**Machine Learning for Signal Processing**

**Detecting faces in images**

Class 7. 19 Sep 2013

Instructor: Bhiksha Raj

---

**Administrivia**

- Project teams?
- Project proposals?

---

**Last Lecture: How to describe a 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 $f = w_1 V_1 + w_2 V_2 + \ldots + w_k V_k$
  - $V_i$ is used to “correct” errors resulting from using only $V_i$
  - $V_j$, corrects errors remaining after correction with $V_j$
  - And so on.
- $V = [V_1, V_2, V_k]$ can be computed through Eigen analysis

---

**Normalizing out variations: HEQ**

- 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

• 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
- 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
    - 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

Face Detection: A Quick Historical Perspective

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

- The Viola Jones method
  - Boosted cascaded classifiers

- But first, what is boosting

Classification

- Multi-class classification
  - Many possible categories
    - E.g. Sounds "AH", "IY", "UW", "AE."
    - 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

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

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 Chimpanzee
    - 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?

Face Detection as Classification

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

- Classification can be abstracted as follows
- $H : X \rightarrow \{+1, -1\}$
- 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

- The total confidence in all classifiers that classify the entity as a chimpanzee is
  $\text{Score}_{\text{chimpanzee}} = \sum_{\text{classifier}} \text{classifier accuracy} \cdot \text{chimpanzee}$
- The total confidence in all classifiers that classify it as a human is
  $\text{Score}_{\text{human}} = \sum_{\text{classifier}} \text{classifier accuracy} \cdot \text{human}$
- If $\text{Score}_{\text{chimpanzee}} > \text{Score}_{\text{human}}$, 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
• Second strong learner: weighted combination of first and second weak learners
  — Decision boundary shown by black lines

• The second strong learner also makes mistakes
  — Errors colored black

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

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

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

• 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 to train 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 \( \alpha < 1 \)
- Black points (incorrectly classified) will have a weight \( \beta (\sim 1/\alpha) > 1 \)
- To compute the optimal second classifier, we minimize the total weighted error
  - Each data point contributes \( \alpha \) or \( \beta \) 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 \( \beta \)

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
  - \(y_i\) is the class for the \(i^{th}\) instance
    - \(y_i\) 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 \(x\) as \(h_i(x)\)
  - \(h_i(x)\) is either -1 or +1 (for a binary classifier)
  - \(y^T h(x)\) 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 \(h(x)\)
  - The function is succinctly represented as \(f(x)\)
The Boosting Concept

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

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 \( x \) : \( \frac{1}{2}(1 - y H(x)) \)
  - 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 \(-1\)

First, some example data

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

Training Data

- Face = \(+1\)
- Non-face = \(-1\)
The ADABoost Algorithm

- Initialize $D_t(x) = 1/N$
- For $t = 1, ..., T$
  - Train a weak classifier $h_t$ using distribution $D_t$
  - Compute total error on training data
    - $e_t = \text{Sum} \{ D_t(x) \ln (1 - y_t, h_t(x)) \}$
  - Set $\alpha_t = \frac{1}{2} \ln \frac{(1 - e_t) / e_t}$
  - For $i = 1, ..., N$
    - set $D_{t+1}(x) \leftarrow D_t(x) \exp(-\alpha_t y_t h_t(x))$
  - Normalize $D_{t+1}$ to make it a distribution
- The final classifier is
  - $H(x) = \text{sign}(\sum \alpha_t h_t(x))$

Training Data

<table>
<thead>
<tr>
<th></th>
<th>-0.2 E1 + 0.4 E2</th>
<th>+0.2 E1 - 0.4 E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The E1 “Stump”

Classifier based on E1:
- If $\text{sign}\{\text{wt}(E1) > \text{thresh} > 0\}$
- face = true
- sign = +1 or -1

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

The E1 “Stump”

Classifier based on E1:
- If $\text{sign}\{\text{wt}(E1) > \text{thresh} > 0\}$
- face = true
- sign = +1 or -1

Threshold
Sign = 1, error = 1/8
Sign = -1, error = 7/8
The Best “Stump”

The Best Error

Computing Alpha

The ADABOOST Algorithm

• Initialize \( D_1(x) = \frac{1}{N} \)
• For \( t = 1, \ldots, T \)
  - Train a weak classifier \( h_t \) using distribution \( D_t \)
  - Compute total error on training data
    * \( e_t = \sum (D_t(x) \cdot (1 - y_t h_t(x))) \)
  - Set \( \alpha_t = \frac{1}{2} \ln \left( \frac{1}{e_t} \right) \)
    * set \( D_{t+1}(x) = D_t(x) \exp(-\alpha_t y_t h_t(x)) \)
  - Normalize \( D_{t+1} \) to make it a distribution
• The final classifier is
  - \( H(x) = \text{sign}(\sum_t \alpha_t h_t(x)) \)
The ADABOOST Algorithm

- Initialize $D_1(x) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier $h_t$ using distribution $D_t$
  - Compute total error on training data
    - $e_t = \text{Average } (\frac{1}{2} (1 - y, h_t(x)))$
    - Set $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - e_t}{e_t} \right)$
  - For $i = 1 \ldots N$
    - $D_{t+1}(x) = D_t(x) \exp(-\alpha_t y_i h_t(x))$
  - Normalize $D_{t+1}$ to make it a distribution
- The final classifier is:
  - $H(x) = \text{sign} \left( \sum_i \alpha_t h_t(x) \right)$

The Best Error

- $D_t = D / \sum(D)$
- $D_{t+1}(x) = D_t(x) \exp(-\alpha_t y_i h_t(x))$
- $D_i = \exp(-\alpha_i) \exp(0.97) = 2.63$
- Multiply the correctly classified instances by 0.38
- Multiply incorrectly classified instances by 2.63

The ADABOOST Algorithm

- Initialize $D_1(x) = 1/N$
- For $t = 1, \ldots, T$
  - Train a weak classifier $h_t$ using distribution $D_t$
  - Compute total error on training data
    - $e_t = \text{Average } (\frac{1}{2} (1 - y, h_t(x)))$
    - Set $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - e_t}{e_t} \right)$
  - For $i = 1 \ldots N$
    - $D_{t+1}(x) = D_t(x) \exp(-\alpha_t y_i h_t(x))$
  - Normalize $D_{t+1}$ to make it a distribution
- The final classifier is:
  - $H(x) = \text{sign} \left( \sum_i \alpha_t h_t(x) \right)$
### 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 $w(E1)$ is the most important feature

### 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

- 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 bars
- 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(2) = Area(A)
  - PixelSum(3) = Area(A) + Area(B)
  - PixelSum(4) = Area(A) + Area(B) + Area(C)
  - PixelSum(1) = 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) x (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

- 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
  - 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 \( k(f) \) and polarity \( p(f) \) \((p(f) = -1 \text{ or } p(f) = +1)\) such
      that \( f > p(f) \cdot k(f) \) 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 \( \text{error}(f) \)
      - Find the feature \( f' \) such that \( \text{error}(f') \) is lowest
      - The weak learner is the test \( (f' > p(f') \cdot k(f')) \Rightarrow \text{face} \)
    - 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 threshold (weighting from three weak learners)

To Train

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

Multiple faces in the picture

- During tests:
  - Given any new 24x24 image
    - \( R = \sum_{i=1}^n (f_i > k(f)) \)
    - 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 N x M picture, we will perform (N x 24) x (M x 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 N x M picture, we will perform (N x 24) x (M x 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 N x M picture, we will perform (N x 24) x (M x 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 c0.
  - Do this for many scaling factors

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: \[ h(x) = \text{sign}(\sum_{i \in S} \alpha_i h_i(x)) \]
  - Modified classifier \[ h(x) = \text{sign}(\sum_{i \in S} \alpha_i h_i(x) + \gamma) \]
    - \( \gamma \) is a measure of certainty
      - The higher it is, the more certain we are that we found a face
      - If \( \gamma \) is large, then we assume the presence of a face even when we are not sure
      - By increasing \( \gamma \), we can reduce false rejection, while increasing false detection

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
ROC

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

Problem: Not accurate enough, too slow

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

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).

![Image of face detection examples](image1.png)

**Sample results using the Viola-Jones Detector**

- Notice detection at multiple scales

![Image of face detection results](image2.png)

**More Detection Examples**

![Image of more face detection examples](image3.png)

**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)