Boosting

Wei Li

Syracuse University

Spring 2021

Boosting trees

Gradient boosting

Tree Size J for boosting

Regularization (choosing M)

Boosting trees

Boosting trees

A CART tree can be expressed as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I\left(x \in R_j\right)$$

with parameters $\Theta = (\gamma_j, R_j)_1^J$.

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L\left(y_i, \gamma_j\right) = \arg\min_{\Theta} \sum_{i=1}^{n} L\left(y_i, T\left(x_i, \Theta\right)\right)$$

- Finding γ_j given R_j : Given the R_j, estimating the γ_j is typically trivial, and often $\hat{\gamma}_j = \bar{y}_j$, the mean of the y_i falling in region R_j. For misclassification loss, $\hat{\gamma}_j$ is the modal class of the observations falling in region R_j.

- Finding R_j : A typical strategy is to use a **greedy, top-down recursive partitioning algorithm** to find the R_j.

It is sometimes necessary to approximate above objective function by a smoother and more convenient criterion for optimizing the R_j

$$\tilde{\Theta} = \arg\min_{\Theta} \sum^{n} \tilde{L}\left(y_i, T\left(x_i, \Theta\right)\right)$$

Then given the $\hat{R}_j = \tilde{R}_j$, the γ_j can be estimated more precisely using the original criterion.

boosted trees

The boosted tree model is a sum of such trees,

$$f_M(x) = \sum_{m=1}^{M} T\left(x; \Theta_m\right)$$

induced in a forward stagewise manner (Algorithm 10.2).

At each step m, given the current tree model $f_{m-1}(x)$, we solve

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{n} L\left(y_i, f_{m-1}\left(x_i\right) + T\left(x_i, \Theta_m\right)\right) \tag{1}$$

where $\Theta_m = \left(\gamma_{jm}, R_{jm}\right), j = 1, \cdots, J_m$ the parameter of the next tree.

- Find γ_{jm} given R_{jm} - easy

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}\left(x_i\right) + \gamma_{jm}\right) \tag{2}$$

- Find R_{jm} 's - not so easy, except in that
  - Squared-error loss. At each stage, the solution (1) to is a fitted regression tree to residuals $y_i - f_{m-1}\left(x_i\right)$, $\hat{\gamma}_{jm}$ is the mean of these residuals in each corresponding region.
  - Exponential loss (binary classification).

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{n} w_i^{(m)} \exp\left[-y_i T\left(x_i; \Theta_m\right)\right]$$

$$\hat{\gamma}_{jm} = \frac{1}{2} \log \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I\left(y_i = 1\right)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I\left(y_i = -1\right)}$$

with weights $w_i^{(m)} = \exp\left\{-y_i f_{m-1}\left(x_i\right)\right\}$

A simple algorithm for boosting regression trees

Assume $J_m = J$ fixed (same amount of terminal nodes).

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $m = 1, 2, \ldots, M$, repeat:
   (a) Fit a tree $\hat{f}^m$ with J terminal nodes to the training data(X, r)
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^m(x)$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^m(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{m=1}^{M} \lambda \hat{f}^m(x)$$

- M: too large of M leads to overfit
- λ: often choose some small $\lambda = 0.01$ or 0.001.
  - Smaller λ leads to larger M needed.
- J controls the interaction depth (need not be too large).
  - $J = 1$ is a stump.

Gradient boosting

Gradient boosting

Consider the numerical optimization

$$\arg \min_{f \in \{f_M\}} L(f) \equiv \sum_{i=1}^{n} L\left(y_i, f\left(x_i\right)\right)$$

The solution f_M is written as a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad h_m \in R^n$$

where $\mathbf{f}_M = \left(f_M\left(x_1\right), \cdots, f_M\left(x_n\right)\right)^T$ and $\mathbf{h}_m$ is the increment vector at the m th step.

The general idea of a greedy algorithm is to solve:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$
$$f_m(x) = f_{m-1}(x) + \arg\min_{h_m} \left[\sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + h_m(x_i))\right]$$

where h_m is a base learner function.

Unfortunately, choosing the best function h at each step for an arbitrary loss function L is a computationally infeasible optimization problem in general.

We consider **functional gradient descent** method.

functional gradient descent

- Steepest descent: choose $\mathbf{h}_m = -\rho_m \mathbf{g}_m$, where ρ_m is a scalar and the ith component of $\mathbf{g}_m$ is given by

$$g_{im} := g_m(x_i) = \left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

- The step length ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{n} L\left(y_i, f_{m-1}\left(x_i\right) - \rho g_m\left(x_i\right)\right)$$

- Solution update: $f_m\left(x_i\right) = f_{m-1}\left(x_i\right) - \rho_m g_m\left(x_i\right)$

A comparison

Compared with Forward stagewise boosting:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{n} L\left(y_i, f_{m-1}\left(x_i\right) + T\left(x_i; \Theta_m\right)\right)$$

- Tree predictions $T\left(x_i; \Theta_m\right)$ are analogous to the negative gradients $-g_m(x_i)$
- But $\mathbf{t}_m = \left\{T\left(x_1; \Theta_m\right), \ldots, T\left(x_n; \Theta_m\right)\right\}$ are no independent, but constrained to be predictions of a J_m-terminal node decision tree, whereas $-\mathrm{g}_m$ is the unconstrained maximal descent direction.
- $\rho_m = \arg\min_\rho L\left(y, f_{m-1} - \rho g_m\right)$ is analogous to $\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}\left(x_i\right) + \gamma_{jm}\right)$ but performs a line search for each terminal node.

The gradient is defined only at the training data points x_i whereas the ultimate goal is to generalize $f_M(x)$ to new data not represented in the training set.

A possible solution is to fit the tree T to the negative gradient values by *least squares*:

- Fit a tree $T\left(x;\Theta_m\right)$ at m th iteration whose predictions $\mathbf{t}_m$ are as close as possible to the negative gradient

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{n} \left(-g_{im} - T\left(x_i;\Theta\right)\right)^2$$

- From the (approximate) solution regions $\tilde{\mathcal{R}}_{jm}$ set,

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in \tilde{\mathcal{R}}_{jm}} L\left(y_i, f_{m-1}\left(x_i\right) + \gamma_{jm}\right)$$

Gradient tree boosting

Algorithm 10.3 (Gradient tree boosting)

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$
2. For $m = 1$ to M :
   (a) For $i = 1, 2, \ldots, n$ compute *pseudo-residuals*

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions

$$R_{jm}, j = 1, 2, \ldots, J_m$$

   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$.

The following table summarizes the gradients for some loss functions.

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

ESL Table. 10.2. Gradients for commonly used loss functions.

Gradient tree boosting for classification

For classification (K-classes), K separate trees are built at each iteration, producing $p_k(x) = e^{f_k} / \sum_l e^{f_l}$ or do classification as $h(x) = \arg max_l p_l(x)$.

Each tree T_{km} is fit to its respective negative gradient vector g_{km}, using multinomial deviance,

$$
\begin{aligned}
-g_{ikm} &= \left[\frac{\partial L\left(y_i, f_1\left(x_i\right), \ldots, f_K\left(x_i\right)\right)}{\partial f_k\left(x_i\right)} \right]_{\mathbf{f}(x_i)=\mathbf{f}_{m-1}(x_i)} \\
&= I\left(y_i = k\right) - p_k\left(x_i\right)
\end{aligned}
$$

Algorithm 10.4 (Gradient tree boosting for classification with multinomial deviance loss)

1. Initialize $f_{k0}(x) = 0, k = 1, 2, \ldots, K$.
2. For $m = 1$ to M :
   (a) Set

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{\ell=1}^{K} e^{f_\ell(x)}}, k = 1, 2, \ldots, K$$

   (b) For $k = 1$ to K :
      i. Compute $r_{ikm} = y_{ik} - p_k(x_i), i = 1, 2, \ldots, n$
      ii. Fit a regression tree to the targets $r_{ikm}, i = 1, 2, \ldots, n$ giving terminal regions $R_{jkm}, j = 1, 2, \ldots, J_m$
      iii. Compute

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}| (1 - |r_{ikm}|)}, j = 1, 2, \ldots, J_m$$

      iv. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I\left(x \in R_{jkm}\right)$
3. Output $\hat{f}_k(x) = f_{kM}(x), k = 1, 2, \ldots, K$.

Tree Size J for boosting

Tree Size J for boosting

At each iteration a J -terminal node regression tree is induced. Thus J is the tuning parameter.

- The interaction level of tree-based approximations is limited by the tree size J.
- No interaction effects of level greater than $J - 1$ are possible.
  - Setting $J = 2$ (single split "decision stump") produces boosted models with only main effects; no interactions are permitted.
- Experience so far indicates that $4 \leq J \leq 8$ works well in the context of boosting.

Regularization (choosing M)

Regularization (choosing M)

Shrinkage:

Line 2(d) of Algorithm 10.3 is replaced by

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I\left(x \in R_{jm}\right)$$

The parameter ν is learning rate.

- smaller values of ν, more shrinkage (less complexity)
- higher M, more complexity
- a tradeoff between them ν and M

Empirically the best strategy is to set ν to be very small ($\nu < 0.1$) and choose M by early stopping.

Subsampling:

With stochastic gradient boosting (Friedman, 1999), at each iteration we sample a fraction η of the training observations (without replacement), and grow the next tree using that subsample. The rest of the algorithm is identical.