CSCI 5525 Machine Learning Fall 2019

Lecture 17: Gradient Boosting

April 2020

Lecturer: Steven Wu *Scribe: Steven Wu*

Boosting through the Lens of Gradient Descent

AdaBoost is an adaptive method that iteratively builds an ensemble predictor of the form

$$f_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

In this lecture, we will introduce another popular boosting algorithm. We will first provide a generic framework for designing boosting methods based on ideas from gradient descent.

Let $X = \{x_1, \ldots, x_n\}$ be the set of feature vectors in the training set. For any $h \in \mathcal{H}$, we will embed h in a vector space and view it as a vector of the form $(h(x_1), \ldots, h(x_n)) \in \mathcal{Y}^n$. We can now think of boosting as a form of gradient descent in the space of $\mathcal{Y}^n$.

Let ℓ denote a (convex and differentiable) loss function (e.g., squared loss, logistic loss, and exponential loss). We will further define $\ell(f)$ as:

$$\ell(f) = n\, \hat{\mathcal{R}}(f) = \sum_{i=1}^{n} \ell(y_i, f(x_i)). \tag{1}$$

Assume we have already finished t iterations and already have an ensemble classifier $f_t(x)$. Now in iteration $t+1$ we want to add one more weak learner h_{t+1} to the ensemble. We would like to find a weak predictor that greedily minimizes the training loss:

$$h_{t+1} = \text{argmin}_{h \in \mathcal{H}}\, \ell(f_t + \alpha h). \tag{2}$$

This then defines the next ensemble: $f_{t+1} := f_t + \alpha h_{t+1}$.

Taylor approximation. Now we will look at the optimization in (2). Suppose the step size α is fixed. Then for any given f, we can take the Taylor Approximation on $\ell(f + \alpha h)$:

$$\ell(f + \alpha h) \approx \ell(f) + \alpha \langle \nabla \ell(f), h \rangle. \tag{3}$$

The *linear* function above is only a good approximation within a small region around $\ell(f)$. Thus, we should choose α to be a small value. Given this linear approximation, we can now find an a weak learner h that solves:

$$\text{argmin}_{h \in \mathcal{H}} \langle \nabla \ell(f), h \rangle = \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} \frac{\partial \ell}{\partial [f(x_i)]} h(x_i) \tag{4}$$

where $\frac{\partial \ell}{\partial [f(x_i)]}$ is the derivative of the loss function $\ell(y_i, f(x_i))$ with respect to the prediction $f(x_i)$.

Putting this back to the context of the optimization at step $t+1$, we will now solve:

$$h_{t+1} = \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} \underbrace{\frac{\partial \ell}{\partial [f(x_i)]}}_{r_i} h(x) \tag{5}$$

for finding the next weak learner to append to the ensemble.

The update rule in (5) provides a generic boosting framework on how to iteratively add weak learners onto the ensemble. AdaBoost is an instantiation of such framework. As we mentioned before, it corresponds to choosing the exponential loss $\ell(f) = \sum_{i=1}^{n} e^{-y_i f(x_i)}$, and the choice of α_t is optimal with a nice closed form. Now we look at another boosting method under this framework.

Gradient Boosted Regression Tree

This boosting method is called *Gradient Boosted Regression Tree* or simply *gradient boosting*. It can be used for classification and regression. This method uses regression trees (decision trees for regression). To derive the algorithm, we will the following assumption for convenience: We assume that $\sum_{i=1}^{n} h^2(x_i) = C$ for some constant C. For classification, we always have this. In general, this can be done by rescaling the predictions given by the weak learner h.

Let $t_i = -r_i$ be the negative derivative for each i. The optimization problem in (5) can be written as:

$$\begin{aligned}
&\text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} r_i h(x_i) \\
&= \text{argmin}_{h \in \mathcal{H}} 2 \sum_{i=1}^{n} -t_i h(x_i) \\
&= \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} \underbrace{t_i^2}_{} - 2 t_i h(x_i) + \underbrace{(h(x_i))^2}_{} \qquad (t_i^2 \text{ is a constant and } \textstyle\sum_{i=1}^{n} h^2(x_i) = C) \\
&= \text{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} (h(x_i) - t_i)^2
\end{aligned}$$

This means, at each round we are using a regression tree to fit the "labels" given by $t_1, \ldots, t_n$ by minimizing the squared loss. Note that while that we are solving the least squared problem at every step, the original loss function does not need to be the squared loss. Nonetheless, the special case of squared loss is intuitive:

Special case of squared loss ℓ. If the loss function ℓ is the squared loss, i.e.

$$\ell(f) = \frac{1}{2} \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

then it is easy to show that

$$t_i = -\frac{\partial \ell}{f(x_i)} = y_i - f(x_i).$$

The term $y_i - f(x_i)$ is also called the residual. Thus, in this special case, the gradient boosting method is iteratively trying to compute a regression tree to fix the residuals.