AdaBoost

Dr. Víctor Uc Cetina

Facultad de Matemáticas
Universidad Autónoma de Yucatán

cetina@informatik.uni-hamburg.de
https://sites.google.com/view/victoruccetina

Content

1. Boosting
2. AdaBoost

Boosting

- Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.

Boosting

- Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- To begin, we define an algorithm for finding the rules of thumb, which we call a weak learner.

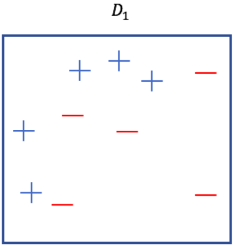
Boosting

- Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- To begin, we define an algorithm for finding the rules of thumb, which we call a weak learner.
- The boosting algorithm repeatedly calls this weak learner, each time feeding it a different distribution over the training data (in Adaboost).

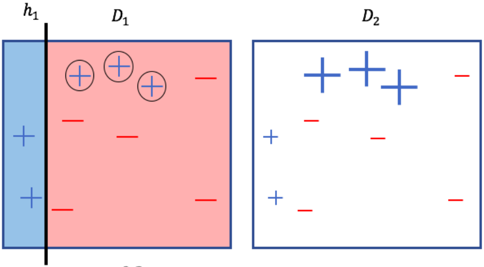
Boosting

- Boosting refers to a general and provably effective method of producing a very accurate classifier by combining rough and moderately inaccurate rules of thumb.
- To begin, we define an algorithm for finding the rules of thumb, which we call a weak learner.
- The boosting algorithm repeatedly calls this weak learner, each time feeding it a different distribution over the training data (in Adaboost).
- Each call generates a weak classifier and we must combine all of these into a single classifier that, hopefully, is much more accurate than any one of the rules.

Boosting



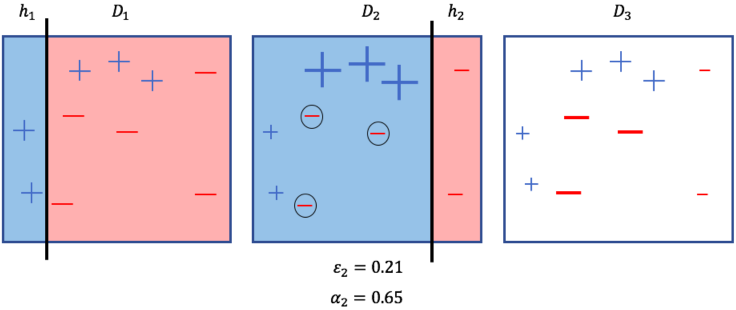
Boosting



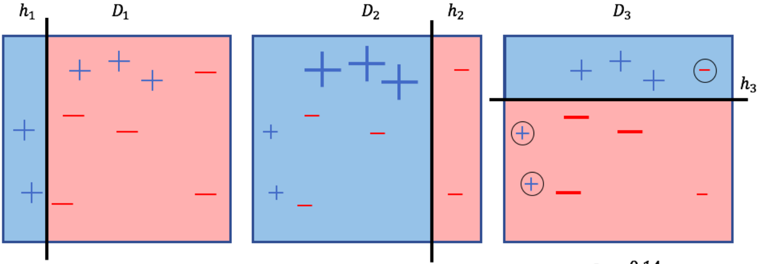
$\varepsilon_1 = 0.3$

$\alpha_1 = 0.42$

Boosting



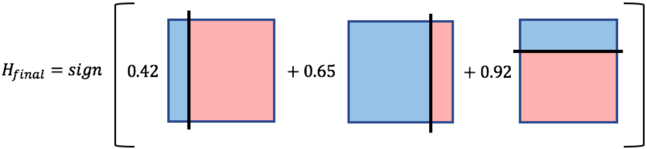
Boosting

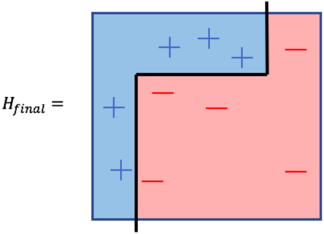


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Boosting





Boosting

- As usual we are given a data set:

$$\mathcal{D} = \left\{(\mathrm{x}_i, y_i) : \mathrm{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, 2, \ldots, m\right\}.$$

- Where x_i could represent the features vector of an image patch, and y_i is whether or not the image is a face.
- First, we need to define a distribution D over the dataset $\mathcal{D}$ such that $\sum_i D(i) = 1$.

Weak classifier

- A weak classifier can be defined as:

$$h_t : \mathbb{R}^d \to \{-1, +1\}.$$

Weak classifier

- A weak classifier can be defined as:

$$h_t : \mathbb{R}^d \to \{-1, +1\}.$$

- A weak classifier must work better than chance. This means it has less than 50% error; if it would have higher than 50% error, just flip the sign.

Weak classifier

- A weak classifier can be defined as:

$$h_t : \mathbb{R}^d \to \{-1, +1\}.$$

- A weak classifier must work better than chance. This means it has less than 50% error; if it would have higher than 50% error, just flip the sign.
- So, we want only a classifier that does not have exactly 50% error (since these classifiers would add no information).

Weak classifier

- A weak classifier can be defined as:

$$h_t : \mathbb{R}^d \rightarrow \{-1, +1\}.$$

- A weak classifier must work better than chance. This means it has less than 50% error; if it would have higher than 50% error, just flip the sign.
- So, we want only a classifier that does not have exactly 50% error (since these classifiers would add no information).
- The error rate of a weak classifier $h_t(x)$ is calculated empirically over the training data:

$$\epsilon(h_t) = \frac{1}{m} \sum_{i=1}^{m} \delta(h_t(x_i) \neq y_i) < \frac{1}{2}.$$

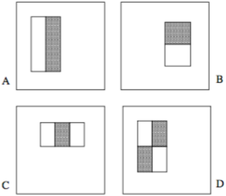
Weak classifier

- Suppose that each x_i is an image patch like the following:

Weak classifier

- Suppose that each x_i is an image patch like the following:

- One possible weak classifier could be a haar-like feature of this kind:



The feature value es the difference of the pixels sums of the white regions and black regions.

Integral image

- Define the integral image as the image whose pixel value at a particular pixel x, y is the sum of the pixel values to the left and above x, y in the original image:

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

where ii is the integral image and i is the original image.

Integral image

- Define the integral image as the image whose pixel value at a particular pixel x, y is the sum of the pixel values to the left and above x, y in the original image:

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

  where ii is the integral image and i is the original image.

- Use the following pair of recurrences to compute the integral image in just one pass.

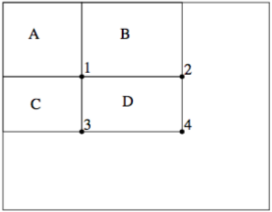
$$s(x, y) = s(x, y - 1) + i(x, y)$$

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

  where we define $s(x, -1) = 0$ and $ii(-1, y) = 0$.

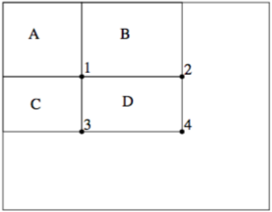
Integral image

- The sum of a particular rectangle can be computed in just 4 references using the integral image.



Integral image

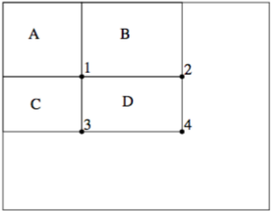
- The sum of a particular rectangle can be computed in just 4 references using the integral image.



- The value at point 1 is the sum of the pixels in rectangle A.

Integral image

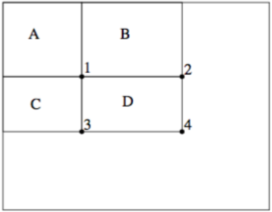
- The sum of a particular rectangle can be computed in just 4 references using the integral image.



- The value at point 1 is the sum of the pixels in rectangle A.
- Point 2 is A+B.

Integral image

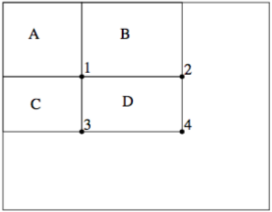
- The sum of a particular rectangle can be computed in just 4 references using the integral image.



- The value at point 1 is the sum of the pixels in rectangle A.
- Point 2 is A+B.
- Point 3 is A+C.

Integral image

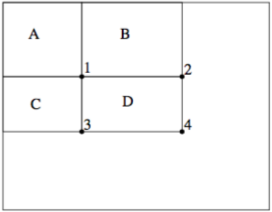
- The sum of a particular rectangle can be computed in just 4 references using the integral image.



- The value at point 1 is the sum of the pixels in rectangle A.
- Point 2 is A+B.
- Point 3 is A+C.
- Point 4 is A+B+C+D.

Integral image

- The sum of a particular rectangle can be computed in just 4 references using the integral image.



- The value at point 1 is the sum of the pixels in rectangle A.
- Point 2 is A+B.
- Point 3 is A+C.
- Point 4 is A+B+C+D.
- So, the sum within D alone is 4-2-3+1.

Boosting

- Each run, the weak learner is designed to select the single rectangle feature which best separates the positive and negative examples.

Boosting

- Each run, the weak learner is designed to select the single rectangle feature which best separates the positive and negative examples.
- The weak learner searches for the optimal threshold classification function, such that the minimum number of examples are misclassified.

Boosting

- Each run, the weak learner is designed to select the single rectangle feature which best separates the positive and negative examples.
- The weak learner searches for the optimal threshold classification function, such that the minimum number of examples are misclassified.
- The weak classifier $h_t(\mathrm{x})$ hence consists of the feature $f_t(\mathrm{x})$, a threshold θ_t, and a parity $p_t = \{-1, +1\}$ indicating the direction of the inequality sign:

$$h_t(\mathrm{x}) = \begin{cases} +1 & \text{if } p_t f_t(\mathrm{x}) < p_t \theta_t \\ -1 & \text{otherwise.} \end{cases}$$

Boosting

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = h_t : t = 1, \ldots, T$$

$$\alpha = \alpha_t : t = 1, \ldots, T$$

Boosting

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = h_t : t = 1, \ldots, T$$

$$\alpha = \alpha_t : t = 1, \ldots, T$$

- Denote the inner product over all weak classifiers as F:

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

Boosting

- Let's assume we have selected T weak classifiers and a scalar constant α_t associated with each:

$$h = h_t : t = 1, \ldots, T$$

$$\alpha = \alpha_t : t = 1, \ldots, T$$

- Denote the inner product over all weak classifiers as F:

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

- Define the strong classifier as the sign of this inner product:

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

The basic AdaBoost algorithm

Given $\mathcal{D} = (x_i, y_i), \ldots, (x_m, y_m)$ as before.

Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.

The basic AdaBoost algorithm

Given $\mathcal{D} = (x_i, y_i), \ldots, (x_m, y_m)$ as before.

Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.

1. Learn weak classifier h_t using distribution D_t.

The basic AdaBoost algorithm

Given $\mathcal{D} = (x_i, y_i), \ldots, (x_m, y_m)$ as before.

Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.

1. Learn weak classifier h_t using distribution D_t.

   Compute the weighted error for each weak classifier.

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(\mathrm{x}_i)\, \delta(h(x_i) \neq y_i), \;\; \forall h.$$

The basic AdaBoost algorithm

Given $\mathcal{D} = (x_i, y_i), \ldots, (x_m, y_m)$ as before.

Initialize the distribution D_1 to be uniform: $D_1(i) = \frac{1}{m}$.

1. Learn weak classifier h_t using distribution D_t.

   Compute the weighted error for each weak classifier.

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(\mathrm{x}_i)\, \delta(h(x_i) \neq y_i), \;\; \forall h.$$

   Select the weak classifier with minimum error.

$$h_t = \arg\min_h \epsilon_t(h)$$

The basic AdaBoost algorithm

2. Set weight α_t based on the error:

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

The basic AdaBoost algorithm

2. Set weight α_t based on the error:

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

3. Update the distribution based on the performance so far:

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

where Z_t is a normalization factor to keep D_{t+1} a distribution.

Note the careful evaluation of the term inside of the exp based on the possible $\{-1, +1\}$ values of the label.

The basic AdaBoost algorithm

- The selected weight for each new weak classifier is always positive.

$$\epsilon_t(h_t) < 1 \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} > 0$$

The basic AdaBoost algorithm

- The selected weight for each new weak classifier is always positive.

$$\epsilon_t(h_t) < 1 \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)} > 0$$

- The smaller the classification error, the bigger the weight and the more this particular weak classifier will impact the final strong classifier.

$$\epsilon(h_A) < \epsilon(h_B) \Rightarrow \alpha_A > \alpha_B$$

The basic AdaBoost algorithm

- The weights of the data points are multiplied by $\exp[-\alpha_t y_i h_t(\mathrm{x}_i)]$.

$$\exp[-\alpha_t y_i h_t(\mathrm{x}_i)] = \begin{cases} \exp[-\alpha_t] < 1 & \text{if } h_t(\mathrm{x}_i) = y_i \\ \exp[\alpha_t] > 1 & \text{if } h_t(\mathrm{x}_i) \neq y_i \end{cases}$$

The basic AdaBoost algorithm

- The weights of the data points are multiplied by $\exp[-\alpha_t y_i h_t(\mathrm{x}_i)]$.

$$\exp[-\alpha_t y_i h_t(\mathrm{x}_i)] = \begin{cases} \exp[-\alpha_t] < 1 & \text{if } h_t(\mathrm{x}_i) = y_i \\ \exp[\alpha_t] > 1 & \text{if } h_t(\mathrm{x}_i) \neq y_i \end{cases}$$

- The weights of correctly classified points are reduced and the weights of incorrectly classified points are increased.

The basic AdaBoost algorithm

- The weights of the data points are multiplied by $\exp[-\alpha_t y_i h_t(\mathrm{x}_i)]$.

$$\exp[-\alpha_t y_i h_t(\mathrm{x}_i)] = \begin{cases} \exp[-\alpha_t] < 1 & \text{if } h_t(\mathrm{x}_i) = y_i \\ \exp[\alpha_t] > 1 & \text{if } h_t(\mathrm{x}_i) \neq y_i \end{cases}$$

- The weights of correctly classified points are reduced and the weights of incorrectly classified points are increased.
- Hence, the incorrectly classified points will receive more attention in the next run.

The basic AdaBoost algorithm

- At each iteration, the weights on the data points are normalized by

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

where $\mathcal{A}$ is the set of correctly classified points:

$$\{\mathrm{x}_i : y_i = h_t(\mathrm{x}_i)\}.$$

The basic AdaBoost algorithm

- At each iteration, the weights on the data points are normalized by

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

  where $\mathcal{A}$ is the set of correctly classified points:

$$\{\mathrm{x}_i : y_i = h_t(\mathrm{x}_i)\}.$$

- We can write these normalization factors as functions of α_t and h_t, then:

$$Z_t = Z_t(\alpha_t, h_t)$$

The basic AdaBoost algorithm

- We want to find h_t and α_t that minimize $Z(h_t, \alpha_t)$:

$$(h_t, \alpha_t)^* = \arg\min Z(\alpha_t, h_t)$$

The basic AdaBoost algorithm

- We want to find h_t and α_t that minimize $Z(h_t, \alpha_t)$:

$$(h_t, \alpha_t)^* = \arg\min Z(\alpha_t, h_t)$$

- Recall, we can separate Z into two parts:

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t]$$

where $\mathcal{A}$ is the set of correctly classified points:

$$\{\mathrm{x}_i : y_i = h_t(\mathrm{x}_i)\}.$$

The basic AdaBoost algorithm

- We want to find h_t and α_t that minimize $Z(h_t, \alpha_t)$:

$$(h_t, \alpha_t)^* = \arg\min Z(\alpha_t, h_t)$$

- Recall, we can separate Z into two parts:

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t]$$

  where $\mathcal{A}$ is the set of correctly classified points:

$$\{\mathrm{x}_i : y_i = h_t(\mathrm{x}_i)\}.$$

- Take the derivative w.r.t. α_t and set it to zero:

$$\frac{dZ(h_t, \alpha_t)}{d\alpha_t} = \sum_{\mathrm{x}_i \in \mathcal{A}} -D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t] = 0$$

The basic AdaBoost algorithm

- So we have

$$\sum_{\mathrm{x}_i \in \mathcal{A}} -D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t] = 0$$

The basic AdaBoost algorithm

- So we have

$$\sum_{\mathrm{x}_i \in \mathcal{A}} -D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t] = 0$$

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \frac{\exp[\alpha_t]}{\exp[-\alpha_t]}$$

The basic AdaBoost algorithm

- So we have

$$\sum_{\mathrm{x}_i \in \mathcal{A}} -D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t] = 0$$

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \frac{\exp[\alpha_t]}{\exp[-\alpha_t]}$$

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[2\alpha_t]$$

The basic AdaBoost algorithm

- So we have

$$\sum_{\mathrm{x}_i \in \mathcal{A}} -D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t] = 0$$

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \frac{\exp[\alpha_t]}{\exp[-\alpha_t]}$$

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[2\alpha_t]$$

- And, by definition, we can write the error as

$$\epsilon_t(h) = \sum_{i=1}^{m} D_t(\mathrm{x}_i)\, \delta(h(x_i) \neq y_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i),\ \forall h$$

The basic AdaBoost algorithm

- Therefore, by substituting

$$\epsilon_t(h) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i), \ \forall h$$

in

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[2\alpha_t]$$

The basic AdaBoost algorithm

- Therefore, by substituting

$$\epsilon_t(h) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i), \ \forall h$$

in

$$\sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) = \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[2\alpha_t]$$

we get

$$\begin{aligned} 1 - \epsilon_t(h) &= \epsilon_t(h) \exp[2\alpha_t] \\ \exp[2\alpha_t] &= \tfrac{1-\epsilon_t(h)}{\epsilon_t(h)} \\ \alpha_t &= \tfrac{1}{2} \ln \left(\tfrac{1-\epsilon_t(h)}{\epsilon_t(h)} \right) \end{aligned}$$

The basic AdaBoost algorithm

Now, by substituting

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

in

$$Z(h_t, \alpha_t) \quad = \quad \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp[-\alpha_t] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp[\alpha_t]$$

The basic AdaBoost algorithm

Now, by substituting

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

in

$$Z(h_t,\alpha_t) \quad = \quad \sum_{\mathrm{x}_i\in\mathcal{A}} D_t(\mathrm{x}_i)\exp[-\alpha_t] + \sum_{\mathrm{x}_i\in\overline{\mathcal{A}}} D_t(\mathrm{x}_i)\exp[\alpha_t]$$

we get

$$Z(h_t,\alpha_t) = \sum_{\mathrm{x}_i\in\mathcal{A}} D_t(\mathrm{x}_i)\exp\left[-\ln\left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)}\right)^{1/2}\right] + \sum_{\mathrm{x}_i\in\overline{\mathcal{A}}} D_t(\mathrm{x}_i)\exp\left[\ln\left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)}\right)^{1/2}\right]$$

The basic AdaBoost algorithm

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp\left[-\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right] +$$

$$\sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp\left[\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right]$$

The basic AdaBoost algorithm

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp\left[-\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp\left[\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right]$$

$$Z(h_t, \alpha_t) = \Big(1-\epsilon_t(h_t)\Big)\sqrt{\frac{\epsilon_t(h_t)}{1-\epsilon_t(h_t)}} + \epsilon_t(h_t)\sqrt{\frac{1-\epsilon_t(h_t)}{\epsilon_t(h_t)}}$$

The basic AdaBoost algorithm

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp\left[-\ln\left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)}\right)^{1/2}\right] +$$

$$\sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp\left[\ln\left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)}\right)^{1/2}\right]$$

$$\begin{aligned} Z(h_t, \alpha_t) &= \Big(1-\epsilon_t(h_t)\Big)\sqrt{\tfrac{\epsilon_t(h_t)}{1-\epsilon_t(h_t)}} + \epsilon_t(h_t)\sqrt{\tfrac{1-\epsilon_t(h_t)}{\epsilon_t(h_t)}} \\ &= 2\sqrt{\epsilon_t(h_t)\Big(1-\epsilon_t(h_t)\Big)} \end{aligned}$$

The basic AdaBoost algorithm

$$Z(h_t,\alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp\left[-\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right] +$$

$$\sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp\left[\ln \left(\frac{1-\epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right]$$

$$\begin{aligned} Z(h_t,\alpha_t) &= \Big(1-\epsilon_t(h_t)\Big)\sqrt{\tfrac{\epsilon_t(h_t)}{1-\epsilon_t(h_t)}} + \epsilon_t(h_t)\sqrt{\tfrac{1-\epsilon_t(h_t)}{\epsilon_t(h_t)}} \\ &= 2\sqrt{\epsilon_t(h_t)\Big(1-\epsilon_t(h_t)\Big)} \end{aligned}$$

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

The basic AdaBoost algorithm

$$Z(h_t, \alpha_t) = \sum_{\mathrm{x}_i \in \mathcal{A}} D_t(\mathrm{x}_i) \exp\left[- \ln \left(\frac{1 - \epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right] + \sum_{\mathrm{x}_i \in \overline{\mathcal{A}}} D_t(\mathrm{x}_i) \exp\left[\ln \left(\frac{1 - \epsilon_t(h)}{\epsilon_t(h)} \right)^{1/2} \right]$$

$$\begin{aligned} Z(h_t, \alpha_t) &= \Big(1 - \epsilon_t(h_t)\Big) \sqrt{\tfrac{\epsilon_t(h_t)}{1 - \epsilon_t(h_t)}} + \epsilon_t(h_t) \sqrt{\tfrac{1 - \epsilon_t(h_t)}{\epsilon_t(h_t)}} \\ &= 2 \sqrt{\epsilon_t(h_t) \Big(1 - \epsilon_t(h_t)\Big)} \\ & \quad \text{Changing variable } \gamma_t = \tfrac{1}{2} - \epsilon_t(h_t), \;\; \gamma_t \in (0, \tfrac{1}{2}] \\ &= \sqrt{1 - 4\gamma_t^2} \;\; \leq \;\; \exp[-2\gamma_t^2] \end{aligned}$$

The basic AdaBoost algorithm

- Therefore, after t steps, the error rate of the strong classifier is bounded on top by

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

The basic AdaBoost algorithm

- Therefore, after t steps, the error rate of the strong classifier is bounded on top by

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

- Hence, each step decreases the upper bound exponentially.

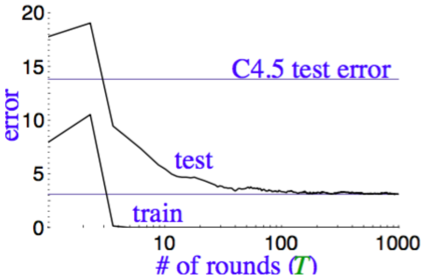
The basic AdaBoost algorithm

- Therefore, after t steps, the error rate of the strong classifier is bounded on top by

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

- Hence, each step decreases the upper bound exponentially.
- And, a weak classifier with small error rate will lead to faster descent.

The basic AdaBoost algorithm



(boosting C4.5 on "letter" dataset)

Plot taken from Schapire's and Corso's slides

Reference

- Jason Corso's slides (University of Michigan)
- Y.Freund and R.E. Schapire. A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. Journal of Computer and System Science, 55(1):119139, 1997.
- R.E. Schapire. The boosting approach to machine learning: an overview. In MSRI Workshop on Nonlinear Estimation and Classification, 2002.
- Paul Viola, Michael Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", CVPR 2001.
- Paul Viola, Michael Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision 57(2), 137-154, 2004.

Thank you!

Dr. Víctor Uc Cetina
cetina@informatik.uni-hamburg.de