Gradient Boosting Machines

Contents

Introduction

The common ensemble techniques like random forests rely on simple averaging of models in the ensemble.The family of boosting methods is based on a different, constructive strategy of ensemble formation.

The main idea of boosting is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.

The first prominent boosting techniques (AdaBoost) were purely algorithm-driven, which made the detailed analysis of their properties and performance rather difficult. This led to a number of speculations as to why these algorithms either outperformed every other method, or on the contrary, were in applicable due to severe overfitting.

In gradient boosting machines,or simply, GBMs, the learning procedure consecutively fits new models to provide a more accurate estimate of the response variable.

The principle idea behind this algorithm is to construct the new base-learners to be maximally correlated with the negative gradient of the loss function, associated with the whole ensemble. The loss functions applied can be arbitrary, but to give a better intuition, if the error function is the classic squared-error loss, the learning procedure would result in consecutive error-fitting.

In general, the choice of the loss function is up to the researcher, with both a rich variety of loss functions derived so far and with the possibility of implementing one's own task-specific loss. This high flexibility makes the GBMs highly customizable to any particular data-driven task. It introduces a lot of freedom into the model design thus making the choice of the most appropriate loss function a matter of trial and error. However, boosting algorithms are relatively simple to implement, which allows one to experiment with different model designs.

Moreover the GBMs have shown considerable success in not only practical applications, but also in various machine-learning and data-mining challenges.

In this unit we present the basic methodology and learning algorithms of the GBMs,as originally derived by Friedman (2001).

Gradient Boosting Methodology

Friedman (2001) showed that other boosting strategies could be obtained by using different minimizing procedures combined with different loss functions. In particular, he adapted the well-known gradient-descent (also known as steepest-descent) algorithm to derive a more general boosting procedure - which he called "gradient boosting"- primarily for regression situations.

The general minimization problem is to find $\hat{f}$ such that the risk function is minimized

$$\hat{f}(x) = \underset{f}{\operatorname{argmin}} E_{XY}\Big[L(Y, f(x))\Big].$$

Note that at this stage, we don't make any assumptions about the form of neither the true functional dependence $f(x)$, nor the form of the function estimate $\hat{f}(x)$.

The response variable y can come from different distributions. This naturally leads to specification of different loss functions L. In particular, if the response variable is binary, i.e., $y \in \{0, 1\}$, one can consider the exponential loss function. If the response variable is continuous,i.e., $y \in \mathbb{R}$, one can use classical L_2 squared loss function or the robust regression Huber loss. For other response distribution families like the Poisson-counts, specific loss functions have to be designed.

To make the problem of function estimating tractable, we can restrict the function search space to a parametric family of functions $f(x, \theta)$. This would change the function optimization problem into the parameter estimation one:

$$\hat{f}(x) = f(x, \hat{\theta})$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} E_{XY}\Big[L(Y, f(x, \theta))\Big].$$

Typically the closed-form solutions for the parameter estimates are not available. To perform the estimation, iterative numerical procedures are considered.

Numerical optimization

Given M iteration steps, the parameter estimates can be written in the incremental form:

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i$$

The simplest and the most frequently used parameter estimation procedure is the steepest gradient descent. Given N data points $(x, y)_{i=1}^{N}$ we want to decrease the empirical loss function $J(\theta)$ over this observed data:

$$J(\theta) = \sum_{i=1}^{N} L(y_i, f(x_i, \theta))$$

The classical steepest descent optimization procedure is based on consecutive improvements along the direction of the gradient of the loss function $\nabla J(\theta)$. As the parameter estimates $\hat{\theta}$ are presented in an incremental way, we would distinguish the estimate notation. By the subscript index of the estimates $\hat{\theta}_t$ we would consider the t-th incremental step of the estimate $\hat{\theta}$. The superscript $\hat{\theta}^t$ corresponds to the collapsed estimate of the whole ensemble, i.e., sum of all the estimate increments from step 1 up till step t.

The steepest descent optimization procedure is organized as follows:

1. Initialize the parameter estimates $\hat{\theta}_0$.

   For each iteration t, repeat:

2. Obtain a compiled parameter estimate $\hat{\theta}^t$ from all the previous iterations:

$$\hat{\theta}^t = \sum_{i=0}^{t} \hat{\theta}_i$$

3. Evaluate the gradient of the loss function $\nabla J(\theta)$, given the obtained parameter estimates

$$\nabla J(\theta) = \{\nabla J(\theta_i)\} = \Big[\frac{\partial J(\theta)}{\partial \theta_i}\Big]_{\theta=\hat{\theta}^t}$$

4. Calculate the new incremental parameter estimate $\hat{\theta}_t$:

$$\hat{\theta}_t \leftarrow -\gamma \nabla J(\theta) \qquad \gamma > 0$$

5. Add the new estimate $\hat{\theta}_t$ to the sequence.

Example. Write multiple linear regression parameter estimation in this framework.

$$Y_{n\times 1} = X_{n\times p}\theta_{p\times 1} + \epsilon_{n\times 1}$$

Ordinary least squares aims to minimize

$$J(\theta) = (Y - X\theta)^\intercal (Y - X\theta) = Y^\intercal Y - 2\theta^\intercal X^\intercal Y + \theta^\intercal X^\intercal X\theta$$

And, the gradient is of the form

$$\nabla J(\theta) = -2X^\intercal Y + 2X^\intercal X\theta$$

Observe that, by equating to zero it turns out the normal equations and the LS estimation of θ:

$$\hat{\theta} = (X^\intercal X)^{-1} X^\intercal Y$$

providing inverse matrix exists.

But, we can also apply the gradient descent method. We can write estimation procedure with the iteration

$$\theta_{t+1} = \theta^t - \gamma \nabla J(\theta^t) = \theta^t - 2\gamma X^\intercal (Y - X\theta^t), \qquad \gamma > 0$$

Observe that, as $X\theta^t$ tends to Y, residual becomes small and $\nabla J(\theta^t)$ tends to zero.

We can write the iterative process as follows:

$$\begin{aligned}
\theta^1 &= \theta^0 - 2\gamma X^\intercal (Y - X\theta^0) = \theta_0 + \theta_1 \\
\theta^2 &= \theta^1 - 2\gamma X^\intercal (Y - X\theta^1) = \theta_0 + \theta_1 + \theta_2 \\
\theta^3 &= \theta^2 - 2\gamma X^\intercal (Y - X\theta^2) = \theta_0 + \theta_1 + \theta_2 + \theta_3 \\
&\vdots \\
\theta^M &= \theta^{M-1} - 2\gamma X^\intercal (Y - X\theta^{M-1}) = \theta_0 + \theta_1 + \theta_2 + \theta_3 + \cdots + \theta_{M-1}
\end{aligned}$$

Optimization in function space

The main difference between boosting methods and conventional machine-learning techniques is that optimization is held out in the function space. That is, we set the function estimate $\hat{f}$ in the additive functional form:

$$\hat{f}(x) = \sum_{i=0}^{M} \hat{f}_i(x)$$

In this representation, M is the number of iterations, $\hat{f}_0(x)$ is the initial guess and $\{\hat{f}_i(x)\}_{i=1}^{M}$ are the function increments, also called as "boosts".

To make the functional approach feasible in practice, one can follow a similar strategy of parameterizing the family of functions. Here we introduce the parameterized "base-learner" functions $h(x, \theta)$ to distinguish them from the overall ensemble function estimates $\hat{f}(x)$.

One can choose different families of base-learners such as decision trees. For example, if we use a J-terminal-node regression tree as a base learner, then $h(x, \theta)$ takes the simple form

$$h(x, \theta) = \sum_{j=1}^{J} \bar{y}_j \mathbf{1}_{\{x \in R_j\}},$$

where the components of the parameter vector $\theta = (\{\bar{y}_j, R_j\})^{\intercal}$ define the entire tree: the $\{R_j\}$ are the J disjoint regions of input space and represent the terminal nodes of the tree, and $\{\bar{y}_i\}$ are terminal node means that define the region boundaries. Note that, determining each region $\{R_j\}$ requires a set of boolean conditions that in turn require choosing the appropriate variables and choosing the respective cut-off points. These choices constitute the parameters θ of $h(x, \theta)$.

We can now formulate the *stagewise* approach of function incrementing with the base-learners. For this purpose the optimal step-size ρ should be specified at each iteration.

For the function estimate at the t-th iteration, the optimization rule is therefore defined as:

$$\hat{f}(x)_t \leftarrow \hat{f}(x)_{t-1} + \rho_t h(x, \theta_t)$$

$$(\rho_t, \theta_t) = \arg\min_{\rho, \theta} \sum_{i=1}^{N} L\big(y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta)\big)$$

Gradient Boosting Algorithm

One can arbitrarily specify both the loss function and the base-learner models on demand. In practice, given some specific loss function $L(y, f)$ and/or a custom base-learner $h(x, \theta)$, the solution to the parameter estimates can be difficult to obtain. To deal with this, it was proposed to choose the new function $h(x, \theta_t)$ to be the most parallel to the negative gradient of the loss function using $f(x)$ to predict y, $L(y, f) = \sum_{i=1}^{n} L(y_i, f(x_i))$, along the observed data $\{(x_i, y_i), i = 1, 2, ..., n\}$.

For example, the gradient at x_i is estimated by

$$g_t(x_i) = \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f(x_i)=f_{t-1}(x_i)}, \qquad i = 1, ..., n$$

Instead of looking for the general solution for the boost increment in the function space, one can simply choose the new function increment to be the most correlated with $-g_t(x)$. This permits the replacement of a potentially very hard optimization task with the classic least-squares minimization one:

$$\theta_t = \arg\min_{\theta} \sum_{i=1}^{N} \big(- g_t(x_i) + h(x_i, \theta) \big)^2$$

The update formula is

$$f_t(x) = f_{t-1} + \rho_t h(x; \theta_t),$$

where ρ_t is found by line search

$$\rho_t = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i; \theta_t))$$

To summarize, we can formulate the complete form of the gradient boosting algorithm, as originall yproposed by Friedman (2001). The exact form of the derived algorithm with all the corresponding formulas will heavily depend on the design choices of $L(y, f)$ and $h(x, \theta)$. One can find some common examples of these algorithms in Friedman (2001).

Algorithm Friedman's Gradient Boost algorithm

Inputs

- input data $\{(x_i, y_i)\}$, $i = 1, 2, ..., n$,
- number of iterations M
- choice of the loss function L
- choice of the base-learner $h(x, \theta)$

Algorithm

1. initialize $\hat{f}_0$ with a constant.
2. `for` $t = 1$ `to` M `do`
3. Compute the negative gradient $g_t(x)$
4. Fit a new base-learner function $h(x, \theta_t)$
5. find the best gradient descent step-size ρ_t

$$\rho_t = \operatorname{argmin}_{\rho} \sum_{i=1}^{n} L(y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i; \theta_t)).$$

6. Set $\hat{f}_t(x) \leftarrow \hat{f}_{t-1}(x) + \rho_t h(x; \theta_t)$
7. `end for`

Loss functions for continuous response

When the response variable y is continuous, a regression task is solved. A classic loss function, which is commonly used in practice is the squared-error L_2 loss:

$$L(y, f) = \frac{1}{2}(y - f)^2$$

In the case of the L_2 loss function, its derivative is the residual $y - f$, which implies that the GBM algorithm simply performs residual refitting. The idea behind this loss function is to penalize large deviations from the target outputs while neglecting small residuals.

Loss functions for categorical response

In the case of categorical response, the response variable y typically takes on binary values $y \in \{0, 1\}$, thus, assuming that it comes from the Bernoulli distribution. To simplify the notation, let us assume the transformed labels $\bar{y}$, putting $\bar{y} = 2y - 1$ and making $\bar{y} \in \{-1, 1\}$. In this case, the probability of class-wise response can be estimated by minimizing the negative loglikelihood, associated with the new class labels:

$$L(y, f) = \log(1 + \exp(-2\bar{y}f))$$

This loss function is commonly referred to as the Bernoulli loss.

Another common choice of categorical loss-function is the simple exponential loss, as it is used in the Adaboost algorithm (Schapire, 2002). Following the same notation as in the Bernoulli loss, the Adaboost loss function is therefore defined as

$$L(y, f) = \exp(-\bar{y}f)$$

References

Friedman,J.(2001). Greedy boosting approximation: a gradient boosting machine. Ann.Stat. 29, 1189–1232.doi:10.1214/aos/1013203451

Schapire,R.(2002).The boosting approach to machine learning: an overview. Nonlin. Estimat. Classif. Lect. NotesStat. 171, 149–171.doi:10.1007/978-0-387- 21579-2_9

The Elements of Statistical Learning: Data Mining, Inference, and Prediction. **Chapter 10**. Second Edition. Springer. 2009.

Natekin Alexey, Knoll Alois. Gradient boosting machines, a tutorial. Frontiers in Neurorobotics, 7, 2013.