CS229 Supplemental Lecture notes

John Duchi

1 Boosting

We have seen so far how to solve classification (and other) problems when we have a data representation already chosen. We now talk about a procedure, known as *boosting*, which was originally discovered by Rob Schapire, and further developed by Schapire and Yoav Freund, that automatically chooses feature representations. We take an optimization-based perspective, which is somewhat different from the original interpretation and justification of Freund and Schapire, but which lends itself to our approach of (1) choose a representation, (2) choose a loss, and (3) minimize the loss.

Before formulating the problem, we give a little intuition for what we are going to do. Roughly, the idea of boosting is to take a *weak learning* algorithm—any learning algorithm that gives a classifier that is slightly better than random—and transforms it into a *strong* classifier, which does much much better than random. To build a bit of intuition for what this means, consider a hypothetical digit recognition experiment, where we wish to distinguish 0s from 1s, and we receive images we must classify. Then a natural weak learner might be to take the middle pixel of the image, and if it is colored, call the image a 1, and if it is blank, call the image a 0. This classifier may be far from perfect, but it is likely better than random. Boosting procedures proceed by taking a collection of such weak classifiers, and then reweighting their contributions to form a classifier with much better accuracy than any individual classifier.

With that in mind, let us formulate the problem. Our interpretation of boosting is as a coordinate descent method in an infinite dimensional space, which—while it sounds complex—is not so bad as it seems. First, we assume we have raw input examples $x \in \mathbb{R}^n$ with labels $y \in \{-1, 1\}$, as is usual in binary classification. We also assume we have an infinite collection of *feature* functions $\phi_j : \mathbb{R}^n \to \{-1, 1\}$ and an infinite vector $\theta = [\theta_1 \ \theta_2 \ \cdots]^T$, but

which we assume always has only a finite number of non-zero entries. For our classifier we use

$$h_\theta(x) = \text{sign}\left(\sum_{j=1}^{\infty} \theta_j \phi_j(x)\right).$$

We will abuse notation, and define $\theta^T \phi(x) = \sum_{j=1}^{\infty} \theta_j \phi_j(x)$.

In boosting, one usually calls the features ϕ_j *weak hypotheses*. Given a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$, we call a vector $p = (p^{(1)}, \ldots, p^{(m)})$ a distribution on the examples if $p^{(i)} \geq 0$ for all i and

$$\sum_{i=1}^{m} p^{(i)} = 1.$$

Then we say that there is a *weak learner with margin* $\gamma > 0$ if for any distribution p on the m training examples there exists one weak hypothesis ϕ_j such that

$$\sum_{i=1}^{m} p^{(i)} 1\left\{y^{(i)} \neq \phi_j(x^{(i)})\right\} \leq \frac{1}{2} - \gamma. \tag{1}$$

That is, we assume that there is *some* classifier that does slightly better than random guessing on the dataset. The existence of a weak learning algorithm is an assumption, but the surprising thing is that we can transform any weak learning algorithm into one with perfect accuracy.

In more generality, we assume we have access to a *weak learner*, which is an algorithm that takes as input a distribution (weights) p on the training examples and returns a classifier doing slightly better than random. We will

(i) **Input:** A distribution $p^{(1)}, \ldots, p^{(m)}$ and training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ with $\sum_{i=1}^m p^{(i)} = 1$ and $p^{(i)} \geq 0$

(ii) **Return:** A weak classifier $\phi_j : \mathbb{R}^n \to \{-1, 1\}$ such that

$$\sum_{i=1}^{m} p^{(i)} 1\left\{y^{(i)} \neq \phi_j(x^{(i)})\right\} \leq \frac{1}{2} - \gamma.$$

Figure 1: Weak learning algorithm

show how, given access to a weak learning algorithm, boosting can return a classifier with perfect accuracy on the training data. (Admittedly, we would like the classifer to generalize well to unseen data, but for now, we ignore this issue.)

1.1 The boosting algorithm

Roughly, boosting begins by assigning each training example equal weight in the dataset. It then receives a weak-hypothesis that does well according to the current weights on training examples, which it incorporates into its current classification model. It then reweights the training examples so that examples on which it makes mistakes receive higher weight—so that the weak learning algorithm focuses on a classifier doing well on those examples—while examples with no mistakes receive lower weight. This repeated reweighting of the training data coupled with a weak learner doing well on examples for which the classifier currently does poorly yields classifiers with good performance.

The boosting algorithm specifically performs *coordinate descent* on the exponential loss for classification problems, where the objective is

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \exp(-y^{(i)} \theta^T \phi(x^{(i)})).$$

We first show how to compute the exact form of the coordinate descent update for the risk $J(\theta)$. Coordinate descent iterates as follows:

(i) Choose a coordinate $j \in \mathbb{N}$

(ii) Update θ_j to

$$\theta_j = \arg\min_{\theta_j} J(\theta)$$

while leaving θ_k identical for all $k \neq j$.

We iterate the above procedure until convergence.

In the case of boosting, the coordinate updates are not too challenging to derive because of the analytic convenience of the exp function. We now show how to derive the update. Suppose we wish to update coordinate k. Define

$$w^{(i)} = \exp\left(-y^{(i)} \sum_{j \neq k} \theta_j \phi_j(x^{(i)})\right)$$

to be a weight, and note that optimizing coordinate k corresponds to minimizing

$$\sum_{i=1}^{m} w^{(i)} \exp(-y^{(i)}\phi_k(x^{(i)})\alpha)$$

in $\alpha = \theta_k$. Now, define

$$W^+ := \sum_{i:y^{(i)}\phi_k(x^{(i)})=1} w^{(i)} \quad \text{and} \quad W^- := \sum_{i:y^{(i)}\phi_k(x^{(i)})=-1} w^{(i)}$$

to be the sums of the weights of examples that ϕ_k classifies correctly and incorrectly, respectively. Then finding θ_k is the same as choosing

$$\alpha = \arg\min_{\alpha} \left\{W^+ e^{-\alpha} + W^- e^{\alpha}\right\} = \frac{1}{2} \log \frac{W^+}{W^-}.$$

To see the final equality, take derivatives and set the resulting equation to zero, so we have $-W^+ e^{-\alpha} + W^- e^{\alpha} = 0$. That is, $W^- e^{2\alpha} = W^+$, or $\alpha = \frac{1}{2} \log \frac{W^+}{W^-}$.

What remains is to choose the particular coordinate to perform coordinate descent on. We assume we have access to a weak-learning algorithm as in Figure 1, which at iteration t takes as input a distribution p on the training set and returns a weak hypothesis ϕ_t satisfying the margin condition (1). We present the full boosting algorithm in Figure 2. It proceeds in iterations $t = 1, 2, 3, \ldots$. We represent the set of hypotheses returned by the weak learning algorithm at time t by $\{\phi_1, \ldots, \phi_t\}$.

2 The convergence of Boosting

We now argue that the boosting procedure achieves 0 training error, and we also provide a rate of convergence to zero. To do so, we present a lemma that guarantees progress is made.

Lemma 2.1. *Let*

$$J(\theta^{(t)}) = \frac{1}{m} \sum_{i=1}^{m} \exp\left(- y^{(i)} \sum_{\tau=1}^{t} \theta_\tau \phi_\tau(x^{(i)}) \right).$$

Then

$$J(\theta^{(t)}) \leq \sqrt{1 - 4\gamma^2} J(\theta^{(t-1)}).$$

For each iteration $t = 1, 2, \ldots$:

(i) Define weights

$$w^{(i)} = \exp\left(- y^{(i)} \sum_{\tau=1}^{t-1} \theta_\tau \phi_\tau(x^{(i)}) \right)$$

and distribution $p^{(i)} = w^{(i)} / \sum_{j=1}^m w^{(j)}$

(ii) Construct a weak hypothesis $\phi_t : \mathbb{R}^n \to \{-1, 1\}$ from the distribution $p = (p^{(1)}, \ldots, p^{(m)})$ on the training set

(iii) Compute $W_t^+ = \sum_{i:y^{(i)}\phi_t(x^{(i)})=1} w^{(i)}$ and $W_t^- = \sum_{i:y^{(i)}\phi_t(x^{(i)})=-1} w^{(i)}$ and set

$$\theta_t = \frac{1}{2} \log \frac{W_t^+}{W_t^-}.$$

Figure 2: Boosting algorithm

As the proof of the lemma is somewhat involved and not the central focus of these notes—though it is important to know one's algorithm will converge!—we defer the proof to Appendix A.1. Let us describe how it guarantees convergence of the boosting procedure to a classifier with zero training error.

We initialize the procedure at $\theta^{(0)} = \vec{0}$, so that the initial empirical risk $J(\theta^{(0)}) = 1$. Now, we note that for any θ, the misclassification error satisfies

$$1\left\{\text{sign}(\theta^T \phi(x)) \neq y\right\} = 1\left\{y\theta^T\phi(x) \leq 0\right\} \leq \exp\left(-y\theta^T\phi(x)\right)$$

because $e^z \geq 1$ for all $z \geq 0$. Thus, we have that the misclassification error rate has upper bound

$$\frac{1}{m} \sum_{i=1}^m 1\left\{\text{sign}(\theta^T\phi(x^{(i)})) \neq y^{(i)}\right\} \leq J(\theta),$$

and so if $J(\theta) < \frac{1}{m}$ then the vector θ makes *no* mistakes on the training data. After t iterations of boosting, we find that the empirical risk satisfies

$$J(\theta^{(t)}) \leq (1 - 4\gamma^2)^{\frac{t}{2}} J(\theta^{(0)}) = (1 - 4\gamma^2)^{\frac{t}{2}}.$$

To find how many iterations are required to guarantee $J(\theta^{(t)}) < \frac{1}{m}$, we take logarithms to find that $J(\theta^{(t)}) < 1/m$ if

$$\frac{t}{2}\log(1-4\gamma^2) < \log\frac{1}{m}, \quad \text{or} \quad t > \frac{2\log m}{-\log(1-4\gamma^2)}.$$

Using a first order Taylor expansion, that is, that $\log(1-4\gamma^2) \le -4\gamma^2$, we see that if the number of rounds of boosting—the number of weak classifiers we use—satisfies

$$t > \frac{\log m}{2\gamma^2} \ge \frac{2\log m}{-\log(1-4\gamma^2)},$$

then $J(\theta^{(t)}) < \frac{1}{m}$.

3 Implementing weak-learners

One of the major advantages of boosting algorithms is that they automatically generate features from raw data for us. Moreover, because the weak hypotheses always return values in $\{-1, 1\}$, there is no need to normalize features to have similar scales when using learning algorithms, which in practice can make a large difference. Additionally, and while this is not theoretically well-understood, many types of weak-learning procedures introduce non-linearities intelligently into our classifiers, which can yield much more expressive models than the simpler linear models of the form $\theta^T x$ that we have seen so far.

3.1 Decision stumps

There are a number of strategies for weak learners, and here we focus on one, known as *decision stumps*. For concreteness in this description, let us suppose that the input variables $x \in \mathbb{R}^n$ are real-valued. A decision stump is a function f, which is parameterized by a threshold s and index $j \in \{1, 2, \ldots, n\}$, and returns

$$\phi_{j,s}(x) = \text{sign}(x_j - s) = \begin{cases} 1 & \text{if } x_j \ge s \\ -1 & \text{otherwise.} \end{cases} \tag{2}$$

These classifiers are simple enough that we can fit them efficiently even to a weighted dataset, as we now describe.

Indeed, a decision stump weak learner proceeds as follows. We begin with a distribution—set of weights $p^{(1)}, \ldots, p^{(m)}$ summing to 1—on the training set, and we wish to choose a decision stump of the form (2) to minimize the error on the training set. That is, we wish to find a threshold $s \in \mathbb{R}$ and index j such that

$$\widehat{\mathrm{Err}}(\phi_{j,s}, p) = \sum_{i=1}^{m} p^{(i)} 1\left\{\phi_{j,s}(x^{(i)}) \neq y^{(i)}\right\} = \sum_{i=1}^{m} p^{(i)} 1\left\{y^{(i)}(x_j^{(i)} - s) \leq 0\right\} \quad (3)$$

is minimized. Naively, this could be an inefficient calculation, but a more intelligent procedure allows us to solve this problem in roughly $O(nm \log m)$ time. For each feature $j = 1, 2, \ldots, n$, we sort the raw input features so that

$$x_j^{(i_1)} \geq x_j^{(i_2)} \geq \cdots \geq x_j^{(i_m)}.$$

As the only values s for which the error of the decision stump can change are the values $x_j^{(i)}$, a bit of clever book-keeping allows us to compute

$$\sum_{i=1}^{m} p^{(i)} 1\left\{y^{(i)}(x_j^{(i)} - s) \leq 0\right\} = \sum_{k=1}^{m} p^{(i_k)} 1\left\{y^{(i_k)}(x_j^{(i_k)} - s) \leq 0\right\}$$

efficiently by incrementally modifying the sum in sorted order, which takes time $O(m)$ after we have already sorted the values $x_j^{(i)}$. (We do not describe the algorithm in detail here, leaving that to the interested reader.) Thus, performing this calcuation for each of the n input features takes total time $O(nm \log m)$, and we may choose the index j and threshold s that give the best decision stump for the error (3).

One *very* important issue to note is that by flipping the sign of the thresholded decision stump $\phi_{j,s}$, we achieve error $1 - \widehat{\mathrm{Err}}(\phi_{j,s}, p)$, that is, the error of

$$\widehat{\mathrm{Err}}(-\phi_{j,s}, p) = 1 - \widehat{\mathrm{Err}}(\phi_{j,s}, p).$$

(You should convince yourself that this is true.) Thus, it is important to also track the smallest value of $1 - \widehat{\mathrm{Err}}(\phi_{j,s}, p)$ over all thresholds, because this may be smaller than $\widehat{\mathrm{Err}}(\phi_{j,s}, p)$, which gives a better weak learner. Using this procedure for our weak learner (Fig. 1) gives the basic, but extremely useful, boosting classifier.

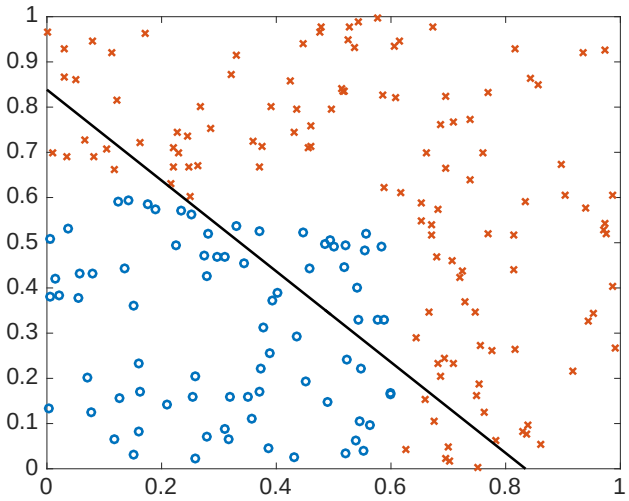

Figure 3: Best logistic regression classifier using the raw features $x \in \mathbb{R}^2$ (and a bias term $x_0 = 1$) for the example considered here.

3.2 Example

We now give an example showing the behavior of boosting on a simple dataset. In particular, we consider a problem with data points $x \in \mathbb{R}^2$, where the optimal classifier is

$$y = \begin{cases} 1 & \text{if } x_1 < .6 \text{ and } x_2 < .6 \\ -1 & \text{otherwise.} \end{cases} \tag{4}$$

This is a simple non-linear decision rule, but it is impossible for standard linear classifiers, such as logistic regression, to learn. In Figure 3, we show the best decision line that logistic regression learns, where positive examples are circles and negative examples are x's. It is clear that logistic regression is not fitting the data particularly well.

With boosted decision stumps, however, we can achieve a much better fit for the simple nonlinear classification problem (4). Figure 4 shows the boosted classifiers we have learned after different numbers of iterations of boosting, using a training set of size $m = 150$. From the figure, we see that the first decision stump is to threshold the feature x_1 at the value $s \approx .23$, that is, $\phi(x) = \text{sign}(x_1 - s)$ for $s \approx .23$.

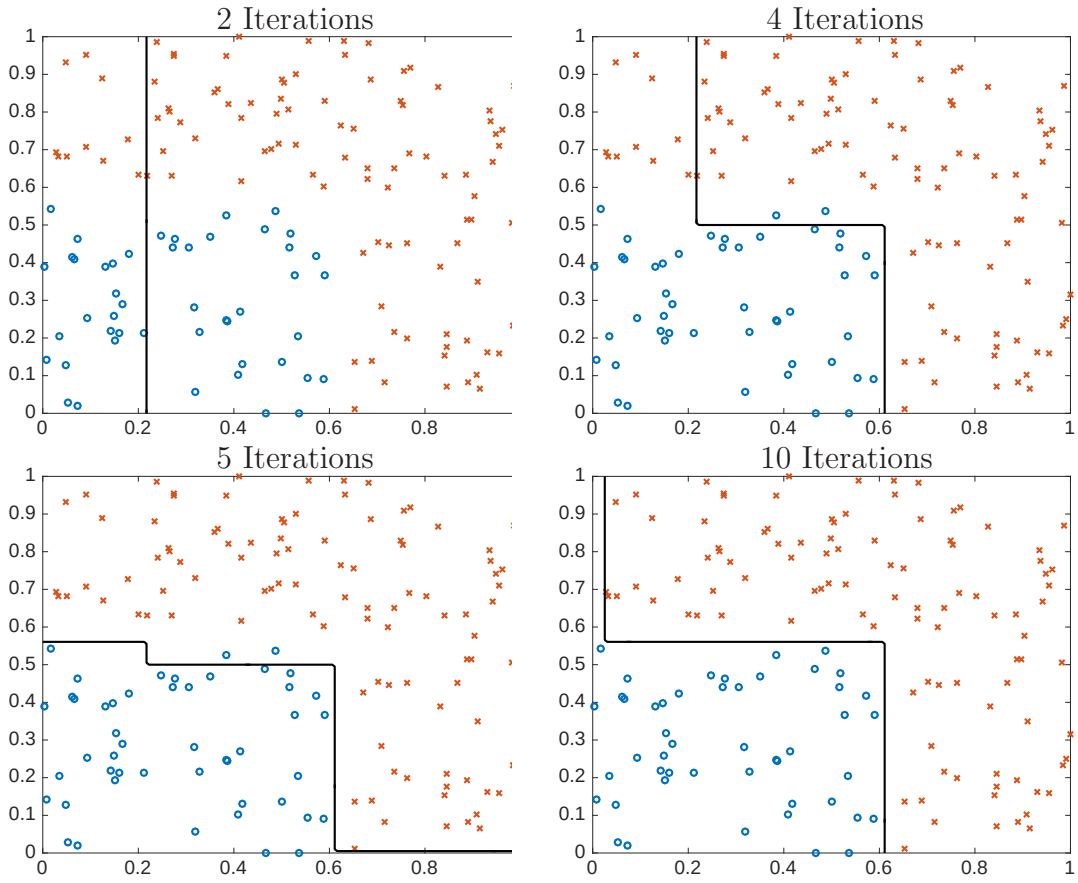

Figure 4: Boosted decision stumps after $t = 2, 4, 5$, and 10 iterations of boosting, respectively.

3.3 Other strategies

There are a huge number of variations on the basic boosted decision stumps idea. First, we do not require that the input features x_j be real-valued. Some of them may be categorical, meaning that $x_j \in \{1, 2, \ldots, k\}$ for some k, in which case natural decision stumps are of the form

$$\phi_j(x) = \begin{cases} 1 & \text{if } x_j = l \\ -1 & \text{otherwise,} \end{cases}$$

as well as variants setting $\phi_j(x) = 1$ if $x_j \in C$ for some set $C \subset \{1, \ldots, k\}$ of categories.

Another natural variation is the *boosted decision tree*, in which instead of a single level decision for the weak learners, we consider conjuctions of features or trees of decisions. Google can help you find examples and information on these types of problems.

A Appendices

A.1 Proof of Lemma 2.1

We now return to prove the progress lemma. We prove this result by directly showing the relationship of the weights at time t to those at time $t-1$. In particular, we note by inspection that

$$J(\theta^{(t)}) = \min_{\alpha} \{W_t^+ e^{-\alpha} + W_t^- e^{\alpha}\} = 2\sqrt{W_t^+ W_t^-}$$

while

$$J(\theta^{(t-1)}) = \frac{1}{m} \sum_{i=1}^{m} \exp\left(- y^{(i)} \sum_{\tau=1}^{t-1} \theta_\tau \phi_\tau(x^{(i)}) \right) = W_t^+ + W_t^-.$$

We know by the weak-learning assumption that

$$\sum_{i=1}^{m} p^{(i)} 1\left\{y^{(i)} \neq \phi_t(x^{(i)})\right\} \leq \frac{1}{2} - \gamma, \quad \text{or} \quad \frac{1}{W_t^+ + W_t^-} \underbrace{\sum_{i : y^{(i)} \phi_t(x^{(i)}) = -1} w^{(i)}}_{= W_t^-} \leq \frac{1}{2} - \gamma.$$

Rewriting this expression by noting that the sum on the right is nothing but W_t^-, we have

$$W_t^- \leq \left(\frac{1}{2} - \gamma\right)(W_t^+ + W_t^-), \quad \text{or} \quad W_t^+ \geq \frac{1+2\gamma}{1-2\gamma} W_t^-.$$

By substituting $\alpha = \frac{1}{2}\log\frac{1+2\gamma}{1-2\gamma}$ in the minimum defining $J(\theta^{(t)})$, we obtain

$$\begin{aligned}
J(\theta^{(t)}) &\leq W_t^+ \sqrt{\frac{1-2\gamma}{1+2\gamma}} + W_t^- \sqrt{\frac{1+2\gamma}{1-2\gamma}} \\
&= W_t^+ \sqrt{\frac{1-2\gamma}{1+2\gamma}} + W_t^- (1 - 2\gamma + 2\gamma)\sqrt{\frac{1+2\gamma}{1-2\gamma}} \\
&\leq W_t^+ \sqrt{\frac{1-2\gamma}{1+2\gamma}} + W_t^- (1-2\gamma)\sqrt{\frac{1+2\gamma}{1-2\gamma}} + 2\gamma \frac{1-2\gamma}{1+2\gamma}\sqrt{\frac{1+2\gamma}{1-2\gamma}} W_t^+ \\
&= W_t^+ \left[\sqrt{\frac{1-2\gamma}{1+2\gamma}} + 2\gamma\sqrt{\frac{1-2\gamma}{1+2\gamma}}\right] + W_t^- \sqrt{1-4\gamma^2},
\end{aligned}$$

where we used that $W_t^- \leq \frac{1-2\gamma}{1+2\gamma} W_t^+$. Performing a few algebraic manipulations, we see that the final expression is equal to

$$\sqrt{1-4\gamma^2}(W_t^+ + W_t^-).$$

That is, $J(\theta^{(t)}) \leq \sqrt{1-4\gamma^2} J(\theta^{(t-1)})$. □