Boosting

CSE 250B

Weak learners

It is often easy to come up with a **weak classifier**, one that is only slightly better than random guessing.

$$\Pr(h(X) \neq Y) = \frac{1}{2} - \epsilon$$

A learning algorithm that can consistently generate such classifiers is called a **weak learner**.

Is it possible to systematically boost the quality of a weak learner?

Blueprint:

- Think of the data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ as a distribution D on $\mathcal{X} \times \mathcal{Y}$
- Repeat for $t = 1, 2, \ldots$:
  - Feed D to the weak learner, get back a weak classifier h_t
  - Reweight D to put more emphasis on points that h_t gets wrong
- Combine all these h_t's somehow

Adaboost

Assume $\mathcal{Y} = \{-1, 1\}$. Given: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \mathcal{Y}$.

1. Initialize $D_1(i) = 1/n$ for all $i = 1, 2, \ldots, n$
2. For $t = 1, 2, \ldots, T$:
   - Give D_t to weak learner, get back some $h_t : \mathcal{X} \to [-1, 1]$
   - Update distribution:

$$r_t = \sum_{i=1}^{n} D_t(i) y^{(i)} h_t(x^{(i)}) \in [-1, 1] \quad (h_t\text{'s margin of success})$$

$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t} \in \mathbb{R} \quad (\text{strength of update})$$

$$D_{t+1}(i) \propto D_t(i) \exp\left(-\alpha_t y^{(i)} h_t(x^{(i)})\right)$$

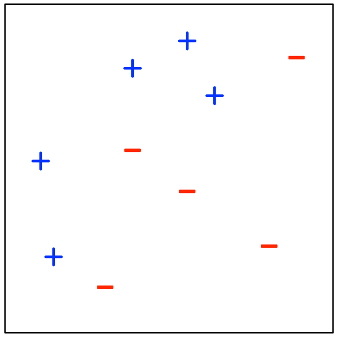
3. Final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Example

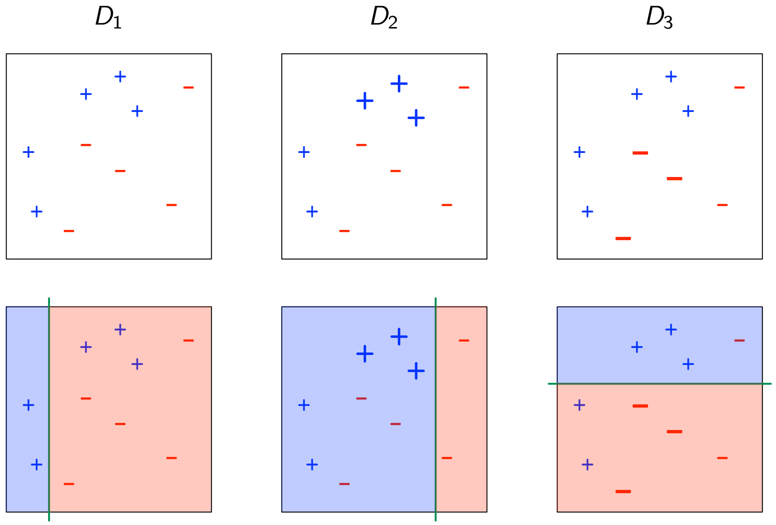
(From Freund and Schapire's tutorial.)

Training set:



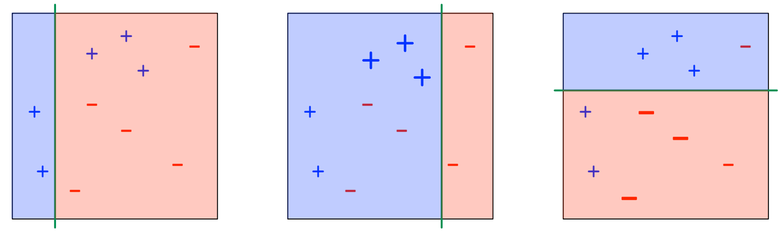
Weak classifiers: single-coordinate thresholds, popularly known as "decision stumps" (in this case, horizontal and vertical lines)

Example, cont'd



h_1 $r_1 = 0.40, \alpha_1 = 0.42$ | h_2 $r_2 = 0.58, \alpha_2 = 0.65$ | h_3 $r_3 = 0.72, \alpha_3 = 0.92$

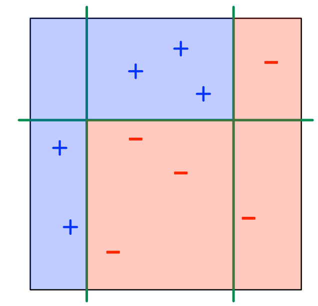
Example, cont'd



h_1 $\alpha_1 = 0.42$ | h_2 $\alpha_2 = 0.65$ | h_3 $\alpha_3 = 0.92$

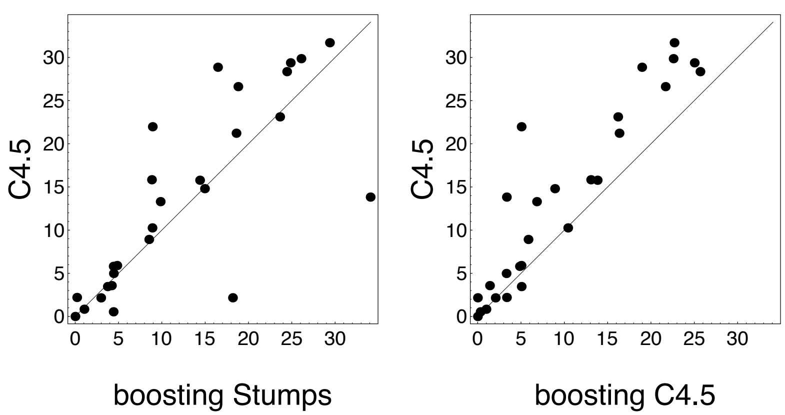
Final classifier:

$\text{sign}(0.42 h_1(x) + 0.65 h_2(x) + 0.92 h_3(x))$



Boosting versus decision tree

Freund and Schapire: results on 27 benchmark data sets from the UCI repository.



Training error dropoff

The surprising power of a weak learner:

Theorem. Suppose that on each round t, the weak learner returns a classifier h_t that differs from random by some margin γ:

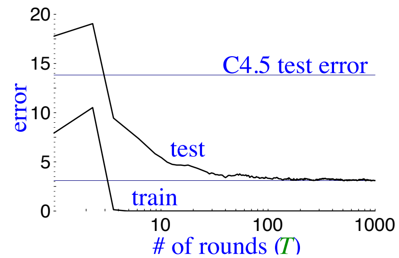
$$\left| \sum_{i=1}^{n} D_t(i) y^{(i)} h_t(x^{(i)}) \right| \geq \gamma.$$

Then after T rounds the training error is at most $e^{-\gamma^2 T/2}$.

Presumably, there will come a time T at which further reductions in training error are simply overfitting and will cause test error to rise?

Overfitting?

Freund and Schapire: boosting decision trees for “letter” dataset.



- Test error does not increase, even after 1000 rounds (total size over 2 million nodes)
- Test error keeps dropping even after training error has gone to zero:

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

Looking at the margin

Recall the final classifier (with weights normalized to sum to 1):

$$H(x) = \text{sign}\underbrace{\left(\frac{\sum_t \alpha_t h_t(x)}{\sum_t |\alpha_t|}\right)}_{\text{call this } f(x)}$$

The **margin** of this classifier on a particular $(x, y) \in \mathcal{X} \times \{-1, 1\}$ is:

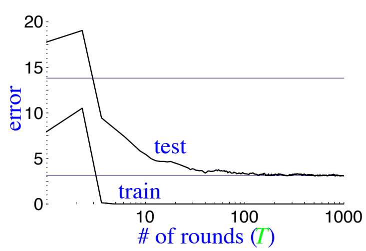
$$(\text{fraction of votes correct}) - (\text{fraction incorrect}) = yf(x) \in [-1, 1].$$

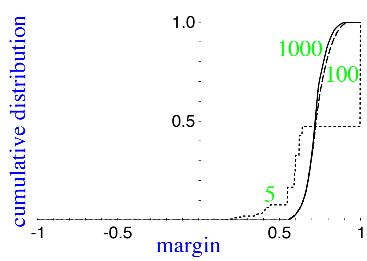
Did H just barely get it right? Or definitively?

- Intuitively: the larger a classifier’s margins on the training data, the better its generalization – that is, the lower its true error.
- There are mathematical results that make this precise.
- Adaboost seems to increase the margins on the training points even after training error has gone to zero.

Example revisited

Look at cumulative distribution of margins of training examples:





	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

Another view of boosting

Let $\mathcal{H}$ denote the set of base classifiers $\mathcal{X} \to \{-1, 1\}$.
For instance, $\mathcal{H} = \{\text{decision stumps}\}$.

Imagine a representation of $x \in \mathcal{X}$ in which each $h \in \mathcal{H}$ corresponds to a feature:

$$\Phi(x) = (h(x) : h \in \mathcal{H})$$

The final classifier found by boosting,

$$H(x) = \text{sign}\underbrace{\left(\sum_t \alpha_t h_t(x)\right)}_{\text{call this } f(x)}.$$

is a linear classifier in this enhanced space.

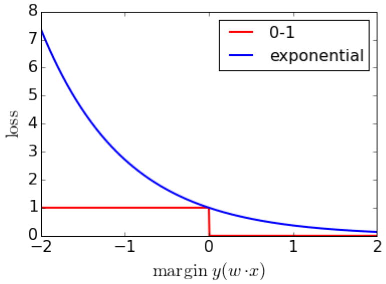
What kind of linear classifier does boosting return? Is it optimizing some loss function?

Minimizing exponential loss

It can be shown that boosting is looking for the linear classifier that minimizes the **exponential loss**:

$$\frac{1}{n}\sum_{i=1}^{n} e^{-y^{(i)} f(x^{(i)})}.$$

This loss function is yet another convex upper bound on 0-1 loss:



Boosting is a **coordinate descent** procedure for minimizing the loss.

Other ensemble methods

1. Endless variants of boosting
2. Bagging (bootstrapping aggregation)
3. Random forests

Bagging

Given a data set S of n labeled points:

- For $t = 1$ to T:
    - Choose n' points randomly, with replacement, from S.
    - Fit a classifier h_t to these points.

(For instance: $T = 100$ or 1000, $n' = n$.)

Final predictor: majority vote of $h_1, \ldots, h_T$.

The resampling and averaging reduces overfitting.

Random forests

Rather like bagging decision trees, but with an additional source of randomization.

Given a data set S of n labeled points:

- For $t = 1$ to T:
    - Choose n' points randomly, with replacement, from S.
    - Fit a decision tree h_t to these points. When building the tree, at each node restrict the split direction to be one of k features chosen at random.

(For instance: $k = \sqrt{p}$ when data is p-dimensional.)

Final predictor: majority vote of $h_1, \ldots, h_T$.