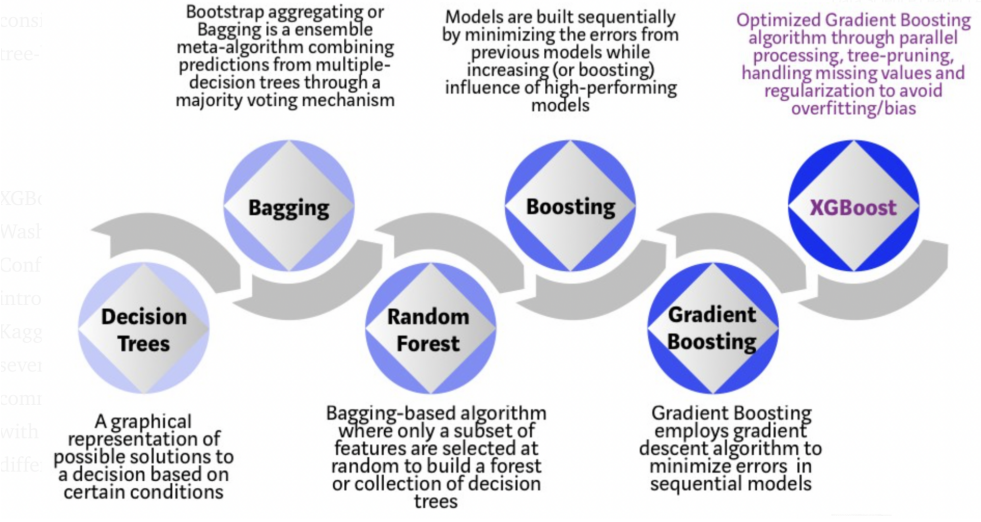
LEAP Climate Prediction Challenges

Tutorial on Decision Trees to XGBootsting

Tian Zheng
Department of Statistics, Columbia University



Source: Towards Data Science: XGBoost Algorithm: Long May She Reign!

Part I: Tree-based methods.

- *Tree-based methods* partition the feature space into a set of rectangles and then fit a simple model in each one.
- It is a *recursive partition method*.

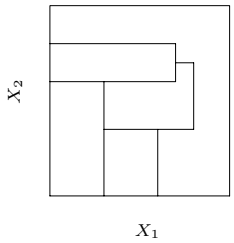
- Data:
  1. p inputs, $x_{i1}, \ldots, x_{ip}$;
  2. one response, y_i;
  3. N observations.
- The partition is in the form a binary splitting tree. For each splitting node, the split is defined based on *one input* and one splitting value, e.g., $x_j < s$.
- Algorithm automatically decides on the splitting variables and split points.

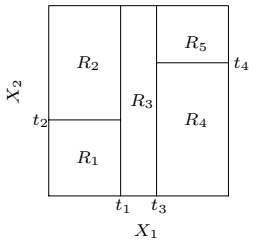
- Given a partition that has M regions, $R_1, \ldots, R_M$.
- Consider a simple model

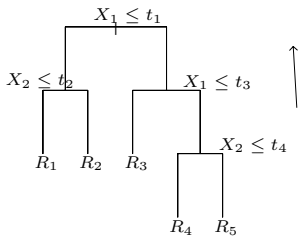
$$f(x) = \sum_{m=1}^{M} c_m \mathbf{1}(x \in R_m).$$

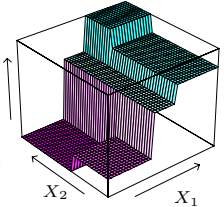
  Naturally, if using the quadratic loss, the expected loss is minimized when $c_m = \mathrm{E}(y|x \in R_m)$, $m = 1, \ldots, M$.
- Thus, the best fitted model for this partition using observed data is

$$\hat{f}(x) = \sum_{m=1}^{M} \bar{y}_{x \in R_m} \mathbf{1}(x \in R_m).$$


X_2
X_1


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_2
X_1


$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5


X_2
X_1

- Using criterion is based on loss function. $\sum(y - \hat{f}(x))^2$, a greedy algorithm grows a tree that leads to a "good" partition.
- The greedy algorithm

step 0 Let $R_{0,1} = \mathbb{R}^p$.

step k For each $1 \leq i \leq 2^{k-1}$, find an input j and a split value s, that define

$$\begin{aligned} R_{k,2i} &= R_{k-1,i} \wedge \{x_{.j} < s\}, \\ R_{k,2i+1} &= R_{k-1,i} \wedge \{x_{.j} \geq s\}, \end{aligned}$$

and minimize

$$\sum_{x_{i.} \in R_{k,2i}} (y_i - \bar{y}_{x_{i.} \in R_{k,2i}})^2 + \sum_{x_{i.} \in R_{k,2i+1}} (y_i - \bar{y}_{x_{i.} \in R_{k,2i+1}})^2$$

- The determination of the best pair of (j, s) is a finite search of complexity $p \cdot (\#x_{i\cdot} \in R_{k,i})$.
- Here k is the number of layers of the tree that the algorithm grows till step k.
- How far should we go in k:
  - The algorithm will stop when there is only one observation left in the node region.
  - Large k will lead to overfitting. Small k may not be provide enough structure to fit the data.
  - Thus, here, the tree size is the tuning parameter for model complexity.

- Use extra sum of squares based criterion as in regression analysis.
- (problem) "a seemingly worthless split might lead to a very good split below it".
- **Pruning:**

step 0 Pre-decide a minimal allowed node region size—number of observations in the node region.

step 1 Grow a tree under the requirement of the minimal allowed node region size.

step 2 Pruning—collapsing some nodes regions to reduce complexity without much increment in the loss.

- Consider the cost-complexity criterion ($|T|$ is the number of terminal nodes in T).

$$C_\alpha(T) = SSE + \alpha|T|,$$

- The cost-complexity criterion has a similar form as the AIC and C_p criteria.
- *Weakest link pruning:* start with the tree in step 1, recursively merging nodes with the lowest increment in ΔSSE, this will result in a sequence of subtrees,
  - partition of one subtree is nested in the one of the next;
  - The $|T|$ decreases by 1.
  - The SSE increases.
- It can be shown that the subtree that minimizes C_α can be found in this sequence.
- α can be determined using cross-validation.

- The tree is grown in the same fashion as for regression tree (greedy algorithm, pruning, etc).
- Define $\hat{p}_m(k) = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbf{1}(y_i \in k)$.
- Different criteria for splitting
  - Misclassification error.
  - Gini index: $\sum \hat{p}(1 - \hat{p})$.
  - Cross-entropy or deviance: $\sum \hat{p} \log \hat{p}$.
- Gini index and entropy are more sensitive to changes in the node probabilities.
- Among classifiers with the same prediction performance, the one with better probability differentiation among classes is believed to have better generalization performance. (An idea also used in SVM.)

- CART!
- Why binary split?
  - Controls the complexity growth of the tree.
  - Multiple split can be rewritten into a unique sequence of binary splits according to the greedy algorithm.
- Why not using more inputs at each splitting node?
  - It does improve predictive power.
  - But it is hard to interpret.
  - It is also hard to search for good tree topology.

- One wrong split due to chance "mess up" all consequent splits.
- A way of adjusting for sampling variability—bagging the trees. This is the idea behind the method of *random forest* (RF).

Part II: Model averaging

- Based on bootstrap samples

$$\hat{f}_{\text{bag}} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

- This is different from model built with bootstrap estimate of the model parameters, when the model is nonlinear.
- For classification problems, the $\hat{f}$ usually takes the form estimated probabilities for classes. And the prediction is usually given as the most probable class.
- The bagged predictor for classification can take two forms:
  - bag the predictions (weighted votes)
  - bag the class probabilities

- *Committee methods*: the final model is an unweighted "average" of the fitted individual $\hat{f}_m$'s.

$$\hat{f}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x; \hat{\theta}_m)$$

- Models M_1, ..., M_M can also be averaged based on $\Pr(M_m|\mathbf{Z})$.

$$\hat{f}(x) = \sum_{m=1}^{M} \hat{f}_m(x; \hat{\theta}_m) \Pr(M_m|\mathbf{Z})$$

- $\Pr(M_m|\mathbf{Z})$ can be well estimated by BIC. Full Bayesian computation can also derive these probabilities but is more computationally intensive, and the computational cost is not justified by performance.

- Models averages from previous slide can be viewed as *weighted average*.

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

- Searching for the best weight, one may consider

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \omega_m \hat{f}_m(x_i))$$

- Will this lead to overfitting?

- Stacking or *stacked generalization*

$$\hat{\omega}^{st} = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \omega_m \hat{f}_m^{-i}(x_i)\right)$$

- If we search ω among vectors with elements 0 or 1, and with sum 1, stacking is equivalent to leave-one-out model selection.

Part III: Boosting.

- "Weak" learners: classifiers with error rates slightly better than random guessing.
- Boosting: combines the outputs of many "weak" learners to produce a better prediction.
- The combined prediction is in the form of a *committee vote*, weighted sum of individual votes.
- The key here is that these weak learners should NOT be highly correlated.
- Boosting procedure sequentially train classifiers (learners) based on current prediction performance and decide the weight of the classifier trained on the fly.
- Each classifier is trained to the training data using weights adaptively updated each step.

- A learning algorithm that produces weak learners—*usually easy to compute and interpret*.
- *A mechanism to generate modified versions of data*—so that the classifiers are less correlated.
- A mechanism to assign weights to learner—*so that the performance of the collection of classifiers can be "optimized"*.

- Class codes: [-1: class 1; 1: class 2]

Step 0 $w_i = 1/N$, $i = 1, \ldots, N$.

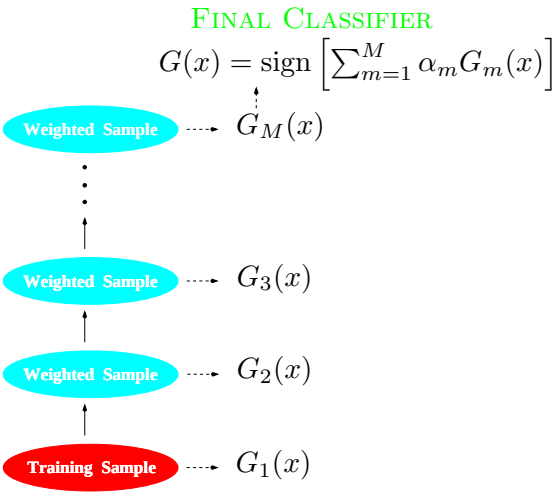
Step m Iterate:

- Fit classifier $G_m(x)$ to the training data with weights w_i.
- Compute

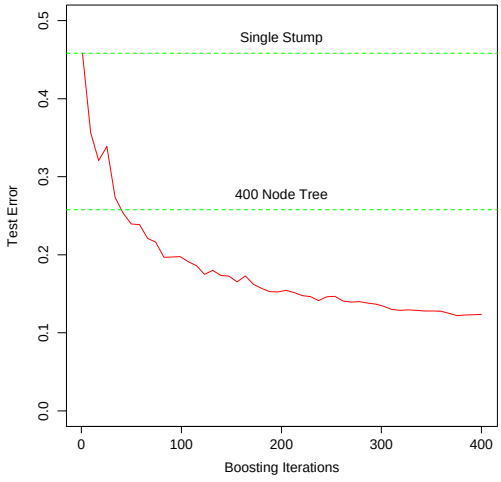
$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

- weight for this classifier: $\alpha_m = \log \frac{1-\text{err}_m}{\text{err}_m}$.
- $w_i = w_i \cdot \exp(\alpha_m \mathbf{1}(y_i \neq G_m(x_i)))$. (For weak learner, err_m might be close to 0.5 but should less than 0.5.)

Final classifier: $G(x) = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$

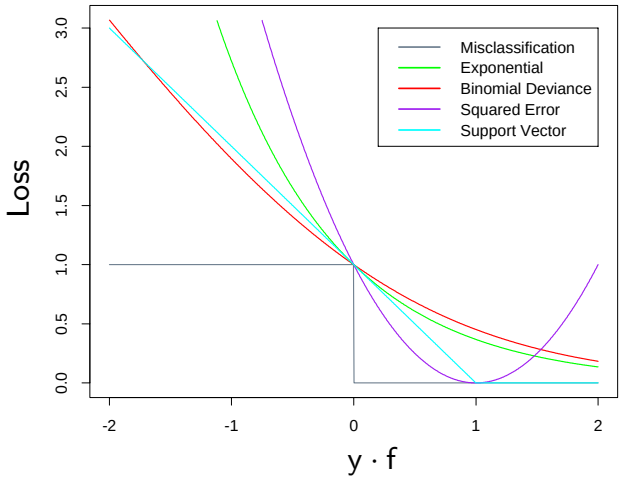

Final Classifier
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

Weak learner used: stumps


Single Stump
400 Node Tree
Test Error
Boosting Iterations
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400

- The AdaBoost.M1 is know as “discrete AdaBoost” since its classifier gives a categorical prediction.
- It can be modified for regression problems.

Part IV: Loss Functions


Misclassification
Exponential
Binomial Deviance
Squared Error
Support Vector
Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
-2
-1
0
1
2
y · f

- Consider classification problems. Outcome Y takes value 1 or -1.
- Let $f(x)$ be a regression function used for classification. The prediction based on $f(x)$ is then $G(x) = \text{sign}(f(x))$.
- Misclassification when $y_i \neq G(x_i)$.
- "margin": $yf(x)$, where f(x) takes continuous value.
- Large positive value of the "margin" means that the prediction is correct and well within the classification boundary.
- Large negative value means the misclassification is very much cross the boundary (thus less like to go away with a small random changes to the data).

- Misclassification rate only penalize misclassifications.
- The exponential loss is a continuous function that approximates the misclassification loss.
- However, the penalty for negative margin increase faster than the "reward" for positive margin.
- The support vector also penalize small positive margins.
- Also from this figure, we see the problem with squared loss in this classification setting.

- It leads to simple modular reweighting AdaBoost Algorithm.
- It is also shown that

$$f^*(x) = \text{argmin}_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2} \log \frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}.$$

- Binomial log-likelihood loss

$$l(Y, f(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x))$$

  where $Y' = (1 + Y)/2$.
- Here

$$-l(Y, f(x)) = log\left(1 + e^{-2Yf(x)}\right).$$

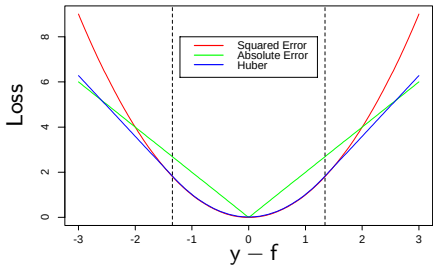


Figure 10.5: *A comparison of three loss functions for regression, plotted as a function of the margin* $y - f$. *The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when* $|y - f|$ *is large.*

Part V: boosting trees

- A tree is represented by the regions $R_j$represented by its terminal nodes.

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

- Once R_j are identified, γ_j is easy to estimate.
- Global optimal R_j's are hard to find. CART uses a top-down greedy algorithm.
- Smoother target functions work better than misclassification loss.

- is a sum of trees

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

- Here Θ_m is a general notation representing both the tree topology (partition) R_j's and the γ_j's at terminal nodes.
- At each step

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

- Finding R_j for Θ_m may be more difficult than fitting a single tree.

- For square-loss, $T(x, \Theta_m)$ can be identified by fitting a single tree to the residuals $y_i - f_{m-1}(x_i)$.
- For exponential loss, the optimal tree is identified by minimizing a weighted misclassificaiton rate.
- For other loss function, we need *gradient boosting*.

- Recall

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

- $T(x_i; \Theta_m)$ should be updated along the steepest decent of $L(y, f)$ at $f = f_{m-1}$.
- Let $\mathbf{g}_m$ be the gradient with components

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

- Problem: a tree may not be able to move along this gradient exactly.

- An approximate step is then introduced.
- A tree is fitted to $-g_{im}$ with squared loss, i.e.,

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2.$$

TABLE 10.2. *Gradients for commonly used 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: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

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
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (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
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

- To avoid overfitting at each step, it is recommended to use trees of a fixed size J.
- J is then becomes a tuning parameter.

- For boosting, regularization means when to stop or how many steps should be used.
- Another strategy is *shrinkage*

$$f_m(x) = f_{m-1}(x) + \nu T(x; \Theta_m)$$

  where $0 < \nu < 1$.
- Subsampling can also be used at each step to avoid overfitting $T(x; \Theta_m)$.

https://tz33cu.shinyapps.io/Tutorial7-GBM/

Part VI: Interpreting Boosted Trees

- Relative importance of each variable X_l is defined as

$$I_l^2 = \frac{1}{M} \_sum_{m=1}^{M} I_l^2(T_m)$$

- where

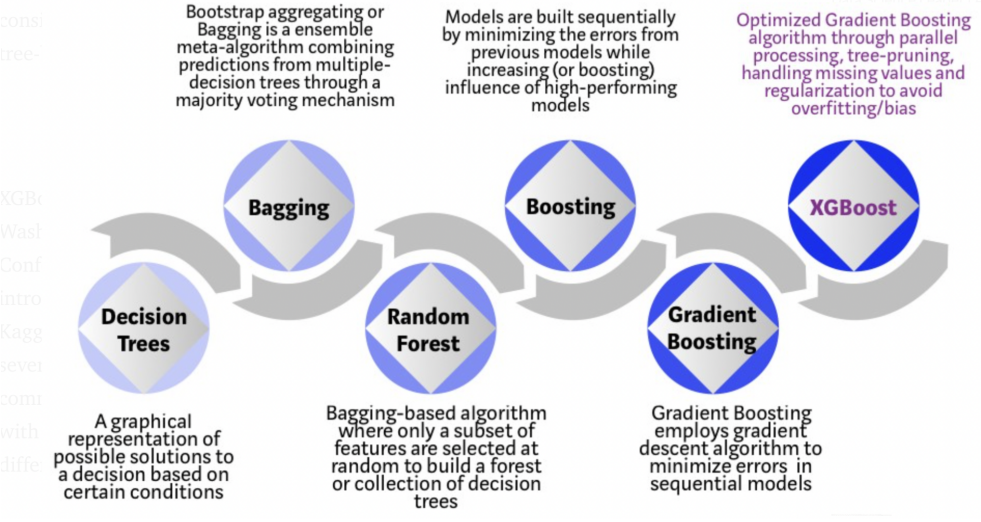
$$I_l^2(T_m) = \sum_{\text{each internal node } t} \Delta_t^2 I(X_l \text{is used for this split})$$

  and Δ_t is the improvement at internal node t.
- This measure depends on data. We can set the largest to 100 and scale the others accordingly.

- $f(x)$ is usually a high-dimensional function that is hard to visualize.
- For "important" variables we are interested to know how they affect $f(x)$.
- We rewrite $f(x)$ into $f(x = (x_s, x_c))$ where x_s is what we want to display.
- We integrate out x_c to derive partial dependence of f on x_s,

$$f_s(x_s) = E_{x_c} f(x_s, x_c).$$

Part VII: XGBoosting



Source: Towards Data Science: XGBoost Algorithm: Long May She Reign!