Statistical Learning and Prediction

Gradient tree boosting

Marjolein Fokkema

Leiden University

Tree ensemble methods

Bagging and random forests:

- Trees are independent, can be fitted in parallel
- Fit large trees (low bias)
- Average over predictions of many trees (lowers variance)

Gradient boosting:

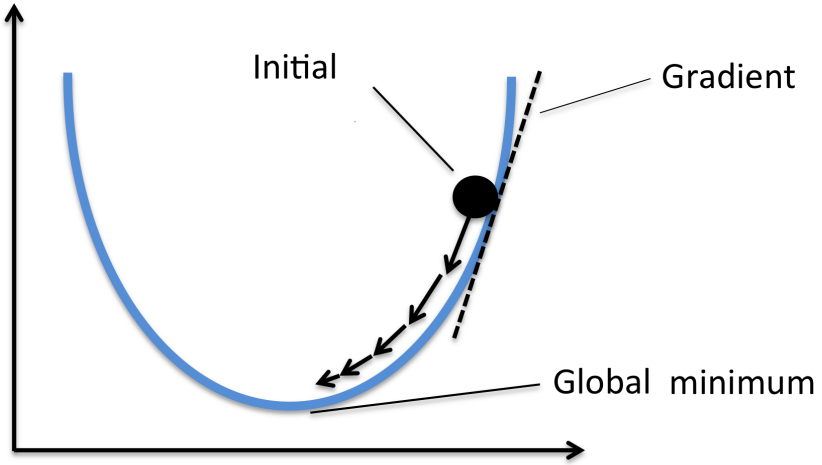
- Trees are fitted sequentially
- Fit small trees (low variance)
- Average over predictions of many trees (lowers bias)

Ensembling can be done with any (set of) baselearners, need not be trees.

Gradient descent

- Iterative procedure for finding the minimum of a function, for which no closed-form formula exists.

- At every step, move in opposite direction of the (approximate) gradient at the current estimate:



x-axis: $\hat{f}(x)$; y-axis: error estimate of $\hat{f}(x)$

Gradient tree boosting

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Gradient tree boosting

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Gradient tree boosting

Algorithm 8.2 *Boosting for Regression Trees*

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

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Gradient boosting

Given training data $\{x_i, y_i\}$ $(i = 1, \ldots, n)$ and learning rate λ:

- Initialize with a constant, e.g.: $F_0(x) = \bar{y}$
- For $b = 1, \ldots, B$:
  - Compute *pseudo-residuals* (a.k.a. *negative gradient*):

$$r_{i,b} = y_i - F_{b-1}(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{b-1}(x_i)}$$

  - Fit a *regression* tree $\hat{f}_b(x)$ to the predictors and pseudo-residuals $\{x_i, r_i\}$
  - Update the model: $F_b(x) = F_{b-1}(x) + \lambda \hat{f}_b(x)$
- Output final ensemble: $\hat{f}(x) = F_B(x)$

Stochastic gradient boosting

Given training data $\{x_i, y_i\}$ with $(i = 1, \ldots, n)$ and learning rate λ:

- Initialize with a constant, e.g.: $F_0(x) = \bar{y}$
- For $b = 1, \ldots, B$:
  - Compute *pseudo-residuals* (a.k.a. *negative gradient*):

$$r_{i,b} = y_i - F_{b-1}(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{b-1}(x_i)}$$

  - Fit a *regression* tree $\hat{f}_b(x)$ to a **sub sample of observations** from $\{x_i, r_i\}$
  - Update the model: $F_b(x) = F_{b-1}(x) + \lambda \hat{f}_b(x)$
- Output final ensemble: $\hat{f}(x) = F_B(x)$

Stochastic gradient boosting with binary outcome

Given training data $\{x_i, y_i\}$ with $(i = 1, \ldots, n)$ and learning rate λ

- Initialize with a constant, e.g.: $F_0(x) = \bar{y}$ and $\eta_0 = \log\left(\frac{\bar{y}}{1-\bar{y}}\right)$
- For $b = 1, \ldots, B$:
  - Compute *pseudo-residuals* (a.k.a. *negative gradient*):

$$r_{i,b} = y_i - F_{b-1}(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{b-1}(x_i)}$$

    where $F_{b-1}(x) = \frac{e^{\eta_{(b-1)}}}{1+e^{\eta_{(b-1)}}}$ and $\eta_{(b-1)} = \eta_0 + \sum_1^{b-1} F_{b-1}(x)$
  - Fit a *regression* tree $\hat{f}_b(x)$ to a subsample of the observations (pseudo-residuals instead of original outcome!) of $\{x_i, r_i\}$
  - Update the model: $F_b(x) = F_{b-1}(x) + \lambda \hat{f}_b(x)$
- Output final ensemble: $\hat{f}(x) = F_B(x)$

Tuning parameters: Boosting

- `shrinkage` (λ, learning rate): Small, non-zero values; `gbm` default of 0.1 is quite high, values of .01 or .001 often better.

- `n.trees`: Number of random samples (trees) to generate. Generally requires values $> (1/\lambda)$.

- `distribution`: Specifies which loss function should be minimized, and thereby how the gradient is computed. For continuous responses specify "gaussian" (or eg., "laplace", to minimize absolute error loss). For binary responses, use "bernoulli". See `?gbm` for more options.

- `interaction.depth` (tree depth; number of splits or nodes). Specifies the level of variable interactions allowed; default is 1, yielding trees with only a single split, i.e., an additive model.

- `bag.fraction`: Fraction of training observations randomly selected to fit each tree. A value of 1 will result in no subsampling, and all training

observations used for inducing each tree. Default is 0.5, yielding a subsample comprising 50% of the training observations.

Tuning parameters: Bagging and random forests

- `ntree`: Number of samples to draw / trees to generate. Default of 500 often suffices, rarely hurts to use more (e.g., 1,000).

- `mtry`: Number of predictors to be considered for each split; p for bagging; random forests use $\sqrt{p}$ for classification, $p/3$ for regression. Can use CV to obtain optimal value.

- `replace` and `sampsize`: Sampling strategy and fraction. Bootstrap sampling is sensible default, but may increase inclusion frequency for noise variables. Subsampling with sampling fraction of .632 may be preferred.

- `maxnodes`: Maximum number of nodes in every tree. By (sensible) default limited by nodesize of 1 (classification) and 5 (regression). Large trees can lead to unstable results when there are many multicollinear predictors at best weakly related to the response (Segal, 2003). Thus, smaller trees (with optimal tree size e.g., determined through CV) may perform better.

Tuning parameters: Bayesian Additive Regression Trees

BARTs default settings work well in many prediction problems already; tuning in general does not make a huge difference.

Package `dbarts` does provide a function `xbart` that allows you to tune the main parameters of BART if you would like to further optimize performance.