Boosted Neural Networks

Alexandre A. Damião

December 1st, 2018

Introduction

In this report I'll explain an implementation of a gradient boosting algorithm where the weak learner of interest is a neural network. This is more of an academic exercise than anything else. I will be using the "Iris Flower Dataset" as the data analyzed by this method. This machine learning technique will be used primarily for classification. In this small report I'll be doing only the derivation of the boosted side of things and later showing a pseudocode for the algorithm itself.

Computation

Suppose we are given a training set $\Omega = \{(x_i, y_i)\}_{i \in I}$ where $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}^m$. A cost function $C : \Omega \to \mathbb{R}$ (for a chosen differentiable, convex function L), based on the predictor $f : \mathbb{R}^n \to \mathbb{R}^m$, is given by:

$$C = \sum_{i \in I} L(y_i, f(x_i)) \ , \tag{1}$$

where:

$$\begin{aligned} L : \mathbb{R}^m \times \mathbb{R}^m &\to \mathbb{R} \\ (y, f(x)) &\mapsto L(y, f(x)) \end{aligned}$$

is a convex function with continuous derivatives. For this report we will be using the well known quadratic form since it simplifies a lot of the calculations:

$$L(y, f(x)) = (y - f(x))^T (y - f(x)) \equiv (y - f)^T (y - f)$$

The idea of the method is to keep updating the predictor by means of a weak learner. The weak learner are usually small decision trees (*XGBoost* for example). However, here we will implement a learner based upon neural networks with one hidden layer.

Suppose we are at the step m of our iterations. This means that we have determined the predictor $f_{m-1}(x)$ and now we want to calculate $f_m(x)$. The final predictor will be of the form:

$$f(x) = \gamma_0 f_0(x) + \gamma_1 f_1(x) + \cdots + \gamma_k f_k(x) \tag{2}$$

where $\gamma_i \in \mathbb{R}$.

Let's now describe the method itself. The first step will be to fit a neural net f_0 to the training points of Ω. This is done just as one would do for a regular neural network. Once we obtain f_0 we will want to determine γ_0. To do so we must find a quantity γ_0 such that we minimize the following cost:

$$C = \sum_i L(y_i, \gamma_0 f_0(x_i)) = \sum_i (y_i - \gamma_0 f_0(x_i))^T (y_i - \gamma_0 f_0(x_i))$$

$$= \sum_i y_i^T y_i - 2\gamma_0 y_i^T f_0(x_i) + \gamma_0^2 f_0(x_i)^T f_0(x_i)$$

Taking the derivative of the cost function with respect to γ_0 we obtain:

$$\frac{dC}{d\gamma_0} = -2\sum_i y_i^T f_0(x_i) + 2\gamma_0 \sum_i f_0(x_i)^T f_0(x_i)$$

and by making $dC/d\gamma_0$ equal to zero we determine that:

$$\gamma_0 = \frac{\sum_i y_i^T f_0(x_i)}{\sum_i f_0(x_i)^T f_0(x_i)} \tag{3}$$

One quick remark: we know that this choice for γ_0 will minimize the cost function because the second derivative of the cost function with respect to γ_0 is strictly positive, since we established that L is a convex function. Therefore we know that γ_0 given by Eq. (3) will minimize the cost function.

For the next step we have to increment our current predictor $\gamma_0 f_0(x)$ by $h(x)$ in such a way that the cost function C decreases as much as spossible. So consider the following:

$$C = \sum_i L(y_i, f_{m-1}(x_i) + h(x_i)) \tag{4}$$

where f_{m-1} can be thought of being the most current predictor. Let's expand L in a Taylor series up to second order with respect to the small variations $h(x_i)$:

$$L(y, f + h) \approx L(y, f) + \nabla L^T h + \frac{1}{2} h^T \nabla^2 L\, h \tag{5}$$

where:

$$\nabla L = \begin{bmatrix} \partial_{f_1} L \\ \partial_{f_2} L \\ \vdots \\ \partial_{f_m} L \end{bmatrix} \quad \text{and} \quad \nabla^2 L = \begin{bmatrix} \partial^2_{f_1 f_1} L & \dots & \partial^2_{f_1 f_m} L \\ \partial^2_{f_2 f_1} L & \dots & \partial^2_{f_2 f_m} L \\ \vdots & \ddots & \vdots \\ \partial^2_{f_m f_1} L & \dots & \partial^2_{f_m f_m} L \end{bmatrix} \tag{6}$$

This yields the following:

$$C = C_{m-1} + \Big(\sum_i \nabla L_i^T h_i + \frac{1}{2} h_i^T \nabla^2 L_i h_i \Big) \tag{7}$$

where the notation used here is such that $\nabla L_i \equiv \nabla L(y_i, f(x_i))$. It is interesting to note that, by writing $\Delta C = \sum_i \nabla L_i^T h_i + \frac{1}{2} h_i^T \nabla^2 L_i h_i$ and taking the gradient with respect to h_i, we obtain:

$$\nabla(\Delta C) = \frac{\partial(\Delta C)}{\partial h_i} = \sum_i (\nabla L_i + \nabla^2 L_i \, h_i)$$

Therefore, by making $\nabla(\Delta C) = 0$ we must enforce that h_i be equal to:

$$-(\nabla^2 L_i)^{-1} \nabla L_i$$

Another remark is that, since the laplacian of C with respect to h_i is strictly positive:

$$\nabla^2(\Delta C) = \sum_i \nabla^2 L_i > 0$$

then the choice of $h_i = -(\nabla^2 L_i)^{-1}\nabla L_i$ will minimize the cost function C. Keep this remark in mind, it is quite important. The increment we will be adding to our predictor will be such that:

$$h \text{ maps } x_i \text{ to } -(\nabla^2 L_i)^{-1}\nabla L_i$$

So we fit a neural net from x_i to $-(\nabla^2 L_i)^{-1}\nabla L_i$. For our nice quadratic function L this is the same as:

$$h \text{ maps } x_i \text{ to } y_i - f_{m-1}(x_i)$$

We finally pick a new constant γ which will help adjust this prediction. In other words, we have found the predictor up to step $m-1$. We have to determine the new direction to proceed by fitting our neural network to $(y_i - f_{m-1}(x_i))$.

$$f_{m-1}(x) = \sum_{i=0}^{m-1} \gamma_i h_i(x)$$

and the cost function we want to minimize is:

$$C = \sum_i L(y_i, f_{m-1}(x_i) + \gamma_m h(x_i))$$

Taking the derivative with respect to γ_m yields the following:

$$\frac{dC}{d\gamma_m} = -2\sum_i \Big(y_i - f_{m-1}(x_i)\Big)^T h(x_i) + 2\gamma_m \sum_i h^T(x_i)h_(x_i)$$

which by making it equal to zero gives us:

$$\gamma_m = \frac{\sum_i \Big(y_i - f_{m-1}(x_i)\Big)^T h(x_i)}{\sum_i h^T(x_i) h(x_i)}$$

and we finally update our predictor:

$$f_m(x) = \gamma_0 h_0(x) + \gamma_1 h_1(x) + \cdots + \gamma_m h_m(x)$$

Algorithm - Pseudocode

I've broken down the whole code into two parts: initialization and loop. They are very similar but I thought it was worth having both pieces here just to make sure it is clear. The second part should be repeated until the desired number of weak learners is achieved.

Algorithm 1 Initialization

```
1: Fit the training data $\Omega$ with a neural network to determine $h_0(x)$.
2: Calculate $\Delta_i = y_i - h_0(x_i)$ and determine:
       $$\gamma_0 = \frac{\sum_i \Delta_i^T h_0(x_i)}{\sum_i h_0^T(x_i) h_0(x_i)}$$
3: Update the predictor: $f_0(x) = \gamma_0 h_0(x)$
4: Fit a neural network to the training data $\{(x_i, \Delta_i)\}_{i \in I}$ to determine $h_1(x)$.
5: Calculate $\Delta_i = y_i - f_0(x_i)$ and determine:
       $$\gamma_1 = \frac{\sum_i \Delta_i^T h_1(x_i)}{\sum_i h_1^T(x_i) h_1(x_i)}$$
6: Update the predictor: $f_1(x) = \gamma_0 h_0(x) + \gamma_1 h_1(x)$
```

Algorithm 2

```
1: Fit a neural network to the data $\{(x_i, y_i - f_{j-1}(x_i))\}_{i \in I}$ to determine
   $h_j(x)$.
2: Determine:
       $$\gamma_j = \frac{\sum_i \Delta_i^T h_j(x_i)}{\sum_i h_j^T(x_i) h_j(x_i)} \quad \text{where} \quad \Delta_i = y_i - f_{j-1}(x_i)$$
3: Update the predictor: $f_j(x) = \sum_{k=0}^{j} \gamma_k h_k(x)$
```