Lecture 19: Gradient Boosting

Statistical Learning and Data Mining

Xingye Qiao
Department of Mathematical Sciences
Binghamton University

E-mail: qiao@math.binghamton.edu

Read: ELSII Ch. 10, and ISLR 8.2

Outline

The next section would be . . . . . .

1. Gradient Boosting
2. XGBoost

Gradient Boosting

- In this lecture, generalize boosting from exponential loss or logistic loss to any arbitrary differentiable loss functions
- Use Gradient Boosting as fast algorithm to achieve additive model
- Like boosting, improve weak learners, typically decision trees.

Boosting Trees

- Boosting can improve weak learner (trees here.)
- For regression tree T: $T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}_{[x \in R_j]}$
- The boosted tree model is a sum of such trees,

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

- Can be induced in a forward stagewise manner: at each step

$$\underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m))$$

- squared-error loss: not hard; simply have the current tree regress on the current residual.
- exponential loss: give rise to AdaBoost.

Challenge in Boosting Trees

- Loss criteria such as the absolute error or the Huber loss in place of squared-error loss for regression, and the deviance in place of exponential loss for classification, will serve to robustify boosting trees.
- Unfortunately, unlike their nonrobust counterparts, these robust criteria do not give rise to simple fast boosting algorithms.
- Simple fast algorithms do not exist for even more general loss criteria

Draw inspiration from gradient descent algorithm

- The loss in using $f(x)$ to predict y on the training data is

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

- Let $\boldsymbol{f} := [f(x_1), \ldots, f(x_n)]' \in \mathbb{R}^n$ be the values of the approximation function at all the data points.
- Essentially we want to find

$$\hat{\boldsymbol{f}} = \underset{\boldsymbol{f}}{\operatorname{argmin}}\, L(\boldsymbol{f})$$

- In boosted tree, $\boldsymbol{f}$ would be $\boldsymbol{f}_m = \sum_{j=0}^{m} \boldsymbol{h}_j = \boldsymbol{f}_{m-1} + \boldsymbol{h}_m$

- In gradient descent, if one wants to minimize $L(\theta)$ with respect to θ, one use the update function

$$\theta_m = \theta_{m-1} - \gamma L'(\theta_{m-1})$$

  Here $L'(\theta_m)$ is the gradient vector of L with respect to vector θ, evaluated at $\theta = \theta_{m-1}$
- Inducing a new tree $\approx$ a step in the gradient descent algorithm, with the goal to minimize L with respect to $\boldsymbol{f}$.
- Naive solution: Let $\boldsymbol{h}_m = -\gamma L'(\theta_m)$. Unfortunately, NOT helpful in generalizing $f_M(x)$ to new data not in training set.
- Resolution: to induce a tree whose predictions are as close as possible to the negative gradients. Using squared error to measure closeness, this amounts to

$$\underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{n} (-L'_{im} - T(x_i; \Theta))^2$$

Gradient Tree Boosting Algorithm - for regression

1. Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
    1. For each i, compute

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

    2. Fit a regression tree with r_{im} as the response, and give leaf node regions R_{jm}
    3. For each leaf node region compute

$$\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

    4. Update $f_m(x) = f_{m-1}(x) + \sum_j \gamma_{jm} \mathbb{1}_{[x \in R_{jm}]}$
3. Output $\hat{f}(x) = f_M(x)$

The algorithm for **classification** is similar.

- Loss function is replaced by multinomial deviance loss.
- Lines 2(a)–(d) are repeated K times at each iteration m, once for each class
- The final result is K different boosted trees.
- The negative gradient for class k is

$$\mathbb{1}_{[y_i=k]} - p_k(x_i)$$

  where $p_k(x) = \frac{e^{f_k(x)}}{\sum_\ell e^{f_\ell(x)}}$

End products:

- Classification: $\operatorname{argmax}_j p_j(x) = \operatorname{argmax}_j f_j(x)$
- Class probability estimate: $\frac{e^{f_k(x)}}{\sum_\ell e^{f_\ell(x)}}$

Tuning parameters.

Two basic tuning parameters are

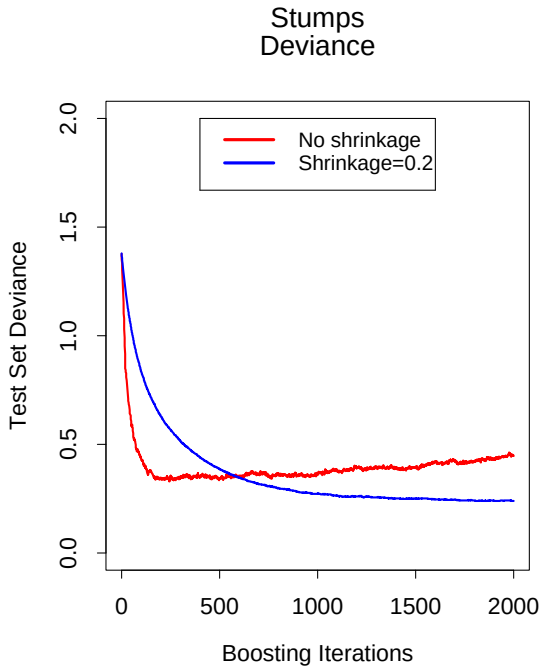
- the number of iterations M and
- the sizes of each of the constituent trees J_m, $m = 1, 2, ..., M$.

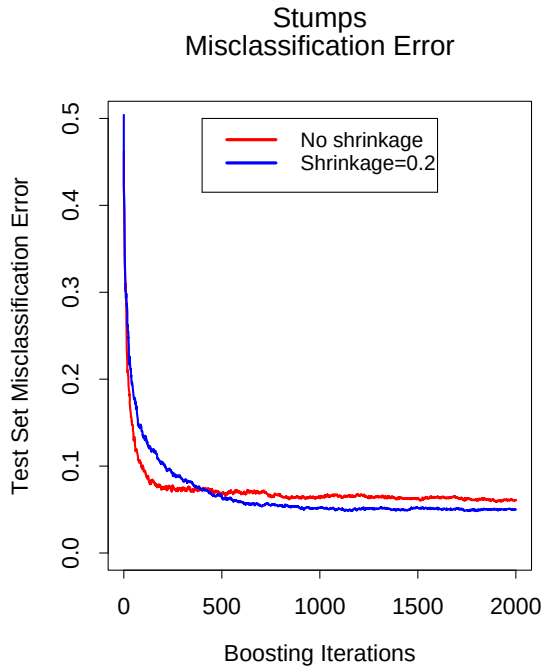
R and commercial packages

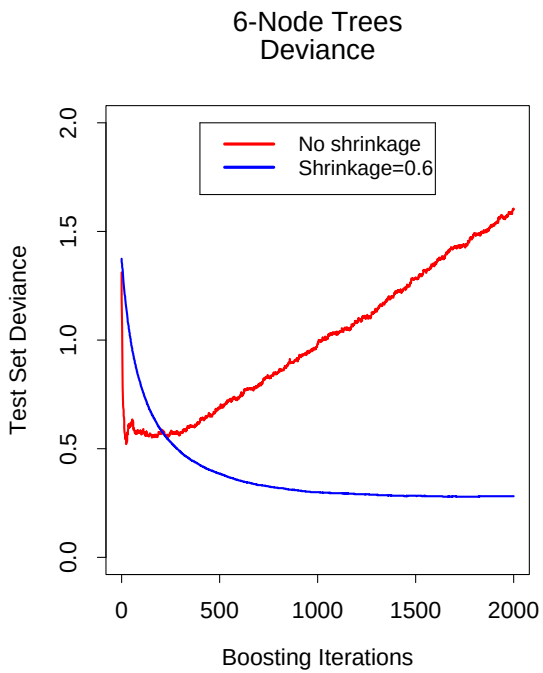
- Implemented in the R `gbm` package
- Another R implementation of boosting is `mboost`
- A commercial implementation of gradient boosting/MART called TreeNet® is available from Salford Systems, Inc.

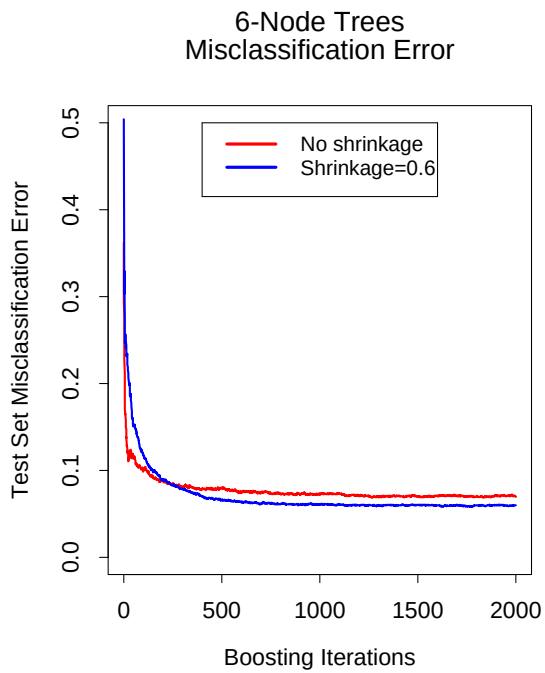
Improvements: Shrinkage

- Scale the contribution of each tree by a factor $0 < \nu < 1$ when it is added to the current approximation.
  $f_m(x) = f_{m-1}(x) + \nu \cdot \sum_j \gamma_{jm} \mathbb{1}_{[x \in R_{jm}]}$
- A trade-off: small ν means more M to achieve small error rates. Hence more computing cost.
- The best strategy appears to be to set ν to be very small (< 0.1) and then choose M by early stopping
- Dramatic improvements (over no shrinkage) for regression and for probability estimation.
- The corresponding improvements in misclassification risk are less, but still substantial.


Stumps
Deviance
No shrinkage
Shrinkage=0.2
Test Set Deviance
0.0
0.5
1.0
1.5
2.0
0
500
1000
1500
2000
Boosting Iterations


Stumps
Misclassification Error
No shrinkage
Shrinkage=0.2
Test Set Misclassification Error
0.0
0.1
0.2
0.3
0.4
0.5
0
500
1000
1500
2000
Boosting Iterations


6-Node Trees
Deviance
No shrinkage
Shrinkage=0.6
Test Set Deviance
0.0
0.5
1.0
1.5
2.0
0
500
1000
1500
2000
Boosting Iterations


6-Node Trees
Misclassification Error
No shrinkage
Shrinkage=0.6
Test Set Misclassification Error
0.0
0.1
0.2
0.3
0.4
0.5
0
500
1000
1500
2000
Boosting Iterations

Improvement: Subsampling

- We saw that bagging can improve performance.
- 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.
- A typical value for η can be 0.5, although for large N, η can be substantially smaller than 0.5.
- Not only does the sampling reduce the computing time by the same fraction η, but in many cases it actually produces a more accurate model.
- (Example next page) Note: it appears here that subsampling without shrinkage does poorly.

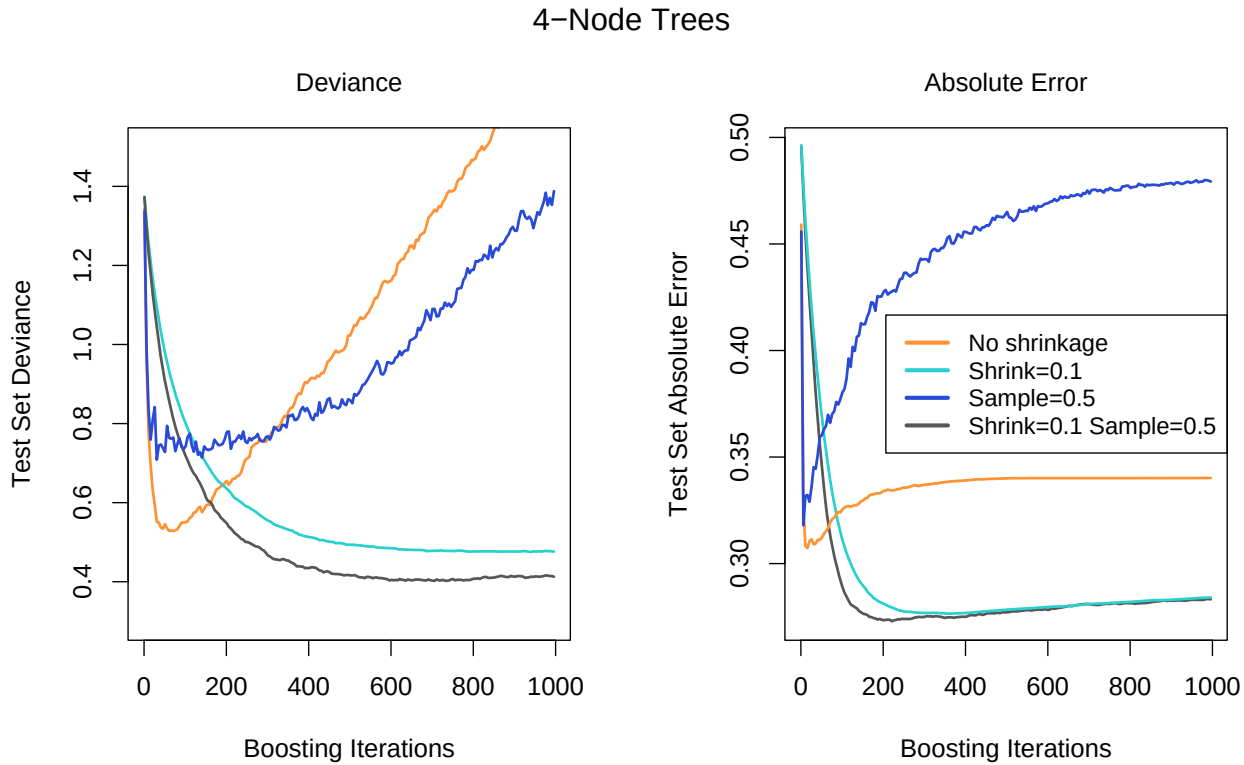


FIGURE 10.12. *Test-error curves for the simulated example (10.2), showing the effect of stochasticity. For the curves labeled "Sample= 0.5", a different 50% subsample of the training data was used each time a tree was grown. In the left panel the models were fit by* `gbm` *using a binomial deviance loss function; in the right–hand panel using square-error loss.*

The next section would be . . . . . .

1. Gradient Boosting
2. XGBoost

Tree Ensemble methods

So far,

- We have known how to ensemble trees using gradient boosting or random forest.
- Almost half of data mining competitions are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization
- Learn higher order interaction between features.
- Can be scalable, and are used in Industry

XGBoost

XGBoost = "Extreme Gradient Boosting"

- Idea: Combination of regularized gradient boosting and random forest.
- Pushed the extreme of the computation limits of machines to provide a scalable, portable and accurate library
- R library: `xgboost`.

Recall: Additive Training

- The prediction at round m is

$$f_m(x) = f_{m-1}(x) + \underbrace{\sum_j \gamma_j \mathbb{1}_{[x \in R_{jm}]}}_{\text{to be determined; call it } h_m}$$

- Overall, minimize

$$\text{Objective} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{m=1}^{M} \underbrace{\Omega(h_m)}_{\text{complexity}}$$

- So the goal at round m is to minimize

$$\text{Objective} = \sum_{i=1}^{n} \ell(y_i, f_{m-1}(x_i) + h_m(x_i)) + \Omega(h_m)$$

Loss function

There are ways to simplify the loss function. Consider square loss.

$$(y_i - f_{m-1}(x_i) - h_m(x))^2 = (2(f_{m-1}(x_i) - y_i)h_m(x) + h_m(x)^2) + \text{const.}$$

- For other loss functions, use Taylor expansion

  $$\ell(y_i, f_{m-1}(x_i) + h_m(x_i)) \approx \ell(y_i, f_{m-1}(x_i)) + r_{im}h_m(x_i) + \frac{1}{2}t_{im}h_m^2(x_i)$$

  where

  $$r_{im} = \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}|_{f=f_{m-1}}, \quad t_{im} = \frac{\partial^2 \ell(y_i, f(x_i))}{\partial f^2(x_i)}|_{f=f_{m-1}}$$

- With constant removed, only need to minimize

  $$\sum_{i=1}^{n} r_{im}h_m(x_i) + \frac{1}{2}t_{im}h_m^2(x_i) + \Omega(h_m)$$

Complexity of a Tree

- Recall $h_m(x) = \sum_{j=1}^{T} \gamma_j \mathbb{1}_{[x \in R_{jm}]}$
- Define its complexity as

$$\Omega(h_m) = \alpha T + \frac{1}{2}\lambda \sum_{j=1}^{T} \gamma_j^2$$

- T: number of leaf nodes
- $\sum_{j=1}^{T} \gamma_j^2$: squared L_2 norm of the predicted values.
- Next: incorporate this to the simplified objective function, and re-order the summations (over i and over j)

$$\sum_{i=1}^{n}\left[r_{im}h_m(x_i)+\frac{1}{2}t_{im}h_m^2(x_i)\right]+\Omega(h_m)$$

$$=\sum_{i=1}^{n}\left[r_{im}\sum_{j=1}^{T}\gamma_j\mathbb{1}_{[x_i\in R_{jm}]}+\frac{1}{2}t_{im}\left\{\sum_{j=1}^{T}\gamma_j\mathbb{1}_{[x_i\in R_{jm}]}\right\}^2\right]+\alpha T+\frac{1}{2}\lambda\sum_{j=1}^{T}\gamma_j$$

$$=\sum_{j=1}^{T}\left[\sum_{i=1}^{n}r_{im}\mathbb{1}_{[x_i\in R_{jm}]}\gamma_j+\frac{1}{2}\left\{\sum_{i=1}^{n}t_{im}\mathbb{1}_{[x_i\in R_{jm}]}+\lambda\right\}\gamma_j^2\right]+\alpha T$$

This is sum of T separate quadratic functions. We know from middle school that $Gx+\frac{1}{2}Hx^2$ has minimal value $-\frac{G^2}{2H}$ at $x=-\frac{G}{H}$.

Hence, for given tree structure, the optimal value for γ and minimal value of the objective are

$$\gamma_j^* = -\frac{\sum_{i=1}^n r_{im} \mathbb{1}_{[x_i \in R_{jm}]}}{\left\{\sum_{i=1}^n t_{im} \mathbb{1}_{[x_i \in R_{jm}]} + \lambda\right\}} \text{ and}$$

$$Obj^* = -\sum_{j=1}^T \frac{\sum_{i=1}^n r_{im}^2 \mathbb{1}_{[x_i \in R_{jm}]}}{2\left\{\sum_{i=1}^n t_{im} \mathbb{1}_{[x_i \in R_{jm}]} + \lambda\right\}} + \alpha T$$

- This becomes a new criterion to consider when splitting a node. Similar to impurity function in CART.
  - Given variable, where to split?
  - Which variable is the best one to split?
  - Whether it is worthwhile to split?

Recap: XGBoost

- Add a new tree in each iteration
- Beginning of each iteration, calculate r_{im} and t_{im}.
- Use a new statistics to greedily grow a tree. The goal is to minimize the objective function.
- Add the resulting tree to the model.
  - Usually, use the shrinkage improvement (see boosting)

Key parameters

- learning rate in shrinkage: `eta`– scaling the contribution of each tree
- α: `gamma` in R. complexity penalty on tree side.
- `max_depth`: maximum depth of a tree
- `subsample`: subsample ratio of the training instance

Recap of the entire course

What have you learned from the course?

- Regression, classification, clustering, dimension reduction, model ensemble.
- Linear and nonlinear methods. Kernel methods.
- Loss and regularization. Sparsity penalties.
- Model selection and assessment. Cross validation.
- Most importantly: the best practice of statistical learning and data analytics

Still a lot to learn

- Optimisation
- Bayesian network (directed graphical model)
- Undirected graphical model
- Neural network (deep learning)
- Compressive sensing
- Reinforcement learning
- Matrix completion
- .....