

Face Detection using Adaboost

CS 763

Ajit Rajwade

What is face detection?

- Task of identifying (marking out) faces in an image.



<http://iomniscient.com/newsletters/HTML/EN/image/FR.gif>

Face detection is not face recognition

- Face recognition asks: what is the identity of the person whose image is given to you?



Face detection is not face verification

- Face verification asks: given two images, do they belong to the same person (we don't care who that person is)?



Challenges in face detection

- Change of facial pose/scale
- Change of illumination conditions
- Occlusions (to a lesser extent)



- Designing a detection system resistant to these changes is challenging!

A simplified problem

- Restrict ourselves to **near frontal views** of the face.
- Assume limited range of illumination conditions!
- Assume little or no facial occlusions.
- Multiple **scales** are allowed!

A simple face detector

- Collect together a bunch of face images of equal size (say 80 x 80) in near frontal pose.
- Extract some features from these faces (example: average histogram of gradient orientations, distribution of eigen-coefficients obtained from PCA).



Image taken from: P. Viola
and M. Jones. Robust real-
time object detection.
*International Journal of
Computer Vision*, 57(2):137–
154, 2004.

A simple face detector

- Given a query image, slide a 80 x 80 window all over.
- Extract the same features from the portion of the image covered by the window.
- Classify it as face or non-face depending on the distance in the feature-space.
- If that distance $>$ some threshold – “non-face”, otherwise “face”.
- For scale-invariance, take windows of other sizes and resize them to 80 x 80 before feature extraction.

A simple face detector – and Adaboost

- Which features should we select?
- Different features will produce different results.
- Adaboost is a technique that does the following:
 - ✓ Uses multiple (weak) classifiers – each based on different features
 - ✓ Combines these different (weak) classifiers into a single powerful classifier

Machine learning 101 jargon

- A classifier is a program that assigns an input vector (i.e. a data item) to one of out $M=2+$ classes.
- If $M = 2$, it is called a binary classifier.
- Formal representation: Given input vector \mathbf{x} , the output of a binary classifier is $h(\mathbf{x}) \in \{-1, +1\}$.
- A classifier needs to be trained on “training data”. During training, it “learns” one or more decision rules.
- It is tested on “test data”, i.e. unseen data so that it can work in the real world.

Adaboost

- Adaboost is an **ensemble learning** algorithm.
- It takes a collection of classifiers – called **weak learners** or **base learners** (like a rule of thumb).
- It combines them to produce a **strong classifier**.
- What's a strong classifier? One that will produce good results on unseen data!
- Face detection requires a binary classifier (face versus non-face).
- Adaboost does have multi-class variants but we stick to the 2-class case.

Adaboost

- The weak learners (rules of thumb) **must** have error-rate less than 50%, i.e. they must be at least slightly better than random guessing.
- If the rule of thumb has more than 50% error, just invert its sign!
- Finding and then combining many simple (weak) rules of thumb is easier than finding one complex (accurate) rule.

Adaboost

- Adaboost was invented by Freund and Schapire in 1997.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1997.

- They won the Gödel prize for this contribution in 2003.
- Adaboost was applied to face detection (with some modifications) by Viola and Jones in 2001.

P. Viola and M. Jones. Robust real-time object detection. International Journal of Computer Vision, 57(2):137–154, 2004.

http://en.wikipedia.org/wiki/G%C3%B6del_Prize

Viola-Jones face detector

- Considered one of the state of the art face detectors.
- The original implementation in 2001/2004 claimed a detection speed of 0.07 seconds per frame of size $\sim 300 \times 300$ on a standard desktop.



Image taken from: P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

Figure 10: Output of our face detector on a number of test images from the MIT+CMU test set.

Adaboost

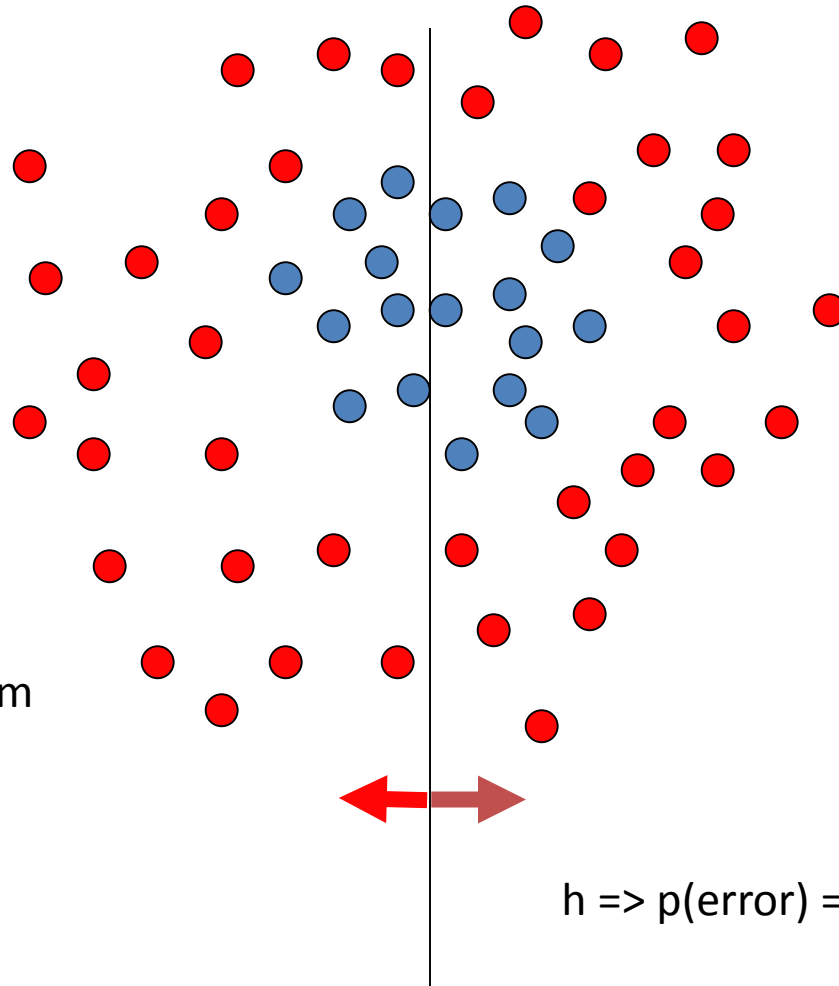
$$H(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x}) + \dots$$

The diagram illustrates the components of the Adaboost equation. It features the equation $H(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x}) + \dots$. Four blue labels with arrows point to specific parts of the equation: 'Strong classifier' points to $H(\mathbf{x})$, 'Feature vector' points to \mathbf{x} , 'Weight' points to α_1 , and 'Weak classifier' points to $h_1(\mathbf{x})$.

Sign of $h_i(\mathbf{x})$ decides to which class point \mathbf{x} is assigned, by the i - th weak classifier.

Sign of $H(\mathbf{x})$ decides to which class point \mathbf{x} is assigned, by the final strong classifier.

Toy Example — taken from Antonio Torralba @MIT



Weak learners from
the family of lines

Each data point has
a class label:

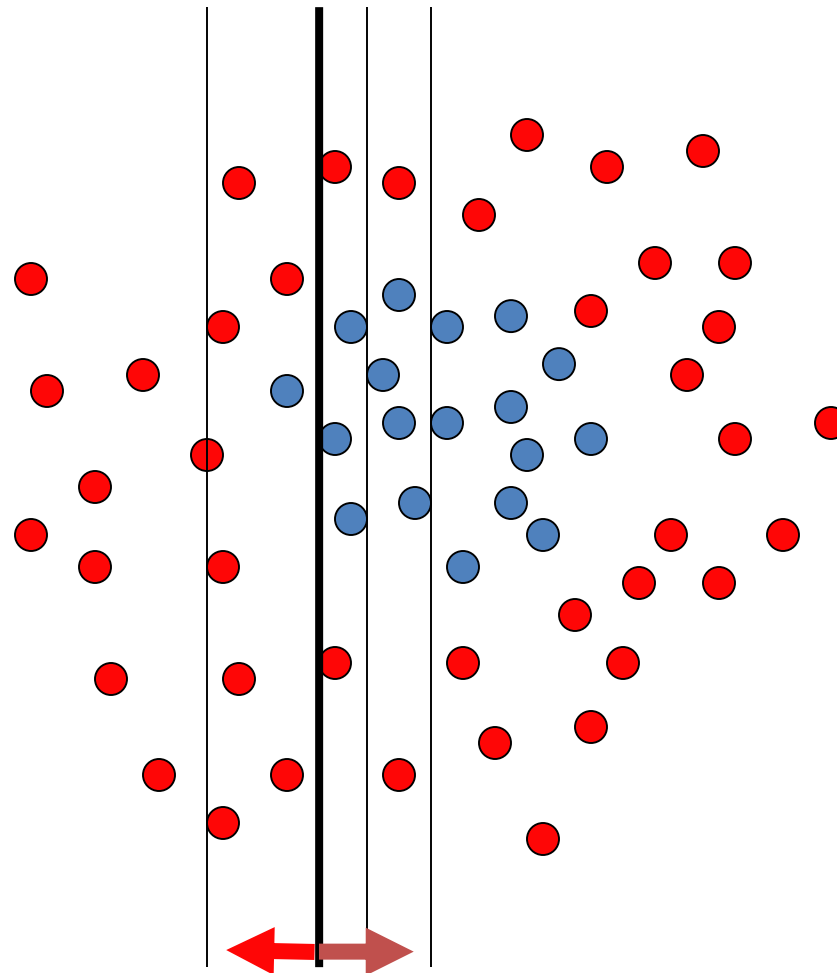
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

and a weight:

$$w_t = 1$$

$h \Rightarrow p(\text{error}) = 0.5$ it is at chance

Toy example



Each data point has
a class label:

$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

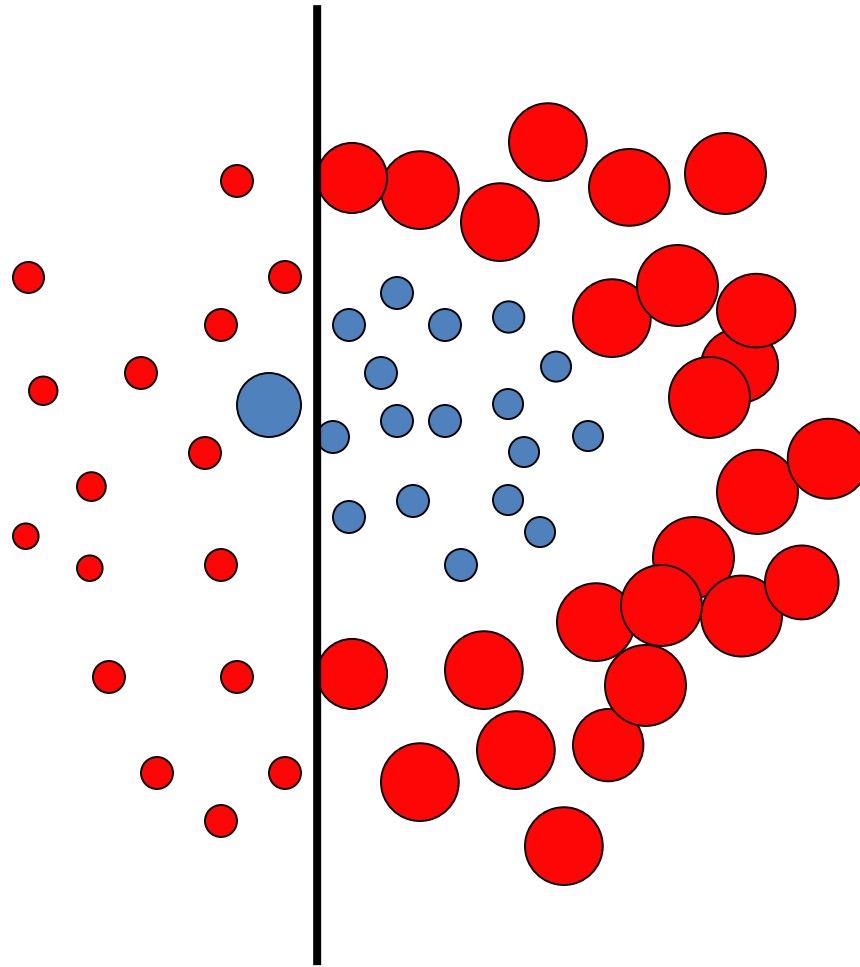
and a weight:

$$w_t = 1$$

This one seems to be the best

This is a '**weak classifier**': It performs slightly better than chance.

Toy example



Each data point has
a class label:

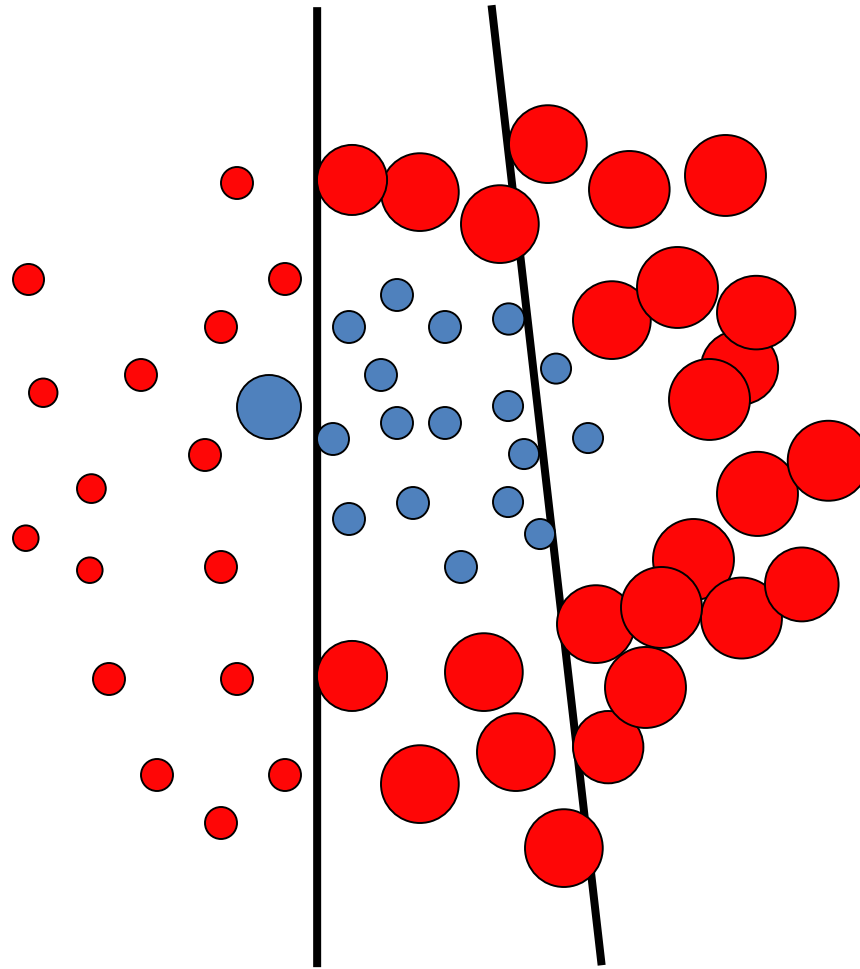
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has
a class label:

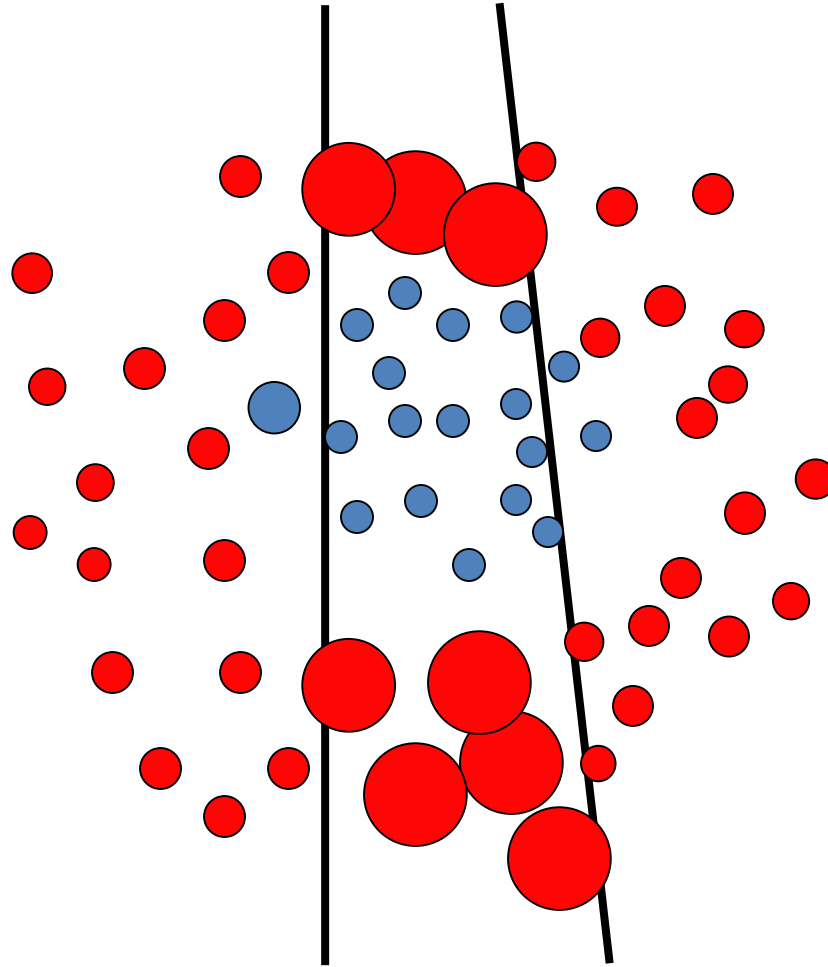
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has
a class label:

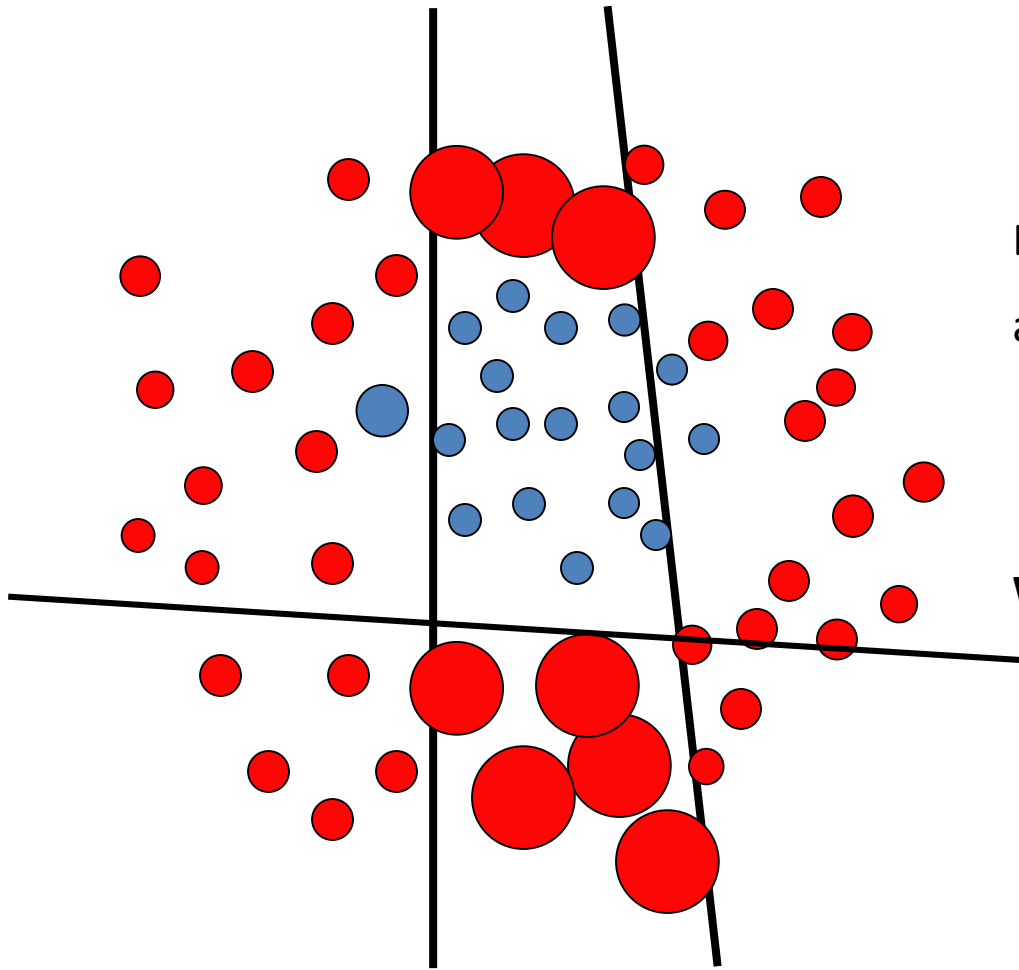
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



Each data point has
a class label:

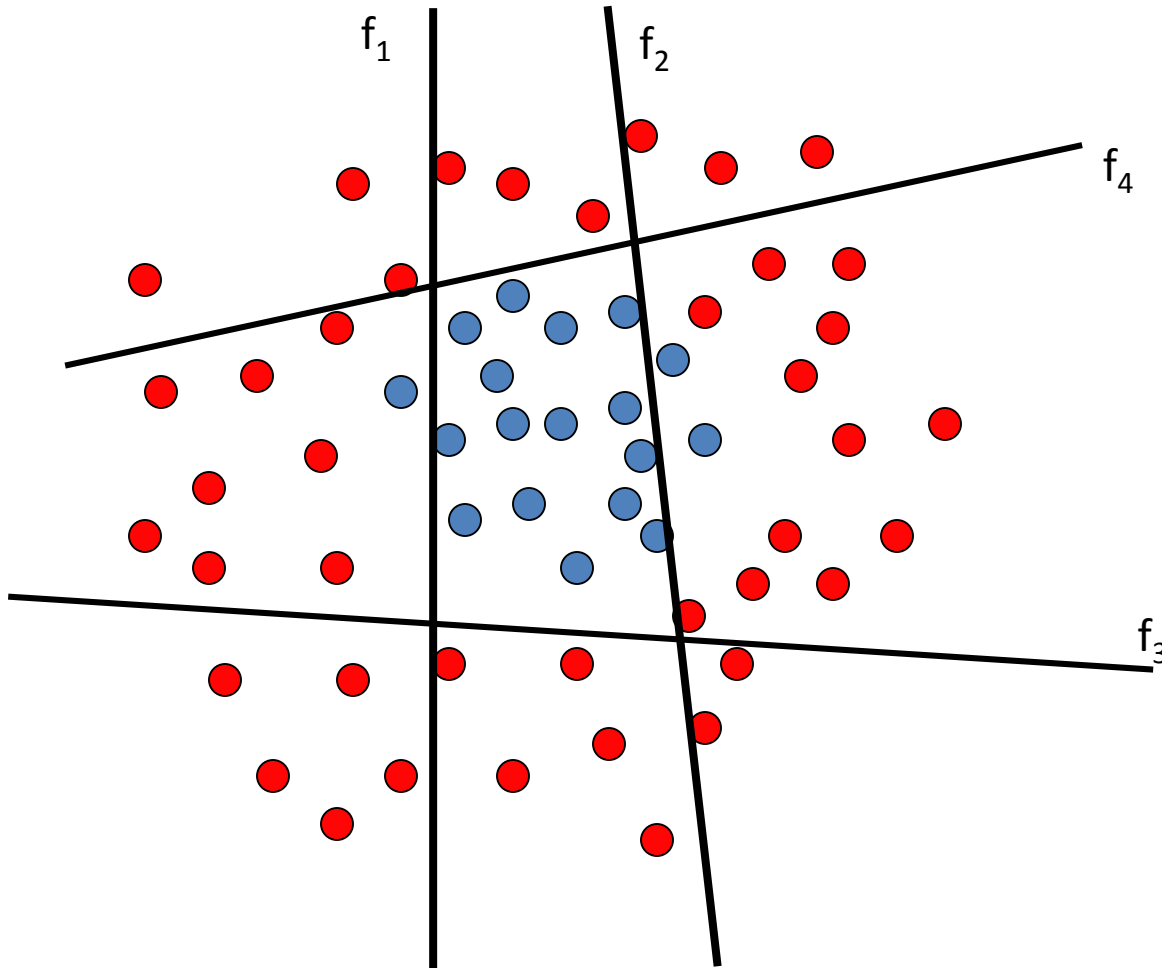
$$y_t = \begin{cases} +1 & (\text{red circle}) \\ -1 & (\text{blue circle}) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Toy example



The strong (non-linear) classifier is built as the combination of all the weak (linear) classifiers. Each weak classifier handles only some of the training samples well and makes errors on the rest. The combined classifier has a much lower error on the training set than any of the weak classifiers.

Formal Procedure of AdaBoost

- given training set $(x_1, y_1), \dots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \dots, T$:

- construct distribution D_t on $\{1, \dots, m\}$
- find weak classifier (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small error ϵ_t on D_t :

$$\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$$

- output final classifier H_{final}

A fancy word for “weights” on the training points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$. But the weights are non-negative and sum to 1.

Just sum up the weights of those points that were misclassified by h_t

Formal Procedure of AdaBoost

- How do you construct the weights? Are they updated? If so, how?
- What is the final classifier? How is it created from the weak classifiers?
- How many weak classifiers are chosen?

Formal Procedure of Adaboost

- constructing D_t :

- $D_1(i) = 1/m$
- given D_t and h_t :

Training samples that are misclassified get more weight. The ones that are correctly classified are given less weight.

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$
$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$
$$= \sum_{i=1}^m D_{t+1}(i)$$
$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i))$$

where Z_t = normalization constant

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

- final classifier:

- $H_{\text{final}}(x) = \text{sign} \left(\sum_t \alpha_t h_t(x) \right)$

These are the weights given to each weak classifier. Not to be confused with the weights on the training samples!

Note that classifiers with **lower** errors get **more** weight. The weights are always positive if the error ≤ 0.5 .

A simple family of classifiers

- Consider N vectors $\{\mathbf{x}_i\}$, $1 \leq i \leq N$, each having d elements.
- Consider the family of weak classifiers:

$$h_t(x_i; j, \theta) = \text{sign}(x_{ij} - \theta)$$
- Choosing the best weak classifier from this family involves choosing a combination of j and θ so as to minimize the weighted training error (given the current set of weights).

A simple family of classifiers.

- Here is a slightly more complicated family of weak classifiers:

$$h_t(\mathbf{x}_i; j, \theta, p) = \text{sign}((x_{ij} - \theta)p) \text{ where } p \in \{-1, +1\}$$

- Choosing the best weak classifier from this family involves choosing a combination of j, p and θ so as to minimize the weighted training error (given the current set of weights).

Error on Training Set

- Theorem:

- write ϵ_t as $1/2 - \gamma_t$
- then

$$\text{training error}(H_{\text{final}}) \leq \exp\left(-2 \sum_t \gamma_t^2\right)$$

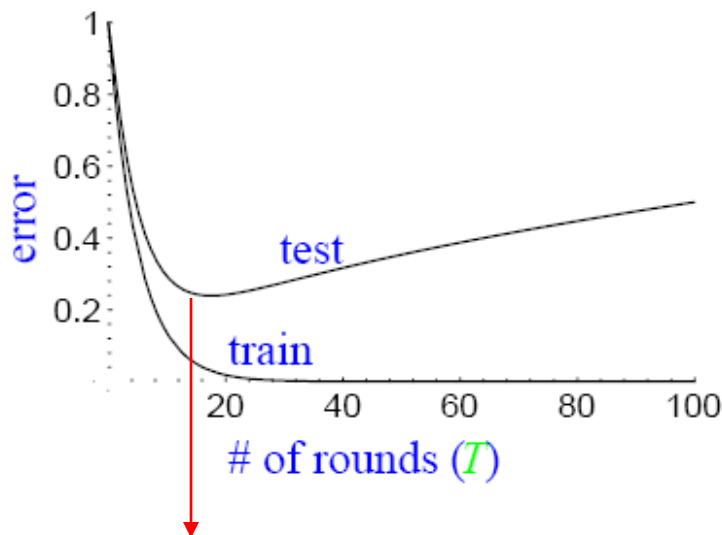
- so: if $\forall t : \gamma_t \geq \gamma > 0$
then $\text{training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$

- AdaBoost is adaptive:

- does **not** need to know γ or T a priori
- can exploit $\gamma_t \gg \gamma$

But we are NOT interested in Training set

- Will Adaboost overfit?



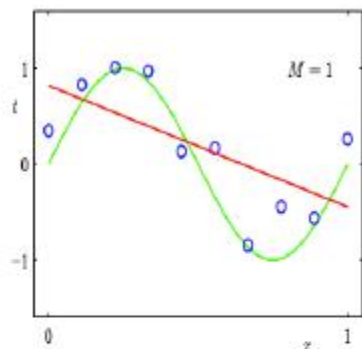
Over fitting: you learn “too much” on the training set, but fail on the test set!

Over-fitting can haunt any machine learning procedure – whether it is classification or regression or probability density estimation.

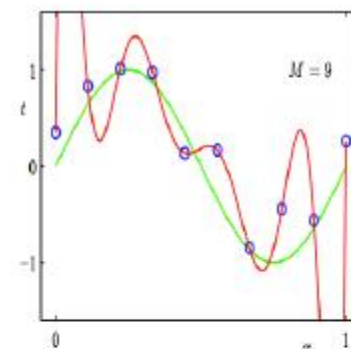
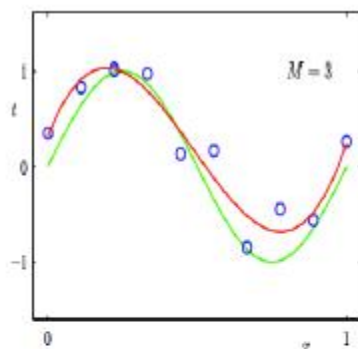
Shall we stop before over fitting? If only over fitting happens with Adaboost.

Phenomenon of Overfitting

Regression:

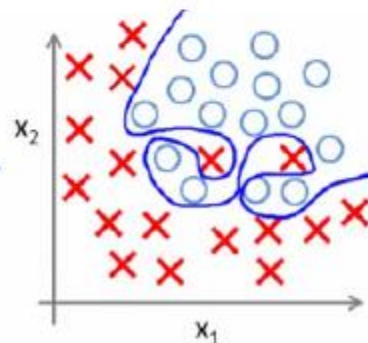
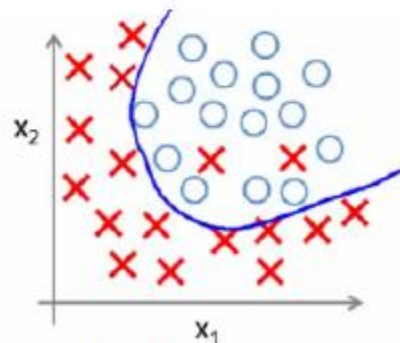
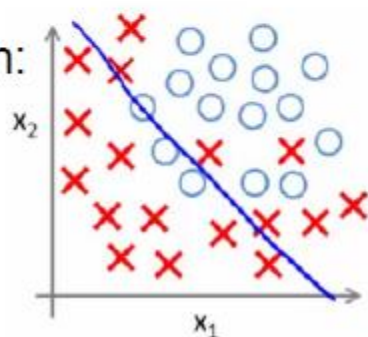


predictor too inflexible:
cannot capture pattern



predictor too flexible:
fits noise in the data

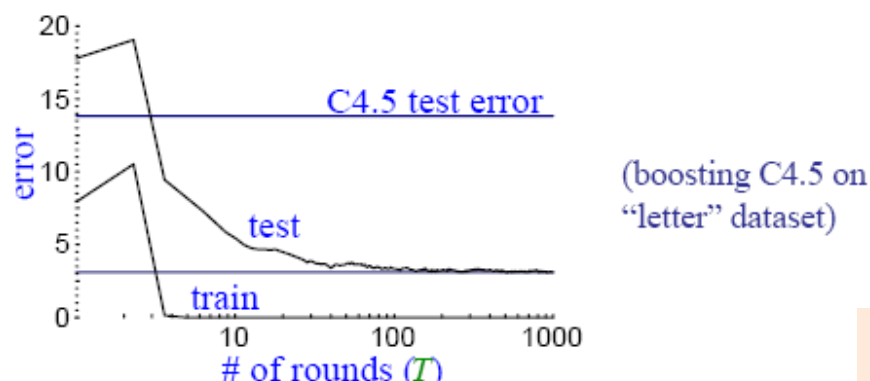
Classification:



Copyright © 2013 Victor Lavevko

<http://www.inf.ed.ac.uk/teaching/courses/iaml/slides/eval-2x2.pdf>

Actual Typical Run of Adaboost



- test error does not increase, even after 1000 rounds
 - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

You should (will!) notice this when you do the assignment!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Back to Viola and Jones face detector

- The detector operates in two phases – (1) training and (2) testing.
- During training, it learns a good classifier that distinguishes between a face and a non-face.
- During testing, the detector is actually deployed on unseen images that were **not** part of the training set.

Back to Viola and Jones face detector

- Input – patches of fixed size (say 24×24) each labeled as “face” (+1) or “non-face” (-1).



- The Viola-Jones detector does **not** use sophisticated features – such as eigen-coefficients, gradient statistics or outputs of Gabor filters!

Viola and Jones face detector: features

- It uses simple sums and differences of rectangles – these features are computationally very efficient.
- These features are called as **Haar-like features**.

Image taken from: P. Viola
and M. Jones. Robust real-
time object detection.
*International Journal of
Computer Vision*, 57(2):137–
154, 2004.

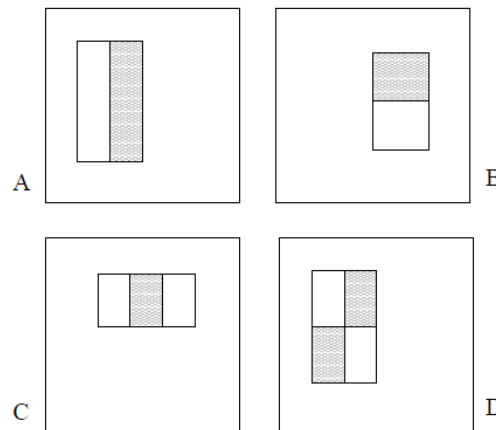


Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

Viola and Jones face detector: features

- For 24×24 patches, there are more than 150,000 such features.

Naively implemented, this Haar-like feature will take $a \times b$ operations to compute where a and b is the height and width of the rectangle.

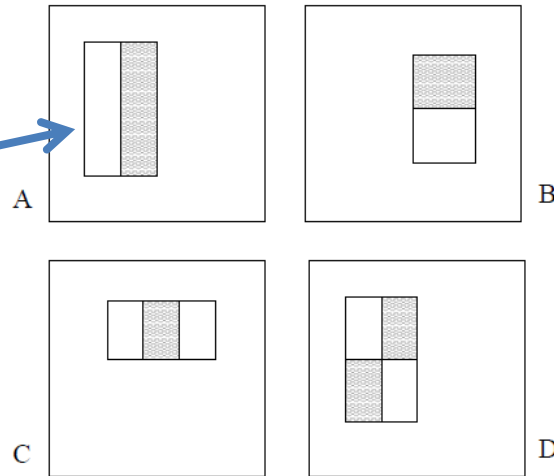


Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

Viola and Jones face detector: features

- These features can be efficiently computed using the so-called integral image defined as follows:

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

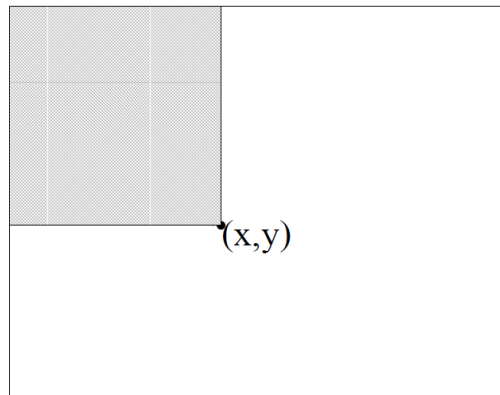


Figure 2: The value of the integral image at point (x, y) is the sum of all the pixels above and to the left.

Viola and Jones face detector: features

- The integral image can be computed with a single pass over the image, if you use the following recurrences:

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

$$s(x, y) \text{ is the cumulative row sum, } s(x, -1) = 0$$

$$ii(-1, y) = 0$$

Viola and Jones face detector: features

- A single rectangle feature can be computed using four array references into the integral image.

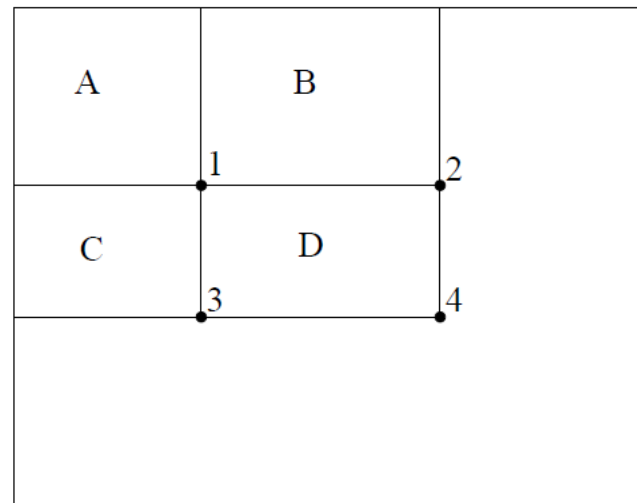


Figure 3: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A . The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Viola and Jones face detector: features

- We have too many features. Which ones should we use? Let the Adaboost algorithm decide!
- In each Adaboost round, pick the rectangle feature that best separates faces from non-faces!
- You also need to decide the optimal threshold and optimal parity. Thus the **weak classifier** is represented as follows:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

Diagram illustrating the weak classifier function $h_j(x)$ with annotations:

- Selected Haar-like feature (points to $f_j(x)$)
- Threshold (points to θ_j)
- Parity (points to p_j)

x is a 24x24 pixel sub-window of an image

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ 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 $t = 1, \dots, T$:

This is different from the earlier $\{-1, +1\}$ convention for the output labels

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 , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
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 $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

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

The expression for the strong classifier is different from the one we have seen so far. This is because we switched from $\{-1, +1\}$ to $\{0, 1\}$ for the output labels

Table 1: The boosting algorithm for learning a query online. T hypotheses are constructed each using a single feature. The final hypothesis is a weighted linear combination of the T hypotheses where the weights are inversely proportional to the training errors.

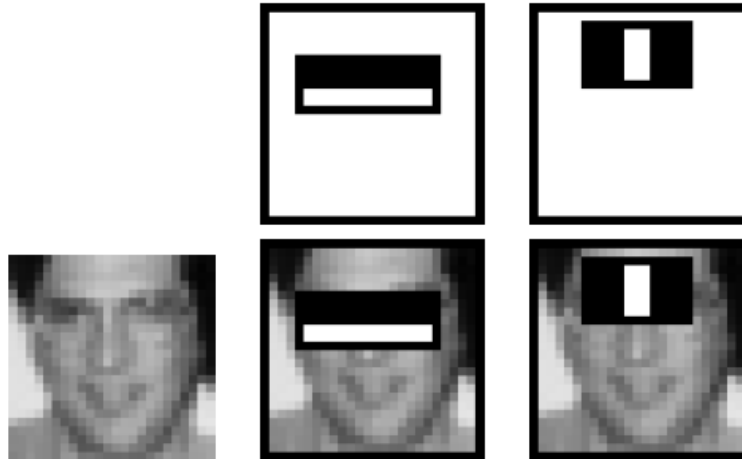


Figure 5: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

Image taken from: P. Viola and M. Jones. Robust real-time object detection.
International Journal of Computer Vision, 57(2):137–154, 2004.

Speeding up: classifier cascade

- Adaboost training is time consuming as a large number of features need to be evaluated.
- So we build a **cascade** of strong classifiers - each classifier in the cascade uses a larger number of processed features than the previous one.
- The first strong classifier uses only **two** features – the ones shown on the previous slide.

(**) The threshold for the strong classifier produced by Adaboost is optimized to yield low error rates. In the Viola Jones paper which uses $\{0,1\}$ as labels instead of $\{-1,+1\}$, the threshold is $\frac{1}{2} \sum_{t=1}^T \alpha_t$. Here, we **deliberately use a lower threshold** to give higher detection rates but higher false positives, taking care that the false positive rate does not exceed a limit, say 40%.

Speeding up: classifier cascade

- The first strong classifier uses only two features – the ones shown on the previous slide.
- The strong classifier threshold can be reduced (**) in order to yield **very low false negative rates** (no “face” should be labeled a “non-face”) allowing **high false positive rates** (some “non-faces” may be labeled as “face”).
- The key idea is to reject as many obvious non-faces as possible early on.
- A go-ahead (positive result) from the first classifier triggers the next classifier in the cascade and so on. A negative result immediately **eliminates** the point.

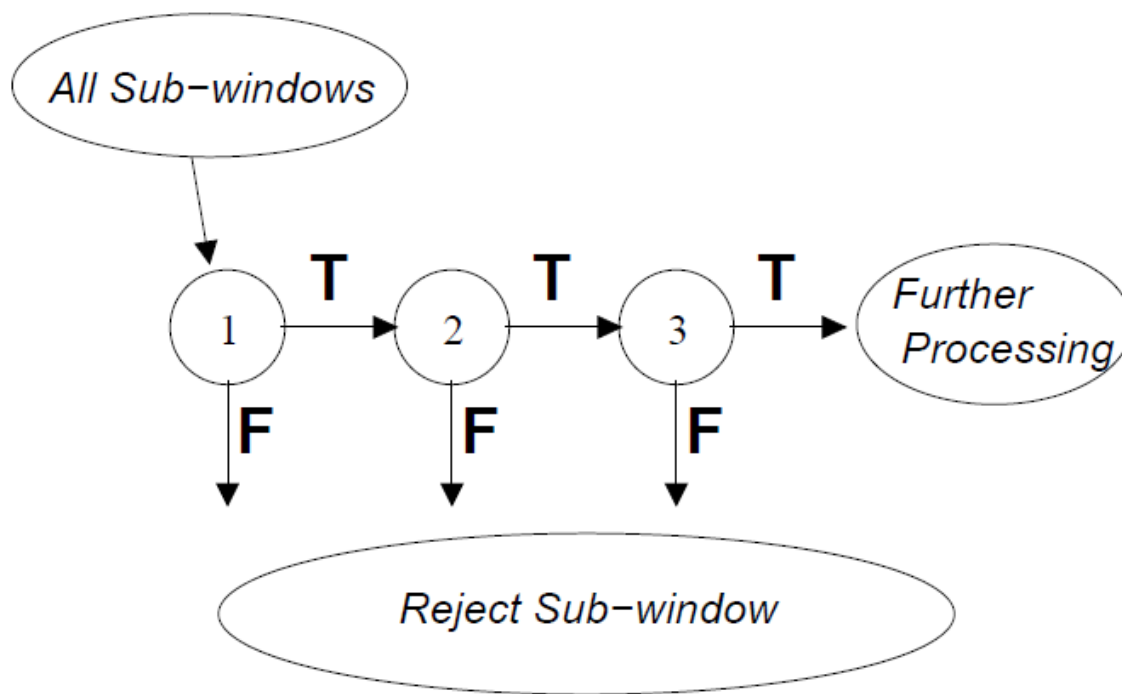


Figure 6: Schematic depiction of a the detection cascade. A series of classifiers are applied to every sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After several stages of processing the number of sub-windows have been reduced radically. Further processing can take any form such as additional stages of the cascade (as in our detection system) or an alternative detection system.

Image taken from: P. Viola and M. Jones. Robust real-time object detection.
International Journal of Computer Vision, 57(2):137–154, 2004.

Speeding up: classifier cascade

- The cascade idea is inspired from the fact that most images will contain many more non-face windows than face windows.
- Quickly discarding several non-faces saves a huge amount of time during training as well as testing.

Cascade: detection rate, false positive rate

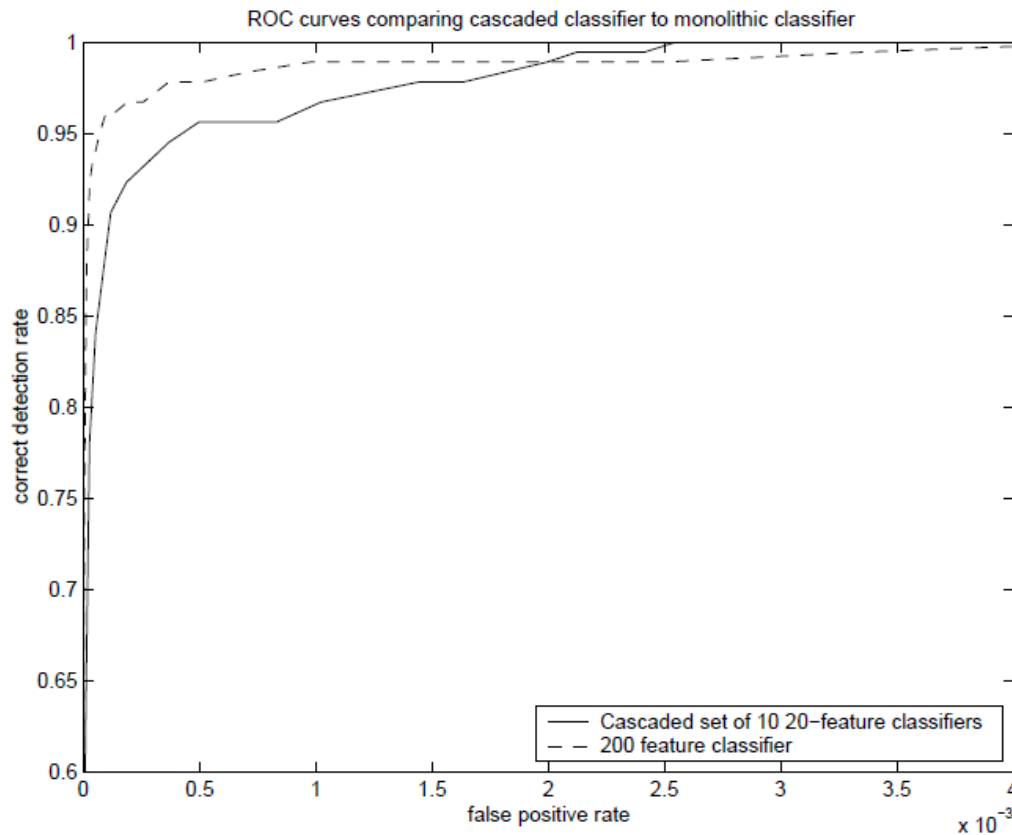
- The overall false positive rate is: $F = \prod_{i=1}^K f_i$,
- The overall detection rate is: $D = \prod_{i=1}^K d_i$,
- Assume $K = 10$. If your final detector must have a 90% detection rate, each individual detector must have a detection rate of at least 99% ($0.99^{10} = 0.9$).
- But the individual detectors in the cascade may have high false positive rates (say 40%). The net false positive rate will be 0.01%.

- User selects values for f , the maximum acceptable false positive rate per layer and d , the minimum acceptable detection rate per layer.
- User selects target overall false positive rate, F_{target} .
- P = set of positive examples
- N = set of negative examples
- $F_0 = 1.0$; $D_0 = 1.0$
- $i = 0$
- while $F_i > F_{target}$
 - $i \leftarrow i + 1$
 - $n_i = 0$; $F_i = F_{i-1}$
 - while $F_i > f \times F_{i-1}$
 - * $n_i \leftarrow n_i + 1$
 - * Use P and N to train a classifier with n_i features using AdaBoost
 - * Evaluate current cascaded classifier on validation set to determine F_i and D_i .
 - * Decrease threshold for the i th classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects F_i)
 - $N \leftarrow \emptyset$
 - If $F_i > F_{target}$ then evaluate the current cascaded detector on the set of non-face images and put any false detections into the set N

Denotes the layer number in the cascade

This refers to the strong classifier produced by the i -th layer of the cascade

The validation set is distinct from P and N . Each data-point in the validation set is also labelled as face or non-face.



Comparison between a single classifier with 200 features (i.e. 200 rounds of Adaboost) and a cascade of 10 classifiers each using 20 features. The first classifier in the cascade was trained on 5000 face images + 10000 non-faces. The second classifier was trained on 5000 faces + 5000 false positives from stage 1, and so on. The detection rates of the cascade is comparable to the full classifier, but its speed is 10 times as high.

Comments on the cascade

- The cascaded classifier saves a great deal of time during testing as it eliminates obvious non-face windows very early on.
- During training, the cascaded classifier still needs to select some features out of several candidate features – which is no doubt expensive. However, the gain is obtained by being able to throw out non-face windows early on.

Details of experiments in the Viola-Jones face detector

- ~5000 faces obtained from a web-crawl, cropped and resized to 24 x 24
- Final detector is a 32-layer cascade with ~4900 features in total
- Around 10,000 non-face images were used. Classifiers at different steps in the cascade trained on different non-faces (around 6000 in number).
- Total training time ~ around a couple of weeks.

Image pre-processing

- All sub-images (size 24 x 24) were made 0 mean and unit variance to induce a very basic form of illumination invariance.

Scale invariance

- This was handled by sampling pixels with larger gaps, producing sub-images of the same size, i.e. 24×24 (eg: instead of sampling consecutive locations for the 24×24 window, you take every second pixel in a 48×48 region to extract a 24×24 window).

Adaboost: Some Theory

References:

- * Many slides adapted from the website of Prof. Jason Corso, SUNY Buffalo

Adaboost: some questions

1. What objective function does Adaboost minimize during training?
2. Why the strange formula for alpha?
3. Can you prove that the training error goes to zero with infinitely many rounds?
4. Why do we pick the next classifier that has the lowest weighted training error (i.e. ϵ_t)?
5. Why the given rule for updating the weights?

(1) Adaboost algorithm: what does it do?

- Our aim is to select classifiers $\{h_t\}$, $1 \leq t \leq T$, and their weights $\{\alpha_t\}$, $1 \leq t \leq T$, so as to minimize the training error of the strong classifier, i.e.:

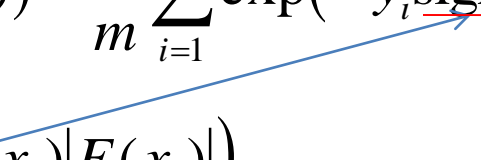
$$\operatorname{argmin} \frac{1}{m} \sum_{i=1}^m \delta(H(\mathbf{x}_i) \neq y_i)$$

This is the Kronecker delta function – it outputs 1 if the predicate passed as argument is true, otherwise it outputs 0.

$$H(\mathbf{x}) = \operatorname{sign} [F(\mathbf{x})] = \operatorname{sign} \left[\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right]$$

(1) Adaboost algorithm: what does it do?

- Adaboost does **not** minimize this classification error (called “empirical risk”) directly.
- But it minimizes the following upper bound on this error (i.e. a quantity which is guaranteed to never be less than the classification error):

$$\begin{aligned}\frac{1}{m} \sum_{i=1}^m \exp(-y_i F(x_i)) &= \frac{1}{m} \sum_{i=1}^m \exp(-y_i \text{sign}(F(x_i)) |F(x_i)|) \\ &= \frac{1}{m} \sum_{i=1}^m \exp(-y_i \text{sign}(F(x_i)) |F(x_i)|)\end{aligned}$$


Adaboost: Optimization – why is it an upper bound on $\text{Err}(H)$?

- To prove the following:

$$\text{Err}(H) = \frac{1}{m} \sum_{i=1}^m \boxed{\delta(H(\mathbf{x}_i) \neq y_i)} \leq Z = \frac{1}{m} \sum_{i=1}^m \boxed{\exp[-y_i F(\mathbf{x}_i)]}$$

$F(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x})$

$$\begin{aligned} F(\mathbf{x}_i) &= \text{sign}(F(\mathbf{x}_i)) |F(\mathbf{x}_i)| \\ &= H(\mathbf{x}_i) |F(\mathbf{x}_i)| \end{aligned}$$

If $H(\mathbf{x}_i) \neq y_i$ then the LHS = 1 \leq RHS = $e^{|F(\mathbf{x}_i)|}$.

If $H(\mathbf{x}_i) = y_i$ then the LHS = 0 \leq RHS = $e^{-|F(\mathbf{x}_i)|}$.

$$\delta(H(\mathbf{x}_i) \neq y_i) \leq \exp[-y_i F(\mathbf{x}_i)] \quad \text{for all } i$$

(2) Let's look at the weights

- Recursive computation of the weights:

$$\begin{aligned} D_{t+1}(i) &= \frac{1}{Z_t} D_t(i) \exp [-\alpha_t y_i h_t(x_i)] & (21) \\ &= \frac{1}{Z_{t-1} Z_t} D_{t-1}(i) \exp \left[-y_i (\alpha_t h_t(x_i) + \alpha_{t-1} h_{t-1}(x_i)) \right] \\ &= \dots \\ &= \frac{1}{Z_1 \dots Z_t} D_1(i) \exp \left[-y_i (\alpha_t h_t(x_i) + \dots + \alpha_1 h_1(x_i)) \right] \end{aligned}$$

- Normalization of the weights at each round:

$$\begin{aligned} Z_t &= \sum_{\mathbf{x}_i} D_t(\mathbf{x}_i) \exp [-y_i \alpha_i h_t(\mathbf{x}_i)] \\ &= \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp [-\alpha_t] + \sum_{\mathbf{x}_i \in \overline{\mathcal{A}}} D_t(\mathbf{x}_i) \exp [\alpha_t] \end{aligned}$$

Set of correctly classified points

(2) Let's look at the weights

- But the weights sum up to 1. So:

$$\sum_{i=1}^m D_{t+1}(\mathbf{x}_i) = \frac{1}{Z_1 \dots Z_t} \frac{1}{m} \sum_{i=1}^m \exp[-y_i F(\mathbf{x}_i)] = 1$$

$$Z = Z_1 \dots Z_t = \frac{1}{m} \sum_{i=1}^m \exp[-y_i F(\mathbf{x}_i)]$$

- We have seen that this quantity Z is an upper bound on the training error:

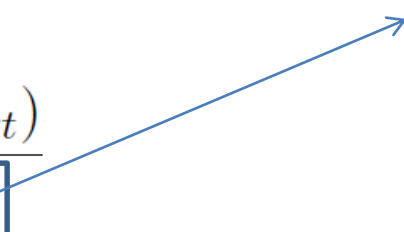
$$\text{Err}(H) \leq Z = Z(\alpha, h) = Z_t(\alpha_t, h_t) \dots Z_1(\alpha_1, h_1)$$

(2) Adaboost: Updating alphas

- Adaboost seeks to minimize Z w.r.t. the classifiers $\{h_t\}$ and also their respective weights $\{\alpha_t\}$, $1 \leq t \leq T$.
- Solving for alphas:

$$Z_t(\alpha_t, h_t) = \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \bar{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t]$$

$$\frac{dZ_t(\alpha_t, h_t)}{d\alpha_t} = \sum_{\mathbf{x}_i \in \mathcal{A}} -D_t(\mathbf{x}_i) \exp[-\alpha_t] + \boxed{\sum_{\mathbf{x}_i \in \bar{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t]} = 0$$

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


(3) Adaboost: Bound on training error

- Plugging in these alphas into the expression for Z , we get:

$$\begin{aligned} Z_t(\alpha_t, h_t) &= \sum_{\mathbf{x}_i \in \mathcal{A}} D_t(\mathbf{x}_i) \exp[-\alpha_t] + \sum_{\mathbf{x}_i \in \bar{\mathcal{A}}} D_t(\mathbf{x}_i) \exp[\alpha_t] \\ &= (1 - \epsilon_t(h_t)) \sqrt{\frac{\epsilon_t(h_t)}{1 - \epsilon_t(h_t)}} + \epsilon_t(h_t) \sqrt{\frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)}} \\ &= 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \end{aligned}$$

- Define:

$$\gamma_t = \frac{1}{2} - \epsilon_t(h_t), \quad \gamma_t \in (0, \frac{1}{2}]$$

(3) Adaboost: Bound on training error

- Plugging in these values, we now get:

$$\begin{aligned} Z_t(\alpha_t, h_t) &= 2\sqrt{\epsilon_t(h_t)(1 - \epsilon_t(h_t))} \\ &= \sqrt{1 - 4\gamma_t^2} \\ &\leq \exp[-2\gamma_t^2] \end{aligned}$$

Since $1+x \leq e^x$ for all x . Put in $x = -4(\gamma_t)^2$

- After T rounds, we get:

$$\text{Err}(H) \leq Z \leq \exp \left[-2 \sum_{t=1}^T \gamma_t^2 \right]$$

(4) Adaboost: How to pick the next classifier?

- Notice that the given algorithm picks classifiers (belonging to a parametric family) that minimizes the weighted training error, given fixed values of the alphas!
- Why so? Because we seek a hypothesis h_t that minimizes Z (see expression for Z in terms of ε_t two slides before), i.e. that minimizes Z_t .
- Z_t is a monotonically increasing function of ε_t for $\varepsilon_t \in (0, 1/2]$, so we seek h_t that has the least possible ε_t .

Coordinate descent

- Consider a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$.
- Consider a multivariate convex, differentiable function $f(\mathbf{x})$ to be minimized w.r.t. \mathbf{x} .
- Coordinate descent is summarized as follows:

$$x_1^{(k)} \in \operatorname{argmin}_{x_1} f(x_1, x_2^{(k-1)}, x_3^{(k-1)}, \dots, x_n^{(k-1)})$$

$$x_2^{(k)} \in \operatorname{argmin}_{x_2} f(x_1^{(k)}, x_2, x_3^{(k-1)}, \dots, x_n^{(k-1)})$$

$$x_3^{(k)} \in \operatorname{argmin}_{x_3} f(x_1^{(k)}, x_2^{(k)}, x_3, \dots, x_n^{(k-1)})$$

...

$$x_n^{(k)} \in \operatorname{argmin}_{x_n} f(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n)$$

- Start with initial guess for \mathbf{x} .
- Order of updates can be arbitrary.
- Use the latest value of every coordinate.
- The value of f is guaranteed to never increase across these updates.
- Repeat these updates until the change is no more significant.

Coordinate descent

- Theoretical treatment on coordinate descent states that such a sequence is guaranteed to converge to the minimizer of f .
- More details:

<https://www.cs.cmu.edu/~ggordon/10725-F12/slides/25-coord-desc.pdf>

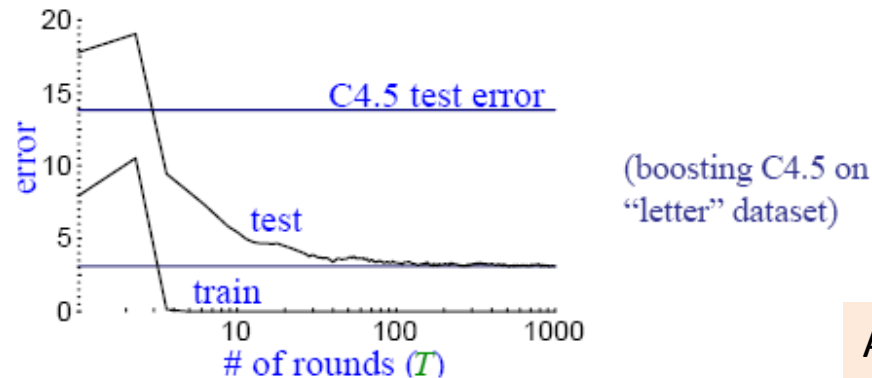
Adaboost is an example of coordinate descent

- Adaboost algorithm = coordinate descent on the function $Z(\{\alpha_t\})$ given a fixed family of finitely many classifiers.
- You find one weight α_t (for some t) at a time using coordinate descent.
- For a fixed number (T) of Adaboost rounds, not all the classifiers from the family may be selected – for those, the weight will be 0.

Adaboost is an example of coordinate descent

- The empirical risk is not a convex function of the weights $\{\alpha_t\}$.
- The upper bound Z defined earlier is a convex function of $\{\alpha_t\}$ and hence easier to minimize.
- Minimizing Z is not the same as minimizing the empirical risk, but we do know that the minimum of the empirical risk is below the minimum of Z .

Generalization capability of Adaboost



- test error does not increase, even after 1000 rounds
 - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

Adaboost has a tendency not to overfit.

You should (will!) notice this when you do the assignment!

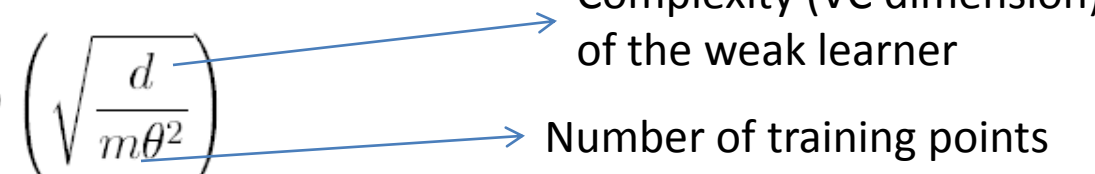
	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Generalization capability of Adaboost

- The aforementioned curious phenomenon has been observed in several experiments on Adaboost.
- It means that often, Adaboost has a tendency **not** to overfit!
- Freund and Schapire explained this observation using the concept of “**margin of a classifier**”.
- The margin of a classifier h on point x is defined as $yh(x)$ – intuitively it tells us how far away x is from the decision boundary (given by $h(x)$). It is like the **confidence** of the vote.

Generalization capability of Adaboost

- The margin of H is given by $\frac{\sum_{t=1}^T \alpha_t y h_t(x)}{\sum_{t=1}^T \alpha_t}$
- **Theorem 1:** The **larger** the *margin*, you have a **better bound** on the value of the *generalization error*. For any $\theta > 0$, the generalization error is upper bounded by the following quantity (with high probability):

$$\hat{\Pr}[\text{margin}(x, y) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right)$$


Complexity (VC dimension) of the weak learner

Number of training points

Generalization capability of Adaboost

- It has been shown that the margin of the Adaboost classifiers **tends to increase** with the number of rounds even after the training error reaches 0.
- The proofs of all these results is beyond the scope of our course.

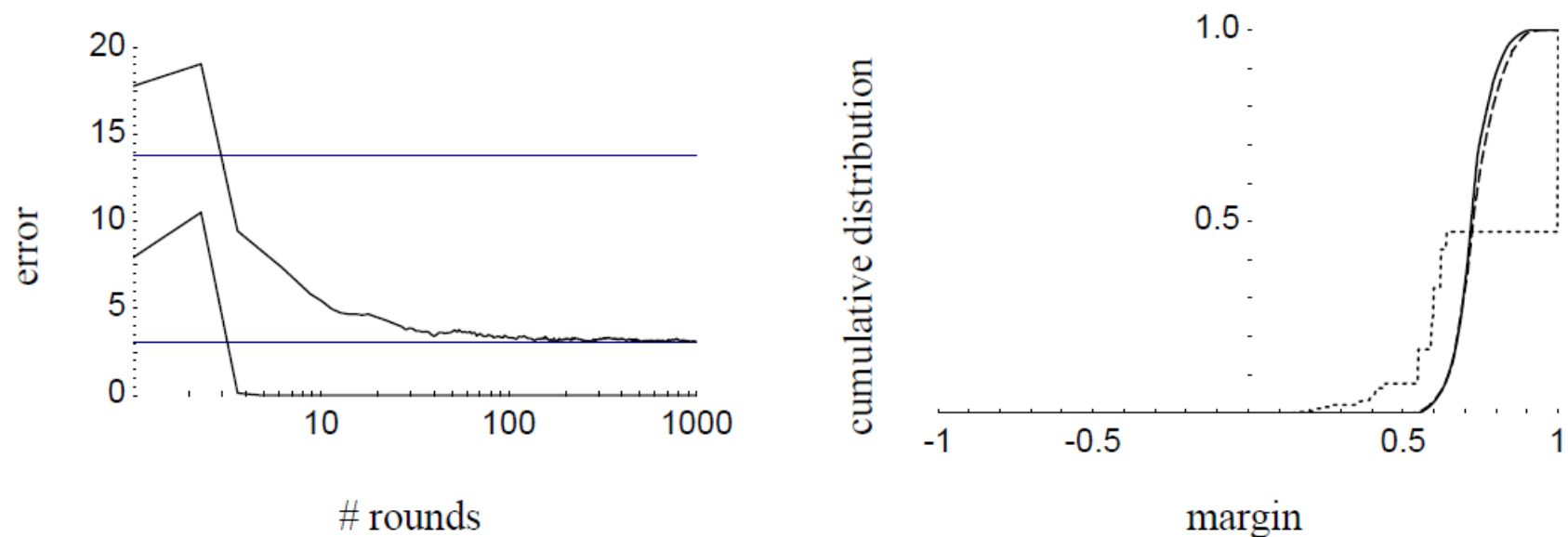


Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [41]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. *Right*: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

This means that with more rounds, the margins of the training samples increase – this pushes the cumulative distribution function (CDF) of the margin rightwards. Recall that the CDF of the margin is given as follows:

$$\text{CDF}_{\text{margin}}(\theta) = \Pr(\text{margin} \leq \theta) = \text{probability that the margin} \leq \theta$$

Adaboost: (Strong!) Positives

- Great theoretical treatment and empirical performance.
- Fast to evaluate (linear combination of classifiers).
- Limited parameter tuning: number of rounds T .
- Simple meta-algorithm, very flexible, can work with any weak learner.

But...Adaboost: Some words of caution

- Performance will depend on data!
- Performance will depend on choice of weak classifier families. Hence can fail if the weak classifiers are either “too weak” or “too complex”.

References

- Tutorial by Freund and Schapire: “A short introduction to boosting”
- Tutorial by Zhou and Yu: “Adaboost”
- Viola and Jones, “Robust real-time object detection”, IJCV 2001.
- [Jason Corso’s \(SUNY Buffalo\) lecture notes on Adaboost](#)
- Adaboost and coordinate descent: [Course notes by Cynthia Rudin](#)