Recognition
Part II: Face Detection via AdaBoost

Linda Shapiro

CSE 455

What's Coming

1. The basic AdaBoost algorithm (next)
2. The Viola Jones face detector features
3. The modified AdaBoost algorithm that is used in Viola-Jones face detection
4. HW 4

Learning from weighted data

sample	class	weight
1.5 2.6	I	1/2
2.3 8.9	II	1/2

Consider a weighted dataset

- D(i) – weight of *i* th training example ($\mathbf{x}^i$,y^i)
- Interpretations:
  - *i* th training example counts as if it occurred D(i) times
  - If I were to "resample" data, I would get more samples of "heavier" data points

Now, always do weighted calculations:

- e.g., MLE for Naïve Bayes, redefine *Count(Y=y)* to be weighted count:

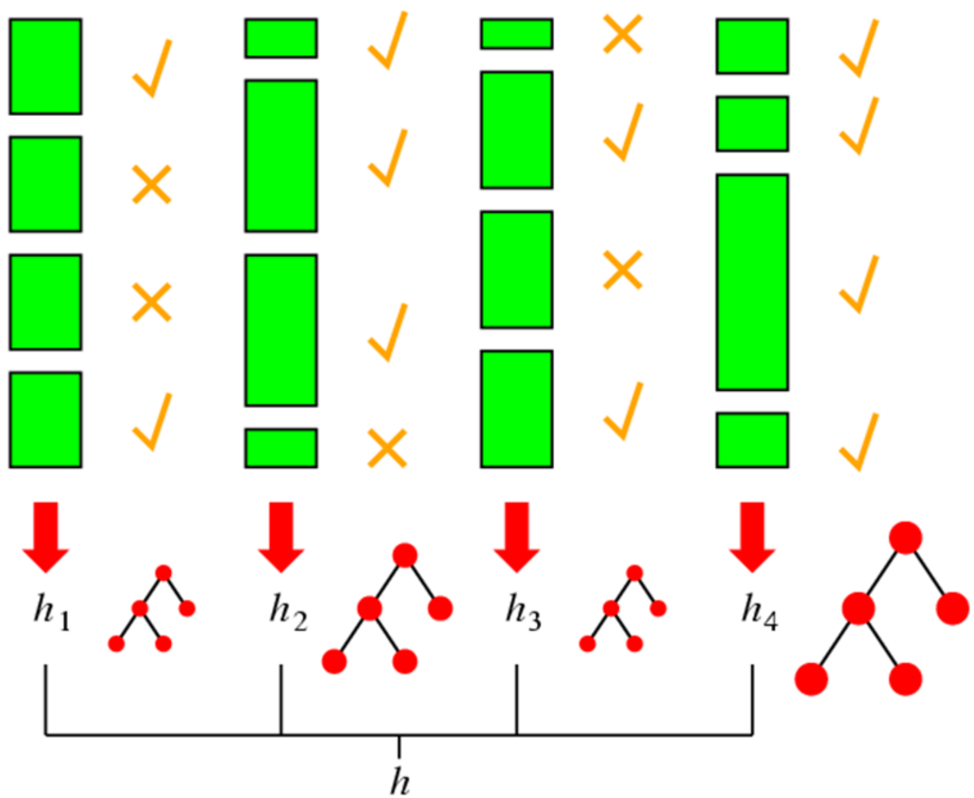
$$Count(Y = y) = \sum_{j=1}^{n} D(j)\delta(Y^j = y)$$

- where δ(P) = 1 when P is true else 0, and
- setting D(j)=1 (or any constant like 1/n) for all j, will recreates unweighted case

AdaBoost Overview

- Input is a set of training examples (X_i, y_i) i = 1 to m.
- We are going to train a sequence of weak classifiers, such as decision trees, neural nets or SVMs. Weak because not as strong as the final classifier.
- The training examples will have weights, initially all equal.
- At each step, we use the current weights, train a new classifier, and use its performance on the training data to produce new weights for the next step.
- But we keep ALL the weak classifiers.
- When it's time for testing on a new feature vector, we will combine the results from all of the weak classifiers.

Idea of Boosting
(from AI text)



m samples 2 classes labeled training data

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$. start with equal weights

For $t = 1, \ldots, T$:

How to choose Many possibilities. Will see one shortly!

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

weight at time *t*+1 for sample *i*

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

update weights

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

sum over *m* samples

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Final Result: linear sum of “base” or “weak” classifier outputs.

Figure 1: The boosting algorithm AdaBoost.

δ(P) = 1 when P is true else 0

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ $y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \ldots, T$:

error $$\boxed{\epsilon_t = \sum_{i=1}^{m} D_t(i)\delta(h_t(x_i) \neq y_i)}$$

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$\boxed{\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)}$$

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

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

α_t is a weight for weak learner h_t.

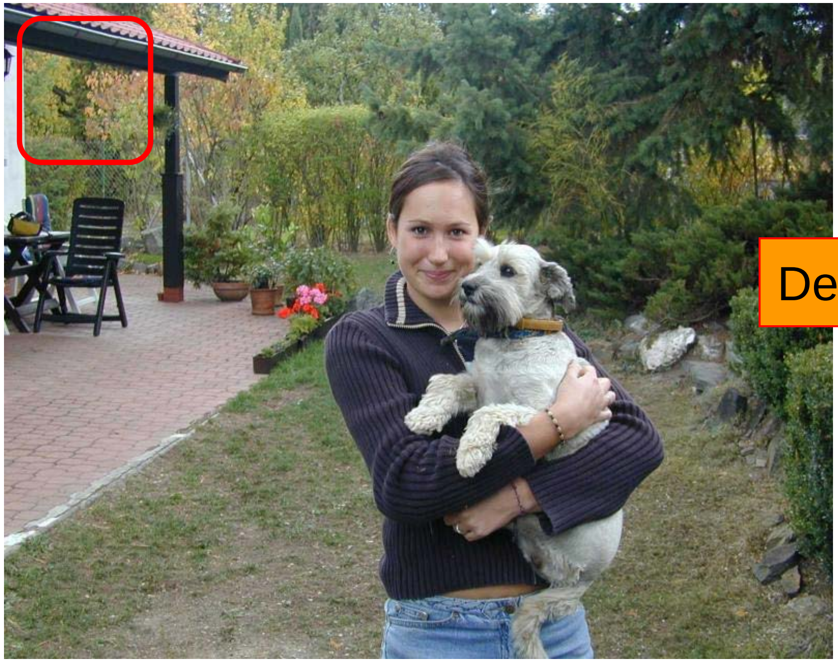
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

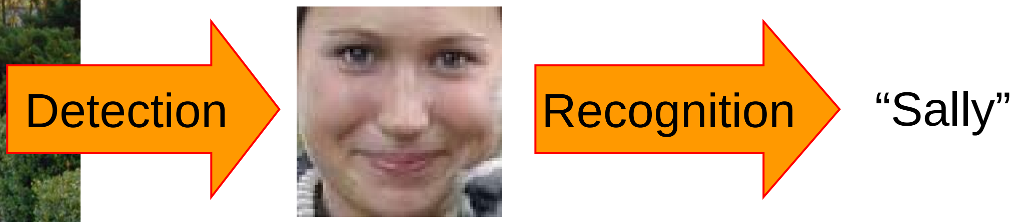
Figure 1: The boosting algorithm AdaBoost.

Face detection

State-of-the-art face detection demo
(Courtesy Boris Babenko)

Face detection and recognition





Face detection

Where are the faces?

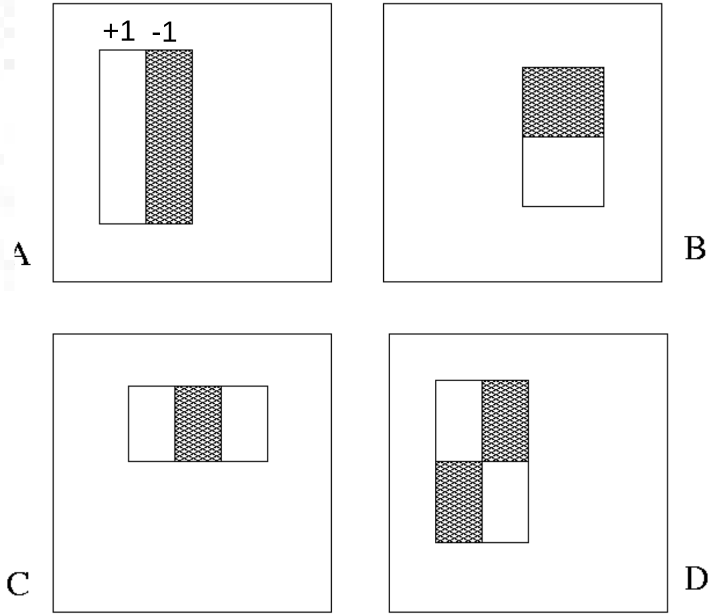
Face Detection

What kind of features?

What kind of classifiers?

Image Features

"Rectangle filters"



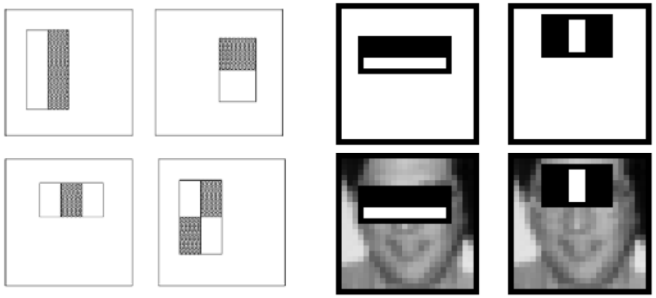
People call them Haar-like features, since similar to 2D Haar wavelets.

Value =

∑ (pixels in white area) –
∑ (pixels in black area)

Feature extraction

“Rectangular” filters



Feature output is difference between adjacent regions

Efficiently computable with integral image: any sum can be computed in constant time

Avoid scaling images scale features directly for same cost

Viola & Jones, CVPR 2001

Recall: Sums of rectangular regions

How do we compute the sum of the pixels in the red box?

After some pre-computation, this can be done in constant time for any box.

This "trick" is commonly used for computing Haar wavelets (a fundemental building block of many object recognition approaches.)

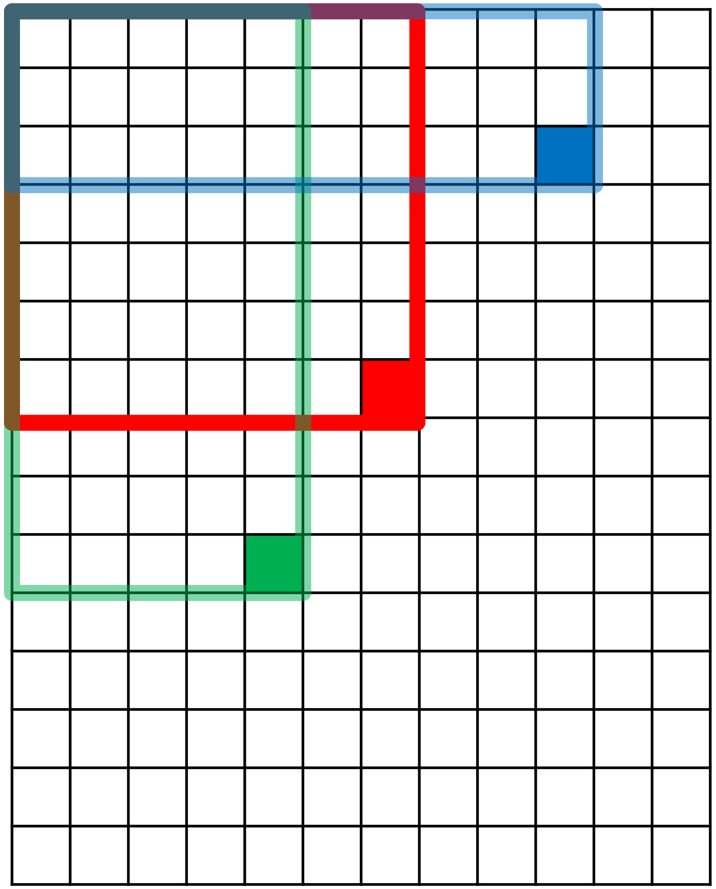
243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	8
17	26	12	160	255	255	109	22	26	19	35	24

Sums of rectangular regions

The trick is to compute an "integral image." Every pixel is the sum of its neighbors to the upper left.

Sequentially compute using:

$$I(x,y) = I(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$



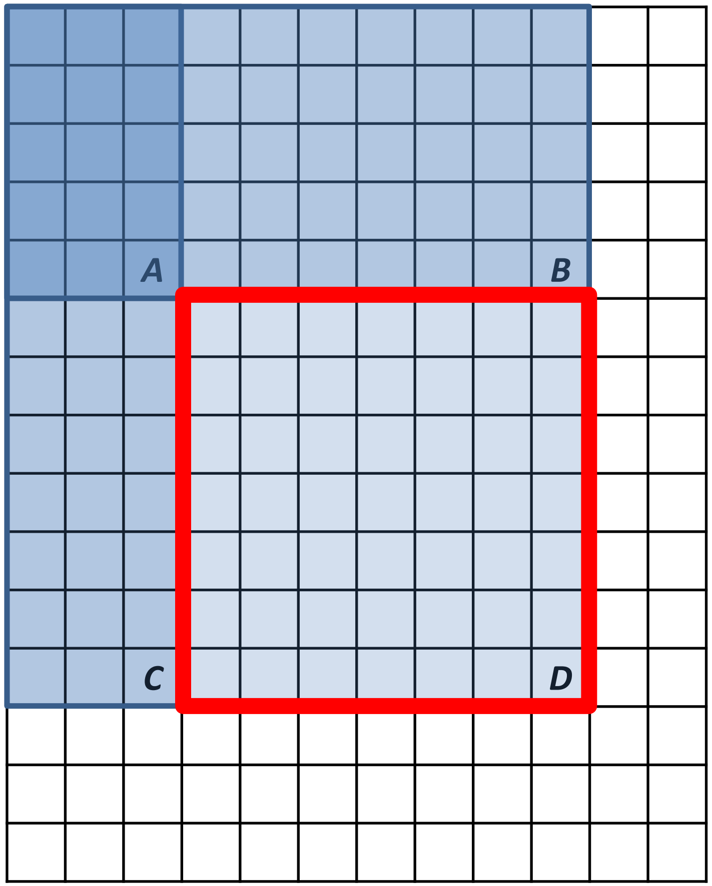
Sums of rectangular regions

Solution is found using:

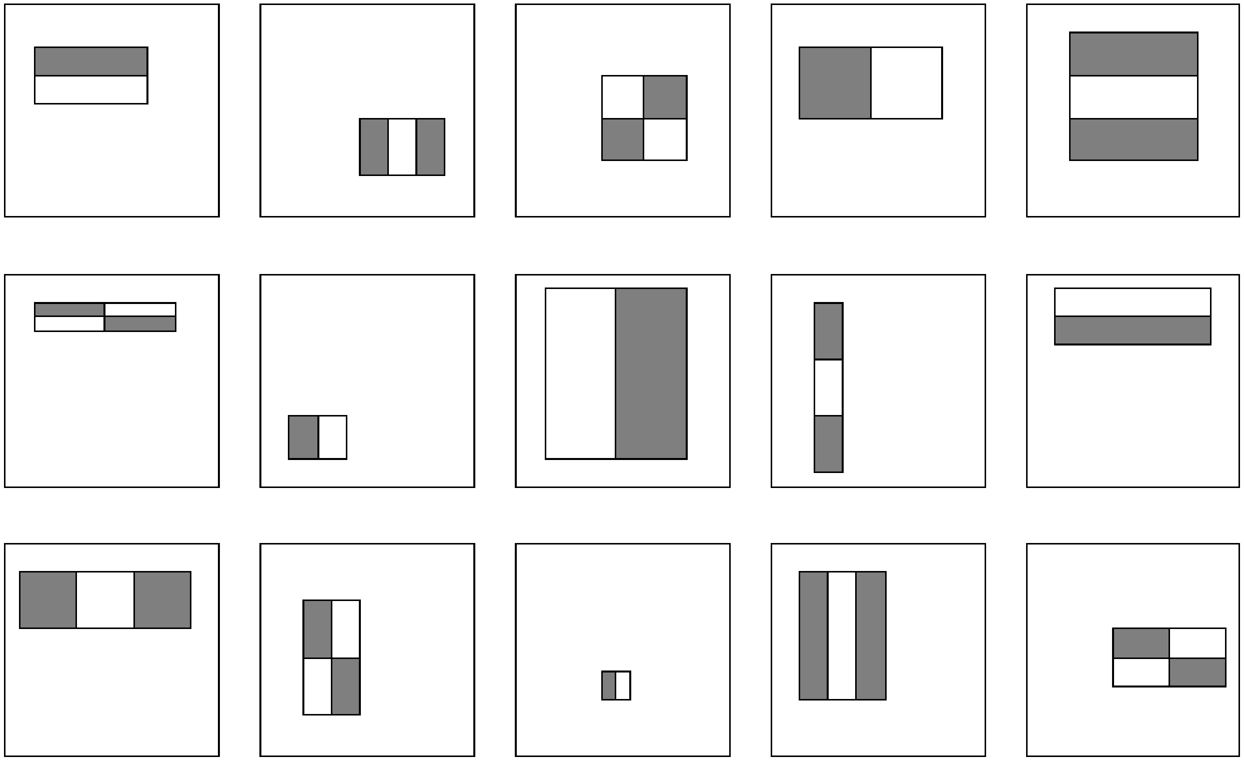
$$A + D - B - C$$

What if the position of the box lies between pixels?

Use bilinear interpolation.



Large library of filters



Considering all possible filter parameters: position, scale, and type:

160,000+ possible features associated with each 24 x 24 window

Use AdaBoost both to select the informative features and to form the classifier

Viola & Jones, CVPR 2001

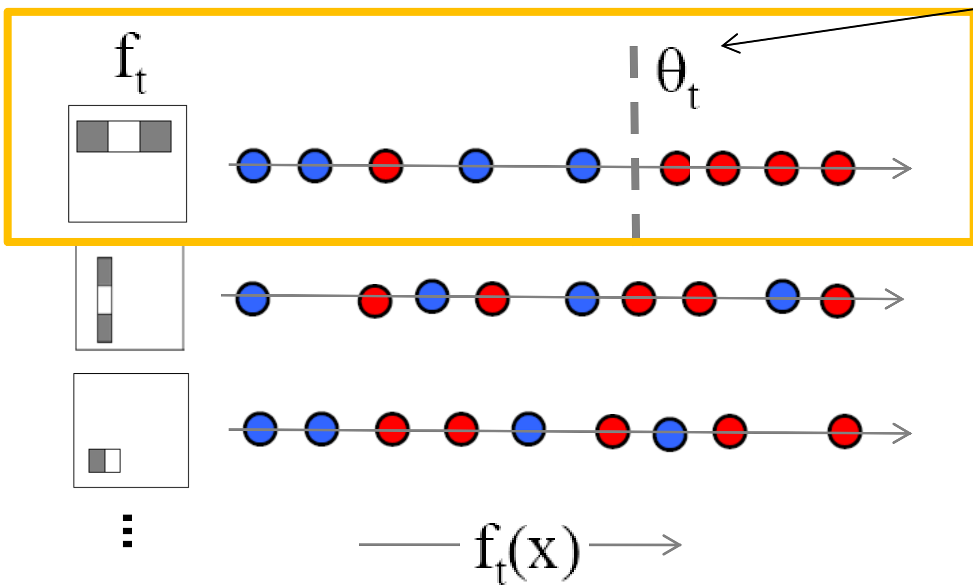
Feature selection

- For a 24x24 detection region, the number of possible rectangle features is ~160,000!
- At test time, it is impractical to evaluate the entire feature set
- Can we create a good classifier using just a small subset of all possible features?
- How to select such a subset?

AdaBoost for feature+classifier selection

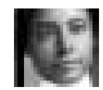
Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of *weighted* error.

θ_t is a threshold for classifier h_t



Outputs of a possible rectangle feature on faces and non-faces.

Resulting weak classifier:



$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ \boxed{0} & \text{otherwise} \end{cases}$$

For next round, reweight the examples according to errors, choose another filter/threshold combo.

Viola & Jones, CVPR 2001

Weak Classifiers

- Each weak classifier works on exactly one rectangle feature.
- Each weak classifier has 3 associated variables
  1. its threshold θ
  2. its polarity p
  3. its weight α

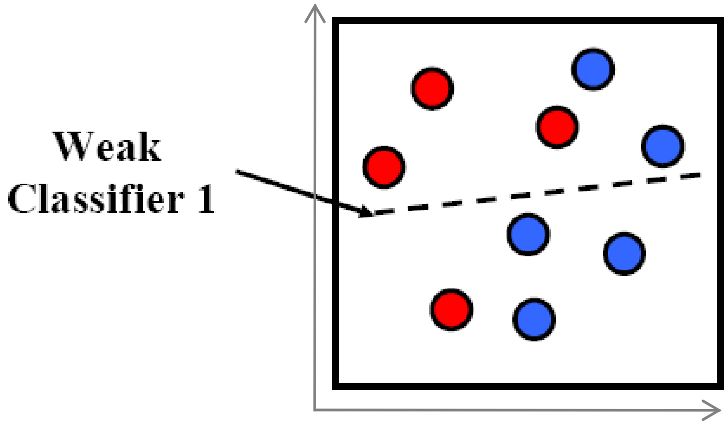
$h(x) = 1$ if $p*f(x) < p\theta$, else 0

used for the combination step

The code does not actually compute h.

- The polarity can be 0 or 1
- The weak classifier computes its one feature f
  - When the polarity is 1, we want $f > \theta$ for face
  - When the polarity is 0, we want $f < \theta$ for face
- The weight will be used in the final classification by AdaBoost.

AdaBoost: Intuition

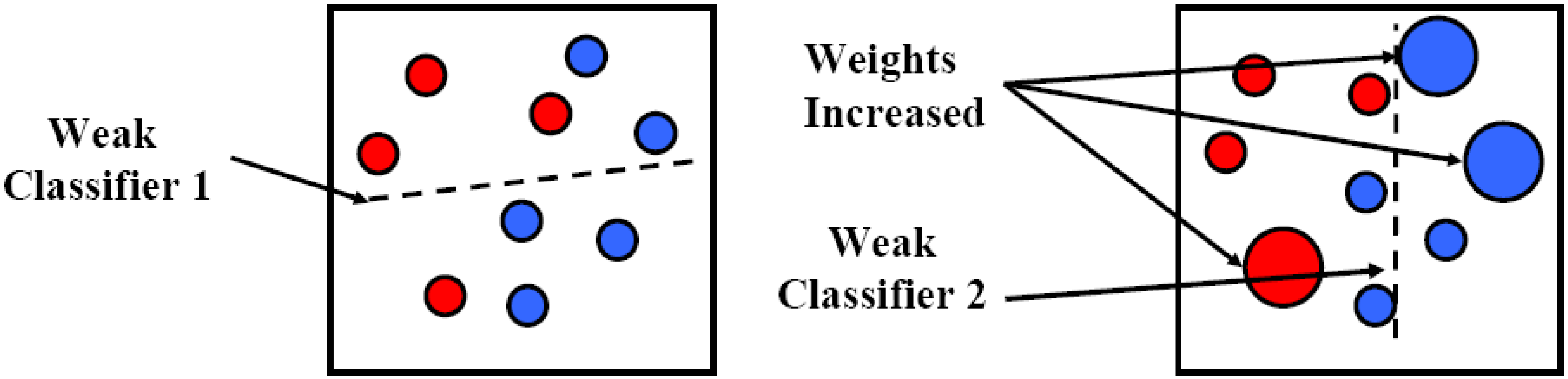


Consider a 2-d feature space with positive and negative examples.

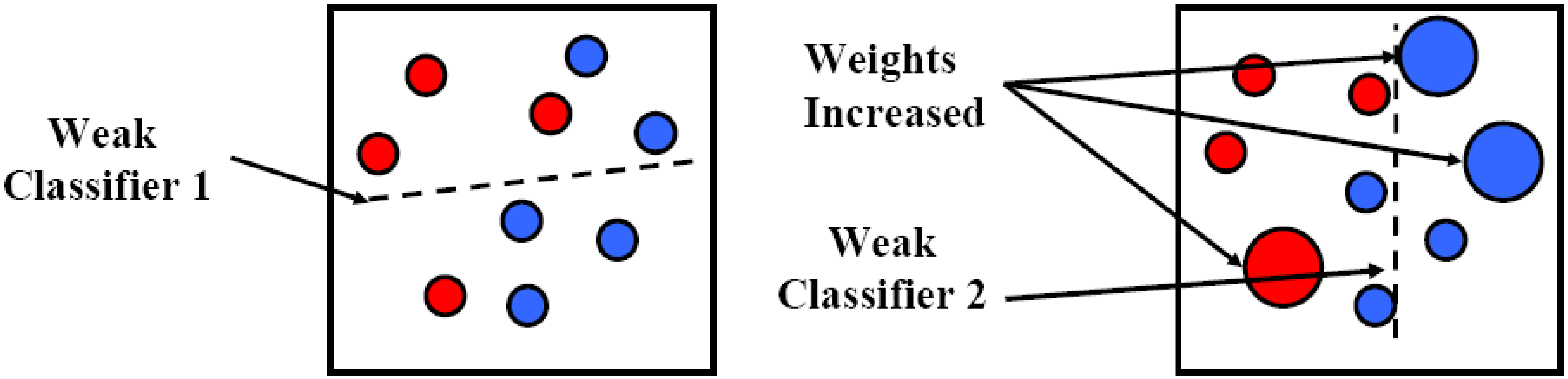
Each weak classifier splits the training examples with at least 50% accuracy.

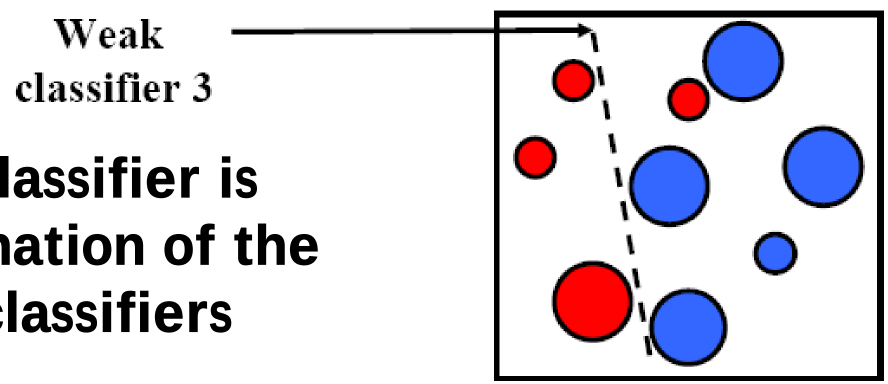
Examples misclassified by a previous weak learner are given more emphasis at future rounds.

AdaBoost: Intuition



AdaBoost: Intuition





Final classifier is combination of the weak classifiers

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

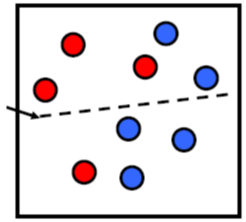
where $\alpha_t = \log \frac{1}{\beta_t}$

$\beta_t = \varepsilon_t / (1- \varepsilon_t)$: the training error of the classifier h_t

Final classifier is combination of the weak ones, weighted according to error they had.

AdaBoost Algorithm modified by Viola Jones

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$. sum over training samples
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.



$\{x_1, \ldots x_n\}$

NOTE: Our code uses equal weights for all samples

For T rounds: meaning we will construct T weak classifiers

Normalize weights

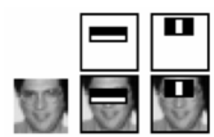
Find the best threshold and polarity for each feature, and return error.

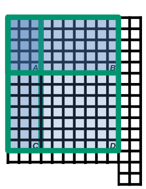
Re-weight the examples:
Incorrectly classified -> more weight
Correctly classified -> less weight

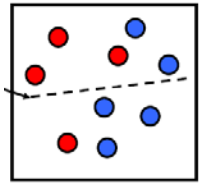
Recall

- Classification
  - Nearest Neighbor
  - Naïve Bayes
  - Decision Trees and Forests
  - Logistic Regression
  - Boosting
  - ....

- Face Detection
  - Simple Features
  - Integral Images
  - Boosting

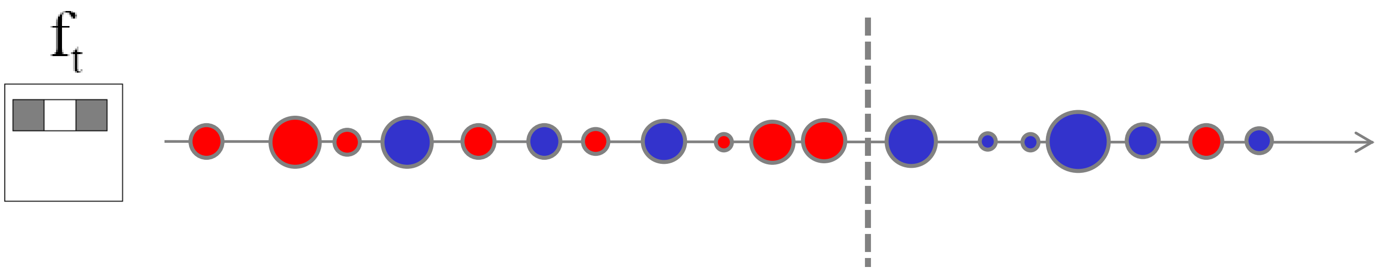






Picking the (threshold for the) best classifier

Efficient single pass approach:



At each sample compute:

$$e = \min\left(S + (T - S),\ S + (T - S) \right)$$

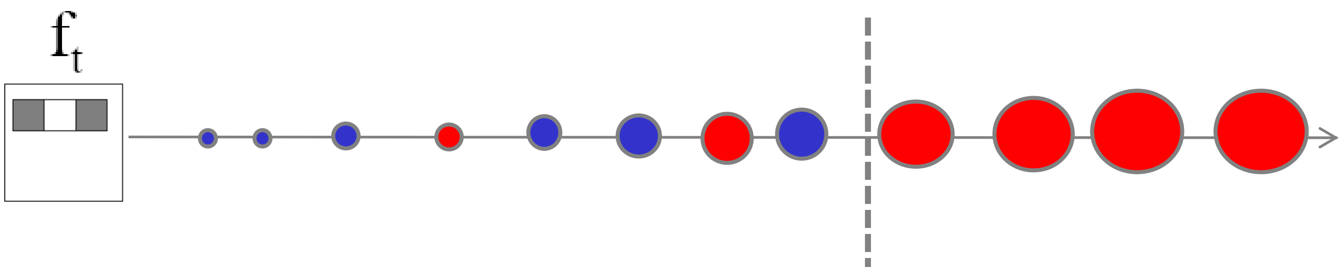
Find the minimum value of e, and use the value of the corresponding sample as the threshold.

S = sum of samples with feature value below the current sample
T = total sum of all samples
S and T are for faces; S and T are for background.

Picking the threshold for the best classifier

Efficient single pass approach:

The features are actually **sorted** in the code according to numeric value!



At each sample compute:

$$e = \min\left(S + (T - S),\ S + (T - S) \right)$$

Find the minimum value of e, and use the value of the corresponding sample as the threshold.

S = sum of weights of samples with feature value below the current sample
T = total sum of all samples
S and T are for faces; S and T are for background.

Picking the threshold for the best classifier

The features for the training samples are actually **sorted** in the code according to numeric value!

Algorithm:

1. find AFS, the sum of the weights of all the face samples
2. find ABG, the sum of the weights of all the background samples
3. set to zero FS, the sum of the weights of face samples so far
4. set to zero BG, the sum of the weights of background samples so far
5. go through each sample s in a loop IN THE SORTED ORDER

   At each sample, add weight to FS or BG and compute:

$$e = \min(BG + (AFS - FS), FS + (ABG - BG))$$

Find the minimum value of e, and use the feature value of the corresponding sample as the threshold.

What's going on?

$$\text{error} = \min\left(\underbrace{BG + (AFS - FS)}_{\text{left}},\ \underbrace{FS + (ABG - BG)}_{\text{right}}\right)$$

- Let's pretend the weights on the samples are all 1's.
- The samples are arranged in a sorted order by feature value and we know which ones are faces (f) and background (b).

- Left is the number of background patches so far plus the number of faces yet to be encountered.

- Right is the number of faces so far plus the number of background patches yet to be encountered.

1+5-0 0+5-1

b	b	b	f	b	f	f	b	f	f
(6,4)	(7,3)	(8,2)	(7,3)	(8,2)	(7,3)	(4,4)	(7,3)	(6,4)	(5,5)
4	3	**2**	3	2	3	4	3	4	5

Measuring classification performance

- Confusion matrix

		Predicted class		
		Class1	Class2	Class3
Actual class	Class1	40	1	6
	Class2	3	25	7
	Class3	4	9	10

- Accuracy
  - (TP+TN)/(TP+TN+FP+FN)
- True Positive Rate=Recall
  - TP/(TP+FN)
- False Positive Rate
  - FP/(FP+TN)
- Precision
  - TP/(TP+FP)
- F1 Score
  - 2*Recall*Precision/(Recall+Precision)

		Predicted	
		Positive	Negative
Actual	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

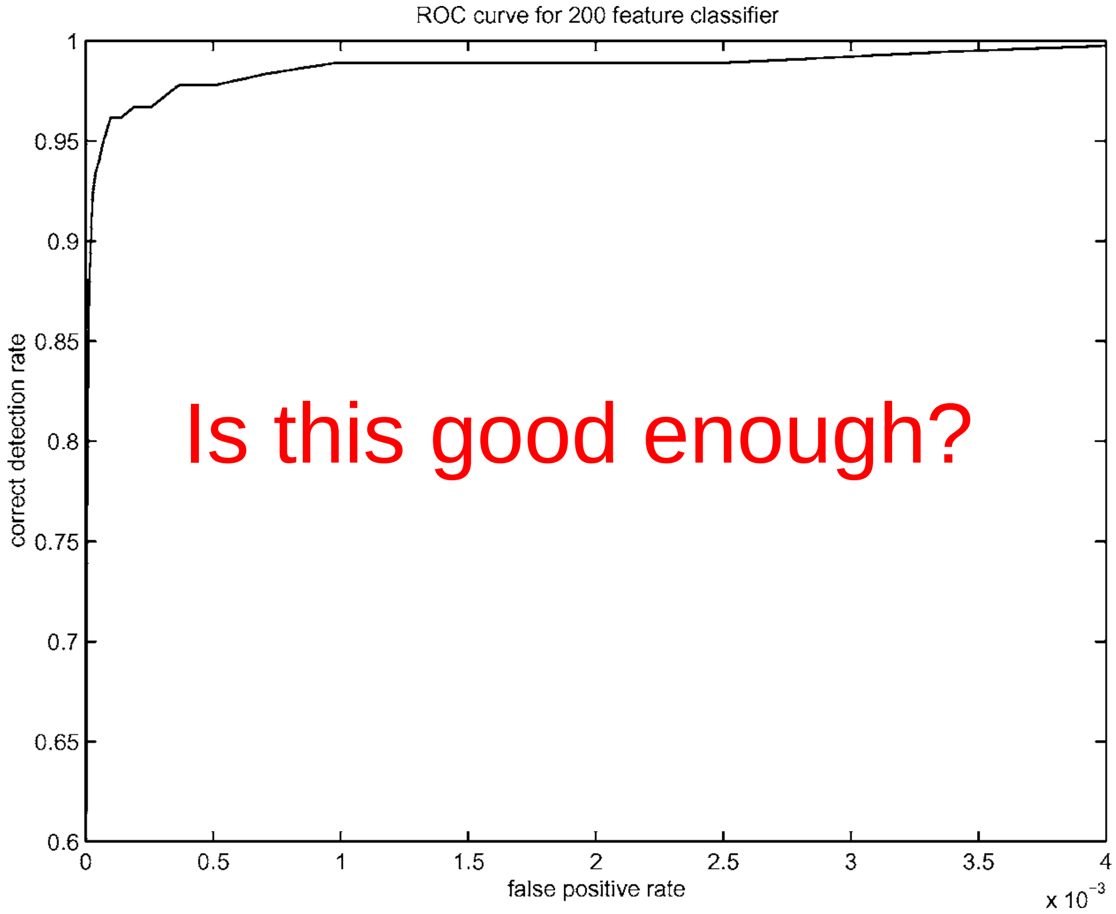
Boosting for face detection

- First two features selected by boosting:

This feature combination can yield 100% detection rate and 50% false positive rate

Boosting for face detection

- A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084

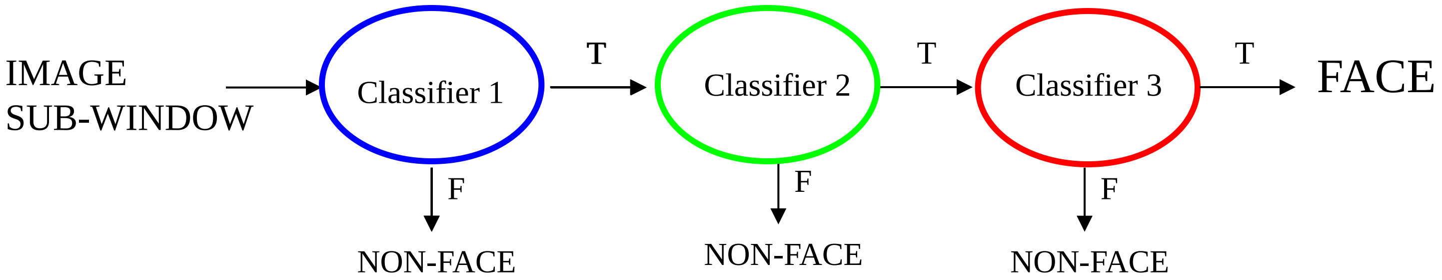


Receiver operating characteristic (ROC) curve

Attentional cascade (from Viola-Jones)

This part will be **extra credit** for HW4

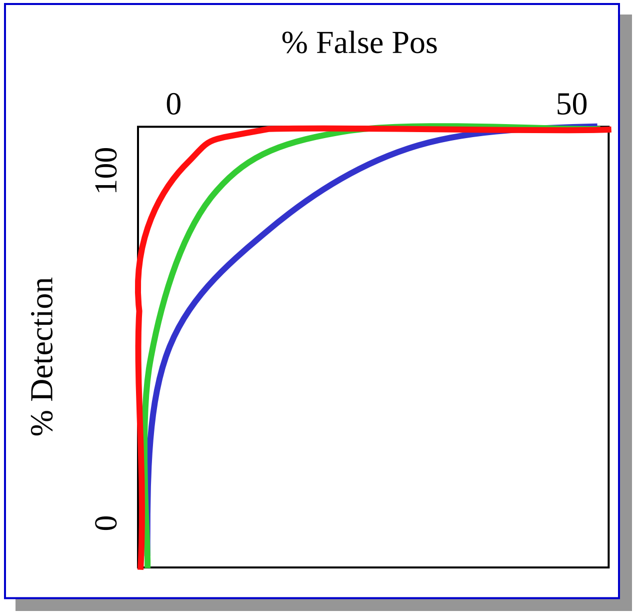
- We start with simple classifiers which reject many of the negative sub-windows while detecting almost all positive sub-windows
- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on
- A negative outcome at any point leads to the immediate rejection of the sub-window

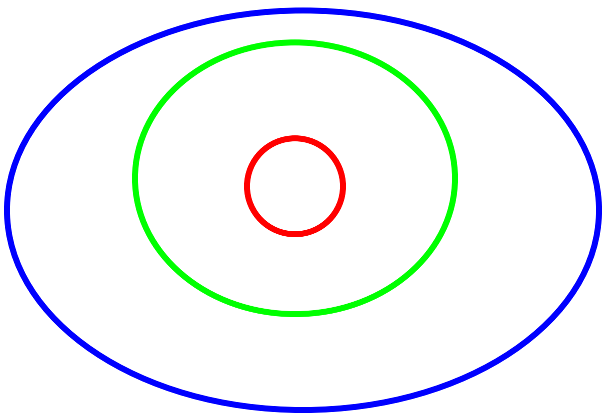


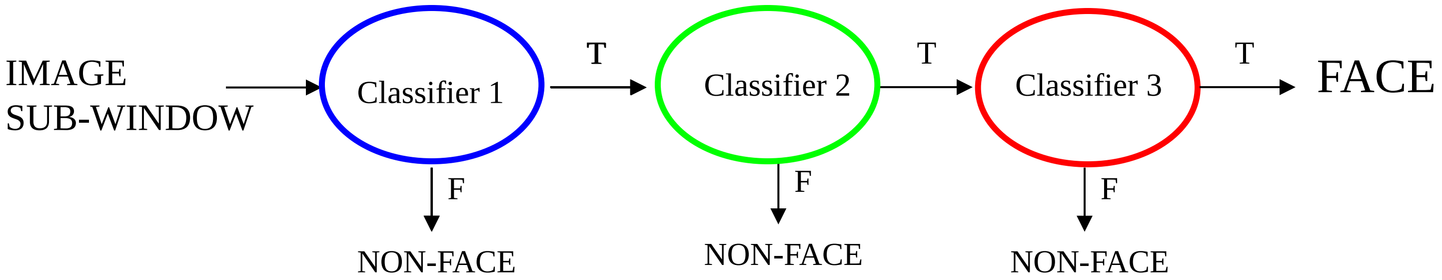
Attentional cascade

- Chain of classifiers that are progressively more complex and have lower false positive rates:

Receiver operating characteristic

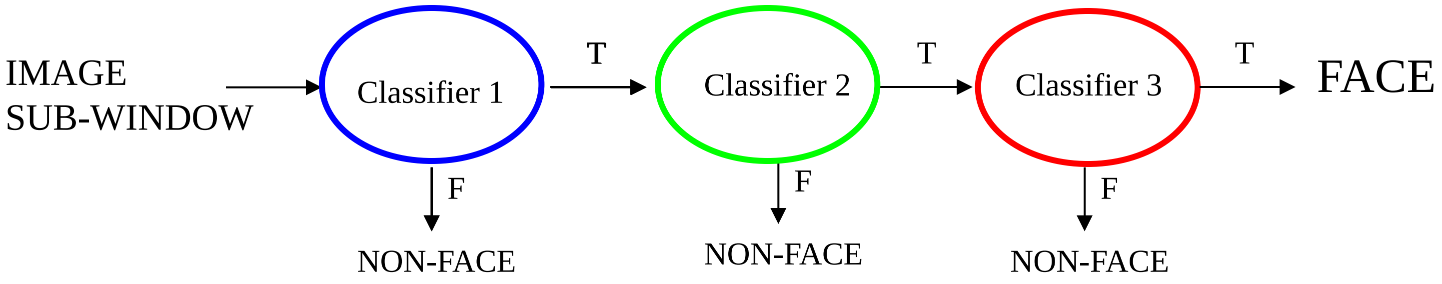






Attentional cascade

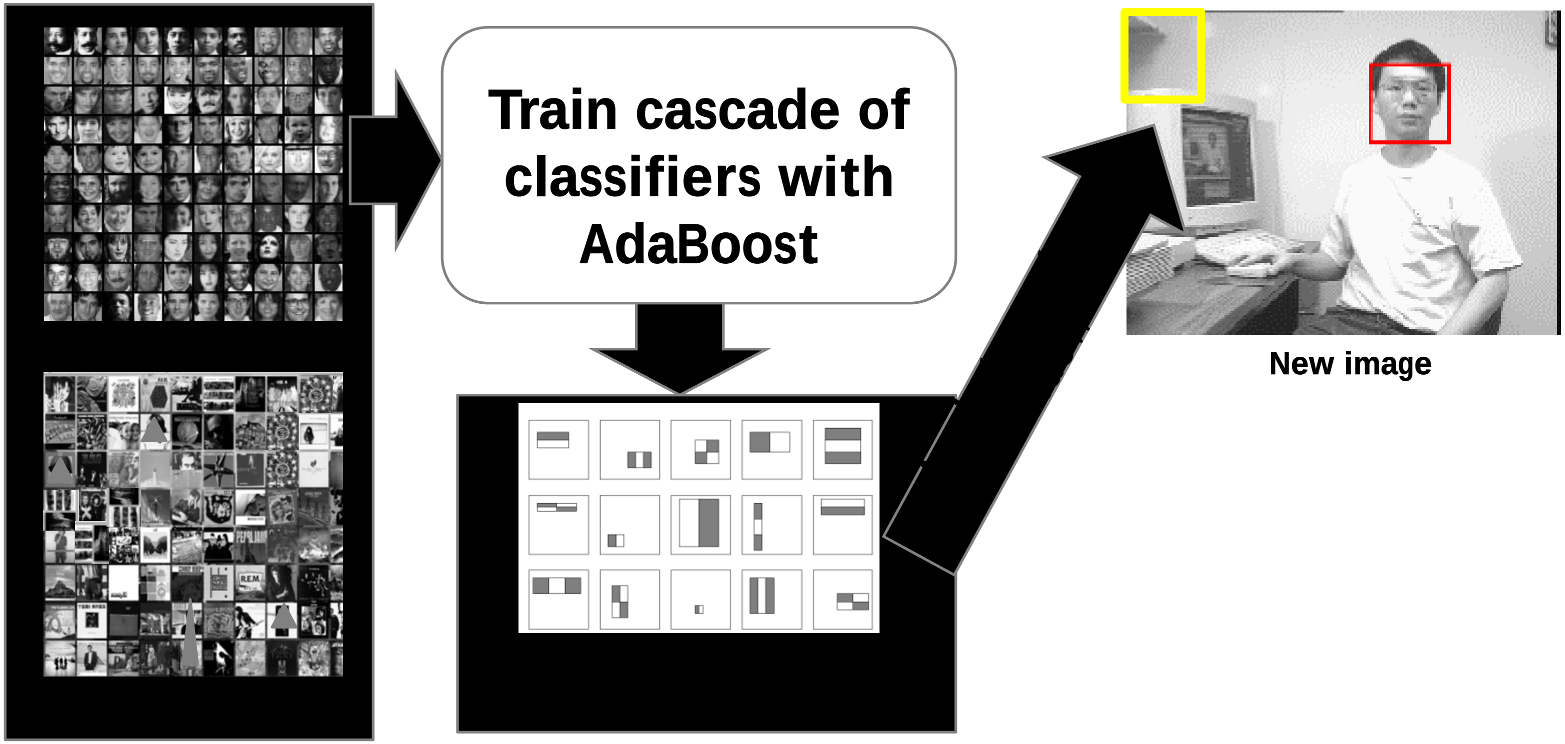
- The detection rate and the false positive rate of the cascade are found by multiplying the respective rates of the individual stages
- A detection rate of 0.9 and a false positive rate on the order of 10^{-6} can be achieved by a 10-stage cascade if each stage has a detection rate of 0.99 ($0.99^{10} \approx 0.9$) and a false positive rate of about 0.30 ($0.3^{10} \approx 6\times10^{-6}$)



Training the cascade

- Set target detection and false positive rates for each stage
- Keep adding features to the current stage until its target rates have been met
  - Need to lower AdaBoost threshold to maximize detection (as opposed to minimizing total classification error)
  - Test on a *validation set*
- If the overall false positive rate is not low enough, then add another stage
- Use false positives from current stage as the negative training examples for the next stage

Viola-Jones Face Detector: Summary



Train with 5K positives, 350M negatives

Real-time detector using 38 layer cascade

6061 features in final layer

[Implementation available in OpenCV:
http://www.intel.com/technology/computing/opencv/]

The implemented system

- Training Data
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 300 million non-faces
    - 9500 non-face images
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose

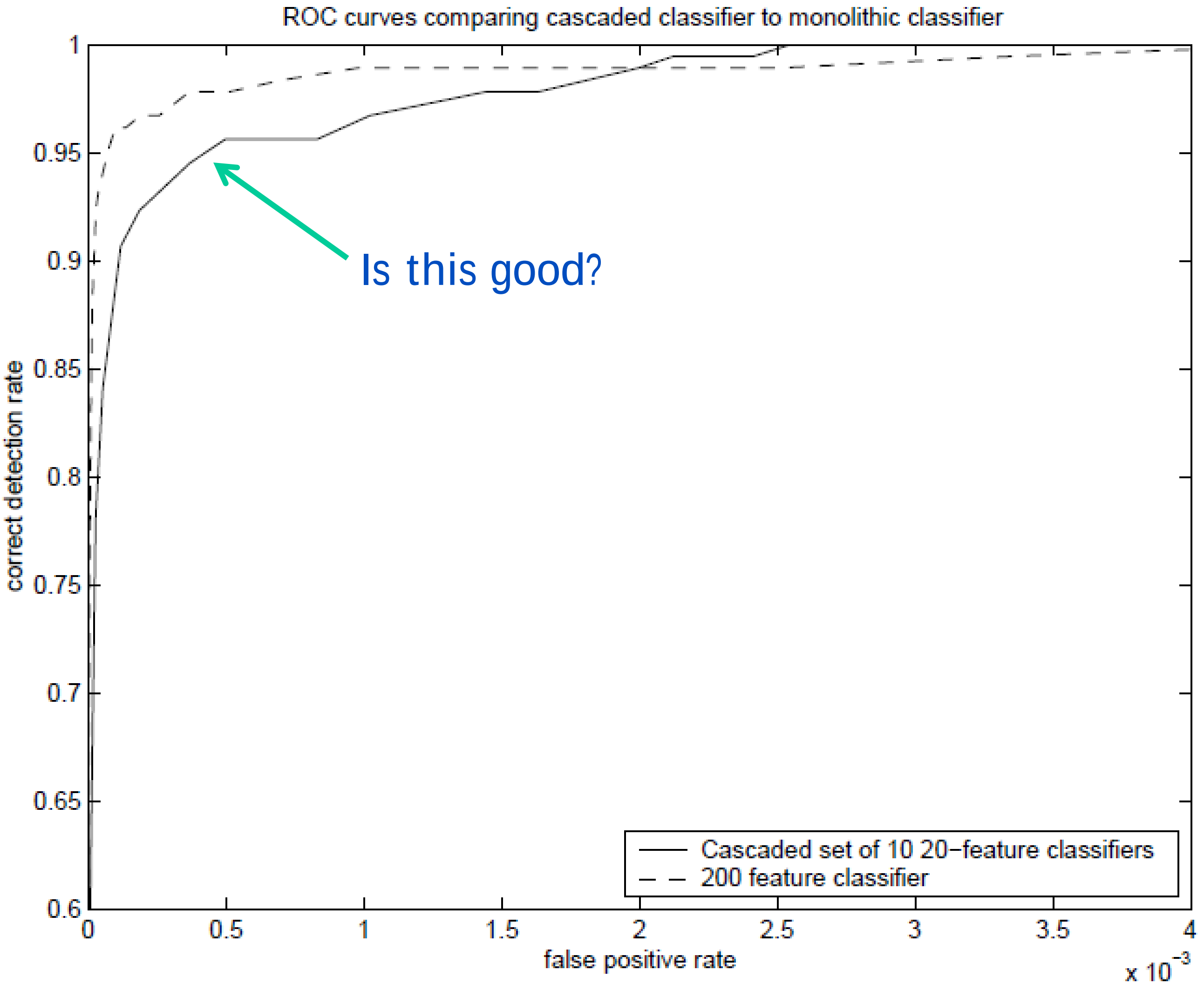
System performance

- Training time: “weeks” on 466 MHz Sun workstation
- 38 layers, total of 6061 features
- Average of 10 features evaluated per window on test set
- “On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds”
    - 15 Hz
    - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)

Non-maximal suppression (NMS)

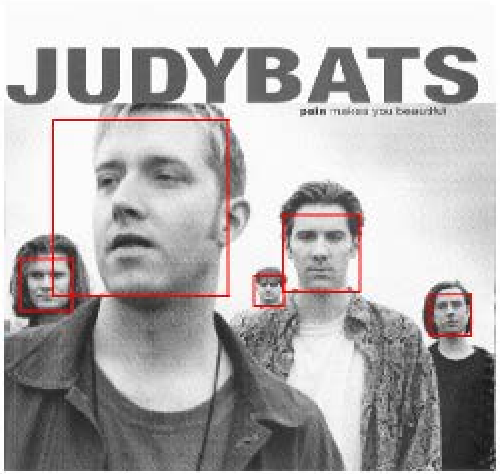
Many detections above threshold.

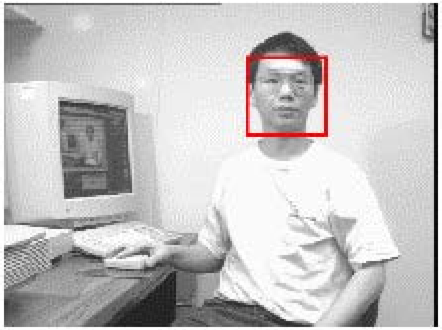
Non-maximal suppression (NMS)

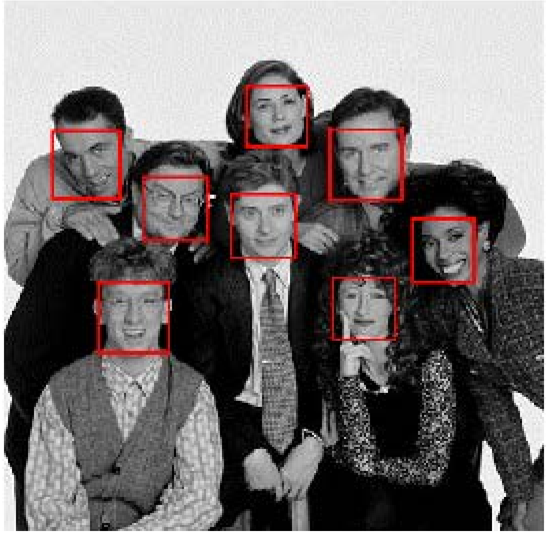


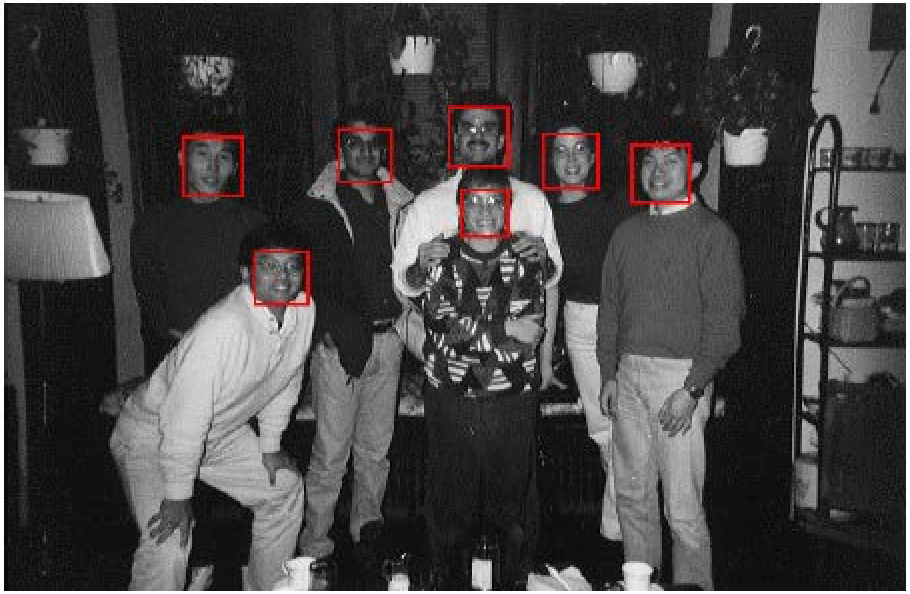
Similar accuracy, but 10x faster

Viola-Jones Face Detector: Results

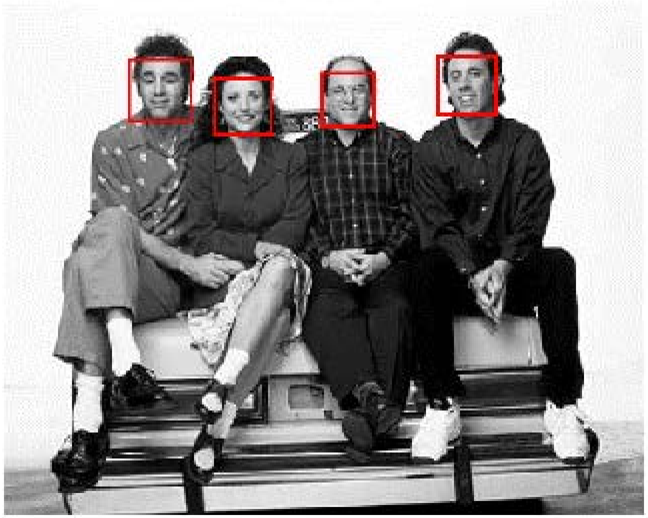


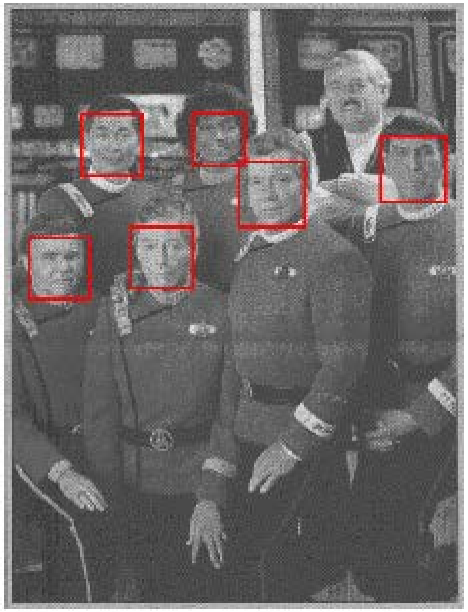


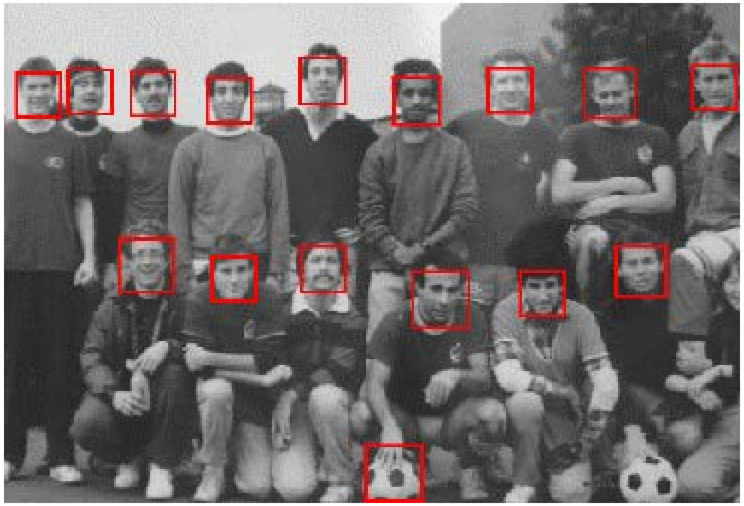


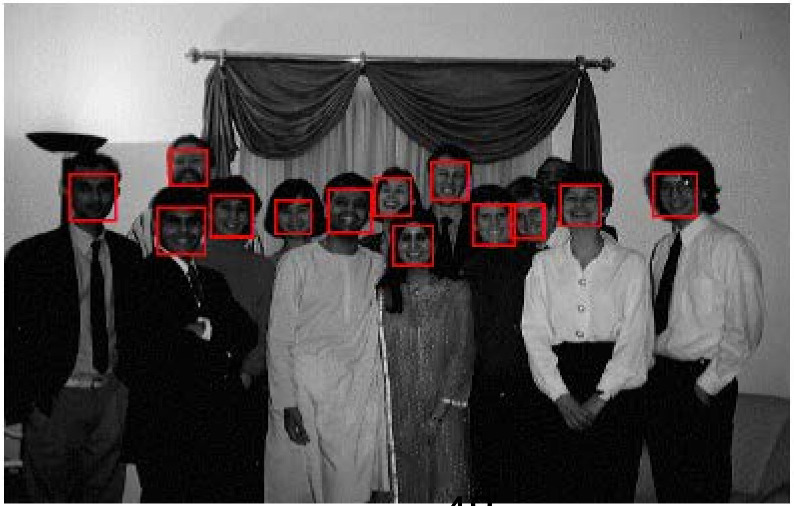


Viola-Jones Face Detector: Results





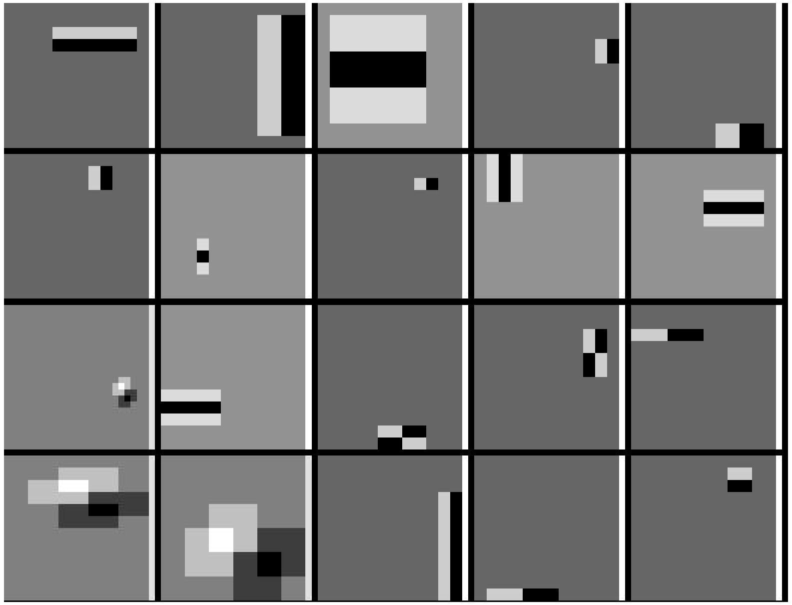




Viola-Jones Face Detector: Results

Detecting profile faces?

Detecting profile faces requires training separate detector with profile examples.



Viola-Jones Face Detector: Results

Summary: Viola/Jones detector

- Rectangle features
- Integral images for fast computation
- Boosting for feature selection
- Attentional cascade for fast rejection of negative windows