Boosting, Gradient Boosting, and Adaptive Boosting

date: 20261121
status: #finished
tags: #machine_learning #ensemble_learning #gradient_boosting #adaboost

---

1. Boosting

Boosting is an ensemble technique, just like bagging, and it aims to construct a strong model (learner), H_T, via **sequentially** combining a set of weak models, $\{h_t \in \mathcal{H}\}_{t=1}^{T}$:

$$H_T = H_0 + \sum_{t=1}^{T} \alpha_t h_t,$$

where H_0 is the initial model.

Specifically, (α_t, h_t) is obtained based on H_{t-1}, and H_t is obtained via

$$H_t = H_{t-1} + \alpha_t h_t.$$

Notice that we present the general pipeline of boosting here, and various boosting models are different in ways of how $\{\alpha_t, h_t\}_t$ are identified.

---

2. Anyboost

(i) Anyboost v.s. Gradient Boosting

Boosting can be formulated as gradient descent on the empirical risk in function space, and this view is called "anyboost" (Mason et al., 2000). Each round picks a weak model h_t that points downhill, i.e., whose inner product with the risk gradient is negative (the condition derived below). The aim of each round is

$$R(H_t) \leq R(H_{t-1}),$$

but this is only guaranteed when α_t is small enough or chosen by a line search, because the downhill condition comes from a first-order approximation. Note that anyboost

doesn't explicitly state how $\{(\alpha_t, h_t)\}_t$ are obtained: the paper leaves the step size open and only asks the weak learner for a downhill direction. Instead, it just gives a general principle, and gradient boosting follows that principle and is specific about how they should be computed. In this section, we are going to introduce the idea of anyboost.

(ii) First-Order Taylor Expansion

To be clear on how gradient descent principle is used, we will apply the first order Taylor expansion on the empirical risk.

The empirical risk is defined as

$$R(H) = \frac{1}{n}\sum_i L(y_i, H(x_i))$$

where $L(y, F) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function. Note that for multi-class classification, the model outputs K scores, and the loss applies the softmax to them to get the distribution over classes, so in this case, $L : \{1, ..., K\} \times \mathbb{R}^K \to \mathbb{R}$ where K is the number of classes. Moreover, although in data, $y_i \in \{1, ..., K\}$, when writing code, y_i is an one-hot encoded K dimensional vector.

In this case, R is a function of a hypothesis (function) H, and we can define another kind of empirical risk, $\tilde{R} : \mathbb{R}^n \to \mathbb{R}$ or $\mathbb{R}^{n\times K} \to \mathbb{R}$ where the input of the function is the vertically stacked model outputs.

$$\tilde{R}(F_1, F_2, ..., F_n) = \frac{1}{n}\sum_i L(y_i, F_i),$$

and $L(y, F)$ is assumed to be differentiable at F. As mentioned, each H_t is obtained by updating from H_{t-1} via the formula of $H_t = H_{t-1} + \alpha_t h_t$. Therefore, to see how α_t and h_t can be identified, we will apply the first order Taylor expansion on

$$\begin{aligned}&\tilde{R}(H_t(x_1), H_t(x_2), ..., H_t(x_n))\\ =\ &\tilde{R}(H_{t-1}(x_1) + \alpha_t h_t(x_1), H_{t-1}(x_2) + \alpha_t h_t(x_2), ..., H_{t-1}(x_n) + \alpha_t h_t(x_n))\end{aligned}$$

around

$$\tilde{R}(H_{t-1}(x_1), H_{t-1}(x_2), ..., H_{t-1}(x_n)).$$

To simplify the notation, we let

$$H_t(X) = \begin{pmatrix} H_t(x_1) \\ H_t(x_2) \\ \vdots \\ H_t(x_n) \end{pmatrix} \in \mathbb{R}^n \text{ or } \mathbb{R}^{n\times K},$$

$$h_t(X) = \begin{pmatrix} h_t(x_1) \\ h_t(x_2) \\ \vdots \\ h_t(x_n) \end{pmatrix} \in \mathbb{R}^n \text{ or } \mathbb{R}^{n\times K},$$

$$L(y, H_t(X)) = \begin{pmatrix} L(y_1, H_t(x_1)) \\ L(y_2, H_t(x_2)) \\ \vdots \\ L(y_n, H_t(x_n)) \end{pmatrix} \in \mathbb{R}^n,$$

and

$$\frac{\partial L}{\partial F}(y, H_t(X)) = \begin{pmatrix} \frac{\partial L}{\partial F}(y_1, H_t(x_1)) \\ \frac{\partial L}{\partial F}(y_2, H_t(x_2)) \\ \vdots \\ \frac{\partial L}{\partial F}(y_n, H_t(x_n)) \end{pmatrix} \in \mathbb{R}^n \text{ or } \mathbb{R}^{n\times K}.$$

Besides, we consider the regression task here for simplicity. For multi-class classification, the expansion below still holds as **one** expansion, with $H_t(X)$, $h_t(X)$, and the gradient in $\mathbb{R}^{n\times K}$, and the inner product sums over both samples and classes. XGBoost and LightGBM build one tree per class at each step, and then each tree's symbols are in $\mathbb{R}^n$. These K trees are **not** one-vs-all, as the gradient of each class still comes from the softmax over all K scores.

Then, we have

$$\begin{aligned} \tilde{R}(H_t(X)) &= \tilde{R}(H_{t-1}(X) + \alpha_t h_t(X)) \\ &\approx \tilde{R}(H_{t-1}(X)) + \alpha_t \left[\nabla \tilde{R}(H_{t-1}(X))\right]' h_t(X). \end{aligned}$$

Note that

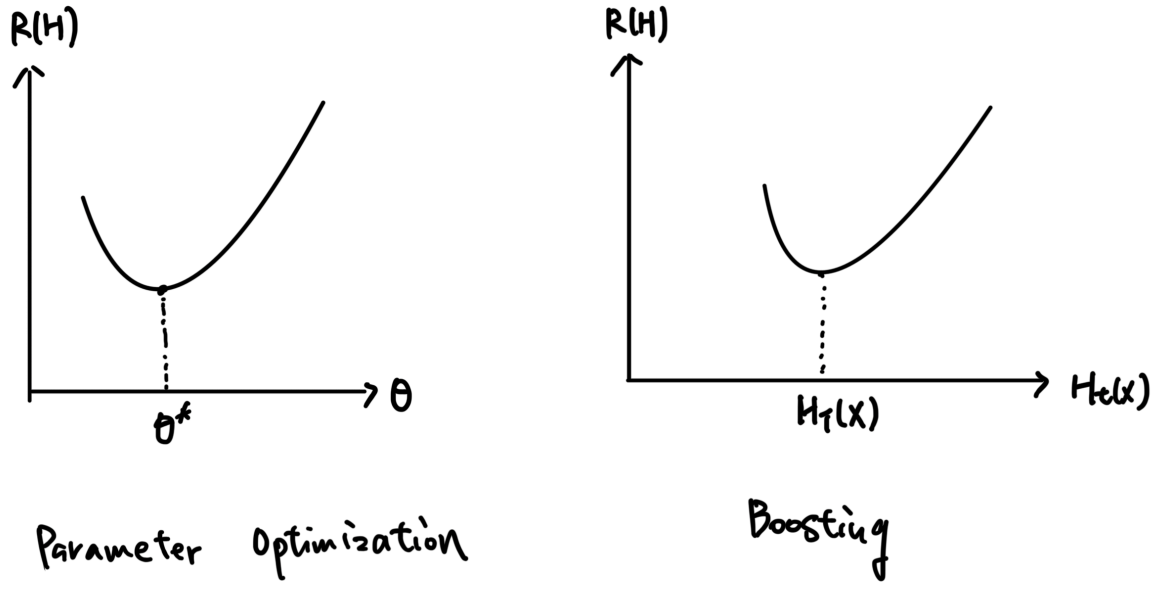
$$\begin{aligned} \nabla \tilde{R}(H_{t-1}(X)) &= \begin{pmatrix} \frac{\partial \tilde{R}}{\partial F_1}(H_{t-1}(X)) \\ \frac{\partial \tilde{R}}{\partial F_2}(H_{t-1}(X)) \\ \vdots \\ \frac{\partial \tilde{R}}{\partial F_n}(H_{t-1}(X)) \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{n}\frac{\partial L}{\partial F}(y_1, H_{t-1}(x_1)) \\ \frac{1}{n}\frac{\partial L}{\partial F}(y_2, H_{t-1}(x_2)) \\ \vdots \\ \frac{1}{n}\frac{\partial L}{\partial F}(y_n, H_{t-1}(x_n)) \end{pmatrix} \\ &= \frac{1}{n}\frac{\partial L}{\partial F}(y, H_{t-1}(X)) \end{aligned}$$

(iii) Gradient Descent Fashion

Therefore, to follow the gradient descent principle, we should choose $\alpha_t > 0$ and h_t such that $\tilde{R}(H_t(X)) \leq \tilde{R}(H_{t-1}(X))$. To first order, that is,

$$\frac{\alpha_t}{n}\left[\frac{\partial L}{\partial F}(y, H_{t-1}(X))\right]' h_t(X) \leq 0$$

It's so amazing that we can formulate boosting as predictions optimization via gradient descent. In parameter optimization, we do it through incremental adjustment, and here, we incrementally adjust predictions, $H_{t-1}(X)$ by $\alpha_t h_t(X)$, and we just need to ensure the increments, $\alpha_t h_t(X)$, has a negative inner product with the gradient of the loss with respect to the predictions, $\frac{\partial L}{\partial F}(y, H_{t-1}(X))$.



With this idea, h_t is sometimes referred to as a "step", and we compute steps $\{h_t(X)\}_t$ so that the empirical risk gradually decreases:

$$\tilde{R}(H_0(X)) \geq \tilde{R}(H_0(X) + \alpha_1 h_1(X)) \geq \ldots \geq \tilde{R}(H_0(X) + \alpha_1 h_1(X) + \ldots + \alpha_T h_T(X)).$$

This holds for the average over samples, not for each sample: the loss $L(y_i, H_t(x_i))$ of a single sample i can go up.

(iv) Anyboost Procedure

The procedure of anyboost is as follows.

Fix $H_0 = 0$, and iterate t from 1 to T {

1. Compute $\frac{\partial L}{\partial F}(y, H_{t-1}(X))$ based on H_{t-1}
2. Choose $h_t \in \mathcal{H}$ with $\left[\frac{\partial L}{\partial F}(y, H_{t-1}(X))\right]' h_t(X) < 0$

3. Choose a coefficient $\alpha_t > 0$
4. Update the model: $H_t = H_{t-1} + \alpha_t h_t$

}

---

3. Gradient Boosting

Gradient Boosting (Friedman, 2001) is a kind of anyboost that suggests α_t and h_t to be identified by the following separate minimizations.

$$h_t = \arg\min_{h \in \mathcal{H}} \left\| h(X) - \left[-\frac{\partial L}{\partial F}(y, H_{t-1}(X)) \right] \right\|^2,$$

(Friedman also fits a free scale β in front of h; a regression tree absorbs it into its leaf values), and

$$\alpha_t = \arg\min_{\alpha \in \mathbb{R}} \tilde{R}(H_{t-1}(X) + \alpha h_t(X)).$$

Note that the gradient doesn't exactly answer how much we should add exactly on the previous prediction, $H_{t-1}(x_i)$ so that the loss will be minimized, and it simply gives the correct direction but the step size is unclear. That's why we have α_t. Besides, it's more often that H_t updates like the follows

$$H_t = H_{t-1} + \nu \alpha_t h_t$$

where $\nu \in (0, 1]$ is a hyperparameter served as the learning rate, which Friedman (2001) calls shrinkage. $\nu = 0$ is excluded because the model would never move.

Note that the initialization is

$$H_0 = \arg\min_{c \in \mathbb{R}} \frac{1}{n} \sum_i L(y_i, c).$$

That is, $\{H_0(x_i)\}_i$ are all identical to that optimal c.

The intuition of the identification rule of $h_t(X)$ is to let it be as close to $-\frac{\partial L}{\partial F}(y, H_{t-1}(X))$ as possible to satisfy anyboost's gradient descent principle, and to let α_t minimize the loss along h_t. The learning rate ν then shrinks that best step on purpose, which regularizes the model.

---

4. Adaptive Boosting

Adaptive boosting (AdaBoost, Freund and Schapire, 1997) chooses (α_t, h_t) from sample weights $D_t(i)$ that it updates each round. It is model-agnostic: h_t can be any weak model that can be trained on weighted samples. "Adaptive" means that α_t and D_{t+1} are computed from the error that h_t actually makes, so no bound on that error is needed in advance.

(i) Task

The original AdaBoost is for binary classification. The paper codes the labels as $\{0, 1\}$, and we use the equivalent coding $y_i \in \{-1, +1\}$. Each weak model outputs $h_t(x) \in \{-1, +1\}$, while the strong model $H_t(x) = \sum_{s \le t} \alpha_s h_s(x)$ is a real score, and the predicted class is $\operatorname{sign}(H_T(x))$.

Multi-class is possible. The original paper's AdaBoost.M1 needs every h_t to have weighted error below $\frac{1}{2}$, which is hard to reach with K classes. SAMME (Zhu et al., 2009) only needs weighted error below $1 - \frac{1}{K}$, i.e., better than random guessing among K classes, by adding $\ln(K-1)$ to α_t. scikit-learn's `AdaBoostClassifier` uses SAMME. Regression is also possible, e.g., AdaBoost.R2 (Drucker, 1997) used by scikit-learn's `AdaBoostRegressor`, but it is rarely used, since gradient boosting with square loss is the usual choice.

(ii) Procedure

Fix $H_0 = 0$ and $D_1(i) = \frac{1}{n}$ for all i, and iterate t from 1 to T {

1. Fit $h_t \in \mathcal{H}$ to minimize the weighted error $\varepsilon_t = \sum_i D_t(i) \mathbb{1}\{h_t(x_i) \neq y_i\}$
2. Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$, which is positive when $\varepsilon_t < \frac{1}{2}$ and larger for a more accurate h_t
3. Reweight the samples: $D_{t+1}(i) = D_t(i) \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, where $Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i))$ makes the weights sum to 1. A misclassified sample has $y_i h_t(x_i) = -1$, so its weight is multiplied by $e^{\alpha_t} > 1$
4. Update the model: $H_t = H_{t-1} + \alpha_t h_t$

}

The original AdaBoost has no learning rate ν, because α_t is already the exact minimizer of the exponential loss along h_t (see (iv)). Implementations add one anyway: scikit-learn's `learning_rate` multiplies α_t, as in the shrinkage update of gradient boosting. With $\nu < 1$ the coin-toss property in (iii) no longer holds exactly.

(iii) Coin-Toss Interpretation

Under the new weights D_{t+1}, the model just trained, h_t, has weighted error exactly $\frac{1}{2}$, like a coin toss. The wrong samples carry weight ε_t under D_t, so $Z_t = (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t}$. With $e^{\alpha_t} = \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$, each term equals $\sqrt{\varepsilon_t(1-\varepsilon_t)}$, so $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$, and

$$\sum_{i:h_t(x_i)\neq y_i} D_{t+1}(i) = \frac{\varepsilon_t e^{\alpha_t}}{Z_t} = \frac{\sqrt{\varepsilon_t(1-\varepsilon_t)}}{2\sqrt{\varepsilon_t(1-\varepsilon_t)}} = \frac{1}{2}.$$

So h_t carries no information about D_{t+1}, and h_{t+1} has to find what h_t missed. The property holds because of the choice of α_t: setting $\frac{\mathrm{d}Z_t}{\mathrm{d}\alpha_t} = -(1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t} = 0$ gives exactly the α_t of step 2, and this condition says the reweighted wrong and right samples have equal weight.

(iv) AdaBoost as Gradient Boosting with Exponential Loss

AdaBoost is gradient boosting with the exponential loss $L(y, F) = e^{-yF}$, $H_0 = 0$, and $\nu = 1$ (Mason et al., 2000; Friedman et al., 2000).

---

5. Reference

- Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. 2000. **Boosting Algorithms as Gradient Descent**. In Advances in Neural Information Processing Systems 12 (NIPS 1999), 512–518. MIT Press. https://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent
- Jerome H. Friedman. 2001. **Greedy Function Approximation: A Gradient Boosting Machine**. The Annals of Statistics 29(5), 1189–1232. https://doi.org/10.1214/aos/1013203451
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. **The Elements of Statistical Learning**, 2nd ed., Chapter 10 (Boosting and Additive Trees). Springer. https://doi.org/10.1007/978-0-387-84858-7
- Yoav Freund and Robert E. Schapire. 1997. **A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting**. Journal of Computer and System Sciences 55(1), 119–139. https://doi.org/10.1006/jcss.1997.1504
- Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2000. **Additive Logistic Regression: A Statistical View of Boosting**. The Annals of Statistics 28(2), 337–407. https://doi.org/10.1214/aos/1016218223
- Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. 2009. **Multi-class AdaBoost**. Statistics and Its Interface 2(3), 349–360. https://doi.org/10.4310/SII.2009.v2.n3.a8
- Harris Drucker. 1997. **Improving Regressors using Boosting Techniques**. In Proceedings of the 14th International Conference on Machine Learning (ICML

1997), 107–115.