ECE 417
Adaboost Face Detection

4/12/2016

Outline

- Today:
  - Integral Image
  - Scalar Classifier
  - Adaboost
- Thursday:
  - Walk-through of MP6 with matlab open on screen
  - Some background theory of adaboost

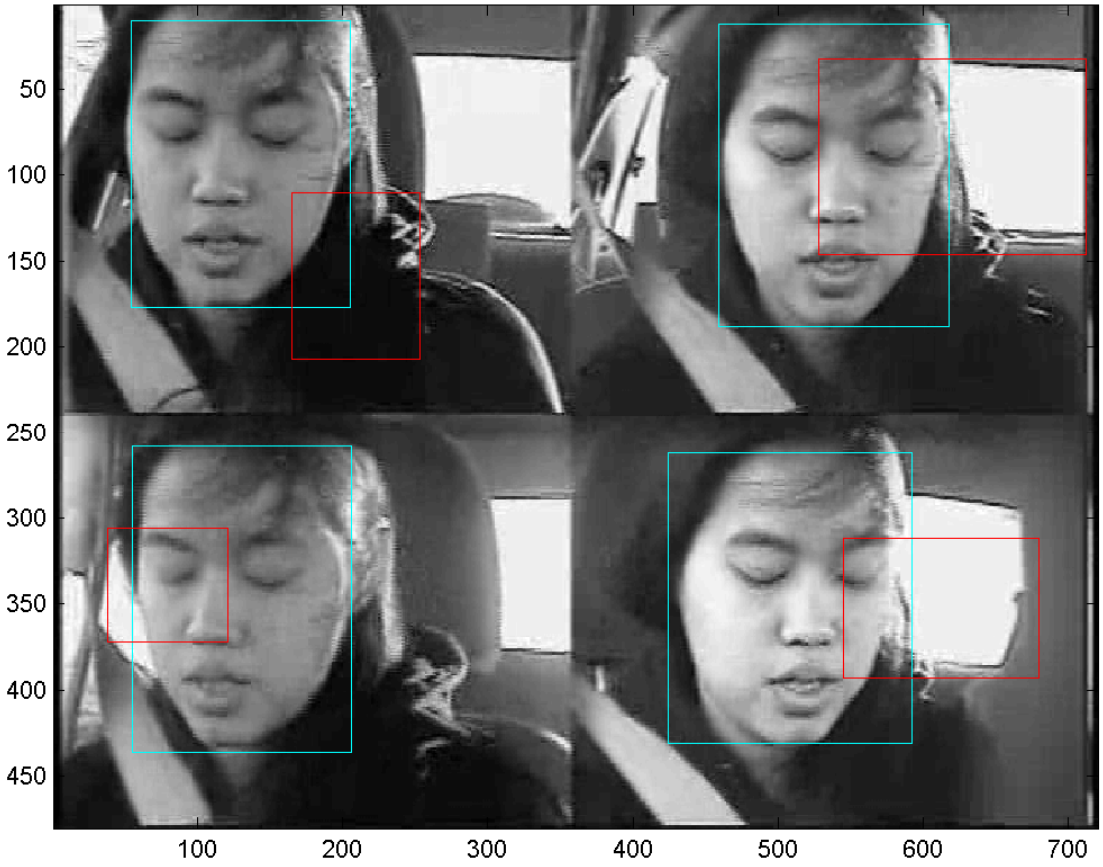
AVICAR database

rects.txt:
12 rectangles per line: lips, face, other
4 ints/rectangle:
[xmin,ymin,width,height]

showrects.m plots
Yellow: lips (first 4/line)
Cyan: face (next 4/line)
Red: other (next 4/line)

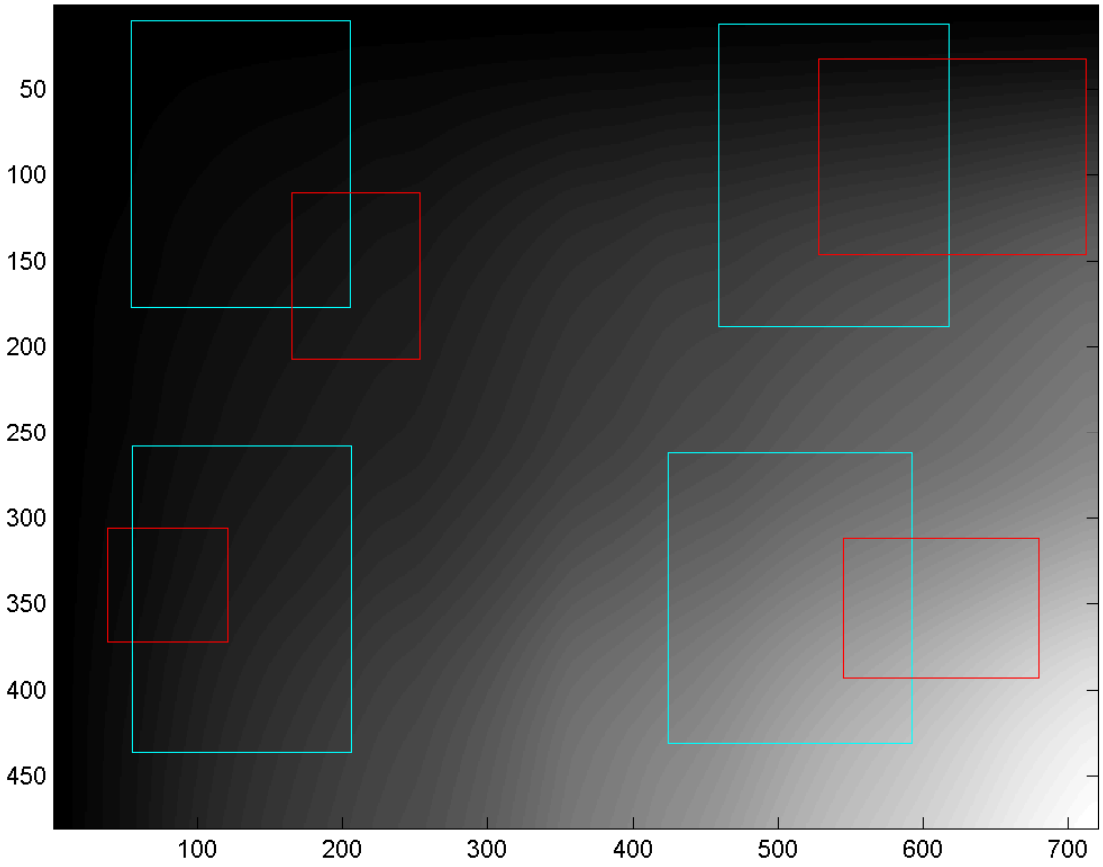
MP:
Discriminate face vs. other

Grayscale image




```
i = sum(a,3);
imagesc(i);
```

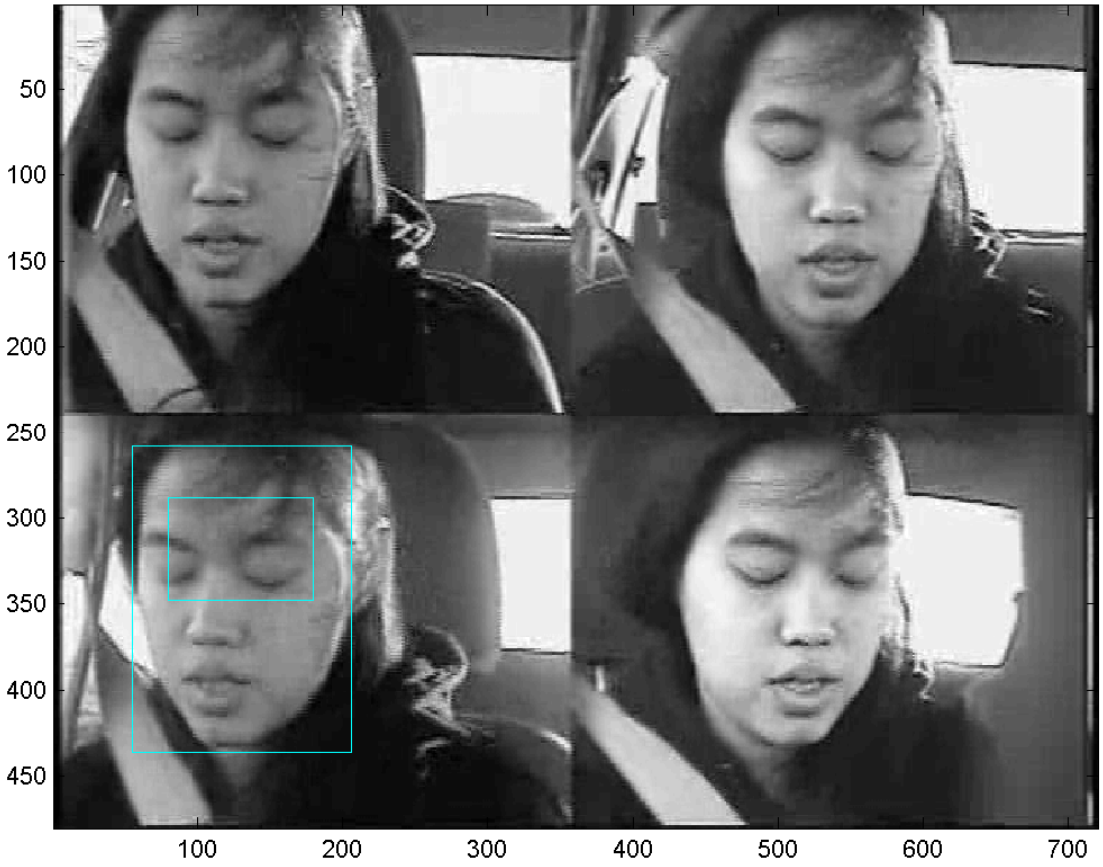
Integral image



$$ii(y,x) = \sum_{x' \le x, y' \le y} i(y', x')$$

ii = cumsum(cumsum(i,2),1);
imagesc(ii);

Scalar features: subrectangles



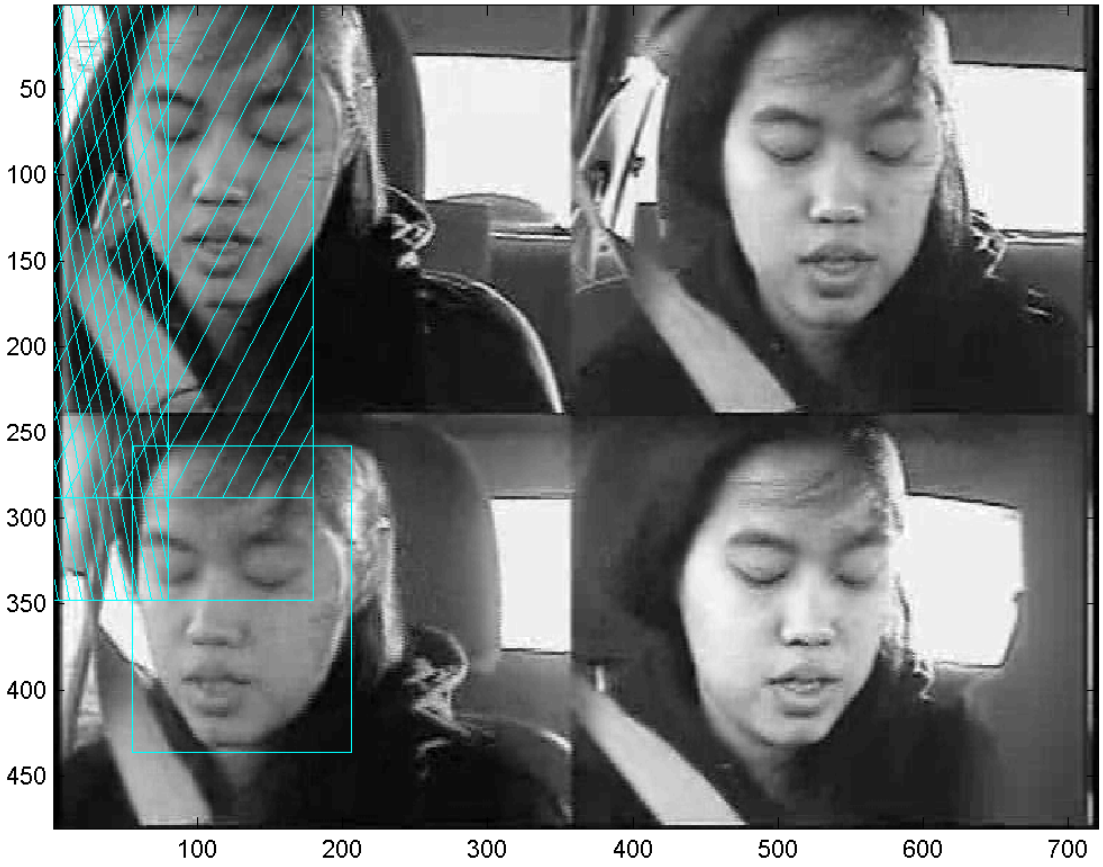
The small cyan rectangle is a sub-rectangle of the big cyan rectangle.

Small rectangle: r'=[xm',ym',w',h']
Big rectangle: r=[xm,ym,w,h]

Relationship:
xm' = xm + fx*w
ym' = ym + fy*h
w' = fw*w
h' = fh*h

The "fractional subrectangle" is
fr = [fx,fy,fw,fh]=[1,1,4,1]/6;

Efficiently computing the sum

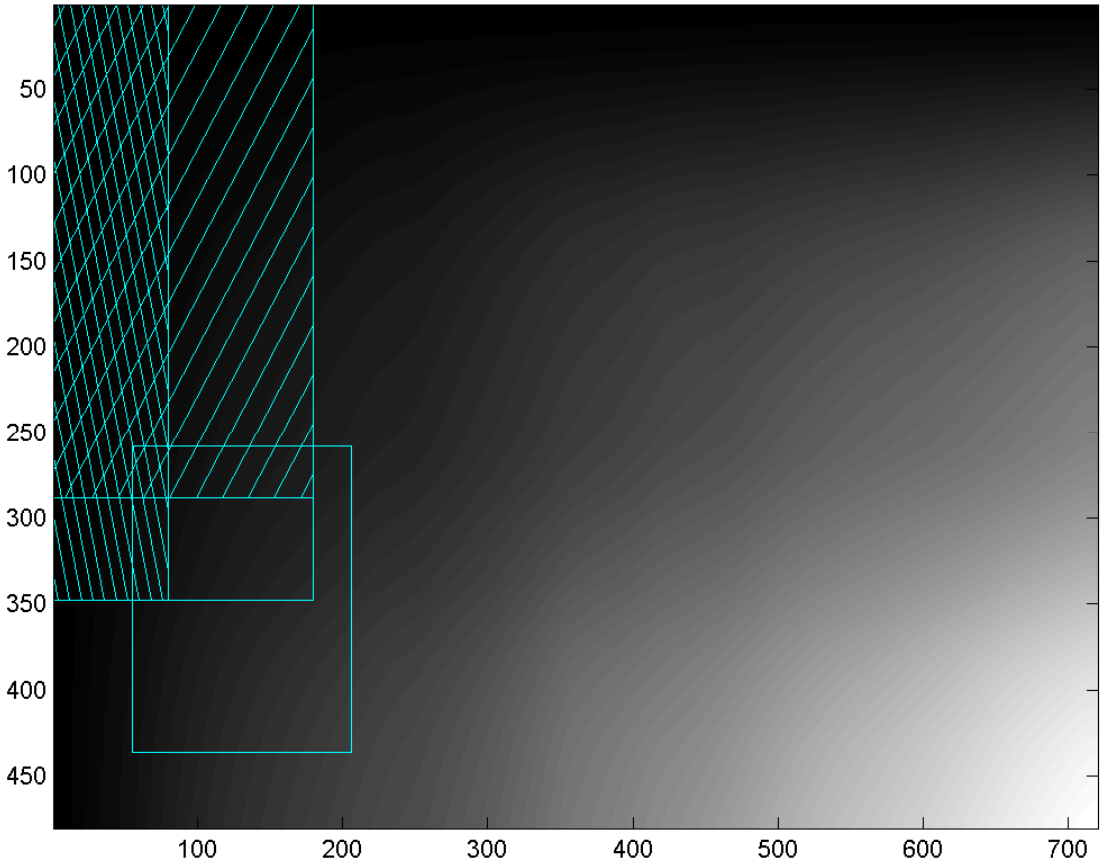


The sum within the subrectangle is:

sum(open pixels) –
sum(/// pixels) –
sum(\\\ pixels) +
sum(### pixels)

The last term is necessary because, by subtracting the two previous terms, we have subtracted the ### pixels twice; it is necessary to compensate.

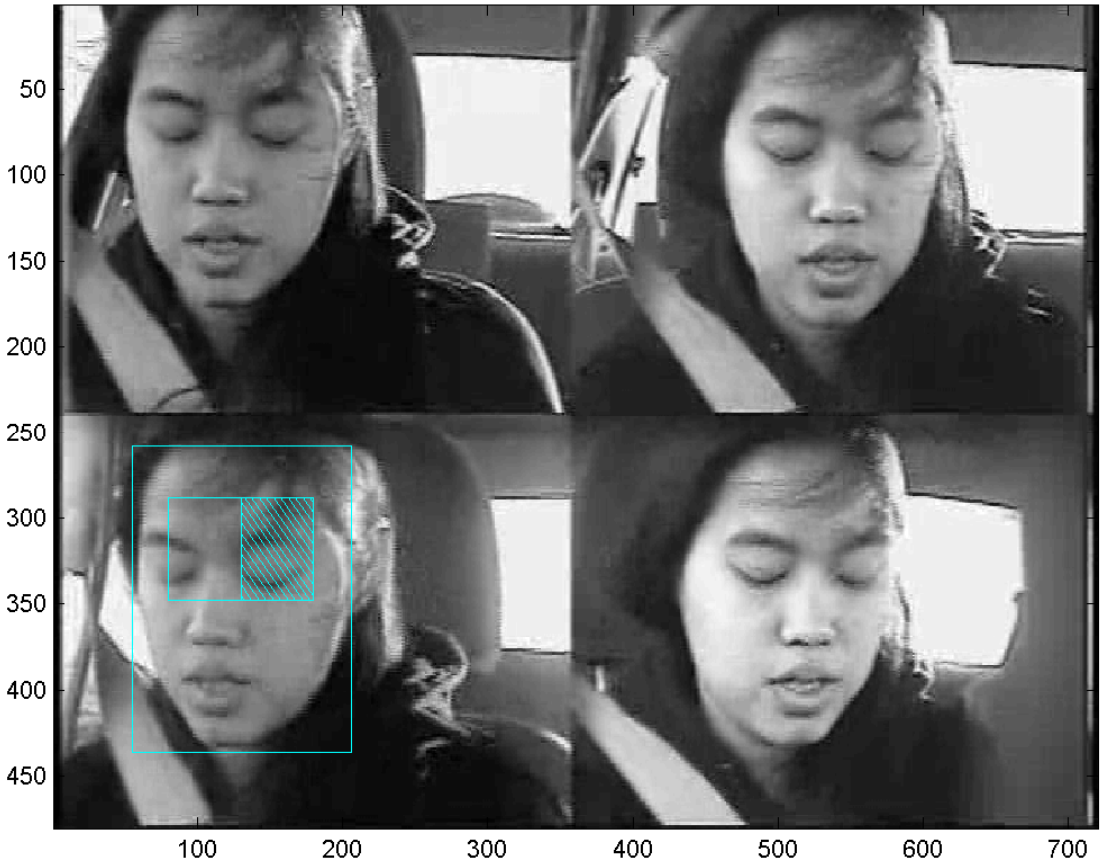
Efficiently computing the sum



In the integral image, each point is a sum! Thus the feature we want is just

ii(y2,x2) –
ii(y1,x2) –
ii(y2,x1) +
ii(y1,x1)

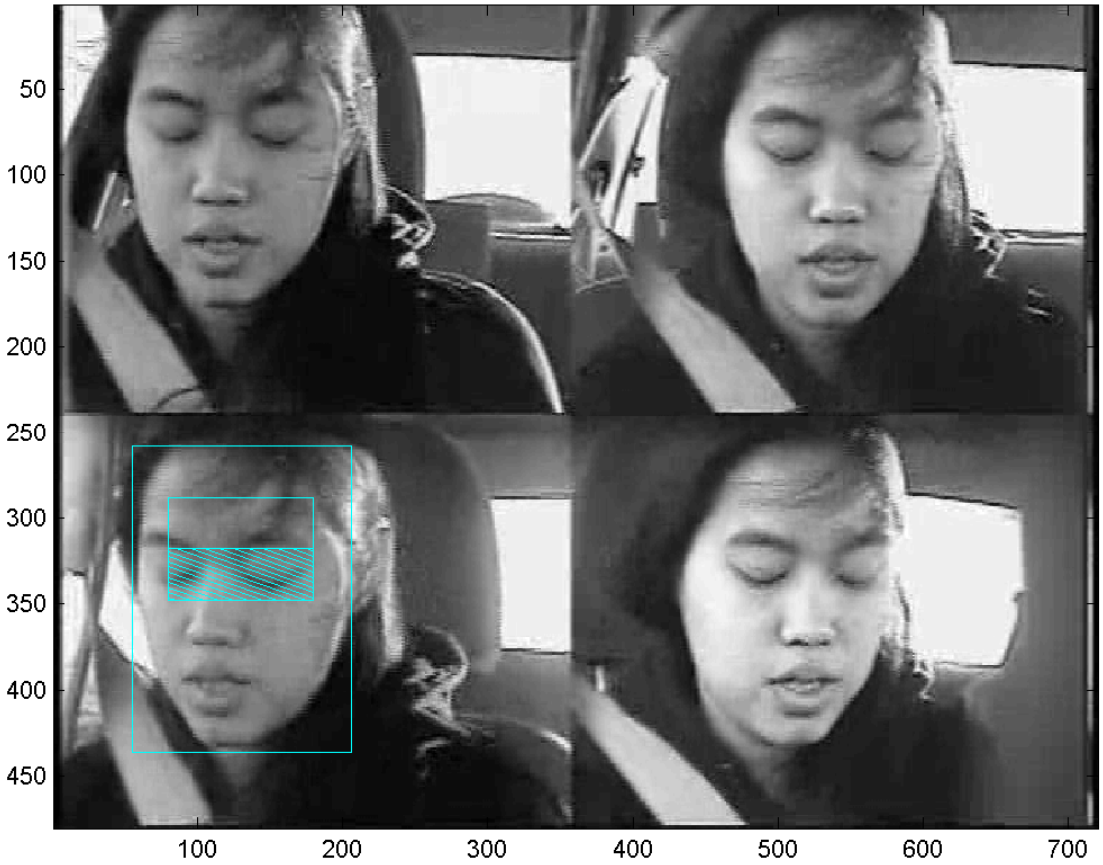
Other useful features: order 2, horizontal



Feature f(x;fr,q=2,v=0)

An order-2 horizontal feature is the sum of the right half, minus the sum of the left half.

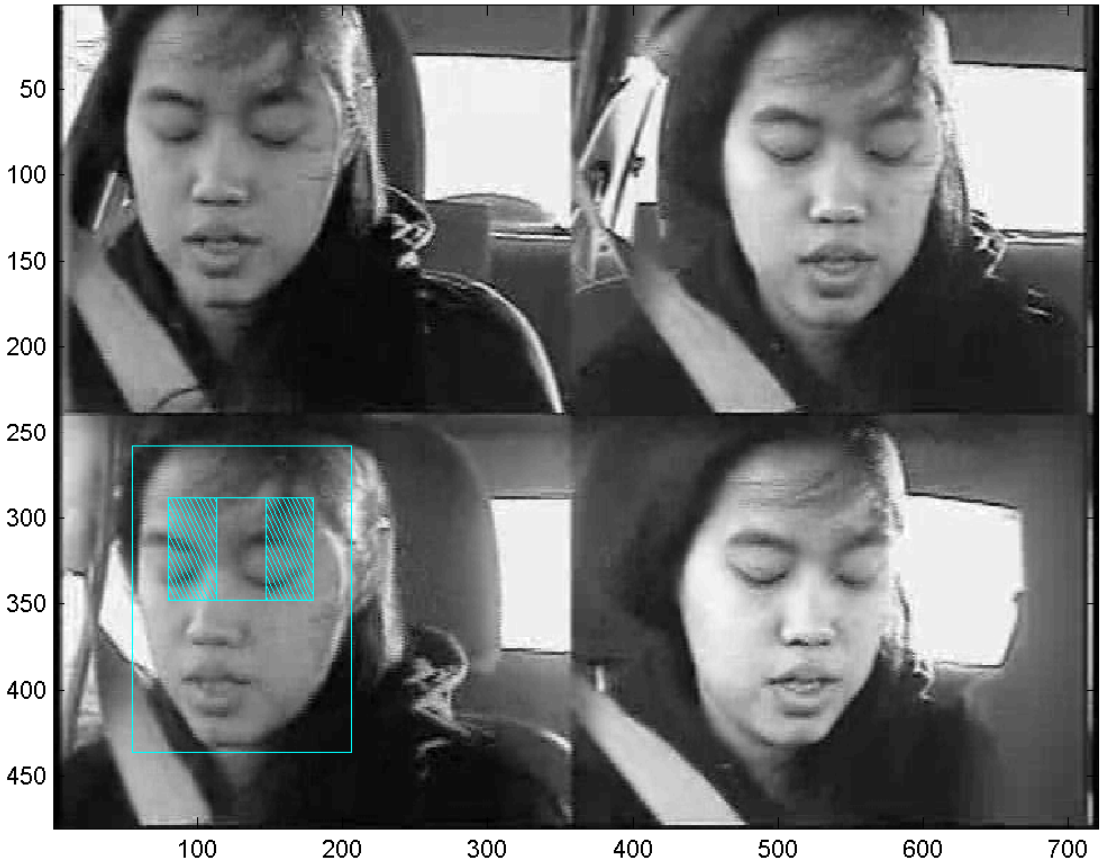
Other useful features: order 2, vertical



Feature f(x;fr,q=2,v=1)

An order-2 vertical feature is the sum of the bottom half, minus the sum of the top half.

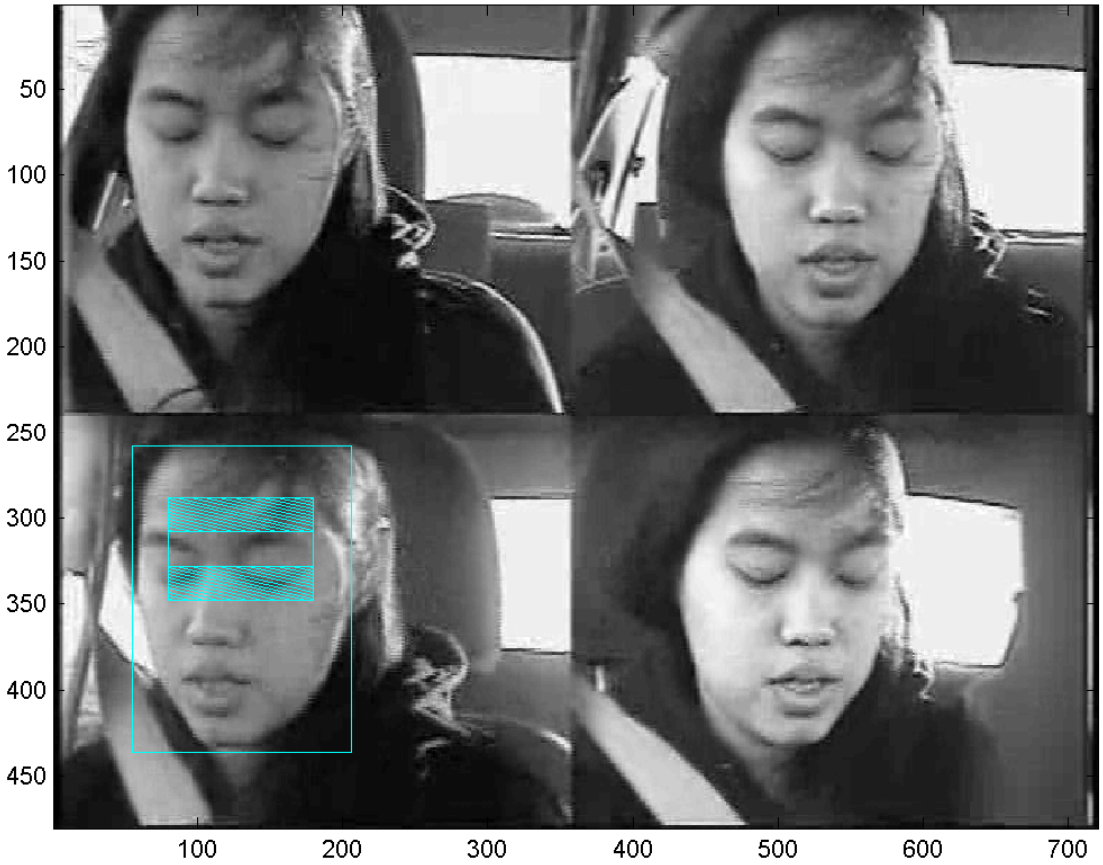
Other useful features: order 3, horizontal



Feature f(x;fr,q=3,v=0)

An order-3 horizontal feature is the sum of the outer thirds, minus the sum of the middle third.

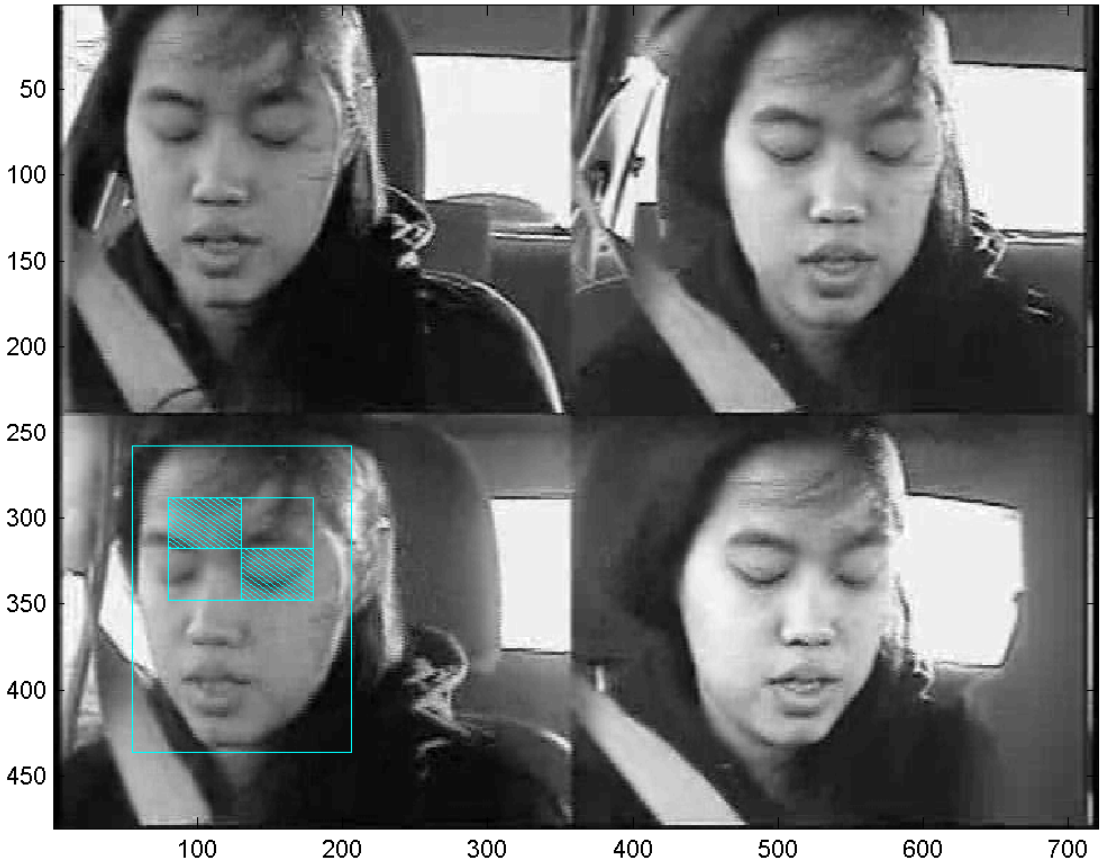
Other useful features: order 3, vertical



Feature f(x;fr,q=3,v=1)

An order-3 vertical feature is the sum of the outer thirds, minus the sum of the middle third.

Other useful features: order 4



Feature f(x;fr,q=4)

An order-4 feature is the sum of the main diagonal quadrants, minus the sum of the off-diagonal quadrants.

Scalar Classifier

$$h(x; fr, q, v, p, \theta) = \begin{cases} 1 & p\, f(x; fr, q, v) < p\, \theta \\ 0 & otherwise \end{cases}$$

In other words, the p=1 classifier is given by

$$h(x; fr, q, v, p = 1, \theta) = \begin{cases} 1 & f(x; fr, q, v) < \theta \\ 0 & otherwise \end{cases}$$

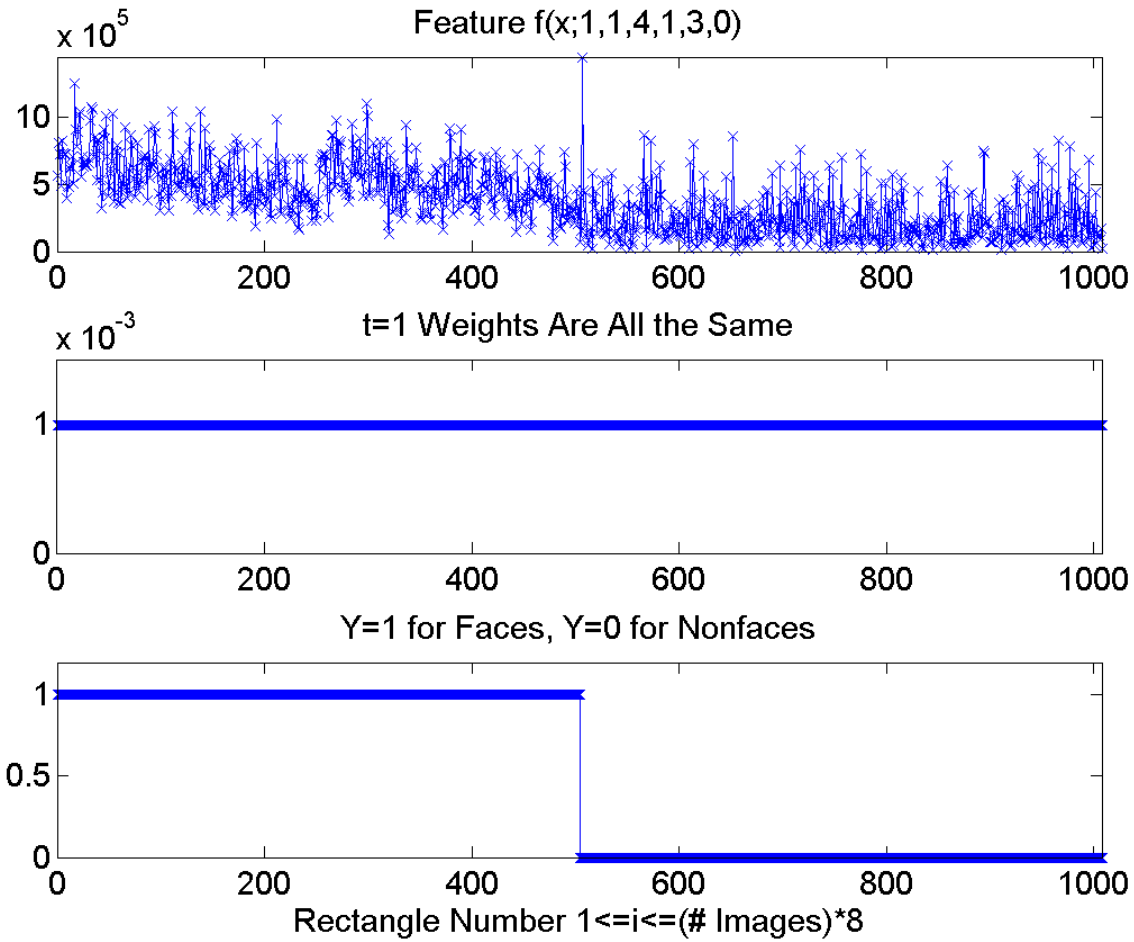
And the p=-1 classifier is given by

$$h(x; fr, q, v, p = -1, \theta) = \begin{cases} 1 & f(x; fr, q, v) \geq \theta \\ 0 & otherwise \end{cases}$$

How to find fx,fy,fw,fh,q,v: Exhaustive search!!!!!!

```
for ix=0:5,
  for iy=0:5,
    for iw=1:(6-ix),
      for ih=1:(6-iy),
        <fr = [ix,iy,iw,ih]/6>
        <compute subrectangle from rectangle>
        for q=1:4,
          for v=0:1,
            <compute features>
            <find the p,Θ that give the lowest weighted error>
            <compare it with the best so far, and save it if it's better>
end; end; end; end; end; end
```

How to find p,Θ: find the minimum error

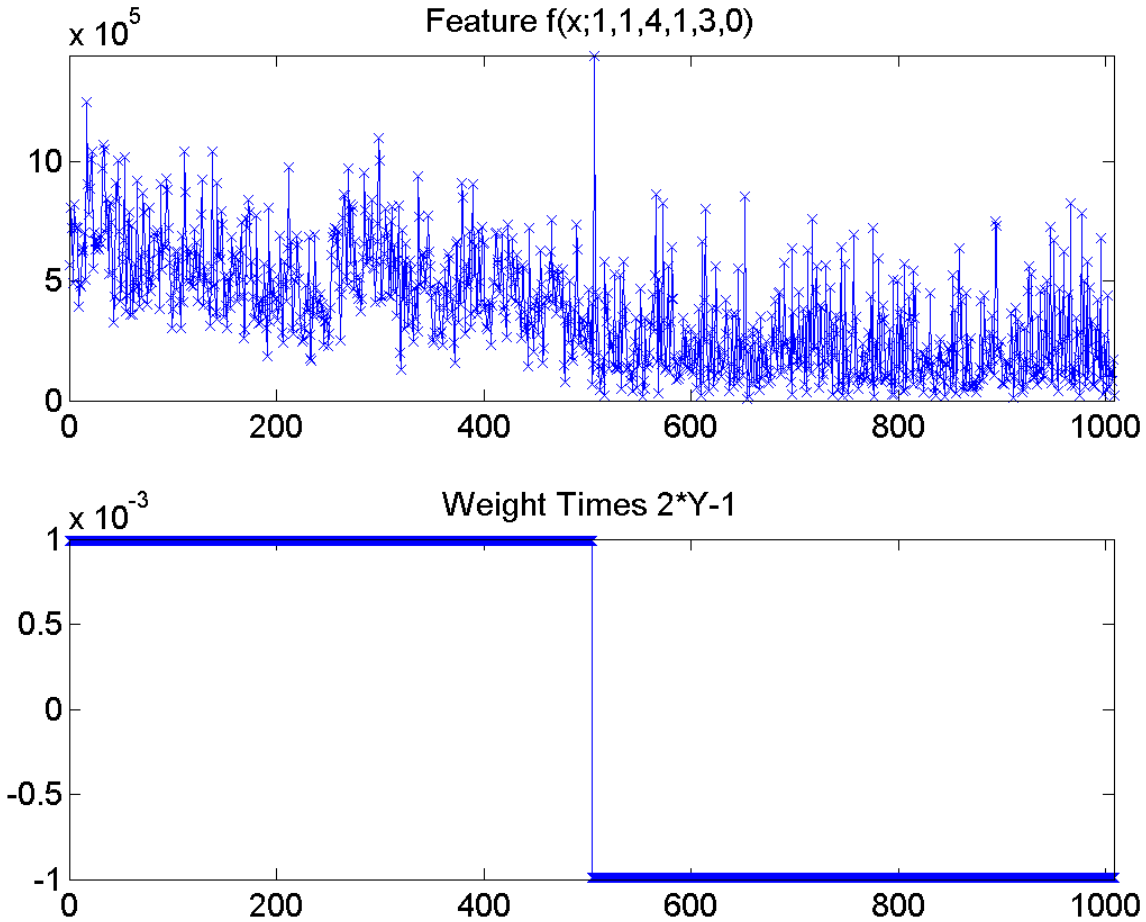


First plot: feature values for one particular feature, computed from all 126 training images, from 8 rectangles/image.

Second plot:
w(t,i) = weight of the i'th training rectangle during the t'th iteration of training.
w(1,i) = 1/(126*8)=1/1008

Third plot: class labels
y=1: face rectangle
y=0: non-face rectangle

“Signed weight” = weight times “sign” of label



First plot: same as before.

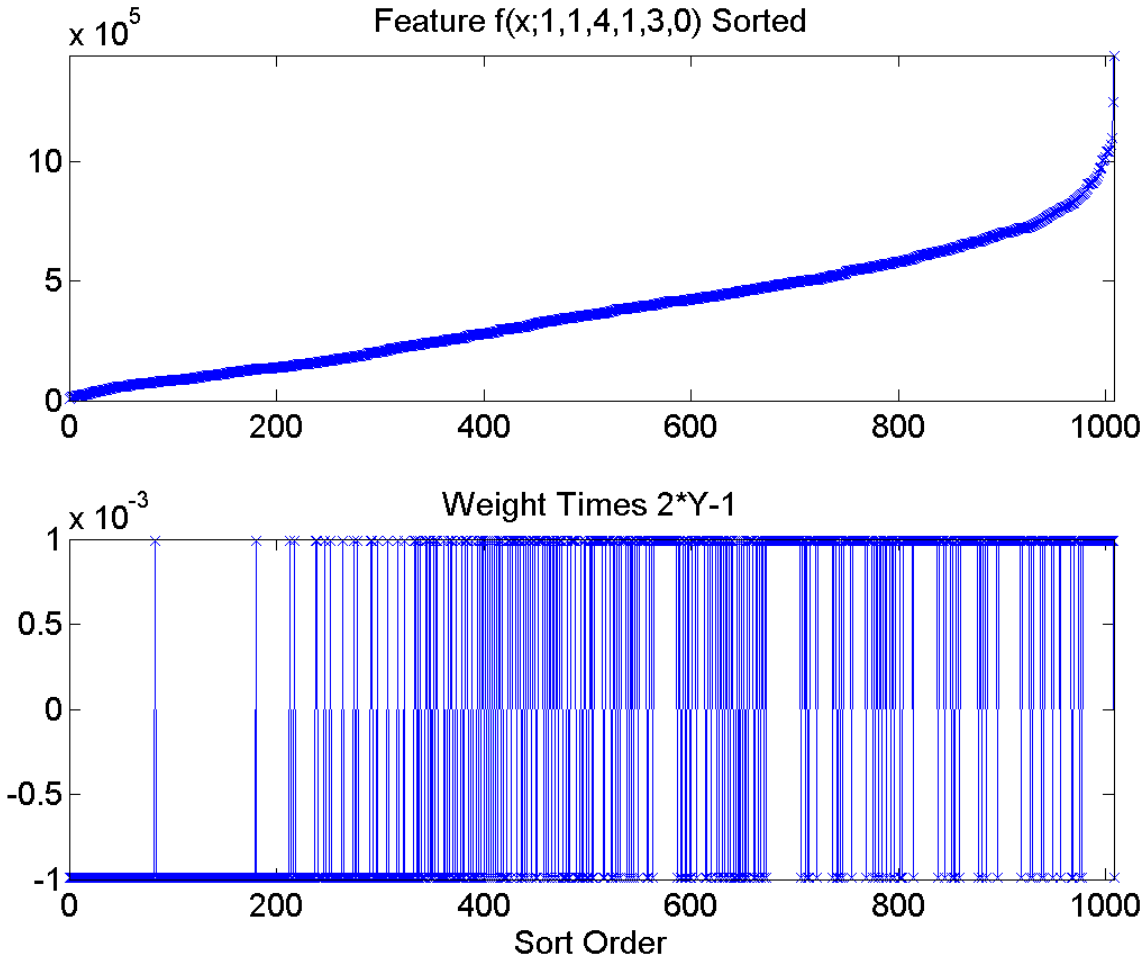
Second plot:
w(t,i)*(2*y(i)-1). Call this the “signed weight.”

If w(t,i) is like the probability of choosing the i’th token,

Then

$$\sum_i w(t,i)(2y(i)-1)$$
$$= \Pr\{Y=1\} - \Pr\{Y=0\}$$

“Signed weight” = weight times “sign” of label



First plot:

```
[f,isort] = sort(f);
plot(1:1008,f);
```

Second plot:

```
w = w(isort);
y = y(isort);
plot(1:1008,w.*(2*y-1));
```

Finding the best scalar classifier out of a set containing tens of thousands of possible scalar classifiers: what I've shown you so far.

1. For every possible feature (fx,fy,fw,fh,q,v),
2. … compute the feature for the whole database…
3. … sort the feature, f, in ascending order. Now you have an ordered list of all of the possible "threshold values" that make sense.
    1. The classifier checks whether or not $f(x)<\Theta$
    2. Suppose that the features are sorted so that $f(1)<f(2)<f(3)$ and so on. Then, as Θ varies from $f(i)\leq\Theta<f(i+1)$, the values of the classifier h(x) don't change.
    3. So the only values of Θ that are meaningful are $\Theta=f(i)$ for some index i. In other words, we should just pick Θ equal to one of the training tokens --- whichever one minimizes the probability of error.
4. … re-arrange w and y into the same order as f, using w(isort) and y(isort). Remember that w(i) is the "probability" of the i'th token, and y(i) is its label. So from this information, we can compute the probability of error. Let's examine this…

Detailed Analysis: the different types of error probabilities

The *a priori* class probabilities are:

$$\pi_1 = \Pr(y = 1) = \sum_{i:y(i)=1} w(t,i)$$

$$\pi_0 = \Pr(y = 0) = \sum_{i:y(i)=0} w(t,i)$$

The different types of error probabilities are:

$P_{FA} = \Pr(y = 0, h(x) = 1)$ false accept (false alarm) probability

$P_{TA} = \Pr(y = 1, h(x) = 1)$ true accept probability

$P_{FR} = \Pr(y = 1, h(x) = 0) = \pi_1 - P_{TA}$ false reject (miss) probability

$P_{TR} = \Pr(y = 0, h(x) = 0) = \pi_0 - P_{FA}$ true reject probability

Scalar Classifier: Error Probabilities

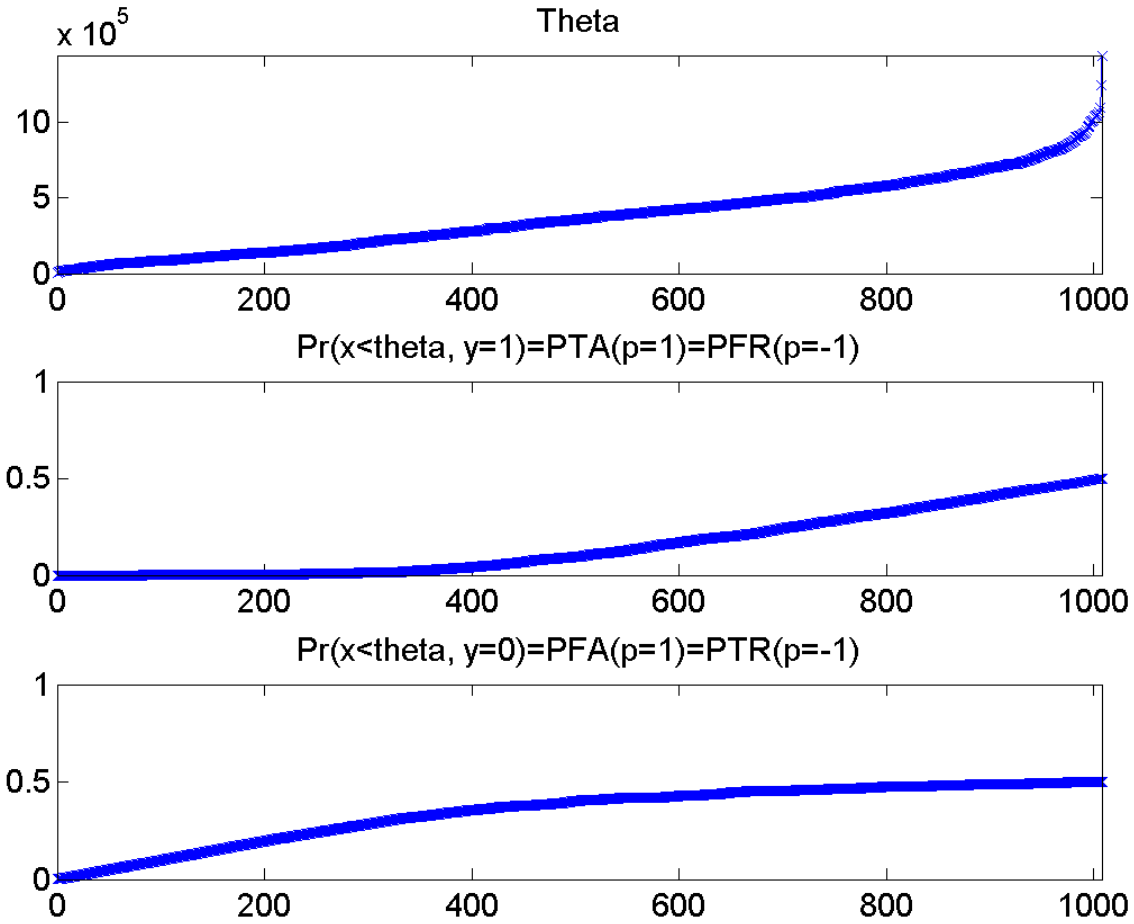
$$f(x; fr, q, v, p, \theta) = \begin{cases} 1 & p\, f(x; fr, q, v) < p\, \theta \\ 0 & otherwise \end{cases}$$

So

$$Pr(x < \Theta, y = 0) = P_{FA}(\Theta, p = 1) = P_{TR}(\Theta, p = -1)$$

$$Pr(x < \Theta, y = 1) = P_{TA}(\Theta, p = 1) = P_{FR}(\Theta, p = -1)$$

Pr(x< Θ,y=1) and P(x<Θ,y=0), every possible Θ



First plot:
[f,isort] = sort(f);
plot(1:1008,f);

Second plot:
cumsum(max(0,w.*(2*y-1)));
Sums up probs of y==1 tokens with feature less than or equal to theta

Third plot:
cumsum(max(0,w.*(1-2*y)));
Sums up probs of y==0 tokens with feature less than or equal to theta

Detailed Analysis: the different types of error probabilities

Probability of Error depends on polarity and theta:

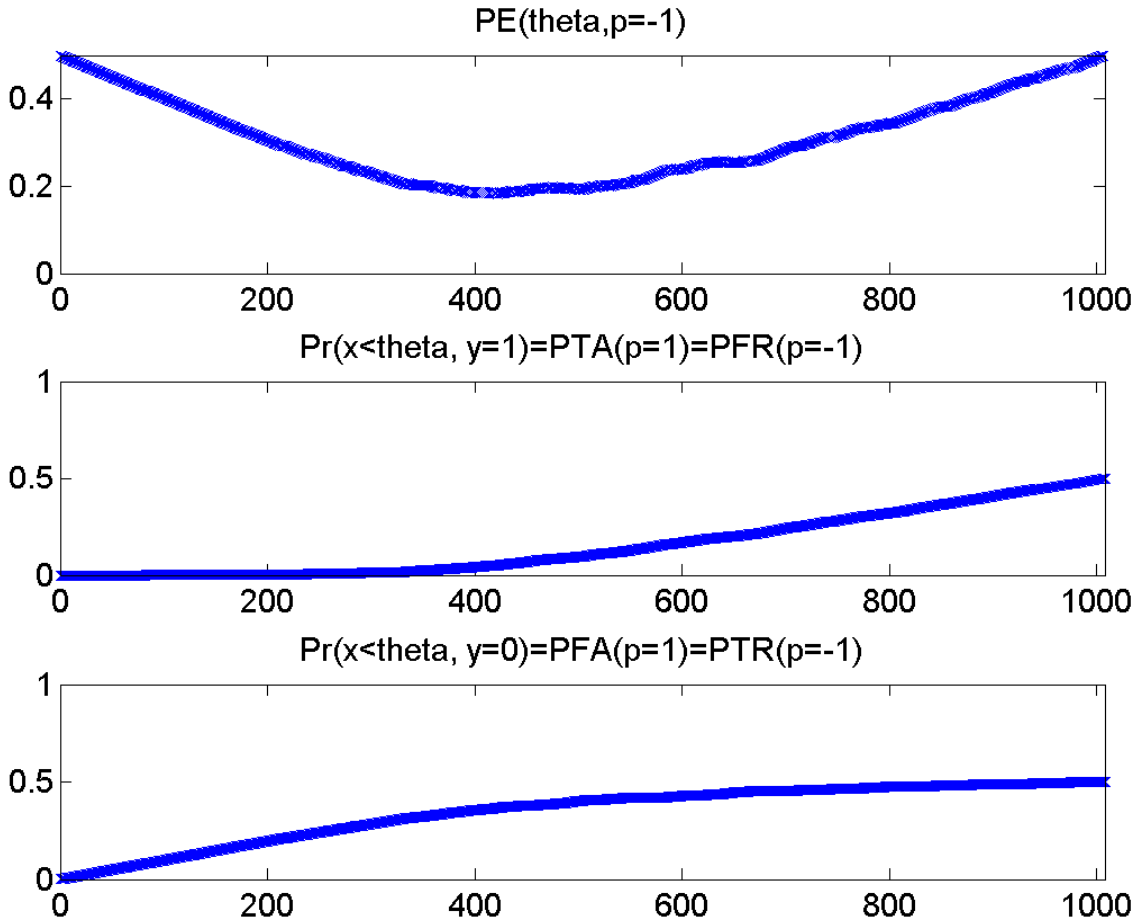
$$P_E(p=1,\theta) = \Pr(x<\theta, y=0) + \Pr(x>\theta, y=1)$$
$$= \pi_1 - \Delta P(\theta)$$

$$P_E(p=-1,\theta) = \Pr(x>\theta, y=0) + \Pr(x<\theta, y=1)$$
$$= \pi_0 + \Delta P(\theta)$$

… but both of them are just …

$$\Delta P(\theta) = Pr(x<\Theta, y=1) - Pr(x<\Theta, y=0)$$

Probability of error, for every possible theta



Second and third plots: same as before.

First plot:
Probability of error, as a function of theta, for polarity p=-1

PE = pi0 + DeltaP

If the minimum in this curve is less than the minimum of pi1 – DeltaP, then p=-1 is best.

Recap: finding the best scalar classifier out of a set containing tens of thousands of possible scalar classifiers

1. For every possible feature (fx,fy,fw,fh,q,v),
2. … compute the feature for the whole database…
3. … sort the feature, f, in ascending order. Now you have an ordered list of all of the possible "threshold values" that make sense
4. … re-arrange w and y into the same order as f. Now, using cumsum, you can quickly find the probability of error for any given "threshold value" and "polarity."
5. … find [pmin1,imin1]=min(pi1-DeltaP) and [pmin0,imin0]=min(pi0+DeltaP). If pmin1 is smaller, then f(imin1) is the threshold, and p=1 is the polarity. Otherwise f(imin0) is the threshold, and p=-1 is the polarity.
6. … if the answer to #5 is the best one you've found so far, keep it.

Adaboost

Suppose somebody told you: I'm going to take a whole bunch of scalar classifiers. Let's use $h_t(x)$ to mean the classifier computed in the t'th training iteration; remember that $h_t(x)$ is either 0 or 1. Then I'm going to add them all together, and the final classifier will be

$$h(x) = \begin{cases} 1 \quad if \sum_t \alpha_t(2h_t(x) - 1) > 0 \\ 0 \quad if \sum_t \alpha_t(2h_t(x) - 1) < 0 \end{cases}$$

How would you choose the classifiers? How would you choose α_t?

(1) Which scalar classifiers should you choose?

- At each training iteration, you have “weights” w(t,i) that tell you the importance of the i’th training token.
- The probability of error is the sum of w(t,i), over all tokens for which the classifier is wrong.
- Choose the classifier with minimum probability of error.
- Now you want the (t+1)st classifier to “fix” the errors made by the (t)th classifier.
- So for every token that ht(x) got wrong, you make w(t+1,i)>w(t,i)
- … and for every token that ht(x) got right, you make w(t,i)<w(t+1,i)

(1) Which scalar classifiers should you choose?

- If the probability of error is always less than ½, the goals on the previous slide can be met by choosing w(t+1,i) so that

$$\sum_{i:\, h_t(x)\, wrong} w(t+1,i) = \sum_{i:\, h_t(x)\, right} w(t+1,i)$$

We can do this by, first, changing only the ones $h_t(x)$ got _right_, to

$$\widetilde{w}(t+i,i) = \left(\frac{P_E}{1-P_E}\right) w(t,i)$$

And then renormalizing, $w(t+i,i)=\widetilde{w}(t+i,i)/\sum_j \widetilde{w}(t+i,j)$

(2) What are the α_t?

- We generally want to give more weight to classifiers that have a lower probability of error. So we want α_t to be a monotonically decreasing function of P_E.
- In fact, traditional logic would say that we only pick the _one_ classifier with the _lowest_ P_E.
- We're not going to do that. We want the ones with higher P_E to hang around, so that they can fix the few errors that the other ones can't fix. But we're going to _really_ scale them down. In fact, we'll _exponentially_ scale them down, like this:

$$\alpha_t = -\log(\frac{P_E}{1 - P_E})$$

(2) What are the α_t?

$$\alpha_t = -\log(\frac{P_E}{1 - P_E})$$

Notice some nice properties:

- $\alpha_t = 0$ if $P_E = 0.5$. So if your training ever gets to the point where it's impossible to find any more classifiers with error less than 50%, that's OK! The α_t for those classifiers will be automatically set to 0.
- $\alpha_t > 0$ if $P_E < 0.5$.
- $\alpha_t = \infty$ if $P_E = 0$. So if you somehow find a classifier that gets every token correct, then Adaboost will only use that classifier --- none of the others will matter.
- This formula has other nice properties that I'm not going to talk about today, because you don't need to know them for either the exam or the MP. But trust me, it's pretty cool.

MP6 recap

- For every training iteration t=1:40,…
- for every possible feature,…
- Compute that feature for every training token in the database.
- Find the Θ and p that minimize the weighted error.
- If this feature gives a better classifier than any so far, keep it.
- end
- alpha(t) = -log(PE(t)/(1-PE(t)))
- end
- Now your complete classifier is sum(alpha(t)(2*h(t,x)-1)). Test this completed classifier on the testing data.