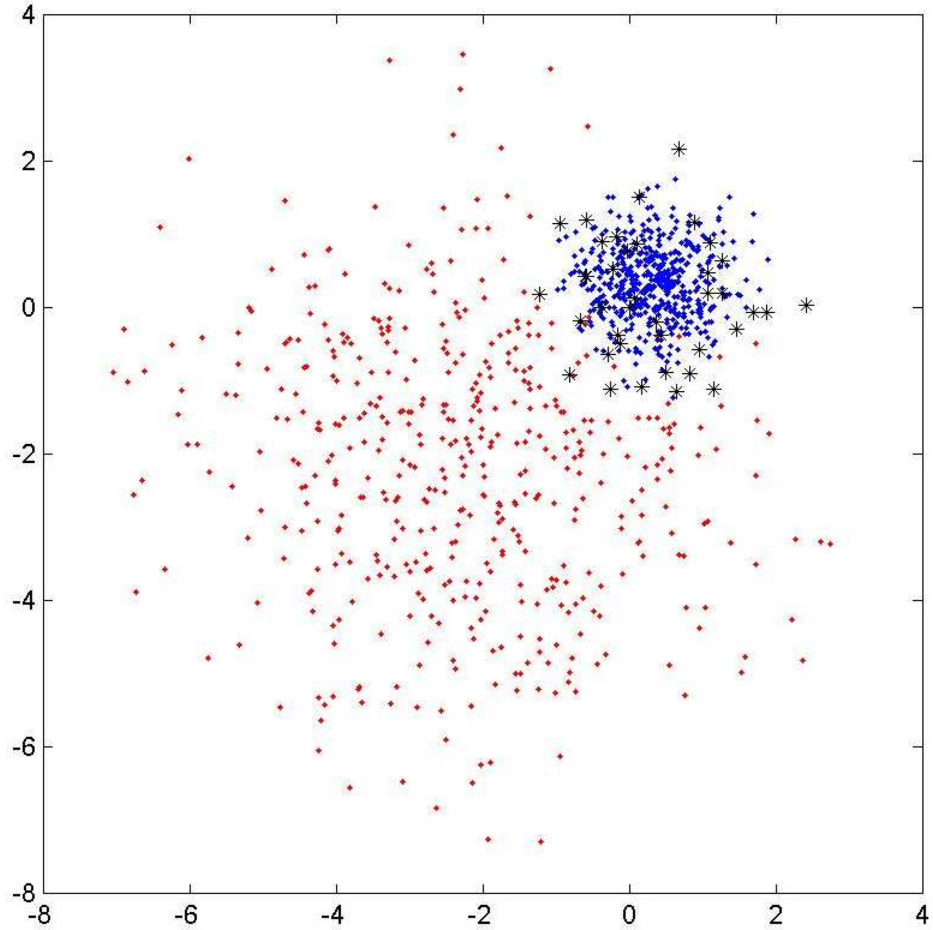
- Third weak learner concentrates on errors made by second strong learner

Boosting: An Example



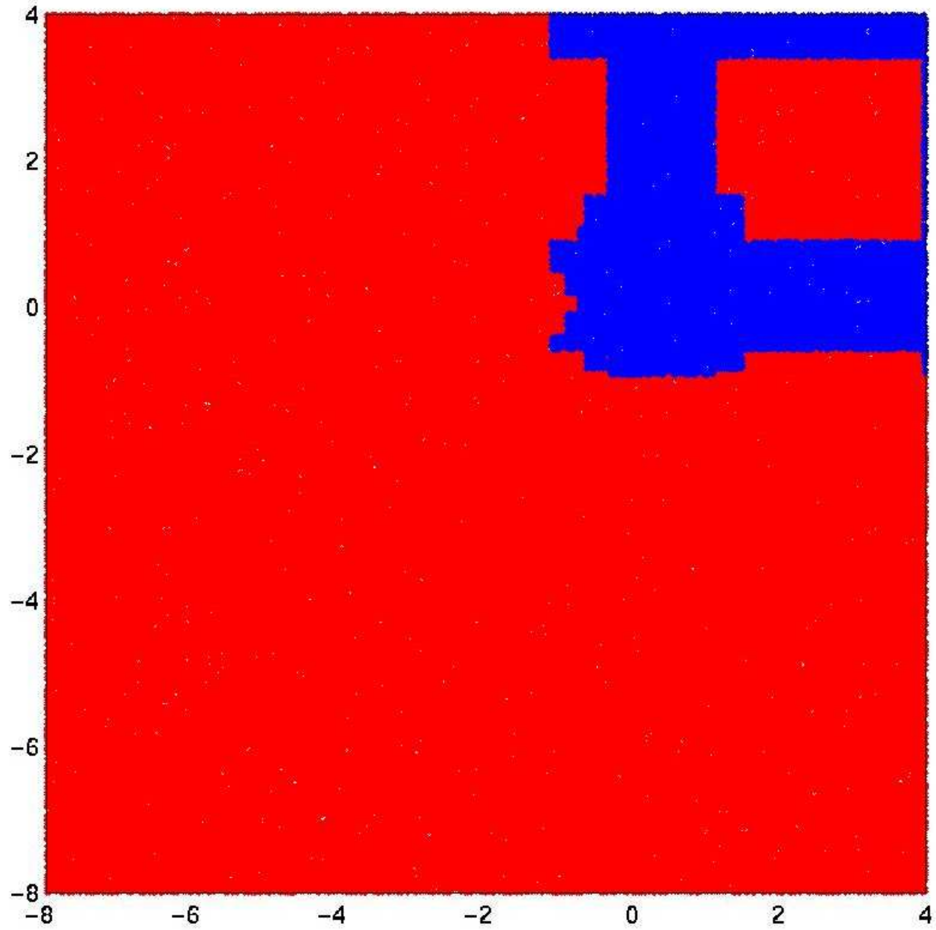
- Third weak learner concentrates on errors made by combination of previous weak learners
- Continue adding weak learners until….

Boosting: An Example



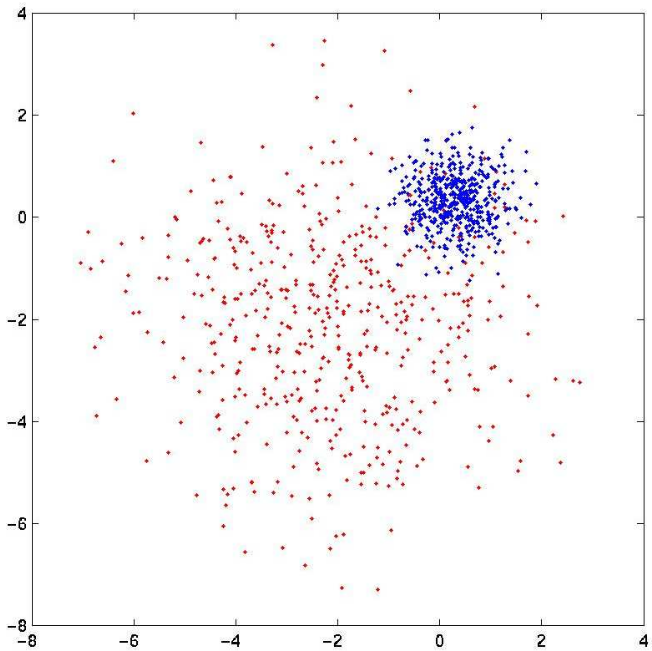
- Voila! Final strong learner: very few errors on the training data

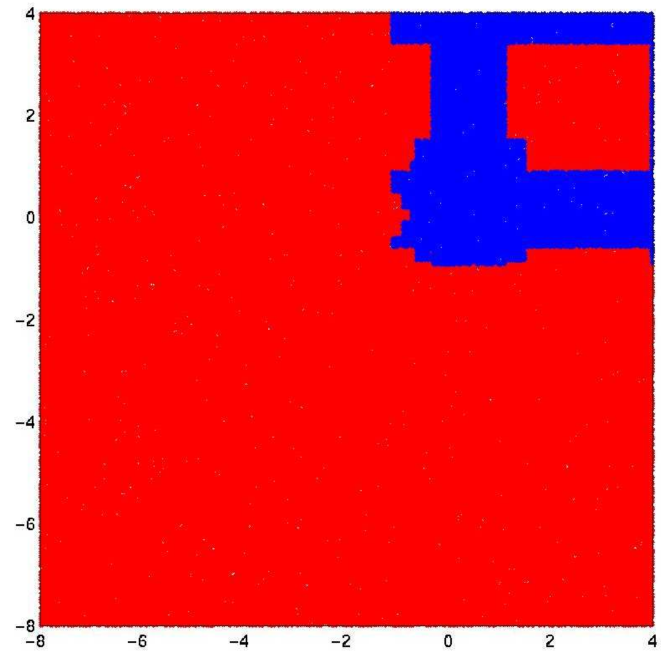
Boosting: An Example



- The final strong learner has learnt a complicated decision boundary

Boosting: An Example

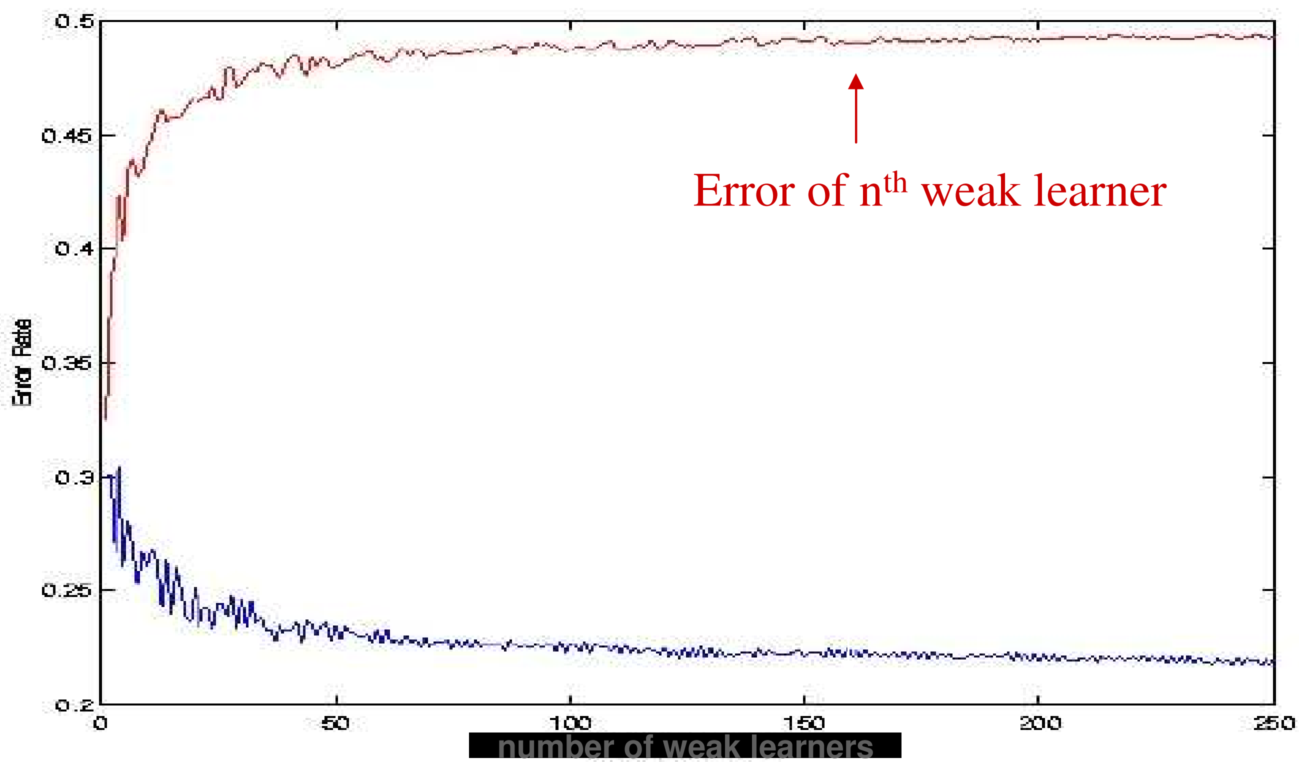




- The final strong learner has learnt a complicated decision boundary
- Decision boundaries in areas with low density of training points assumed inconsequential

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



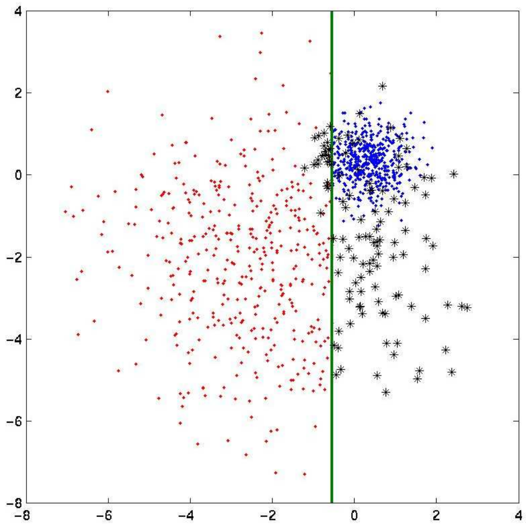
ADABoost

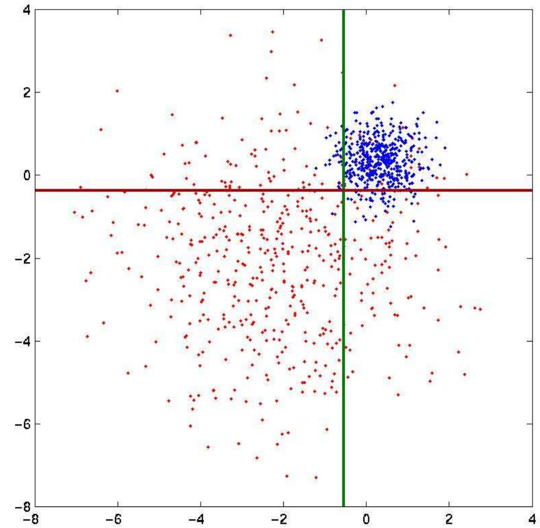
- Cannot just add new classifiers that work well only the the previously misclassified data

- Problem: The new classifier will make errors on the points that the **earlier** classifiers got right
  - Not good
  - On test data we have no way of knowing which points were correctly classified by the first classifier

- Solution: Weight the data when training the second classifier
  - Use all the data but assign them weights
    - Data that are already correctly classified have less weight
    - Data that are currently incorrectly classified have more weight

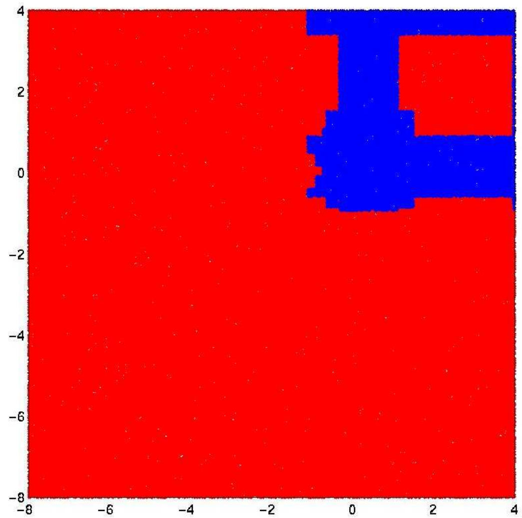
ADA Boost





- The red and blue points (correctly classified) will have a weight $a < 1$
- Black points (incorrectly classified) will have a weight b $(= 1/a) > 1$
- To compute the optimal second classifier, we minimize the total weighted error
  - Each data point contributes a or b to the total count of correctly and incorrectly classified points
    - E.g. if one of the red points is misclassified by the new classifier, the total error of the new classifier goes up by a

ADA Boost



- Each new classifier modifies the weights of the data points based on the accuracy of the *current* classifier
- ***The final classifier too is a weighted combination of all component classifiers***

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 $h_i(x)$

Adaptive Boosting

- As before:
  - y is either -1 or +1
  - $H(x)$ is +1 or -1
  - If the instance is correctly classified, both y and $H(x)$ will have the same sign
    - The product y.H(x) is 1
    - For incorrectly classified instances the product is -1
- Define the error for x : $\frac{1}{2}(1 - yH(x))$
  - For a correctly classified instance, this is 0
  - For an incorrectly classified instance, this is 1

The ADABoost Algorithm

- Given: a set $(x_1, y_1), \ldots (x_N, y_N)$ of training instances
  - x_i is the set of attributes for the i^{th} instance
  - y_i is the class for the i^{th} instance and can be either +1 or -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
    - $e_t = \text{Sum}\ \{½\ (1 – y_i\ h_t(x_i))\}$
  - Set $a_t = ½ \ln((1 – e_t) / e_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- a_t\ y_i\ h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(S_t\ a_t\ h_t(x))$

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

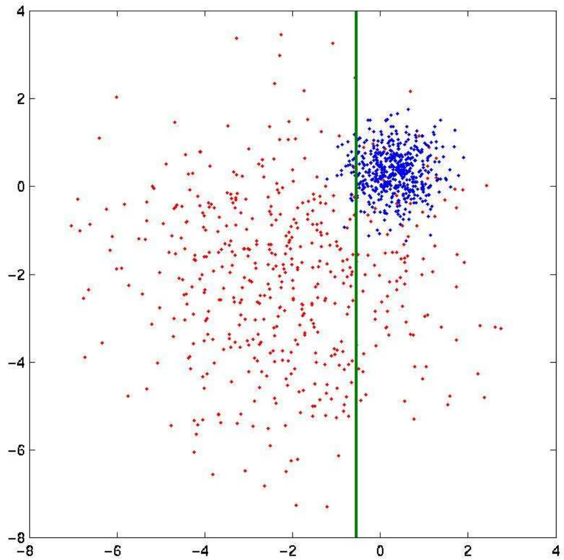
ADA Boost

- Initialize $D_1(x_i) = 1/N$
- Just a normalization: total weight of all instances is 1
  - Makes the algorithm invariant to training data set size

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
    - $e_t = \text{Sum}\ \{½\ (1 - y_i\ h_t(x_i))\}$
  - Set $a_t = ½ \ln ((1 - e_t) / e_t)$
  - For $i = 1 \ldots N$
    - set $D_{t+1}(x_i) = D_t(x_i) \exp(- a_t\ y_i\ h_t(x_i))$
  - Normalize D_{t+1} to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(S_t\ a_t\ h_t(x))$

ADA Boost



- Train a weak classifier h_t using distribution D_t
- Simply train the simple classifier that that classifies data with error 50%
  - Where each data *x* point contributes *D*(*x*) towards the count of errors or correct classification
  - Initially D(x) = 1/N for all data
- Better to actually train a *good* classifier

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
    - $\epsilon_t = \mathrm{Sum}\ \{½\ (1 - y_i\ h_t(x_i))\}$
  - Set $\alpha_t = ½ \ln ((1 - \epsilon_t) / \epsilon_t)$
  - 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
  - $H(x) = \mathrm{sign}(\Sigma_t\ \alpha_t\ h_t(x))$

ADA Boost

- Compute total error on training data
  - e_t = Sum {½ $(1 - y_i\, h_t(x_i))$}

- For each data point x, ½$(1\text{-}y.h(x))$ = 0 for correct classification, 1 for error

- e_t is simply the sum of the weights $D(x)$ for all points that are misclassified by the latest classifier $h_t(x)$
  - Will lie between 0 and 1

$$e_t = \sum_{x \text{ such that } x \text{ is misclassified by } h_t(x)} D(x)$$

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
    - $\varepsilon_t = \text{Sum}\,\{½\,(1 - y_i\, h_t(x_i))\}$
  - 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
  - $H(x) = \text{sign}(\Sigma_t\, \alpha_t\, h_t(x))$

Classifier Weight

- Set $a_t = ½ \ln ((1 - e_t)/e_t)$
- The a_t for any classifier is always positive
- The weight for the t^{th} classifier is a function of its error
  - The poorer the classifier is, the closer a_t is to 0
  - If the error of the classifier is exactly 0.5, a_t is 0.
    - We don't trust such classifiers at all J
  - If the error approaches 0, a_t becomes high
    - We trust these classifiers

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

ADA Boost

- 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

- Readjusting the weights of all training instances
  - If the instance is correctly classified, multiply its weight by β (= $\exp(-\alpha_t)$) < 1
  - If it is *misclassified*, multiply its weight by β (= $\exp(\alpha_t)$) > 1

- Renormalize, so they all sum to 1
  - $$D_{renormalized}(x) = D(x) / \sum_{x'} D(x')$$

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

ADA Boost

- 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

Next Class

- Fernando De La Torre
- We will continue with Viola Jones after a few classes