COMS 4721: Machine Learning for Data Science
Lecture 13, 3/2/2017

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

BOOSTING

General method for taking any binary classifier and improving it.

Robert E. Schapire and Yoav Freund, *Boosting: Foundations and Algorithms*, MIT Press, 2012.
See this textbook for many more details. (I borrow some figures from that book.)

Algorithm: Bagging binary classifiers

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

sample uniformly with replacement

- For $b = 1, \ldots, B$
  - same size as original dataset.
  - Sample a bootstrap dataset $\mathcal{B}_b$ of size n. For each entry in $\mathcal{B}_b$, select (x_i, y_i) with probability $\frac{1}{n}$. Some (x_i, y_i) will repeat and some won't appear in $\mathcal{B}_b$.
  - Learn a classifier f_b using data in $\mathcal{B}_b$.

    (So we now have B classifiers where each one is learned on a specific bootstrapping dataset.)
- Define the classification rule to be

$$f_{bag}(x_0) = \text{sign}\left(\sum_{b=1}^{B} f_b(x_0)\right).$$

bagging of classifiers

evaluate x_0 on each classifier & then simply take a majority vote.

- 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

(development of this idea, motivation: how can we take a mediocre classifier, and boost them so the sum of these classifiers gives a very accurate prediction label

How is it that a committee of blockheads can somehow arrive at highly reasoned decisions, despite the weak judgment of the individual members?

- Schapire & Freund, "Boosting: Foundations and Algorithms"

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.

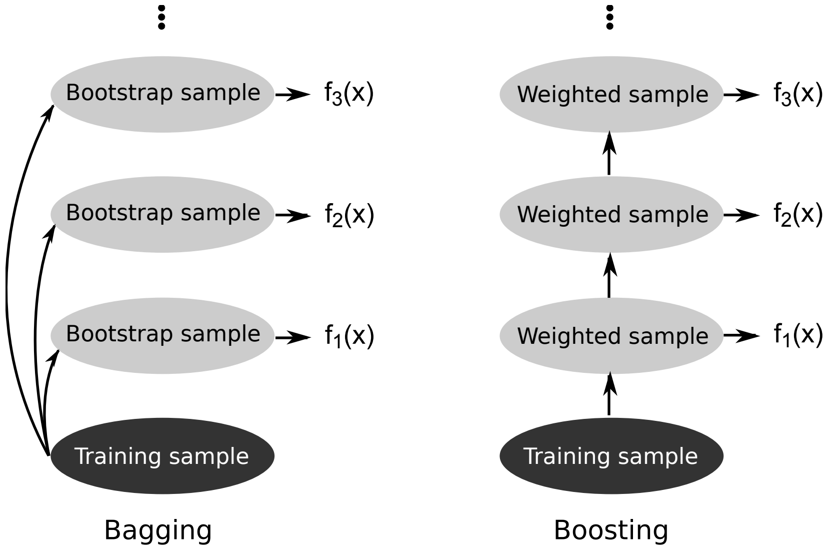
1989 : Robert Schapire creates first boosting algorithm.

1990 : Yoav Freund creates an optimal boosting algorithm.

1995 : Freund and Schapire create AdaBoost (Adaptive Boosting), the major boosting algorithm.

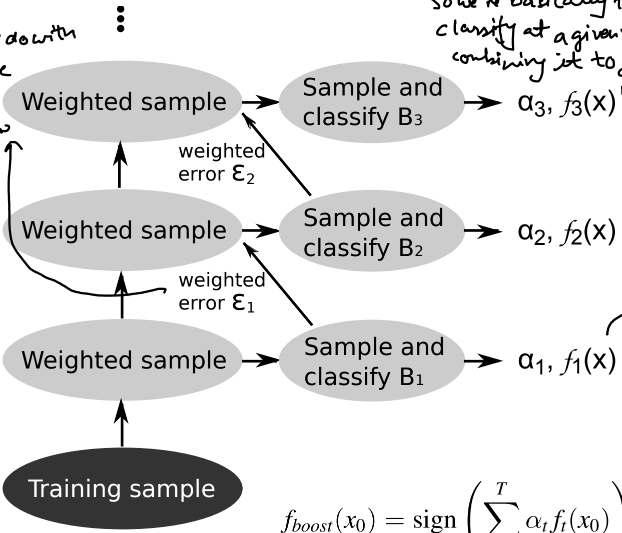
So boosting at a very high level is almost the same but
different in a very crucial way.
So what boosting does is at first creates
a weighted sample of our data set to learn a classifier.
And then given this weighted sample it generates a new weighted sample.
So it's not pulling from the original training sample.
It's pulling from the weighted sample of the previous round of classification.
To generate a new data set for which we get a new classifier.

Bagging vs Boosting (overview)



So we're basically taking how we classify at a given round and then combining it to get a new weighted dataset.

Based on how we do with this classifier, we get some sort of an error weight ϵ. Now we feed that into creating a second sample of our dataset.



classifier (same as with bagging)

Boosting

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

weight decision by α term

THE ADABOOST ALGORITHM (SAMPLING VERSION)

Algorithm: Boosting a binary classifier

data point id.

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

↳ iteration no.

Like a prob. distribution over the data. Initialise it to be uniform. [same we used in bagging]

- For $t = 1, \dots, T$ *
  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 $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$. **

     Calculate the prob. of error at iteration t acc. to the distribution on the data at iteration t.
  4. Scale $\hat{w}_{t+1}(i) = w_t(i) e^{-\alpha_t y_i f_t(x_i)}$ (→ +1, -1) and set $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}$

     → update the weights at iteration t+1 for each of the n data points.

     because we want to turn it into a prob. we simply normalize.
- Set the classification rule to be ***

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

Comment: Description usually simplified to "learn classifier f_t using distribution w_t."

* 1^{st}:

So on the first iteration we sample a bootstrap data set
of size n using a uniform distribution on our data. (Idea from bagging)

t^{th}:

However, at iteration t this distribution on the data might be different.
And so whatever the distribution on our data set is at iteration t,
we sample n times from that to construct a bootstrap sample bt.
Then we learn a classifier at iteration t using the data in bt.

Difference from bagging:

1. So this is exactly the same thing as was done in bagging.
The only difference is that the way that we constructed this data set
was different because it's not necessarily with a uniform distribution.
So that's the key difference.

2. i) Another key difference is that after we construct this
classifier, before we can go to the next iteration.
We need to first decide,
how do we update this weight probability distribution on our data?

ii) And also we have to get these alphas that I've discussed previously that are used in the classification.

** We then calculate the probability of error
at iteration t according to the distribution on the data at iteration t.
So what that's saying is that the error at iteration t is going to be equal.
To the sum of all of the probability weights
on the data that are misclassified.
According to the classifier that we learned only at iteration t.

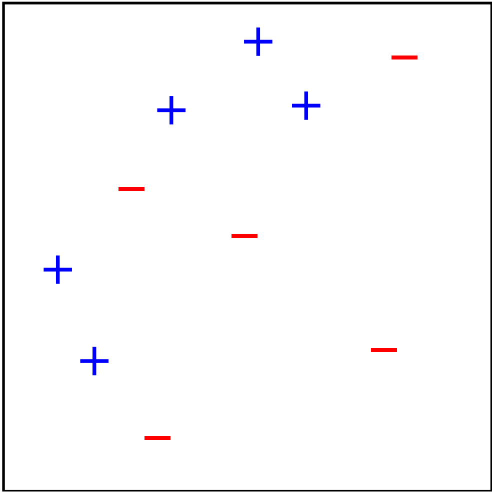
So, we learned ft from the data in bt,
we then evaluate that classifier at all n data points in our original data set
And if our classifier gets it wrong then we include that probability
of that observation and sum up those misclassified weights.

We're summing the weights only on the misclassified data according to the classifier that we learnt at iteration t.

So we do this many times, capital T times, where T is arbitrary but large.
And then we get out of this T pairs, capital T pairs of classifier and
its associated weight alpha.
And then for a new data point, the boosted classifier is equal to
the sign of each individual classifier's prediction.
Weighted by alpha for that particular classifier.
So we sum that up.
And then again it's like a majority vote except it's a majority weighted
vote in this case. And then we take the sign, that's our final classification.

BOOSTING A DECISION STUMP (EXAMPLE 1)

Take weak classifier & boost it to make it good.

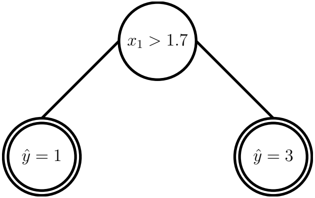


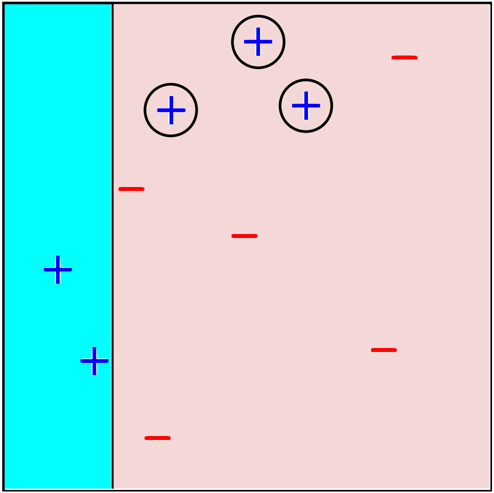
Original data

every sample has some weight.

Uniform distribution, w_1
Learn *weak classifier*

Here: Use a decision stump

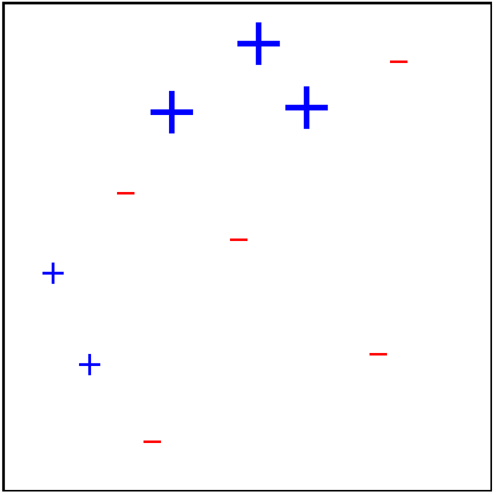




Round 1 classifier

Weighted error: $\epsilon_1 = 0.3$
Weight update: $\alpha_1 = 0.42$

$\frac{1}{2} \ln \left(\frac{0.7}{0.3} \right)$



Now we weight data

Weighted data

After round 1

data point i

$\hat{w}_2(i) = w_1(i) e^{-\alpha y_i f_1(x_i)}$

iteration 2

Correctly classified:

−1 label

predicted class −1

so $y_i f(x_i) = +1$

$\alpha_1 > 0$ always, because we're assuming our classifier is better than random guessing

$$\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$1-\epsilon_t$ → getting right

ϵ_t → getting wrong

> 0 if we are assuming it is better than random guessing. So that's gonna be a +ve no.

For correctly classified points.

$e^{-\alpha_i(1)}$ because $\alpha_1 > 0$

→ is $e^{-\alpha} < 1$

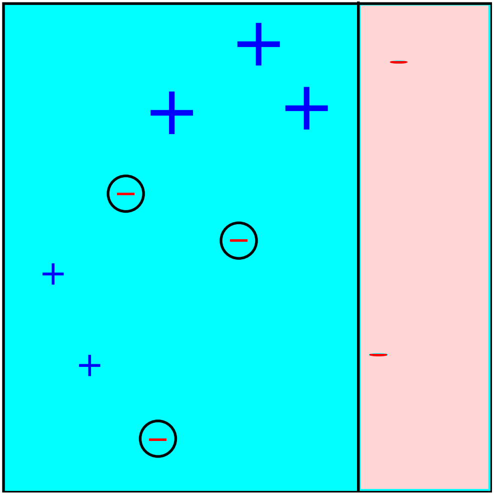
So we are downweighting all of the points that we got correct.
Whatever the weight was at previous iteration we multiply by a no. b/w 0 and 1, we downweight it.

Misclassified points → upweighting
Correctly points →

BOOSTING A DECISION STUMP (EXAMPLE 1)

Step 3: we take weights on these 3 points from the previous round.

The updated weights and sum them up.



Round 2 classifier

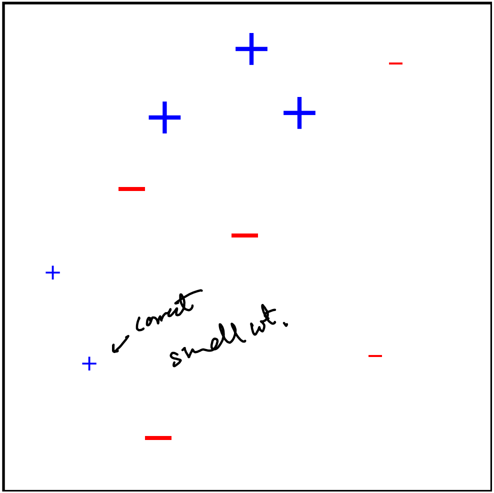
Weighted error: $\epsilon_2 = 0.21$
Weight update: $\alpha_2 = 0.65$

In the next bootstrap round, we have more of these missclassified points.

$\alpha_2 \to$ sum of weighted errors

And update weights. All correct $\to$ downweight
" Incorrect $\to$ upweight

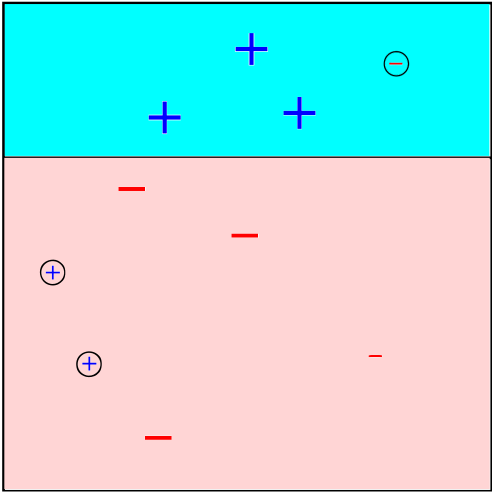
BOOSTING A DECISION STUMP (EXAMPLE 1)



Weighted data

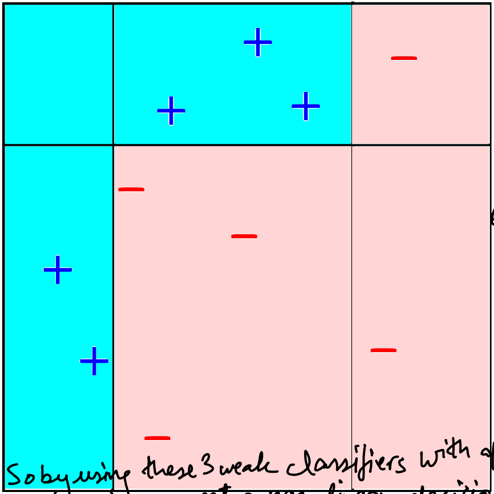
After round 2

BOOSTING A DECISION STUMP (EXAMPLE 1)

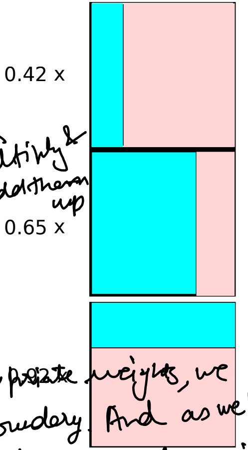


Round 2 classifier

Weighted error: $\epsilon_3 = 0.14$
Weight update: $\alpha_3 = 0.92$



Classifier after three rounds



+ 1st

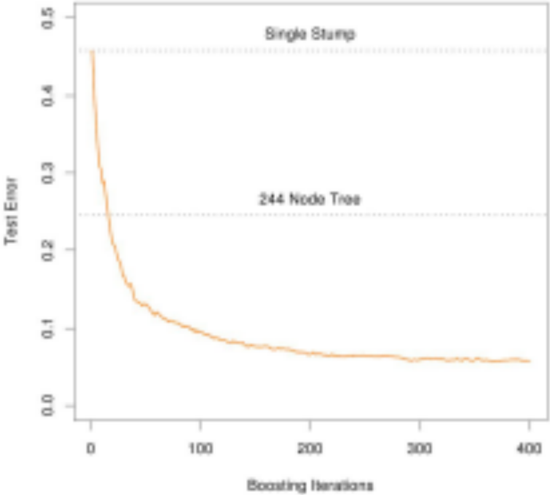
multiply & add them up

+ 2nd

3rd

So by using these 3 weak classifiers with appropriate weights, we can get a non-linear decision boundary. And as we have more and more classifiers, we can get more and more complex decision boundaries.

BOOSTING A DECISION STUMP (EXAMPLE 2)



Example problem

Random guessing
50% error

Decision stump
45.8% error

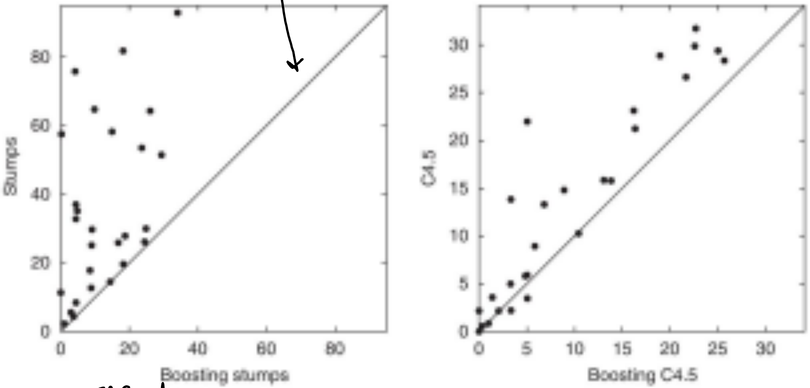
Full decision tree
24.7% error

Boosted stump
5.8% error

Many datasets

each point corresponds to a dataset

above the line here means boosting improved results.



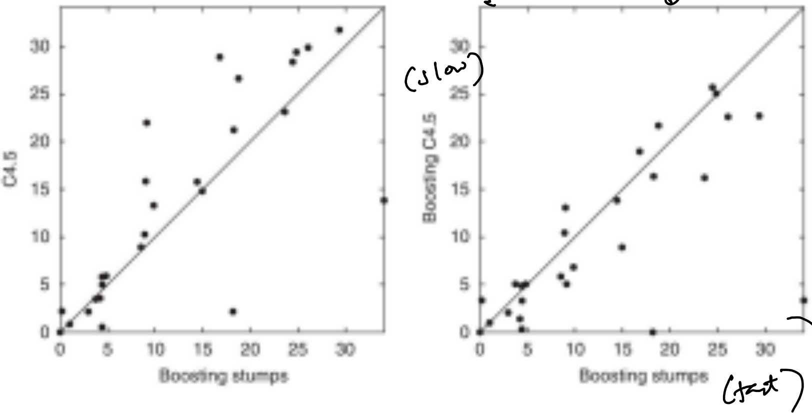
error rate.

Point = one dataset. Location = error rate w/ and w/o boosting. The boosted version of the same classifier almost always produces better results.

BOOSTING

Good result

not obvious advantage.



(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.

Overall not clear that boosting a decision stump is in general worse than boosting a very good decision tree.

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

+1 or -1

$$\text{decision of each classifier} \quad f_{boost}(x_0) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(x_0)\right).$$

1000 or more.

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 $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_T]^\top$ corresponds to a hyperplane.
- So the classifier can be written $f_{boost}(x_0) = \text{sign}(\phi(x_0)^\top \boldsymbol{\alpha})$. dot product
- Boosting learns the feature mapping and hyperplane simultaneously.

Boosting is trying to learn in an online way (adaptive online method) learn a high dimensional feature mapping along with the decision. The linear decision boundary defined by α such that we can classify all of the data in our training set.

↳ Something prove: That's going to learn a linear classifier that perfectly classifies our training set.

APPLICATION: FACE DETECTION

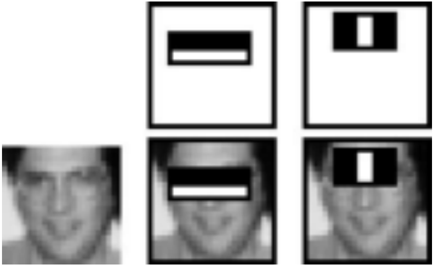
Face Detection (Viola & Jones, 2001)

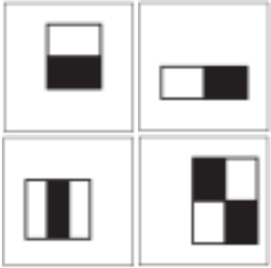
Boosting allows us to define a classifier that does ok and then boost it so that we get excellent performance.

Problem: Locate the faces in an image or video.

Processing: Divide image into patches of different scales, e.g., 24×24, 48×48, 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).

Face detection (example results)

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 ϵ_t is the weighted error of classifier f_t, then for the classifier $f_{boost}(x_0) = \text{sign}(\sum_{t=1}^{T} \alpha_t f_t(x_0))$,

goes to 0 (adding small nos.)

small no.

$$\text{training error } = \underbrace{\frac{1}{n}}_{\text{data set size}} \sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}(x_i)\} \;\leq\; \exp\Big(-2 \overbrace{\sum_{t=1}^{T}}^{\text{big no. } T} (\tfrac{1}{2} - \epsilon_t)^2\Big).$$

misclassified point

sum of each round of boosting.

Even if each ϵ_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:

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

So this theorem shows that a bunch of these weak classifiers that don't do very well on their own when combined, pushes the training error down to 0 as the no. of classifiers we learn increases to ∞.

PROOF OF THEOREM

Setup

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

$$\text{if} \quad \underbrace{a < b}_{\text{Step 2}} \quad \text{and} \quad \underbrace{b < c}_{\text{Step 3}} \quad \text{then} \quad \underbrace{a < c}_{\text{conclusion}}$$

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

Also recall the following step from AdaBoost:

- Update $\hat{w}_{t+1}(i) = w_t(i)e^{-\alpha_t y_i f_t(x_i)}$.
- Normalize $w_{t+1}(i) = \dfrac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)} \quad \longrightarrow \quad$ Define $Z_t = \sum_j \hat{w}_{t+1}(j)$.

PROOF OF THEOREM $(a \leq \boldsymbol{b} \leq c)$

Step 1

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

$$w_{T+1}(i) = \frac{1}{n}\frac{e^{-y_i \sum_{t=1}^{T} \alpha_t f_t(x_i)}}{\prod_{t=1}^{T} Z_t} = \frac{1}{n}\frac{e^{-y_i f_{boost}^{(T)}(x_i)}}{\prod_{t=1}^{T} Z_t} \quad (f_{boost}^{(T)} \text{ is up to step } T)$$

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 $(\boldsymbol{a} \leq \boldsymbol{b} \leq c)$

Step 2

Next show that the training error of $f_{boost}^{(T)}$ after T steps is $\leq \prod_{t=1}^{T} Z_t$.

From Step 1: $w_{T+1}(i) = \frac{1}{n} \frac{e^{-y_i f_{boost}^{(T)}(x_i)}}{\prod_{t=1}^{T} Z_t} \implies w_{T+1}(i) \prod_{t=1}^{T} Z_t = \frac{1}{n} e^{-y_i f_{boost}^{(T)}(x_i)}$

Derivation of Step 2:

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

$$
\begin{aligned}
\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}^{(T)}(x_i)\} &\leq \frac{1}{n} \sum_{i=1}^{n} e^{-y_i f_{boost}^{(T)}(x_i)} \\
&= \sum_{i=1}^{n} w_{T+1}(i) \prod_{t=1}^{T} Z_t \\
&= \prod_{t=1}^{T} Z_t
\end{aligned}
$$

"a" is the training error – the quantity we care about.

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

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i f_t(x_i)} \\ &= \sum_{i\,:\,y_i = f_t(x_i)} e^{-\alpha_t} w_t(i) + \sum_{i\,:\,y_i \neq f_t(x_i)} e^{\alpha_t} w_t(i) \\ &= e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t \end{aligned}$$

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 $Z_t = e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t$.

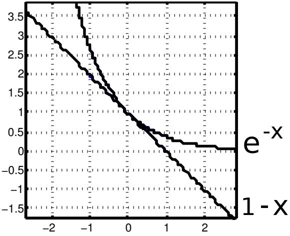
We want the training error to be small, so we pick α_t to *minimize* Z_t. Minimizing, we get the value of α_t used by AdaBoost:

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

Plugging this value back in gives $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$.

PROOF OF THEOREM $(a \leq \boldsymbol{b} \leq \boldsymbol{c})$

Derivation of Step 3 (continued):



Next, re-write Z_t as

$$\begin{aligned} Z_t &= 2\sqrt{\epsilon_t(1-\epsilon_t)} \\ &= \sqrt{1-4(\frac{1}{2}-\epsilon_t)^2} \end{aligned}$$

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

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

PROOF OF THEOREM

Concluding the right inequality $(a \leq \boldsymbol{b} \leq \boldsymbol{c})$

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

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

This concludes the "$b \leq c$" portion of the proof.

Combining everything

$$\text{training error} \;=\; \overbrace{\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}(x_i)\}}^{a} \;\leq\; \overbrace{\prod_{t=1}^{T} Z_t}^{b} \;\leq\; \overbrace{e^{-2\sum_{t=1}^{T}(\frac{1}{2}-\epsilon_t)^2}}^{c}.$$

We set out to prove "$a < c$" and we did so by using "b" as a stepping-stone.

Training vs Testing error

Q: Driving the training error to zero leads one to ask, does boosting overfit?

A: Sometimes, but very often it doesn't!