Algorithm: Bagging binary classifiers

Given \((x_1, y_1), \ldots, (x_n, y_n), x \in \mathcal{X}, y \in \{-1, +1\}\)

- For \(b = 1, \ldots, B\)
  - Sample a bootstrap dataset \(B_b\) of size \(n\). For each entry in \(B_b\), select \((x_i, y_i)\) with probability \(\frac{1}{n}\). Some \((x_i, y_i)\) will repeat and some won’t appear in \(B_b\).
  - Learn a classifier \(f_b\) using data in \(B_b\).
- Define the classification rule to be
  \[
  f_{bag}(x_0) = \text{sign} \left( \sum_{b=1}^{B} f_b(x_0) \right).
  \]

- With bagging, we observe that a committee of classifiers votes on a label.
- Each classifier is learned on a bootstrap sample from the data set.
- Learning a collection of classifiers is referred to as an ensemble method.
Boosting is another powerful method for ensemble learning. It is similar to bagging in that a set of classifiers are combined to make a better one.

It works for any classifier, but a “weak” one that is easy to learn is usually chosen. (weak = accuracy a little better than random guessing)

**Short history**

1984: Leslie Valiant and Michael Kearns ask if “boosting” is possible.
1990: Yoav Freund creates an optimal boosting algorithm.
Bagging vs Boosting (overview)

Bagging:
- Training sample
- Bootstrap sample → $f_1(x)$
- Bootstrap sample → $f_2(x)$
- Bootstrap sample → $f_3(x)$

Boosting:
- Training sample
- Weighted sample → $f_1(x)$
- Weighted sample → $f_2(x)$
- Weighted sample → $f_3(x)$
The AdaBoost Algorithm (Sampling Version)

Training sample
Weighted sample
Weighted sample
Weighted sample
α₃, \( f_3(x) \)
Sample and classify \( B_3 \)

Weighted sample
Sample and classify \( B_2 \)
Weighted sample
Sample and classify \( B_1 \)

Sample and classify \( B_2 \)
Sample and classify \( B_1 \)

Weighted sample
Sample and classify \( B_3 \)

f_{boost}(x₀) = \text{sign} \left( \sum_{t=1}^{T} α_t f_t(x₀) \right)
The AdaBoost Algorithm (Sampling Version)

Algorithm: Boosting a binary classifier

Given \((x_1, y_1), \ldots, (x_n, y_n), x \in \mathcal{X}, y \in \{-1, +1\}\), set \(w_1(i) = \frac{1}{n}\) for \(i = 1 : n\)

- For \(t = 1, \ldots, T\)
  1. Sample a bootstrap dataset \(\mathcal{B}_t\) of size \(n\) according to distribution \(w_t\).
     Notice we pick \((x_i, y_i)\) with probability \(w_t(i)\) and not \(\frac{1}{n}\).
  2. Learn a classifier \(f_t\) using data in \(\mathcal{B}_t\).
  3. Set \(\epsilon_t = \sum_{i=1}^{n} w_t(i) \mathbb{1}\{y_i \neq f_t(x_i)\}\) and \(\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)\).
  4. Scale \(\hat{w}_{t+1}(i) = w_t(i)e^{-\alpha_t y_i f_t(x_i)}\) and set \(w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}\).

- Set the classification rule to be

\[
f_{\text{boost}}(x_0) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t f_t(x_0) \right).
\]

Comment: Description usually simplified to “learn classifier \(f_t\) using distribution \(w_t\).”
**Boosting a Decision Stump (Example 1)**

**Original data**

Uniform distribution, $w_1$

Learn *weak classifier*

Here: Use a decision stump

- $x_1 > 1.7$
  - $\hat{y} = 1$
  - $\hat{y} = 3$
Boosting a decision stump (example 1)

Round 1 classifier

Weighted error: $\epsilon_1 = 0.3$
Weight update: $\alpha_1 = 0.42$
Boosting a decision stump (example 1)

Weighted data

After round 1
Round 2 classifier

Weighted error: $\epsilon_2 = 0.21$
Weight update: $\alpha_2 = 0.65$
Boosting a decision stump (example 1)

Weighted data

After round 2
Round 2 classifier

Weighted error: $\epsilon_3 = 0.14$
Weight update: $\alpha_3 = 0.92$
Boosting a decision stump (example 1)

Classifier after three rounds

0.42 \times

0.65 \times

0.92 \times

+ + ++

- - -

+ + ++

- - -

+ + ++

- - -

+ + ++

- - -
Example problem
Random guessing
50% error
Decision stump
45.8% error
Full decision tree
24.7% error
Boosted stump
5.8% error
Point = one dataset. Location = error rate w/ and w/o boosting. The boosted version of the same classifier almost always produces better results.
(left) Boosting a bad classifier is often better than not boosting a good one. (right) Boosting a good classifier is often better, but can take more time.
**Boosting and Feature Maps**

**Q:** What makes boosting work so well?

**A:** This is a well-studied question. We will present one analysis later, but we can also give intuition by tying it in with what we’ve already learned.

The classification for a new \( x_0 \) from boosting is

\[
f_{\text{boost}}(x_0) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t f_t(x_0) \right).
\]

Define \( \phi(x) = [f_1(x), \ldots, f_T(x)]^\top \), where each \( f_t(x) \in \{-1, +1\} \).

- We can think of \( \phi(x) \) as a high dimensional feature map of \( x \).
- The vector \( \alpha = [\alpha_1, \ldots, \alpha_T]^\top \) corresponds to a hyperplane.
- So the classifier can be written \( f_{\text{boost}}(x_0) = \text{sign}(\phi(x_0)^\top \alpha) \).
- Boosting learns the feature mapping and hyperplane simultaneously.
APPLICATION: FACE DETECTION
**Problem:** Locate the faces in an image or video.

**Processing:** Divide image into patches of different scales, e.g., $24 \times 24$, $48 \times 48$, etc. Extract *features* from each patch.

**Classify** each patch as face or no face using a *boosted decision stump*. This can be done in real-time, for example by your digital camera (at 15 fps).

- One patch from a larger image. Mask it with many “feature extractors.”
- Each pattern gives one number, which is the sum of all pixels in black region minus sum of pixels in white region (total of 45,000+ features).
6. Conclusions

We have presented an approach for face detection which minimizes computation time while achieving high detection accuracy. The approach was used to construct a face detection system which is approximately 15 times faster than any previous approach. Preliminary experiments, which will be described elsewhere, show that highly efficient detectors for other objects, such as pedestrians or automobiles, can also be constructed in this way.

This paper brings together new algorithms, representations, and insights which are quite generic and may well have broader application in computer vision and image processing.

The first contribution is a new technique for computing a rich set of image features using the integral image. In order to achieve true scale invariance, almost all face detection systems must operate on multiple image scales. The integral image, by eliminating the need to compute a multi-scale image pyramid, reduces the initial image processing required for face detection.
ANALYSIS OF BOOSTING
Training error theorem

We can use analysis to make a statement about the accuracy of boosting on the training data.

Theorem: Under the AdaBoost framework, if $\epsilon_t$ is the weighted error of classifier $f_t$, then for the classifier $f_{boost}(x_0) = \text{sign}(\sum_{t=1}^{T} \alpha_t f_t(x_0))$,

$$\text{training error} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}(x_i)\} \leq \exp \left( -2 \sum_{t=1}^{T} \left( \frac{1}{2} - \epsilon_t \right)^2 \right).$$

Even if each $\epsilon_t$ is only a little better than random guessing, the sum over $T$ classifiers can lead to a large negative value in the exponent when $T$ is large.

For example, if we set:

$\epsilon_t = 0.45, \; T = 1000 \rightarrow \text{training error} \leq 0.0067.$
Setup

We break the proof into three steps. It is an application of the fact that

\[
\text{if } a < b \quad \text{and} \quad b < c \quad \text{then} \quad a < c
\]

Step 2  Step 3  conclusion

- Step 1 calculates the value of \( b \).
- Steps 2 and 3 prove the two inequalities.

Also recall the following step from AdaBoost:

- Update \( \hat{w}_{t+1}(i) = w_t(i)e^{-\alpha ty_if_i(x_i)} \).
- Normalize \( w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)} \) \( \rightarrow \) Define \( Z_t = \sum_j \hat{w}_{t+1}(j) \).
Theorem \((a \leq b \leq c)\)

**Step 1**

We first want to expand the equation of the weights to show that

\[ w_{T+1}(i) = \frac{1}{n} \frac{e^{-y_i \sum_{t=1}^{T} \alpha_t f_t(x_i)}}{\prod_{t=1}^{T} Z_t} := \frac{1}{n} \frac{e^{-y_i h_T(x_i)}}{\prod_{t=1}^{T} Z_t} \rightarrow h_T(x) := \sum_{t=1}^{T} \alpha_t f_t(x_i) \]

**Derivation of Step 1:**

Notice the update rule: 

\[ w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{-\alpha_t y_i f_t(x_i)} \]

Do the same expansion for \( w_t(i) \) and continue until reaching \( w_1(i) = \frac{1}{n} \),

\[ w_{T+1}(i) = w_1(i) \frac{e^{-\alpha_1 y_i f_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-\alpha_T y_i f_T(x_i)}}{Z_T} \]

**The product** \( \prod_{t=1}^{T} Z_t \) **is “b” above.** We use this form of \( w_{T+1}(i) \) in Step 2.
**Proof of Theorem** \((a \leq b \leq c)\)

**Step 2**

Next show the training error of \(f_{\text{boost}}^{(T)}\) (boosting after \(T\) steps) is \(\leq \prod_{t=1}^{T} Z_t\). Currently we know

\[
w_{T+1}(i) = \frac{1}{n} \frac{e^{-y_i h_T(x_i)}}{\prod_{t=1}^{T} Z_t} \Rightarrow w_{T+1}(i) \prod_{t=1}^{T} Z_t = \frac{1}{n} e^{-y_i h_T(x_i)} \quad \& \quad f_{\text{boost}}^{(T)}(x) = \text{sign}(h_T(x))
\]

**Derivation of Step 2:**

Observe that \(0 < e^{z_1}\) and \(1 < e^{z_2}\) for any \(z_1 < 0 < z_2\). Therefore

\[
\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{\text{boost}}^{(T)}(x_i)\} \leq \frac{1}{n} \sum_{i=1}^{n} e^{-y_i h_T(x_i)} = \sum_{i=1}^{n} w_{T+1}(i) \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} Z_t
\]

“\(a\)” is the training error – the quantity we care about.
**Proof of Theorem** \( (a \leq b \leq c) \)

**Step 3**

The final step is to calculate an upper bound on \( Z_t \), and by extension \( \prod_{t=1}^{T} Z_t \).

**Derivation of Step 3:**

This step is slightly more involved. It also shows why \( \alpha_t := \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \).

\[
Z_t = \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_t f_t(x_i)}
\]

\[
= \sum_{i : y_i = f_t(x_i)} e^{-\alpha_t} w_t(i) + \sum_{i : y_i \neq f_t(x_i)} e^{\alpha_t} w_t(i)
\]

\[
= e^{-\alpha_t} (1 - \epsilon_t) + e^{\alpha_t} \epsilon_t
\]

Remember we defined \( \epsilon_t = \sum_{i : y_i \neq f_t(x_i)} w_t(i) \), the probability of error for \( w_t \).
Derivation of Step 3 (continued):
Remember from Step 2 that

\[
\text{training error} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}(x_i)\} \leq \prod_{t=1}^{T} Z_t.
\]

and we just showed that \(Z_t = e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t}\epsilon_t\).

We want the training error to be small, so we pick \(\alpha_t\) to minimize \(Z_t\). Minimizing, we get the value of \(\alpha_t\) used by AdaBoost:

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right).
\]

Plugging this value back in gives \(Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}\).
Derivation of Step 3 (continued):

Next, re-write $Z_t$ as

$$Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

$$= \sqrt{1 - 4\left(\frac{1}{2} - \epsilon_t\right)^2}$$

Then, use the inequality $1 - x \leq e^{-x}$ to conclude that

$$Z_t = \left(1 - 4\left(\frac{1}{2} - \epsilon_t\right)^2\right)^{\frac{1}{2}} \leq \left(e^{-4\left(\frac{1}{2} - \epsilon_t\right)^2}\right)^{\frac{1}{2}} = e^{-2\left(\frac{1}{2} - \epsilon_t\right)^2}.$$
**PROOF OF THEOREM**

Concluding the right inequality \((a \leq b \leq c)\)

Because both sides of \(Z_t \leq e^{-2(\frac{1}{2} - \epsilon_t)^2}\) are positive, we can say that

\[
\prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} e^{-2(\frac{1}{2} - \epsilon_t)^2} = e^{-2 \sum_{i=1}^{T} (\frac{1}{2} - \epsilon_t)^2}.
\]

This concludes the “\(b \leq c\)” portion of the proof.

Combining everything

\[
\text{training error} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{y_i \neq f_{\text{boost}}(x_i)\} \leq \prod_{t=1}^{T} Z_t \leq e^{-2 \sum_{i=1}^{T} (\frac{1}{2} - \epsilon_t)^2}.
\]

We set out to prove “\(a < c\)” and we did so by using “\(b\)” as a stepping-stone.
Q: Driving the training error to zero leads one to ask, does boosting overfit?
A: Sometimes, but very often it doesn’t!