Development of Boosting algorithms: from AdaBoost to LightGBM

Wenyue LIU
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599

February 1, 2017

1 Introduction

With the development of computer calculation ability, data scientists have more advanced tools to manage big data and complex algorithms. Combing basic algorithms to improve classification/regression accuracy is a straight forward idea. One of the most famous ensemble algorithms is boosting, which is first introduced by Robert Schapire in a paper published in 1990. The basic logic of boosting algorithm is combining multiple base classifier or weak classifier to generate a classifier committee, which have a better classification accuracy. Compared with another committee based algorithm bootstrap aggregating(bagging), the major difference is that boosting's base classifiers are trained by sequence, bagging use average answers from each members of committee. Start from random guess, boosting add new classifier iteratively to previous classifier. To add a new classifier, classifier function depends on a relationship to previous weak learner's accuracy. Considering all added weak classifier's performance together will provide the final strong classifier. Not limited to classification, application on regression also achieved high prediction accuracy. Some commonly used boosting algorithms are AdaBoost, L2Boost, LogitBoost, Gradient Boosting Machine(GBM). Several popular GBMs frame are Multivariate adaptive regression trees(MART), XGBoost, LightGBM. Boosting and the idea of ensemble learning are highly evolved in data mining competition and industrial application.

2 Boosting algorithms

2.1 General algorithm

The primary goal of boosting algorithm is to minimize loss function, i.e. minimize the misclassification ratio or deviance for classification issue. For binary classification, several loss functions such as misclassification: $I(\text{sgn}(f(x)) \neq y)$, exponential: $exp(-y * f)$, log-likelihood(entropy): $\log(1 + exp(-2yf))$, euclidean distance: $(y - f)^2$, and hinge loss:

$(1 - yf)_+$ are used in boosting algorithm. Comparison of loss reduction curve is shown in following.

A general boosting algorithm called Forward Stage-wise Additive Modeling(FSAM) approximate minimization of (1) via adding new classifiers iteratively. Detail work path of FSAM is shown in Algorithm 1.

$$\min_{\beta_m, \gamma_m{}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right). \tag{1}$$

Algorithm 1 Forward Stage-wise Additive Modeling

```
Initialize $f_0(x) = 0$
for $m = 1 : M$ do
    Compute $(\beta_m, \gamma_m) = \arg\ min \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$;
      ▷ M:iteration times; L:loss function; $\beta$ :expansion coefficient; $b$:weak classifier
    Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$
Return $f(x) = f_M(x)$
```

By using exponential loss function, a well-known AdaBoost(Algorithm 2) is derived from FSAM. However, in AdaBoost algorithm, error observations are blown up by exponential item in $\omega_i \leftarrow \omega_i \exp[\alpha_m \mathbb{1}(\tilde{y}_1 \neq \phi_m(x_i))]$, which will cause lacking of robustness for datasets with high outlier observations. LogitBoost, which updates classification rule linearly(Algorithm 3), is a reasonable substitute to AdaBoost. Comparison of L2Boosting, AdaBoost and LogitBoost is clearly shown in Table 1.

Algorithm 2 AdaBoost, for binary classification with exponential loss

```
$\omega_i = 1/N$;
for $m = 1 : M$ do
    Fit a classifier $\phi_m(\mathbf{x})$ to the training set using weight $\mathbf{w}$;
    Compute $err_m = \frac{\sum_{i=1}^{N} \omega_{i,m} \mathbb{1}(\tilde{y}_1 \neq \phi_m(x_i))}{\sum_{i=1}^{N} \omega_{i,m}}$;
    Compute $\alpha_m = \log[(1 - err_m)/err_m]$;
    Set $\omega_i \leftarrow \omega_i \exp[\alpha_m \mathbb{1}(\tilde{y}_1 \neq \phi_m(x_i))]$;
Return $f(x) = \text{sgn}\left[\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x})\right]$
```

Table 1: Comparison of various boosting algorithms

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(x_i))^2$	$(y_i - f(x_i)$	$\mathbb{E}[y\|x_i]$	L2Boosting
Exponential loss	$exp(-\tilde{y}_i f(x_i))$	$-\tilde{y}_i exp(-\tilde{y}_i f(x_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

Algorithm 3 LogitBoost, for binary classification with log-loss

```
    $\omega_i = 1/N, \pi_i = 1/2$;
    for $m = 1 : M$ do
        Compute the working response $z_i = \frac{y_i^\star - \pi_i}{\pi_i(1-\pi_i)}$;
        Compute $\omega_i = \pi_i(1 - \pi_i)$;
        $\phi_m = \arg\ min_\phi \sum_i^N w_i(z_i - \phi(x_i))^2$;
        Update $f(x) \leftarrow f(x) + \frac{1}{2}\phi_m(x)$;
        Compute $\pi_i = 1/(1 + exp(-2f(x_i)))$
    Return $f(x) = \text{sgn}\left[\sum_{m=1}^M \phi_m(\mathbf{x})\right]$
```

2.2 Gradient Boosting model

In general boosting algorithm, the object is to minimize summation of loss function by adjust weight on each observation in each iteration steps. To descent summation of loss function steeply, gradient boosting model shows efficient performance. In (2), $r_{im} \in \mathbb{R}^N$ is the gradient of loss function. In each iteration, classification rule $f(x)$ is updated by $-\rho_m \mathbf{r}_m$ as $f_m = f_{m-1} - \rho_m \mathbf{r}_m$. Here, step length ρ_m is obtained from $\rho_m = \arg\ minL(\mathbf{f}_{m-1} - \rho_m \mathbf{r}_m)$. Gradient boosting is detailed in algorithm 4.

$$r_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \tag{2}$$

For regression, absolute error, quadratic error and Huber error(3) are three commonly used loss function. Gradients of them are shown in table 2.

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for} \quad |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise} \end{cases} \tag{3}$$

Algorithm 4 Gradient boosting

```
    Initialize $f_0(x) = \arg\ min_\gamma \sum_{i=1}^N L(y_i, \phi(x_i; \gamma))$;
    for $m = 1 : M$ do
        Compute the gradient residual using $r_{im} = -\left[\frac{\partial L(y_i, f(x_i)}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$;
        Use the weak learner to compute $\gamma_m$ which minimize $\sum_{i=1}^N (r_{im} - \phi(x_i; \gamma_m))^2$;
        Update $f_m(x) = f_{m-1}(x) + \nu\phi(x; \gamma_m)$;
    Return $f(x) = f_M(x)$
```

Table 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	$\lvert y_i - f(x_i) \rvert$	sign$\lvert y_i - f(x_i) \rvert$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i) \leq \delta_m$ δ_m sign$[y_i - f(x_i)]$ for $4\lvert y_i - f(x_i) \leq \delta_m$ where $\delta_m = \alpha th - quantile\lvert y_i - f(x_i)\rvert$

2.3 Regularization

Even the gradient descent algorithm could greedily find the rule with lowest training loss, the risk of overfitting will be more severe with the increasing of iterations M. Regularization is needed to suppress overfitting. Using early stopping to find optimal iterations is an efficient strategy. The optimal M is selected when evaluation metric on test data stop optimizing after user defined iterations, for example AUC values stop increasing within 60 new added trees after local maximum point attained.

Another regularization method is setting learning rate ν. Like ridge or LASSO regression, a penalization item is added to balance variance-bias trade-off. In algorithm 4, $f_m(x)$ is updated by $\nu\phi(x; \gamma_m)$. An empirical selection of ν is 0.1, it is obvious from the algorithm that the smaller the ν, the bigger the M.

Sub-sampling is another regularization method. The idea of bagging is use bootstrap method to repeatably sub-sampling from full dataset, and decrease variance via averaging. Implying sub-sampling idea in boosting algorithm, not only reducing prediction variance, but also significantly releasing computation time.

For boosting with tree as weak classifier, by controlling other parameters like maximum depth of each tree, minimum loss reduction needed for leaf node spiting and proportion of 1 and 0 in binary classification etc., over-fitting is better controlled.

2.4 XGBoost

A popular tree based boosting algorithm in data mining competitions is XGBoost, developed by Tianqi Chen in 2014. In his method, both first and second derivatives of loss function are employed to approximate loss function minimization. Also, number of leafs and L2 norm of leaf scores (like ridge penalization) are considered in the structure complexity $\Omega(f_t)$. Objective of XGBoost is shown in following:

$$
\begin{aligned}
Obj^(t) &= \sum_{i=1}^{n} L(y_i, \hat{y_i}) + \sum_{k} \Omega(f_t)) \\
&\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \quad \triangleright \text{Taylor expansion} \\
&= \sum_{i=1}^{n} \left[g_i \omega_{q(x_i)} + \frac{1}{2} h_i \omega_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} \omega_j^2 \quad \triangleright \text{Number of leafs and L2 norm of scores} \\
&= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) \omega_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) \omega_j^2 \right] + \gamma T \quad \triangleright \text{Restructure} \\
&= \sum_{j=1}^{T} [G_j \omega_j + \frac{1}{2} (H_j + \lambda) \omega_j^2] + \gamma T \quad \triangleright G_j = \sum_{i \in I_j} g_i ; H_j = \sum_{i \in I_j} h_i
\end{aligned}
\tag{4}
$$

From equation 4, the optimal objective value is $Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$. Through greedy learning of each tree, we would like to find node split with biggest gain through enumeration scan, which is defined as $Gain = \frac{1}{2} [\frac{G_L eft^2}{H_L eft + \lambda} + \frac{G_R ight^2}{H_R ight + \lambda} - \frac{(G_L eft + G_R ight)^2}{H_L eft + H_R ight + \lambda}] - \gamma$. Where first item is 'goodness' of left child, second item is 'goodness' of right child, third item is 'goodness' if no split applied and forth term is complexity cost of one additional leaf.

2.5 LightGBM

A recent gradient boosting machine frame developed by Microsoft open source team and community optimized GBM in several ways. The pre-sorted algorithms (XGBoost used to find node spiting) need $O(\#data)$ time to find best split point. In LightGBM, histogram based algorithm, which bucketing continuous features into discrete bins, significantly reduce computation cost. Compared to pre-sorted algorithm cost $O(\#data)$, histogram algorithm only ask for $O(\#bins)$+ $O^*(\#data)$, in which O represents for complex gain calculation operation, O^* represents for simple sum-up calculation. Although histogram based algorithms face problem that inaccuracy on spiting node searching, LightGBM still shows better prediction performance than XGBoost in several benchmark datasets. It is understandable by treating the inaccurate splitting as a kind of regularization.

In addition, unlike level-wise tree growing method used in XGBoost, LightGBM uses leaf-wise growing algorithm since some leafs with low splitting gain do not need to splitting. Therefore, when growing same number of leaf, leaf-wise algorithm will reduce more loss than level-wise algorithm.

3 Application

Using XGBoost and LightGBM will achieve high prediction accuracy for both regression and classification problems. Among 29 challenge winning solutions on Kaggle during 2015, 17 of them used XGBoost. In Kaggle competition ends in start of Nov. 2016, winning competitors used LightGBM, which is published in Oct. 2016. In Kaggle's BOSCH production line performance competition, I used XGBoost to win 100/1391 rank. Full codes were posted in `https://github.com/brucebismarck/Kaggle-Bosch-Production-Line-Performance`.

4 Bibliography

Trevor Hastie, Robert Tibshirani and Jerome Friedman (2008). The Elements of Statistical Learning Second Edition Chapter 10, 15, 16. *Springer.*

Christopher M. Bishop. (2006). Pattern Recognition and Machine Learning Chapter 14*Springer.*

Kevin P. Murphy (2012). Machine Learning A Probabilistic Perspective Chapter 16. *MIT Press.*

Tianqi Chen and Carlos Guestrin. (2016) XGBoost: A Scalable Boosting System *KDD '16, August 13-17, 2016, San Francisco, CA, USA*.

Microsoft Open Source Team (2016) LightGBM: Light Gradient Boosting Machine *https://github.com/Microsoft/LightGBM.*

Robert E. Schapire. (1990) The Strength of Weak Learnability *Machine Learning* **5**, 197-227.