Gradient Boosting & Accelerations

Réalisé par :
Redwan MEKRAMI

Travail présenté à
Gérard BIAU

Table des matières

1 Introduction

As Trevor Hastie correctly described it, Gradient Boosting is one of the best "Off-The-Shelf" algorithms in today's world of Machine Learning. Indeed, *XGBoost* is a technique that garners great success in competitions taking place on Kaggle and other similar platforms. However, it still remains a black box for a vast majority of its users. Gradient Boosting is a boosting technique, which consists in constructing a very accurate prediction rule by combining several relatively weak and imprecise rules. A surprisingly rich theory has developed around boosting, connecting to a broad range of topics including statistics, game theory, convex optimization, and information geometry.
In this paper, we will try to explain how Gradient Boosting works, and how we can accelerate its training phase using Optimization techniques. In order to do that, we will start off by going over basic optimization definitions, followed by a presentation of different gradient descent techniques that will, hopefully, let us achieve great results. Next, we will present the theory behind the Classic Gradient Boosting algorithm, and talk about some of its variants. Finally, we will conclude our research on this matter by implementing our algorithms and testing their performances on a given dataset.

2 Fast Recall

The acceleration methods that we will explain are based on the gradient method that we will see later. These methods are based on the notions of convexity and smoothness. Let's take a few minutes to remind the reader of some basic optimization notions.

2.1 Convex Functions

A convex function is a function defined on a convex domain such that, for any two points in the domain, the segment between the two points lies above the function curve between them. We restate these results more precisely :

Definition 1. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if dom(f) is a convex set and if $\forall x, y \in dom(f), \forall \theta \in [0, 1]$, we have :*

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y).$$

There are multiple ways to characterize a convex function, each of which may by convenient or insightful in different contexts. Below we explain the most commonly used criteriea.

Theorem 1. *Let f be a differentiable function. Then f is convex iff dom(f) is a convex set and :*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

Besides, if f is twice differentiable, it is equivalent to :

$$H(x) = \nabla^2 f(x) \succeq 0.$$

2.2 Smoothness

Definition 2. *If $f : \mathbb{R}^d \to \mathbb{R}$ is a convex L -smooth function, we remind that for all $x, x^{prime} \in \mathbb{R}^d$:*

$$\|\nabla f(x') - \nabla f(x)\| \leqslant L\|x' - x\|$$

Besides, if f is twice differentiable, it is equivalent to :

$$0 \preccurlyeq \nabla^2 f(x) \preccurlyeq LI$$

Theorem 2. *(Properties of smooth convex functions) : Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex L -smooth function. Then, we have the following inequalities :*

1. *Quadratic upper-bound :* $0 \leqslant f(x') - f(x) - \nabla f(x)^\top (x' - x) \leqslant \frac{L}{2}\|x' - x\|^2$

2. *Co-coercivity :* $\frac{1}{L}\left\|\nabla f(x') - \nabla f(x)\right\|^2 \leqslant \left[\nabla f(x') - \nabla f(x)\right]^\top (x' - x)$

3. *Lower bound :* $f(x') \geqslant f(x) + \nabla f(x)^\top (x' - x) + \frac{1}{2L}\left\|\nabla f(x') - \nabla f(x)\right\|^2$

4. *"Distance" to optimum :* $f(x) - f(x_*) \leqslant \nabla f(x)^\top (x - x_*)$

3 Gradient Descent and Accelerations

Now that we have introduced the notions of convexity and smoothness, we will use them to build optimization algorithms. The first algorithm and the most natural one is the vanilla gradient descent. Later on, we are going to see how we can improve this simple idea in order to accelerate the algorithm.

3.1 Vanilla Gradient Descent

Consider the unconstrained smooth convex optimization :

$$\min_x f(x) \tag{1}$$

where f is convex and differentiable with $\text{dom}(f) = \mathbb{R}^n$. Gradient descent method is an iterative method that takes a step along the negative gradient direction at each iteration. It produces a sequence of points $x^{(t)}$ with $t = 1,2,3,\cdots$. The sequence follows the following update rule :

$$x_{t+1} = x_t - \alpha_t \nabla f(x_{t+1}) \tag{2}$$

where $\alpha_t > 0$ is the step size chosen for the t-th iteration. This formulation is equivalent to minimizing a quadratic approximation of f at x_t. To see this, consider the following quadratic approximation of $f(x)$:

$$f(x) \approx f(x_t) + \nabla f(x_t)^\top (x - x_t) + \frac{1}{2\alpha_t}\|x - x_t\|_2^2 \tag{3}$$

To minimize the right-hand side term in Equation 3, we take the first order derivative w.r.t x and set it to zero :

$$\nabla f(x_t) + \frac{1}{\alpha_t}(x - x_t) = 0 \tag{4}$$

Clearly, this is equivalent to Equation 2.

Theorem 3. *Suppose the function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$,. Then if we run gradient descent for t iterations with a fixed step size $t \leq 1/L$, it will yield a solution $f^{(t)}$ which satisfies*

$$f(x_t) - f(x^*) \leq \frac{\|x_0 - x^*\|_2^2}{2\alpha_t}$$

where $f(x^*)$ is the optimal value. Intuitively, this means that gradient descent is guaranteed to converge and that it converges with rate $O(1/t)$.

3.2 Nesterov's acceleration

Nesterov's accelerated gradient descent method is a well-known method of accelerating the convergence rate of gradient descent, achieving an optimum convergence rate for first-order approaches for convex problems. Nesterov's method is an extrapolation method that defines carefully calibrated weight sequences for the so-called momentum term that updatse and iterates in the direction of the previous update.

Consider the problem of minimizing a function $f(x)$, Nesterov's accelerated gradient descent starts with an initial guess x_1. For $t \geq 1$, given x_t, a new iterate x_{t+1} is obtained by first adding a multiple of the *momentum* $x_t - x_{t-1}$ to x_t to obtain an auxiliary variable y_t, and then performing a gradient descent step at y_t. The update equations at iteration $t \geq 1$ are as follows :

$$y_t = x_t + \beta_t(x_t - x_{t-1}), \quad x_{t+1} = y_t - \alpha_t \nabla f(y_t), \tag{5}$$

where the gradient descent step length α_t and the momentum weight β_t are suitably chosen numbers, and $x_0 = x_1$ so that the first iteration is a simple gradient descent.
There are a number of ways to choose the α_t and β_t so that Nesterov's accelerated gradient descent converges at the optimal $O(1/t^2)$ in function value for smooth convex functions. For example, when $f(x)$ is a convex function with L-Lipschitz gradient, by choosing $\alpha_t = \frac{1}{L}$, and β_t as

$$\lambda_0 = 0, \quad \lambda_t = \frac{1 + \sqrt{1 + 4\lambda_{t-1}^2}}{2}, \quad \beta_t = \frac{\lambda_{t-1} - 1}{\lambda_t}. \tag{6}$$

Theorem 4. *Let f be L-smooth and convex, then the solution x_t from the Nesterov Accelerated Gradient Descent algorithm satisfies :*

$$f(x_t) - f(x_*) \leq \frac{2L \|x_1 - x_*\|_2^2}{t^2}$$

where x^* is a minimizer of f.

Démonstration. We begin by stating that :

$$f(x_{t+1}) - f(x) = f(x_{t+1}) - f(y_t) + f(y_t) - f(x)$$

Also, since $f(x_{t+1}) - f(y_t) \leq -\frac{1}{2L} \|\nabla f(y_t)\|_2^2 \quad (Lowerbound)$
and $f(y_t) - f(x) \geq \nabla f(y_t)^T (y_t - x) \quad (Theorem 1)$
we deduce :

$$f(x_{t+1}) - f(x) \leq -\frac{1}{2L} \|\nabla f(y_t)\|_2^2 + \nabla f(y_t)^T (y_t - x)$$

Using the fact that : $x_{t+1} = y_t - \frac{1}{L}\nabla f(y_t)$, we have :

$$f(x_{t+1}) - f(x) \leq -\frac{L}{2}\|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T(y_t - x) \tag{7}$$

We will then create two inequalities from (7), for $x = x_t$ and then $x = x_*$:

$$[f(x_{t+1}) - f(x_t)](\lambda_t - 1) \leq \left[-\frac{L}{2}\|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T(y_t - x_t)\right](\lambda_t - 1)$$

$$[f(x_{t+1}) - f(x_*)] \leq \left[-\frac{L}{2}\|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T(y_t - x_*)\right], \lambda_t \geq 1, \forall t.$$

By adding these two together, we have :

$$\lambda_t f(x_{t+1}) - (\lambda_t - 1) f(x_t) - f(x_*) \leq -\frac{\lambda_t L}{2}\|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T(\lambda_t y_t + (\lambda_t - 1)x_t - x_*)$$

Setting $\epsilon_t = f(x_t) - f(x_*)$, we obtain :

$$\lambda_t \epsilon_{t+1} - (\lambda_t - 1)\epsilon_t \leq -\frac{\lambda_t L}{2}\|x_{t+1} - y_t\|_2^2 - L(x_{t+1} - y_t)^T(\lambda_t y_t + (\lambda_t - 1)x_t - x_*)$$

Muliplying both sides by λ_t :

$$\lambda_t^2 \epsilon_{t+1} - \lambda_t(\lambda_t - 1)\epsilon_t \leq -\frac{L}{2}\left[\lambda_t^2\|x_{t+1} - y_t\|_2^2 + 2\lambda_t L(x_{t+1} - y_t)^T(\lambda_t y_t + (\lambda_t - 1)x_t - x_*)\right]$$

By definition of λ_t, we have $\lambda_{t+1}^2 - \lambda_{t+1} = \lambda_t^2$. Then :

$$\lambda_t^2 \epsilon_{t+1} - \lambda_{t-1}^2 \epsilon_t \leq -\frac{L}{2}\left(\|\lambda_t x_{t+1} - (\lambda_t - 1)x_t - x_*\|^2 - \|\lambda_t y_t - (\lambda_t - 1)x_t - x_*\|^2\right)$$

Therefore, using the definition of y_{t+1}, we then have :

$$\lambda_t x_{t+1} - (\lambda_t - 1)x_t - x_* = \lambda_{t+1} y_{t+1} - (\lambda_{t+1} - 1)x_{t+1} - x_*$$

Hence, defining $u_0 = x_0, u_t = \lambda_t y_t - (\lambda_t - 1)x_t - x_*, \forall t \geq 1$, the above equation simplifies to

$$\lambda_t^2 \epsilon_{t+1} - \lambda_{t-1}^2 \epsilon_t \leq -\frac{L}{2}\left(\|u_{t+1}\|^2 - \|u_t\|^2\right)$$

By induction, we have

$$\lambda_{t-1}^2 \epsilon_t \leq \frac{L}{2}\|u_0\|^2$$

From this, we realize the following :

$$\epsilon_t \leq \frac{L\|x_0\|^2}{2\lambda_{t-1}^2}$$

Again, by induction, one can show that :

$$\lambda_{t-1} \geq \frac{t}{2}, \forall t \geq 1$$

Finally :

$$\epsilon_t \leq \frac{2L \|x_0\|^2}{t^2}$$

□

One then obtains a $O(1/t^2)$ convergence rate for this algorithm, which justifies the use of the acceleration.

3.3 Other options for acceleration

We can also look at other options to accelerate the algorithm and make it more efficient. Here's two of the algorithms we stumbled upon during our research :

3.3.1 Nesterov'88 acceleration

Input : $\boldsymbol{x}_0 \in \mathbb{R}^d, \boldsymbol{y}_0 = \boldsymbol{x}_0$ For $t = 0, \cdots, N-1$

$$\begin{cases} x_{t+1} = y_t - \frac{1}{L}\nabla f(y_t) \\ y_{t+1} = x_{t+1} + \frac{t}{t+3}(x_{t+1} - x_t) \end{cases}$$

3.3.2 Algorithm OGM1 acceleration

Input : $x_0 \in \mathbb{R}^d, y_0 = x_0, \theta_0 = 1$. For $i = 0, \cdots, N-1$

$$\begin{cases} y_{i+1} = x_i - \frac{1}{L} f'(x_i) \\ \theta_{i+1} = \begin{cases} \frac{1+\sqrt{1+4\theta_i^2}}{2}, & i \leq N-2 \\ \frac{1+\sqrt{1+8\theta_i^2}}{2}, & i = N-1 \end{cases} \\ x_{i+1} = y_{i+1} + \frac{\theta_i - 1}{\theta_{i+1}}(y_{i+1} - y_i) + \frac{\theta_i}{\theta_{i+1}}(y_{i+1} - x_i) \end{cases}$$

In our practical part, we will also incorporate these two algorithms and look at their performances. Hopefully, they will produce some satisfying results.

4 Gradient boosting

4.1 Introduction

Boosting is one of the most powerful learning ideas introduced in the last twenty years. It was originally designed for classification problems but it can profitably be extended to regression as well. The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers to produce a powerful "committee."

Consider a two-class problem, with the output variable coded as $Y \in \{-1,1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1,1\}$. The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} 1_{(y_i \neq G(x_i))}$$

and the expected error rate on future predictions is $\mathrm{E}_{XY} 1_{(Y \neq G(X))}$. A weak classifier is one whose error rate is only slightly better than random guessing. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x), m = 1,2,\ldots,M$.
The predictions from all of them are then combined through a weighted majority vote to produce the final prediction :

$$G(x) = \text{sign}\left(\sum_{j=1}^{M} {}_j h_j(x)\right)$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm, and weight the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence. The data modifications at each boosting step consist of applying weights $w_1, w_2, \ldots, w_N$ to each of the training observations $(x_i, y_i), i = 1,2,\ldots,N$.
Initially all of the weights are set to $w_i = 1/N$, so that the first step simply trains the classifier on the data in the usual manner. For each successive iteration $m = 2,3,\ldots,M$, the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step m, those observations that were misclassified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly. Thus as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Each successive classifier is thereby forced.

4.2 Analysis of the learning model

To sum up the basic idea of Boosting is to convert a set of weak learners (i.e. classifiers that do better than random, but have high error probability) into a strong one by using

the weighted average of weak learners' opinions. More precisely, we consider the following function class

$$\mathcal{F} = \left\{ \sum_{j=1}^{M} \theta_j h_j(\cdot) : |\theta|_1 \leq 1, h_j : \mathcal{X} \mapsto [-1,1], j \in \{1,2,\ldots,M\} \text{ are classifiers} \right\}$$

and we want to upper bound $\mathcal{R}_n(\mathcal{F})$ for this choice of $\mathcal{F}$.
Let $(Z_i)_{\{1,\ldots,n\}} = (X_i, Y_i)_{\{1,\ldots,n\}}$, we have :

$$\mathcal{R}_n(\mathcal{F}) = \sup_{Z_1,\ldots,Z_n} \mathbb{E}\left[\sup_{f\in\mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^{n} \sigma_i Y_i f(X_i) \right| \right] = \frac{1}{n} \sup_{Z_1,\ldots,Z_n} \mathbb{E}\left[\sup_{|\theta|_1\leq 1} \left| \sum_{j=1}^{M} \theta_j \sum_{i=1}^{n} Y_i \sigma_i h_j(X_i) \right| \right]$$

Let $g(\theta) = \left| \sum_{j=1}^{M} \theta_j \sum_{i=1}^{n} Y_i \sigma_i h_j(X_i) \right|$, $g(\theta)$ is a convex function, thus $\sup_{|\theta|_1\leq 1} g(\theta)$ is achieved at a vertex of the unit ℓ_1 ball $\{\theta : \|\theta\|_1 \leq 1\}$. Define the finite set

$$B_{\mathbf{X},\mathbf{Y}} \triangleq \left\{ \pm \begin{pmatrix} Y_1 h_1(X_1) \\ Y_2 h_1(X_2) \\ \vdots \\ Y_n h_1(X_n) \end{pmatrix}, \pm \begin{pmatrix} Y_1 h_2(X_1) \\ Y_2 h_2(X_2) \\ \vdots \\ Y_n h_2(X_n) \end{pmatrix}, \ldots, \pm \begin{pmatrix} Y_1 h_M(X_1) \\ Y_2 h_M(X_2) \\ \vdots \\ Y_n h_M(X_n) \end{pmatrix} \right\}$$

There :

$$\mathcal{R}_n(\mathcal{F}) = \sup_{\mathbf{X},\mathbf{Y}} R_n(B_{\mathbf{X},\mathbf{Y}})$$

Notice $\max_{b\in B_{\mathbf{x},\mathbf{Y}}} |b|_2 \leq \sqrt{n}$ and $|B_{\mathbf{X},\mathbf{Y}}| = 2M$. Therefore, using the lemma :

Lemma 1. *For any $B \subset \mathbb{R}^n$, such that $|B| < \infty$:, it holds*

$$\mathcal{R}_n(B) = \mathbb{E}\left[\max_{b\in B} \left| \frac{1}{n} \sum_{i=1}^{n} \sigma_i b_i \right| \right] \leq \max_{b\in B} |b|_2 \frac{\sqrt{2\log(2|B|)}}{n}$$

where $|\cdot|2$ denotes the Euclidean norm.

we get

$$\mathcal{R}_n(B\mathbf{X}, \mathbf{Y}) \leq \left[\max_{b\in B_{\mathbf{X},\mathbf{Y}}} |b|_2 \right] \frac{\sqrt{2\log\left(2\left|B_{\mathbf{X},\mathbf{Y}}\right|\right)}}{n} \leq \sqrt{\frac{2\log(4M)}{n}}$$

Thus for Boosting,

$$R_\varphi(\hat{f}) \leq R_\varphi(\bar{f}) + 8L\sqrt{\frac{2\log(4M)}{n}} + 2L\sqrt{\frac{\log(2/\delta)}{2n}} \quad \text{with probability } 1-\delta$$

To get some ideas of what values L usually takes, consider the following examples :

(1) for hinge loss, i.e. $\varphi(x) = (1+x)_+, L = 1$.

(2) for exponential loss, i.e. $\varphi(x) = e^x, L = e$.

(3) for logistic loss, i.e. $\varphi(x) = \log_2(1+e^x), L = \frac{e}{1+e}\log_2(e) \approx 2.43$

Now we have bounded $R_\varphi(f) - R_\varphi(\bar{f})$, but this is not yet the excess risk. Excess risk is defined as $R(\hat{f}) - R(f^*)$, where $f^* = \operatorname{argmin}_f R_\varphi(f)$. The following theorem provides a bound for excess risk for Boosting.

Theorem 5. *Let* $\mathcal{F} = \left\{ \sum_{j=1}^M \theta_j h_j : \|\theta\|_1 \leq 1, h_j \text{ is a weak classifier} \right\}$ *and* φ *is an* L *Lipschitz convex surrogate. Define* $\hat{f} = \operatorname{argmin}_{f\in\mathcal{F}} R_{\varphi,n}(f)$ *and* $\hat{h} = \operatorname{sign}(\hat{f})$. *Then*

$$R(\hat{h}) - R^* \leq 2c\left(\inf_{f\in\mathcal{F}} R_\varphi(f) - R_\varphi(f^*)\right)^\gamma + 2c\left(8L\sqrt{\frac{2\log(4M)}{n}}\right)^\gamma + 2c\left(2L\sqrt{\frac{\log(2/\delta)}{2n}}\right)^2$$

with probability $1-\delta$

Démonstration.

$$\begin{aligned} R(\hat{h}) - R^* &\leq 2c\left(R_\varphi(f) - R_\varphi(f^*)\right)^\gamma \\ &\leq 2c\left(\inf_{f\in\mathcal{F}} R_\varphi(f) - R_\varphi(f^*) + 8L\sqrt{\frac{2\log(4M)}{n}} + 2L\sqrt{\frac{\log(2/\delta)}{2n}}\right)^\gamma \\ &\leq 2c\left(\inf_{f\in\mathcal{F}} R_\varphi(f) - R_\varphi(f^*)\right)^\gamma + 2c\left(8L\sqrt{\frac{2\log(4M)}{n}}\right)^\gamma + 2c\left(2L\sqrt{\frac{\log(2/\delta)}{2n}}\right)^\gamma \end{aligned}$$

Here the first inequality uses Zhang's lemma (see further) and the last one uses the fact that for $a_i \geq 0$ and $\gamma \in [0,1]$, $(a_1+a_2+a_3)^\gamma \leq a_1^\gamma + a_2^\gamma + a_3^\gamma$ □

Lemma 2. *Zhang's Lemma :*
Let :

$$\eta(X) = \mathbb{P}(Y = 1 \mid X)$$

and :

$$H_\eta(\alpha) = \eta(x)\varphi(-\alpha) + (1-\eta(x))\varphi(\alpha)$$

Let $\varphi : \mathbb{R} \mapsto \mathbb{R}_+$ *be a convex non-decreasing function such that* $\varphi(0) = 1$. *Define for any* $\eta \in [0,1]$

$$\tau(\eta) := \inf_{\alpha\in\mathbb{R}} H_\eta(\alpha)$$

If there exists $c > 0$ *and* $\gamma \in [0,1]$ *such that*

$$\left|\eta - \frac{1}{2}\right| \leq c(1 - \tau(\eta))^{\gamma}, \quad \forall \eta \in [0,1]$$

then

$$R(\text{sign}(f)) - R^* \leq 2c \left(R_\varphi(f) - R_\varphi^*\right)^{\gamma}$$

4.3 Gradient Boosting Algorithm

4.3.1 Gradient Boosting Machine

The original version of GBM presented in Algorithm 1, can be viewed as minimizing the loss function by applying an approximated steepest descent algorithm : In other words, the goal of GBM is to obtain a good estimate of the function f that approximately minimizes the empirical loss :

$$L^\star = \min_{f \in \text{lin}(\mathcal{B})} \left\{ L(f) := \sum_{i=1}^{n} \ell\left(y_i, f\left(x_i\right)\right) \right\}$$

GBM starts from a null function $f^0 \equiv 0$ and at each iteration computes the pseudo-residual r^m (namely, the negative gradient of the loss function with respect to the predictions so far f^m) :

$$r_i^m = -\frac{d\ell\left(y_i, f^m\left(x_i\right)\right)}{df^m\left(x_i\right)}$$

Then a weak-learner, that best fits the current pseudo-residual in terms of the least squares loss, is computed as follows :

$$\tau_m = \arg\min_{\tau \in \mathcal{T}} \sum_{i=1}^{n} \left(r_i^m - b_\tau\left(x_i\right)\right)^2$$

This weak-learner is added to the model with a coefficient found via a line search. As the iterations progress, GBM leads to a sequence of functions $\{f^m\}_{m \in [M]}$ (where $[M]$ is a shorthand for the set $\{1, \ldots, M\}$). The usual intention of GBM is to stop early - before one is close to a minimum of Problem (2) - with the hope that such a model will lead to a good predictive performance.

Algorithm 1 Gradient Boosting Machine (GBM)

Initialization. Initialize with $f^0(x) = 0$.
For $m = 0, \ldots, M-1$ do :
Perform Updates :
(1) Compute pseudo residual : $r^m = -\left[\frac{\partial \ell(y_i, f^m(x_i))}{\partial f^m(x_i)}\right]_{i=1,\ldots,n}$.
(2) Find the parameters of the best weak-learner : $\tau_m = \arg\min_{\tau \in \mathcal{T}} \sum_{i=1}^{n} \left(r_i^m - b_\tau\left(x_i\right)\right)^2$
(3) Choose the step-size η_m by line-search : $\eta_m = \arg\min_\eta \sum_{i=1}^{n} \ell\left(y_i, f^m\left(x_i\right) + \eta b_{\tau_m}\left(x_i\right)\right)$
(4) Update the model $f^{m+1}(x) = f^m(x) + \eta_m b_{\tau_m}(x)$.

Output. $f^M(x)$.

4.3.2 Acceleration of Gradient Boosting

For optimizing a smooth convex function, we showed that the standard gradient descent (GD) algorithm can be made much faster, resulting in the *accelerated* gradient descent method. While GD requires $O(1/\varepsilon)$ iterations, accelerated gradient methods only require $O(1/\sqrt{\varepsilon})$.

4.4 The AGB algorithm

Algorithm 2

1: **Require** $T \geq 1$ (number of iterations), $k \geq 1$ (number of terminal nodes in the trees), $0 < \nu < 1$ (shrinkage parameter).

2: **Initialize** $F_0 = G_0 = \arg\min_z \sum_{i=1}^n \psi(z,Y_i)$, $\lambda_0 = 0$, $\gamma_0 = 1$.

3: **for** $t = 0$ to $(T-1)$ **do**

4: For $i = 1, \ldots, n$, **compute** the negative gradient instances

$$Z_{i,t+1} = -\nabla C_n(G_t)(X_i).$$

5: **Fit** a regression tree to the pairs $(X_i,Z_{i,t+1})$, giving terminal nodes $R_{j,t+1}$, $1 \leq j \leq k$.

6: For $j = 1, \ldots, k$, **compute**

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,t+1}} \psi(G_t(X_i) + w, Y_i).$$

7: **Update**

1. $F_{t+1} = G_t + \nu \sum_{j=1}^k w_{j,t+1} 1_{R_{j,t+1}}$.
2. $G_{t+1} = (1-\gamma_t)F_{t+1} + \gamma_t F_t$.
3. $\lambda_t = \frac{1+\sqrt{1+4\lambda_{t-1}^2}}{2}$, $\lambda_{t+1} = \frac{1+\sqrt{1+4\lambda_t^2}}{2}$.
4. $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}$.

8: **end for**

9: **Output** F_T.

We see that in the algorithm above, $(F_t)_t$ and $(G_t)_t$ represent the sequences $(x_t)_t$ and $(y_t)_t$ of Nesterov's acceleration scheme.

$(G_t)_t$ ensures the acceleration and the algorithm outputs F_T after T iterations. Line (2) ini-

tializes to the optimal constant model.
Every time the algorithm is in Step (6), it will select a tree that ensures that the squared-error is minimal, and then perform an update of G_t. The only thing different with the "Classic" Gradient Boosting algorithm is that the $(G_t)_t$ functional will provide the much-needed momentum to $(F_t)_t$ so that our output get closer than usual to the minimum of our loss function.

Let $f_{t+1} = \sum_{j=1}^{k} \beta_{j,t+1} 1_{R_{j,+1}}$ be the approximate-gradient tree output at line 6 of the algorithm. The next logical step is to perform a line search to find the step size and update the model accordingly, as follows :

$$w_{t+1} \in \arg\min_{w>0} \sum_{i=1}^{n} \psi\left(G_t\left(X_i\right) + w f_{t+1}\left(X_i\right), Y_i\right), \quad F_{t+1} = G_t + w_{t+1} f_{t+1}$$

However, if we were to choose Friedman's gradient tree boosting, we would have to choose a separate optimal value $w_{j,t+1}$ for each of the tree's regions, instead of a single w_{t+1} for the whole tree. The coefficients $\beta_{j,t+1}$ from the tree-fitting procedure can be then simply discarded, and the model update rule at epoch t becomes, for each $j = 1, \ldots, k$,

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,+1}} \psi\left(G_t\left(X_i\right) + w, Y_i\right), \quad F_{t+1} = G_t + v \sum_{j=1}^{k} w_{j,t+1} 1_{R_{j,t+1}}$$

which corresponds to lines 6 and $7(a)$ of our algorithm.
We also note that the contribution of the approximate gradient is scaled by a factor $0 < v < 1$ when it is added to the current approximation. The parameter v can be regarded as controlling the learning rate of the boosting procedure. The important thing to know is that a robust model can only be built if we are able to find a good trade-off between the number of trees T and v. All in all, both v and T control prediction risk on the training data but these parameters do not operate independently !

4.5 Acceleration with Nesterov 88 Method

We propose here an other acceleration method base on the Nesterov 88 Method. This Algorithm is the twin of the AGB algorithm except that we modify γ_t according to the acceleration of the Nesterov 88 Method.

Algorithm 3

1: **Require** $T \geq 1$ (number of iterations), $k \geq 1$ (number of terminal nodes in the trees), $0 < \nu < 1$ (shrinkage parameter).

2: **Initialize** $F_0 = G_0 = \arg\min_z \sum_{i=1}^n \psi(z,Y_i)$, $\lambda_0 = 0$, $\gamma_0 = 1$.

3: **for** $t = 0$ to $(T-1)$ **do**

4: For $i = 1, \ldots, n$, **compute** the negative gradient instances

$$Z_{i,t+1} = -\nabla C_n(G_t)(X_i).$$

5: **Fit** a regression tree to the pairs $(X_i,Z_{i,t+1})$, giving terminal nodes $R_{j,t+1}$, $1 \leq j \leq k$.

6: For $j = 1, \ldots, k$, **compute**

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,t+1}} \psi(G_t(X_i) + w,Y_i).$$

7: **Update**

1. $F_{t+1} = G_t + \nu \sum_{j=1}^k w_{j,t+1} 1_{R_{j,t+1}}$.
2. $G_{t+1} = (1-\gamma_t)F_{t+1} + \gamma_t F_t$.
3. $\gamma_t = \frac{t}{t+3}$.

8: **end for**

9: **Output** F_T.

4.6 Acceleration with optimized gradient method

Another modification which can be interesting is to introduce the coefficients θ_t according to the OGM1 Algorithm and to modify the iterations of $(G_t)_t$.

Algorithm 4

1: **Require** $T \geq 1$ (number of iterations), $k \geq 1$ (number of terminal nodes in the trees), $0 < \nu < 1$ (shrinkage parameter).

2: **Initialize** $F_0 = G_0 = \arg\min_z \sum_{i=1}^n \psi(z,Y_i)$, $\lambda_0 = 0$, $\gamma_0 = 1$.

3: **for** $t = 0$ to $(T-1)$ **do**

4: For $i = 1, \ldots, n$, **compute** the negative gradient instances

$$Z_{i,t+1} = -\nabla C_n(G_t)(X_i).$$

5: **Fit** a regression tree to the pairs $(X_i,Z_{i,t+1})$, giving terminal nodes $R_{j,t+1}$, $1 \leq j \leq k$.

6: For $j = 1, \ldots, k$, **compute**

$$w_{j,t+1} \in \arg\min_{w>0} \sum_{X_i \in R_{j,t+1}} \psi(G_t(X_i) + w, Y_i).$$

7: **Update**

1. $F_{t+1} = G_t + \nu \sum_{j=1}^k w_{j,t+1} 1_{R_{j,t+1}}$.
2. $G_{t+1} = F_{t+1} + (\theta_t - 1)(F_{t+1} - F_t) + (\theta_t - 1)(F_{t+1} - G_t)$.
3. $\theta_{t+1} = \begin{cases} \frac{1+\sqrt{1+4\theta_t^2}}{2}, & i \leq N-2 \\ \frac{1+\sqrt{1+8\theta_t^2}}{2}, & i = N-1 \end{cases}$

.

8: **end for**

9: **Output** F_T.

5 Numerical studies

5.1 Quick description of the Dataset

To measure the performance and the quality of our models, we will test them on the "Communities and Crime Data Set". The data combines socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR. It is a dataset which connects each city to its crime ratio through different features such as its population, the average rent in the city etc.

5.2 Simulations

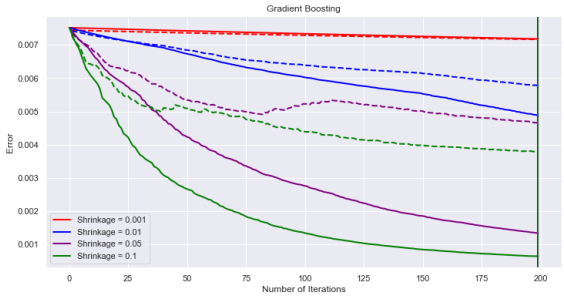

FIGURE 1 – Gradient Boosting

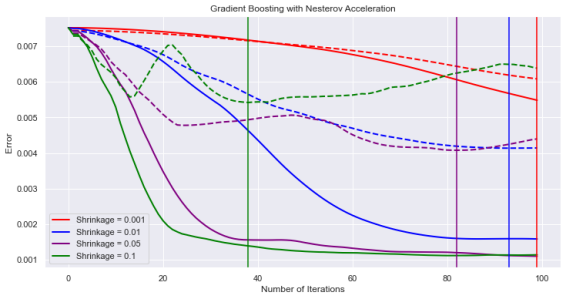

FIGURE 2 – Nesterov Acceleration

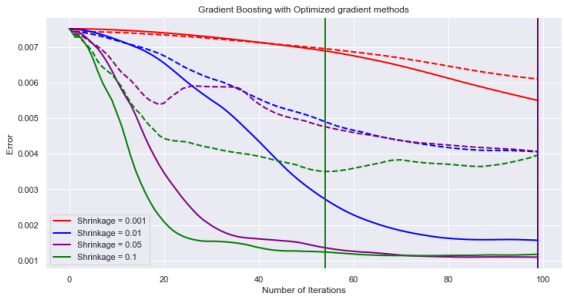

FIGURE 3 – Nesterov'88 Acceleration

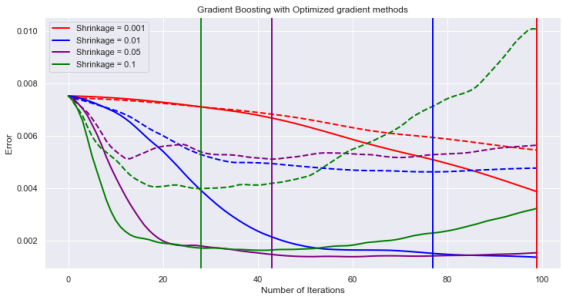

FIGURE 4 – OGM Gradient Boosting

In our experiments, all algorithms use CART trees as the weak learners and least squares loss. The Figures above show the training loss varying with the number of trees with a fixed step-size $\nu \in \{0.001, 0.01, 0.05, 0.1\}$ for both training loss and validation loss. As we can see, when ν is large, the validation loss for Gradient Boosting and Nesterov'88 Acceleration converges faster whereas for AGB and OGM Gradient Boosting, it may diverge. When ν gets smaller, the training loss for OGM Gradient Boosting may decay faster than AGB and Nesterov'88 Acceleration. For the Nesterov acceleration and his variant, the Nesterov 88, there are only a few differences when ν varies. However, Nesterov 88 seems to achieve a lower validation error when ν is large. Finally, accelerated boosting methods depends a lot on the parameter ν but always decreases the training error faster than gradient boosting but ν must be well chosen in order to avoid overfitting.

6 Further developments

Quasi-Newton methods are methods used to either find zeroes or local maxima and minima of functions, as an alternative to Newton's method. They can be used if the Jacobian or Hessian is unavailable or is too expensive to compute at every iteration. The "full" Newton's method requires the Jacobian in order to search for zeros, or the Hessian to find extrema.

6.1 The Barzilai-Borwein Step Size

The BB method, proposed by Barzilai and Borwein has been proved to be very successful in solving nonlinear optimization problems. The key idea behind the BB method is motivated by quasi-Newton methods. Suppose we want to solve the unconstrained minimization problem (1) where f is differentiable. A typical iteration of quasi-Newton methods takes the following form :

$$x_{t+1} = x_t - B_t^{-1}\nabla f(x_t), \tag{8}$$

where B_t is an approximation of the Hessian matrix of f at the current iterate x_t. Different choices of B_t give different quasi-Newton methods. The most important feature of B_t is that it must satisfy the so-called secant equation :

$$B_t s_t = y_t, \tag{9}$$

where $s_t = x_t - x_{t-1}$ and $y_t = \nabla f(x_t) - \nabla f(x_{t-1})$ for $t \geq 1$. It is noted that in (8) one needs to solve a linear system, which may be time consuming when B_t is large and dense.

One way to alleviate this burden is to use the BB method, which replaces B_t by a scalar matrix $\frac{1}{\eta_t}I$. However, one cannot choose a scalar η_t such that the secant equation :(9) holds with $B_t = \frac{1}{\eta_t}I$. Instead, one can find η_t such that the residual of the secant equation is minimized, i.e.,

$$\min_{\eta_t} \left\| \frac{1}{\eta_t} s_t - y_t \right\|_2^2,$$

which leads to the following choice of η_t : $\eta_t = \frac{\|s_t\|_2^2}{s_t^T y_t}$.

Therefore, a typical iteration of the BB method for solving 1 is $x_{t+1} = x_t - \eta_t \nabla f(x_t)$.

6.2 Newton boosting

Remember that the goal of boosting is to find a minimizer F^* of the risk $R(F)$ which is defined as the expected loss

$$R(F) = E_{Y,X}(L(Y, F(X)))$$

where $F(\cdot)$ is a function in a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$ given by

$$\langle F, F \rangle = E_X \left(F(X)^2 \right)$$

For Newton boosting, we assume that $R(F)$ is two times Gâteau differentiable and denote the second Gâteau derivative by

$$d^2R(F,f) = \frac{d^2}{d\epsilon^2}R(F+\epsilon f)\Big|_{\epsilon=0}, \quad F, f \in \Omega_{\mathcal{S}}$$

Newton boosting chooses f_m as the minimizer of a second-order Taylor approximation around F_{m-1} :

$$f_m = \underset{f\in\mathcal{S}}{\operatorname{argmin}} R\left(F_{m-1}\right) + dR\left(F_{m-1}, f\right) + \frac{1}{2}d^2R\left(F_{m-1}, f\right) \tag{10}$$

If we assume the P-almost all existence and integrability of the second derivative of $L(Y,F)$ with respect to F, then (10) can be written as

$$\begin{aligned} f_m &= \underset{f\in\mathcal{S}}{\operatorname{argmin}} E_{Y,X}\left(g_m(Y,X)f(X) + \frac{1}{2}h_m(Y,X)f(X)^2\right) \\ &= \underset{f\in\mathcal{S}}{\operatorname{argmin}} E_{Y,X}\left(h_m(Y,X)\left(-\frac{g_m(Y,X)}{h_m(Y,X)} - f(X)\right)^2\right) \end{aligned} \tag{11}$$

where the gradient $g_m(Y,X)$ is defined as

$$g_m(Y,X) = \frac{\partial L(Y,F)}{\partial F}\Big|_{F=F_{m-1}(X)}$$

and $h_m(Y,X)$ is the second derivative of $L(Y,F)$ with respect to F at F_{m-1} :

$$h_m(Y,X) = \frac{\partial^2 L(Y,F)}{\partial F^2}\Big|_{F=F_{m-1}(X)}$$

The last line in Equation (11) shows that f_m is the weighted L^2 approximation to negative ratio of the gradient over the Hessian $-\frac{g_m(Y,X)}{h_m(Y,X)}$ and the weights corresponds to the second derivative $h_m(Y,X)$

If the following expression is well defined for P-almost all X, we can again calculate the pointwise minimizer of (9) as :

$$f_m(X) = \underset{f\in\mathcal{S}}{\operatorname{argmin}} E_{Y|X}\left(h_m(Y,X)\left(-\frac{g_m(Y,X)}{h_m(Y,X)} - f(X)\right)^2\right)$$

Note that gradient boosting can be seen as a special case of Newton boosting.

7 References

Accelerated gradient boosting, Gerard Biau, Benoît Cadre and Laurent Rouviere,

Boosting Foundations and Algorithms, Robert E. , Schapire Yoav Freund. .

Decaying momentum helps neural network training, John Chen and Anastasios Kyrillidis

On empirical comparisons of optimizers for deep learning, Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl

Accelerating Gradient Boosting Machine Haihao Lu, Sai Praneeth Karimireddy, Natalia Ponomareva, Vahab Mirrokni

Gradient descent with momentum — to accelerate or to super-accelerate ? Goran Nakerst, John Brennan and Masudul Haque

The Elements of Statistical Learning Trevor Hastie, Robert Tibshirani and Jerome Friedman, Springer Series in Statistics,

On the convergence analysis of the optimized gradient method, Donghwan Kim, Jeffrey A.Fessler

Accelerated proximal boosting, Erwan Fouillen, Claire Boyer and Maxime Sangnier

Acceleration in Optimization. Optimization and Control, Damien Scieur

Gradient and Newton Boosting for Classification and Regression, Fabio Sigrist