Universal Boost Decision Trees: A Unified Framework to Translate Optimization Methods to Boosting Algorithms

Zhang Jinxiong

November 9, 2019

Abstract

A general paradigm is developed to translate optimization methods to 'boosting' algorithms. The gradient boost decision trees inspire higher order boost decision trees. And we translate the augmented Lagrangian methods to boosting algorithms with regularized cost fucntion. The interaction of optimization methods and boosting algorithms is discussed in surrogate boosting decision trees. It is first time to connent almost all optimization methods and boosting algorithms. It is why we call it as 'universal boost'. Although it is limited on boosted decision trees, the idea is easy to generalize to any boosting machine.

1 Introduction

Jerome Friedman et al assert that the Discrete AdaBoost algorithm (population version) builds an additive logistic regression model via Newton-like updates for minimizing $E(e^{-yf(x)})$. Jyrki Kivinen, Manfred K. Warmuth, Chunhua Shen, Hanxi Li study some boosting algorithms from a entropy minimization perspective. The gradient boost machine algorithms seems far from continuous optimization methods. Gradient boost machine is usually considered as functional gradient descent. There are many *gradient-like boosting machines* to fit the profitable directions different from negative gradients such as AnyBoost, Functional Frank-Wolfe Boosting, Historical Gradient Boost Machine, XGBoost, NGBoost.

It seems that optimization methods can be mapped to boosting algorithms. However, it is not clear how to translate continuous optimization methods to their functional version such as ADMM. We will show that there is a unified framework to translate each optimization method to boosting algorithm.

2 Gradient Boost Decision Trees

Boosted decision tree or multiple additive regression tree is the sum of successive decision tree

$$F(x) = \sum_{n=1}^{T} f_t(x)$$

where f_n relies on the outputs of its 'parent tree' f_{n-1} for $n = 2, 3, \cdots, N$.

The following algorithm describe the gradient boost decision as boosted decision trees

Algorithm 1 Gradient Boost Decision Trees

Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
Initialize $f_0 = \arg\min_\gamma \sum_{i=1}^{N} L(x_i, \gamma)$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    compute $r_{i,t} = -[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=F^{(t-1)}}]$.
  end for
  Fit a regression tree to the targets $r_{i,t}$ giving terminal regions

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

  for $j = 1, 2, \ldots, J_m$ **do**
    compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, F^{(t-1)}(x_i) + \gamma)$.
  end for
  $f_t = \sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m})$
  Update $F^{(t)} = F^{(t-1)} + \nu f_t, \nu \in (0, 1)$
end for
Output $F^{(T)}$.

In gradient boost decision tree, it takes additive training:

$$\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, \underbrace{F^{(t-1)}(x_i) + \gamma}_{\text{additive training}})$$

which results as weighted sum of decision trees. Here $\gamma_{j,t}$ is expected to be parallel of $-[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=F^{(t-1)}}] = -g_{t-1,i}$ thus $F^{(t)}(x_i) \approx F^{t-1}(x_i) - \alpha g_{t-1,i}$.

If let $F^{(t-1)}(x_i) = \tilde{y}_i$, the following inequality holds for some $\alpha \in (0, 1)$ if the loss function L is smooth

$$\sum_{i=1}^{N} L(\mathrm{y}_i, \tilde{y}_i) \geq \sum_{i=1}^{N} L(\mathrm{y}_i, \tilde{y}_i - \alpha[\frac{\partial L(\mathrm{y}_i, \tilde{y}_i)}{\partial \tilde{y}_i}]).$$

This is basic idea of gradient descent. In gradient boost decision trees, a tree f_t is used to fit the negative gradient $-\frac{\partial L(\mathrm{y}_i, \tilde{y}_i)}{\partial \tilde{y}_i}$ so that it brings some noise or error in this step.

First we start with the gradient descent

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_t \nabla_\theta L(\theta^{(t)})$$

for $t = 0, 1, 2, \cdots, T-1$. The final result of gradient descent is

$$\theta^{(T)} = \theta^{(0)} - \sum_{t=1}^{T-1} \alpha_t \nabla_\theta L(\theta^{(t)}).$$

If we replace the $\theta^{(T)}, \theta^{(0)}$ with the gradient boost decision trees $F^{(T)}, f_0$ and $-\nabla_\theta L(\theta^{(t)})$ with the decision tree f_t for $t = 1, 2, \cdots, T-1$, we will obtain

$$F^{(T)} = f_0 + \sum_{t=1}^{T-1} \alpha_t f_t.$$

In some sense, a decision tree is to fit $-\alpha_t \nabla_\theta L(\theta^{(t)})$ in gradient boost decision trees. So we say that it is analogous to gradient descent.

In optimization, we call the point γ at point $\tilde{y}_i$ as profitable direction if

$$\sum_{i=1}^{N} L(\mathrm{y}_i, \tilde{y}_i) \geq \sum_{i=1}^{N} L(\mathrm{y}_i, \tilde{y}_i + \gamma).$$

There are many *gradient-like boosting machines* to fit the descent directions different from negative gradients such as Functional Frank-Wolfe Boosting, Historical Gradient Boost Machine, XGBoost, NGBoost. It is also the basic idea behind AnyBoost.

Any techniques in gradient-based optimization methods can be applied to gradient guide decision tree such as the distributed optimization techniques, acceleration techniques, variance reduction techniques.

Note that the following relation always holds for most loss function:

$$\ell(\mathrm{y}_i, f(x_i)) = 0 \iff \mathrm{y}_i = f(x_i) \iff \frac{\partial \ell(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=F^{(t-1)}} = 0$$

which means if $f(x_i) = \mathrm{y}_i$ the value 0 will be learnt in next decision tree. In another words, the gradient is always to be 0 when the loss is 0. And the outputs is a constant in vanilla decision tree for different input in the same leaf so that the new targets $r_{i,t}$ and $r_{j,t}$ are equal if x_i, x_j are output in the same leaf and $y_i = y_j$. As a result, such pair x_i, x_j tends to be in the same terminal region in the next tree. The samples in the same terminal region with different targets tend to be seperated in the next tree. The gradients is used to build the next decision tree.

"If I have seen further, it is by standing on the shoulders of giants." The new targets $\{f_{t-1}(x_i) - \alpha_{i,t} g_{i,t}\}$ are the shoulders of giants in decision tree for $i = 1, 2, \cdots, N, t = 1, 2, \cdots, T$.

3 Higher Order Boost Decision Trees

xGBoost extend the gradient boost to the second order boost methods. We can generalize much higher order boost decision tree.

The first one is to translate the Halley's method to a third order boosting method. At point X_n, the next point X_{n+1} is found by Halley's Method:

$$X_{n+1} = X_n - \frac{2f(X_n)f'(X_n)}{2[f'(X_n)]^2 - f(X_n)f''(X_n)}. \tag{1}$$

It is proved to be cubical convergence insofar as the number of significant digits eventually triples with each iteration.

Halley's Method modefied Helley's method

Algorithm 2 Higher Order Boost Decision Trees

Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
Initialize $f_0 = \arg\min_\gamma \sum_{i=1}^{N} L(x_i, \gamma)$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    Compute $r_{i,t} = T_t(x_i)$
  end for
  Fit a regression tree to the targets $r_{i,t}$ giving terminal regions

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

  for $j = 1, 2, \ldots, J_m$ **do**
    Compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, F^{(t-1)}(x_i) + \gamma)$.
  end for
  $f_t = \sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m})$.
  Update $F^{(t)} = F^{(t-1)} + f_t$.
end for
Output $F^{(T)}(x)$.

Here $T_t(x_i) = -\frac{2g_t(x_i)g_t'(x_i)}{2[g_t'(x_i)]^2 - g_t(x_i)g_t''(x_i)}$ and $g_t(x_i) = \frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=F^{(t-1)}}$. Note that it works only when $2[g_t'(x_i)]^2 - g_t(x_i)g_t''(x_i) \neq 0$.And when $g(x) = 0 \forall x \in \mathbb{R}$, it is exactly the Newton's method.

4 ADMM Boost Decision Trees

Now we take alternating direction method of multipliers (ADMM) into consideration.

ADMM is aimed to solve the following convex optimization problem:

$$\min F(x, y) \{= f(x) + g(y)\} \text{ subject to } \quad Ax + By = b \tag{2}$$

where $f(x)$ and $g(y)$ is convex; A and B are matrices. Its augmented Lagrangian is defined as $L_\beta(x,y) = f(x) + g(y) - \lambda^T(Ax + By - b) + \frac{\beta}{2}\|Ax + By - b\|_2^2$. ADMM at step t is described as following:

1. $x^{k+1} = \arg\min_{x\in\mathbf{X}} L_\beta(x, y^k, \lambda^k)$;
2. $y^{k+1} = \arg\min_{y\in\mathbf{Y}} L_\beta(x^{k+1}, y, \lambda^k)$;
3. $\lambda^{k+1} = \lambda^k - \beta(Ax^{k+1} + By^{k+1} - b)$.

The aim boosting algorithms is to find a decision tree f^* to minimize the following function

$$\arg\min_f \sum_{i=1}^N \ell(y_i, f(x_i)) + r\sum_{i=1}^N |f(x_i)| \text{ subject to } z_i = f(x_i)$$

where $\ell(y_i, f(x_i))$ is the loss of sample (x_i, y_i) and $r\sum_{i=1}^N |f(x_i)|$ is the regularization term. Note that the loss function $\ell(\cdot,\cdot)$ is usually convex and f is a decision tree (region-wise linear function), so the above cost function is convex.

Algorithm 3 ADMM Boost Decision Trees

Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
Initialize $f_0 = \arg\min_\gamma \sum_{i=1}^N L(x_i, \gamma)$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    Compute $r_{i,t} = \arg\min_{\gamma_i} L_\beta^{(t)}(\gamma_i, z_i, \lambda_i)$.
  end for
  Fit a regression tree to the targets $r_{i,t}$ giving terminal regions

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

  for $j = 1, 2, \ldots, J_m$ **do**
    Compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, \gamma)$.
  end for
  $f_t = \sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m})$.
  Update $F^{(t)} = F^{(t-1)} + f_t$.
end for
Compute $z^{t+1} = \arg\min_z \sum_{i=1}^N L_\beta^{(t)}(r_{i,t}, z_i, \lambda_i^t)$.
Compute $\lambda^{t+1} = \arg\min_\lambda \sum_{i=1}^N L_\beta^{(t)}(r_{i,t}, z_i^{t+1}, \lambda_i)$
Output $F^{(T)}$.

Here the augmented Lagragian is given by

$$\begin{aligned} L_{\beta}^{(t)}(\gamma_i, z_i, \lambda_i) = & \ell(y_i, f_{t-1}(x_i) + \gamma_i) \\ & + r|z_i|_1 - \langle z_i - (f_{t-1}(x_i) + \gamma_i), \lambda_i \rangle \\ & + \frac{\beta}{2}|z_i - (f_{t-1}(x_i) + \gamma_i)|. \end{aligned}$$

And $f(\cdot)$ is a decision tree; $\ell(\cdot, \cdot)$ is the loss function.

This scheme is more suitable for the regularized cost function. In general, all the optimization methods for the optimizaion problem 2 can be tranlated to boosting algorithms. It is the first time to translate the augmented Lagragian method to boosting algorithms.

5 Surrogate Boost Decision Trees

The surrogate principle in continuous optimization palys a lead role in finding new targets $r_{i,t}$. The surrogate principle is to optimze a simple surrogate objective function at each iteration instead of the original complex objective function. For example, our aim is to solve an optimization problem of the following form

$$\arg\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}). \tag{3}$$

Given initial estimates $\boldsymbol{x}_0$ and surrogate objective function $Q(x_t, x)$, it is updated via

$$\boldsymbol{x}_{t+1} = \arg\min_{x} Q(x_t, x),$$

and it is usually $f(\boldsymbol{x}_{t+1}) \leq f(\boldsymbol{x}_t)$. Generally, it is required that $\lim_{t\to\infty} x_t = x^* \in \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$. The surrogate objective function $Q(x_t, x)$ is usually simpler to optimze than original function $f(\boldsymbol{x})$.

In this framework, we can extend the NGBoost to 'expectation maximization' type boosting algorithms.

A unifying principle: surrogate minimization

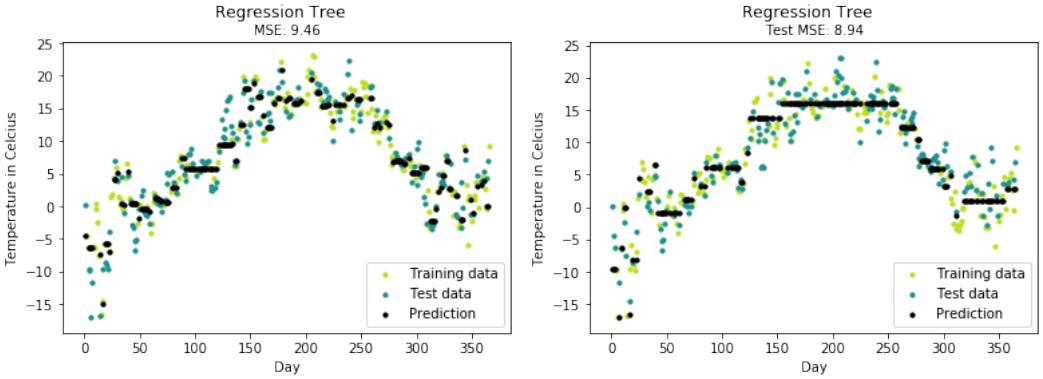
Algorithm 4 Surrogate Boost Decision Trees

Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
Initialize $f_0 = \arg\min_\gamma \sum_{i=1}^{N} L(x_i, \gamma)$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    Compute $r_{i,t} = \arg\min_x Q(f_t(x_i), F^{(t-1)}(x_i) + x)$.
  end for
  Fit a regression tree to the targets $r_{i,t}$ giving terminal regions

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

  for $j = 1, 2, \ldots, J_m$ **do**
    Compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, F^{(t-1)}(x_i) + \gamma)$.
  end for
  $f_t = \sum_{j=1}^{J_m} \gamma_{j,t} \mathbb{I}(x \in R_{j,m})$.
  Update $F^{(t)} = F^{(t-1)} + f_t$.
end for
Output $F^{(T)}$.

6 Experiments



7 Conclusion