Machine Learning – Brett Bernstein

Recitation 9: Gradient Boosting

Intro Question

1. Suppose 10 different meteorologists have produced functions $f_1, \ldots, f_{10} : \mathbb{R}^d \to \mathbb{R}$ that forecast tomorrow's noon-time temperature using the same d features. Given 1000 past data points $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ of similar forecast situations, describe a method to forecast tomorrow's noon-time temperature.

Review of AdaBoost

Assume we have access to a learning algorithm that, given a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, and a weighting $w_1, \ldots, w_n > 0$ will produce a decision function f such that

$$\frac{1}{n}\sum_{i=1}^{n} w_i \, \mathbf{1}(f(x_i) \neq y_i) < \gamma \leq 0.5.$$

We want to use this learning algorithm to build an aggregate classifier of the form

$$G(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m G_m\right).$$

AdaBoost is a method for doing this, that fits each successive G_m by reweighting the training examples according to the misclassifications of G_{m-1}. AdaBoost is is an example of the more general class of additive models.

Additive Modeling

Additive models over a *base hypothesis space* $\mathcal{H}$ take the form

$$\mathcal{F} = \{f(x) = \sum_{m=1}^{M} \nu_m h_m(x) \mid h_m \in \mathcal{H}, \nu_m \in \mathbb{R}\}.$$

Since we are taking linear combinations, we assume the h_m functions take values in $\mathbb{R}$ or some other vector space. Empirical risk minimization over $\mathcal{F}$ tries to find

$$\operatorname*{arg\,min}_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i)).$$

This in general is a difficult task, as the number of base hypotheses M is unknown, and each base hypothesis h_m ranges over all of $\mathcal{H}$. One approach to fitting additive models is to proceed stagewise in a greedy fashion.

Forward Stagewise Additive Modeling (FSAM)

The FSAM method fits additive models using the following algorithmic structure:

1. Initialize $f_0 \equiv 0$.
2. For stage $m = 1, 2, \ldots$:
   (a) Choose $h_m \in \mathcal{H}$ and $\nu_m \in \mathbb{R}$ so that
   $$f_m = f_{m-1} + \nu_m h_m$$
   has the minimum empirical risk.
   (b) The function f_m has the form
   $$f_m = \nu_1 h_1 + \cdots + \nu_m h_m.$$

When choosing h_m, ν_m during stage m, we must solve the minimization

$$(\nu_m, h_m) = \operatorname*{arg\,min}_{\nu \in \mathbb{R}, h \in \mathcal{H}} \sum_{i=1}^{n} \ell(y_i, f_{m-1}(x_i) + \nu h(x_i)).$$

Depending on the base hypothesis space $\mathcal{H}$ and loss function ℓ, this can be a difficult task. The approach we discuss next will leverage the optimization/calculus skills we have used throughout the class.

Gradient Boosting

Instead of determining how to find the optimal (ν, h) pair, we solve an easier local problem using derivatives. We can look at the equation

$$f_m(x) = f_{m-1}(x) + \nu_m h_m(x)$$

as starting from the function f_{m-1} and taking a step in the direction h_m with step length ν_m. We are looking for a step that will minimize the objective

$$\ell(y_1, f_{m-1}(x_1) + \nu_m h_m(x_1)) + \cdots + \ell(y_1, f_{m-1}(x_n) + \nu_m h_m(x_n)).$$

Suppose that instead of using a base classifier h_m, we are allowed to take a small step in a direction $d \in \mathbb{R}^n$:

$$J(d) = \ell(y_1, f_{m-1}(x_1) + d_1) + \cdots + \ell(y_1, f_{m-1}(x_n) + d_n).$$

Which direction d gives us the steepest descent? The solution is the negative gradient (or negative subgradient where not differentiable)

$$-\nabla_d J(0) = -(\partial_2 \ell(y_1, f_{m-1}(x_1)), \ldots, \partial_2 \ell(y_n, f_{m-1}(x_n)))^T.$$

This vector is sometimes called the *pseudoresidual*. Here ∂_2 means to take the partial derivative of ℓ with respect to its second argument. This is sometimes written as

$$\left.\frac{\partial}{\partial f(x_i)}\sum_{i=1}^n \ell(y_i, f(x_i))\right|_{f(x_i)=f_{m-1}(x_i)}.$$

While the negative gradient does give us the steepest descent, it may not correspond to a base hypothesis. To address this, our next step is to find the base hypothesis that is closest to the negative gradient. This is done by solving the following minimization problem

$$h_m := \operatorname*{arg\,min}_{h\in\mathcal{H}}(-\partial_2\ell(y_1, f_{m-1}(x_1)) - h(x_1))^2 + \cdots + (-\partial_2\ell(y_n, f_{m-1}(x_n)) - h(x_n))^2.$$

In words, we must find the base hypothesis $h \in \mathcal{H}$ whose values on the data are closest to the pseudoresidual vector (in Euclidean distance). Suppose we have a learning algorithm that given a dataset will (approximately) determine the ERM for the square loss. We can then create a mock dataset

$$\mathcal{D}^{(m)} = \{(x_1, -\partial_2\ell(y_1, f_{m-1}(x_1))), \ldots, (x_n, -\partial_2\ell(y_n, f_{m-1}(x_n)))\}$$

and feed it into our learning algorithm. The output of the learning algorithm will be the h_m we desired above.

Once we know h_m, the step length ν_m can be determined in several ways:

1. Perform a line search:

$$\nu_m := \operatorname*{arg\,min}_{\nu}\sum_{i=1}^n \ell(f_{m-1}(x_i), \nu h_m(x_i)).$$

2. Used a fixed constant $\nu_m \in (0, 1)$. The value 0.1 is typical, but this value can be optimized as a hyperparameter via validation.

The algorithm explained above is sometimes called *functional gradient descent* or *anyboost*. The most commonly used base hypothesis space for gradient boosting is small regression trees (HTF recommend between 4 and 8 leaves).

Examples of Gradient Boosting

Example 1 (Using $\ell(y, a) = (y - a)^2/2$)**.** To compute an arbitrary pseudoresidual we first note that

$$\partial_a(y - a)^2/2 = -(y - a)$$

giving

$$-\partial_2\ell(y_i, f_{m-1}(x_i)) = (y_i - f_{m-1}(x_i)).$$

In words, for the square loss, the pseudoresiduals are simply the residuals from the previous stage's fit. Thus, in stage m our step direction h_m is given by solving

$$h_m := \operatorname*{arg\,min}_{h\in\mathcal{H}}\sum_{i=1}^n((y_i - f_{m-1}(x_i)) - h(x_i))^2.$$

Example 2 (Using $\ell(y,a) = |y-a|$). Note that

$$\partial_a |y-a| = -\operatorname{sgn}(y-a)$$

giving

$$-\partial_2 \ell(y_i, f_{m-1}(x_i)) = \operatorname{sgn}(y_i - f_{m-1}(x_i)).$$

The absolute loss only cares about the sign of the residual from the previous stage's fit. Thus, in stage m our step direction h_m is given by solving

$$h_m := \arg\min_{h\in\mathcal{H}} \sum_{i=1}^{n} (\operatorname{sgn}(y_i - f_{m-1}(x_i)) - h(x_i))^2.$$

Example 3 (Using $\ell(y,a) = e^{-ya}$). Note that

$$\partial_a e^{-ya} = -ye^{-ya}$$

giving

$$-\partial_2 \ell(y_i, f_{m-1}(x_i)) = y_i e^{-y_i f_{m-1}(x_i)}.$$

Thus, in stage m our step direction h_m is given by solving

$$h_m := \arg\min_{h\in\mathcal{H}} \sum_{i=1}^{n} (y_i e^{-y_i f_{m-1}(x_i)} - h(x_i))^2.$$

As an aside, we will now sketch an argument that shows that if we have learners in the sense of AdaBoost (i.e., they produce classification functions that minimize a weighted $0-1$ loss), we can use them with GBM and the exponential loss to recover the AdaBoost algorithm. Let

$$\vec{r} = \left(y_i e^{-y_i f_{m-1}(x_i)}\right)_{i=1}^{n} \quad \text{and} \quad \vec{h} = (h(x_i))_{i=1}^{n}.$$

Then we have

$$h_m = \arg\min_{h\in\mathcal{H}} \|\vec{r} - \vec{h}\|_2^2 = \|\vec{r}\|_2^2 + \|\vec{h}\|_2^2 - 2\langle \vec{r}, \vec{h}\rangle.$$

Note that $\vec{h} \in \{-1,1\}^n$ so $\|\vec{h}\|_2^2 = n$, i.e., a constant. Thus this minimization is equivalent to

$$\arg\max_{h\in\mathcal{H}} \langle \vec{r}, \vec{h}\rangle.$$

Plugging in, we have

$$h_m = \arg\max_{h\in\mathcal{H}} \sum_{i=1}^{n} h(x_i) y_i e^{-y_i f_{m-1}(x_i)}.$$

Note that

$$h(x_i) y_i = 1 - 2\cdot \mathbf{1}(h(x_i) \neq y_i)$$

so

$$\begin{aligned} h_m &= \operatorname*{arg\,max}_{h \in \mathcal{H}} \sum_{i=1}^{n} e^{-y_i f_{m-1}(x_i)} - 2 \sum_{i=1}^{n} e^{-y_i f_{m-1}(x_i)} \mathbf{1}(h(x_i) \neq y_i) \\ &= \operatorname*{arg\,min}_{h \in \mathcal{H}} \sum_{i=1}^{n} e^{-y_i f_{m-1}(x_i)} \mathbf{1}(h(x_i) \neq y_i). \end{aligned}$$

Thus we see that h_m minimizes a weighted $0-1$ loss. The weights are

$$e^{-y_i f_{m-1}(x_i)} = e^{-y_i (\sum_{i=1}^{m-1} \nu_i h_i(x_i))} = \prod_{i=1}^{m-1} e^{-y_i \nu_i h_i(x_i)} = \prod_{i=1}^{m-1} e^{-\nu_i (1-2\,\mathbf{1}(h_i(x_i) \neq y_i))}.$$

By solving for the optimal step size ν_m it can be shown (we omit this) that the resulting function f_m is the same as produced by AdaBoost.

Next we apply GBM to square loss and absolute loss on a simple 1-d data set. We use decision stumps as our base hypothesis space. Run gbm.py to see the output.