Lecture 25: Boosting and Gradients

Big Data and Machine Learning for Applied Economics
Econ 4676

Ignacio Sarmiento-Barbieri

Universidad de los Andes

November 10, 2020

Agenda

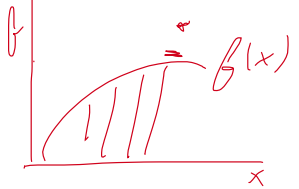
Causal Trees for HTE

- Problem: we never observe t_i unlike prediction that we observe Y_i
- Causal Trees search for leaves with
  - HTE across leaves
  - precisely-estimated leaf effects
- Key is the honest Criterion
- Work well with RCTs

Boosting: Motivation

- Boosting is one of the most powerful learning ideas introduced in the last twenty years.
- It was originally designed for classification problems, but can 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 ensemble, "committee."
- The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.
- Bagging and random forests a committee of trees each cast a vote for the predicted class.
- Boosting is like as a committee method as well, although unlike random forests, the committee of weak learners evolves over time, and the members cast a weighted vote.

Detour: Gradient-Based Optimization



- Most learning algorithms, especially learning, involve optimization of some sort.
- Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x
- We usually phrase most optimization problems in terms of minimizing $f(x)$.
- Maximization may be accomplished via a minimization algorithm by minimizing $-f(x)$.

Goodfellow → Deep Learning Ch 4

Detour: Gradient-Based Optimization

- Suppose we have a function $y = f(x)$, where both x and y are real numbers.
- The derivative $f'(x)$ gives the slope of $f(x)$ at the point x
- It specifies how to scale a small change in the input to obtain the corresponding change in the output

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{1}$$

- The derivative is therefore useful for minimizing a function because it tells us how to change x in order to make a small improvement in y
- We can thus reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative.
- This technique is called gradient descent (Cauchy, 1847).

Detour: Gradient-Based Optimization

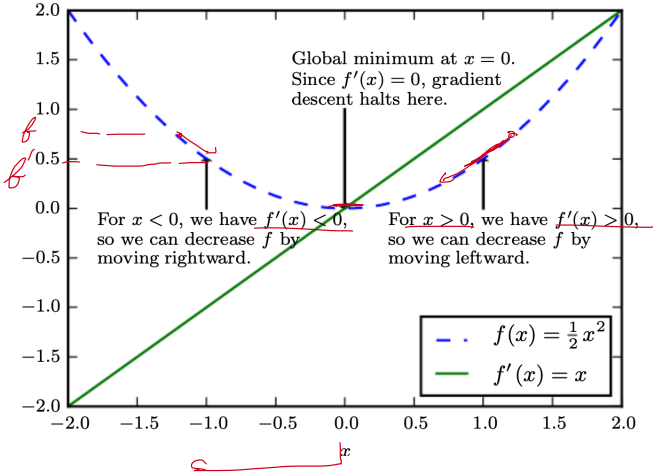


Figure 4.1: Gradient descent. An illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.

Detour: Gradient-Based Optimization

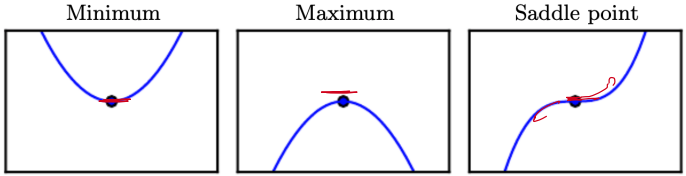


Figure 4.2: Types of critical points. Examples of the three types of critical points in one dimension. A critical point is a point with zero slope. Such a point can either be a local minimum, which is lower than the neighboring points; a local maximum, which is higher than the neighboring points; or a saddle point, which has neighbors that are both higher and lower than the point itself.

Detour: Gradient-Based Optimization

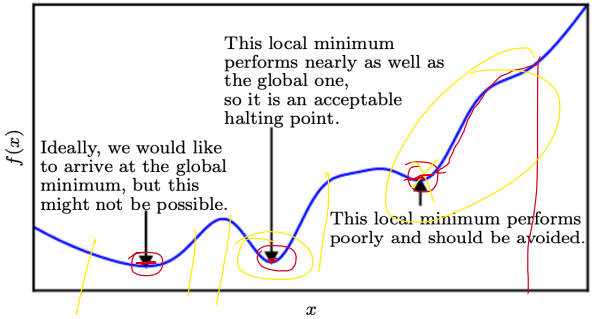


Figure 4.3: Approximate minimization. Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. In the context of deep learning, we generally accept such solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

Detour: Gradient-Based Optimization

- The directional derivative in direction $\mathbf{u}$ (a unit vector) is the slope of the function f in direction $\mathbf{u}$.
- In other words, the directional derivative is the derivative of

$$f(x + \alpha u) \tag{2}$$

- with respect to α, evaluated at $\alpha = 0$

$$\frac{\partial}{\partial \alpha} = \mathbf{u}' \nabla_x f(\mathbf{u}) \tag{3}$$

Detour: Gradient-Based Optimization

- To minimize f, we would like to find the direction in which f decreases the fastest.
- We can use the directional derivative:

$$\min_{u,u'u=1} \mathbf{u}'\nabla_x f(\mathbf{u}) \tag{4}$$

$$\min_{u,u'u=1} ||\mathbf{u}||_2 ||\nabla_x f(\mathbf{u})||_2 \cos\theta \tag{5}$$

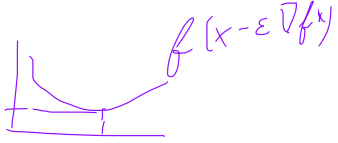
- Where θ is the angle between $\mathbf{u}$ and the gradient
- Substituting $u'u = 1$

$$\min_{u,u'u=1} \cos\theta \tag{6}$$

- This is minimized when $\mathbf{u}$ points in the opposite direction as the gradient.
- In other words, the gradient points directly uphill, and the negative gradient points directly downhill.
- We can decrease f by moving in the direction of the negative gradient.

Detour: Gradient-Based Optimization

- Steepest descent proposes a new point



$$x' = x - \epsilon \nabla_x f(x) \tag{7}$$

- where ϵ is the learning rate, a positive scalar determining the size of the step.
- We can choose ϵ in several different ways:
  - A popular approach is to set ϵ to a small constant.
  - Sometimes, we can solve for the step size that makes the directional derivative vanish.
  - Another approach is to evaluate $f(x - \epsilon \nabla_x f(x))$ for several values of ϵ and choose the one that results in the smallest objective function value. This is called a line search.
- Steepest descent converges when every element of the gradient is zero (or, in practice, very close to zero).
- In some cases, we may be able to avoid running this iterative algorithm and just jump directly to the critical point by solving the equation $\nabla_x f(x) = 0$ for x.

Boosting

- The goal here is to solve something which looks like

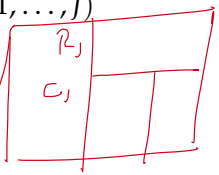
$$f^\star = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) \right\} \tag{8}$$

- for some loss function L, and for some set of predictors $\mathcal{F}$.
- This is an optimization problem.
- Note that here in a function space, so we are solving for a function not a point.

Boosting Trees

- In the case of trees, a constant is assigned to each region ($j = 1, \ldots, J$)
- The predictive rule is

$$x \in R_j \implies f(x) = c_j \tag{9}$$



- A tree can be formally expressed as

$$T(x; \Omega) = \sum_{j=1}^{K} c_j I(x \in R_j) \tag{10}$$

- with $\Omega = \{R_j, c_j\}_1^J$.
- A Boosted tree model is a sum of trees

$$f_M(x) = \sum_{m=1}^{M} T(x, \Omega_m) \tag{11}$$

Boosting Trees

Numerical Optimization via Gradient Boosting

- The objective is

$$f^{\star} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) \right\} \tag{12}$$

- where $f(\mathbf{x}_i)(x) = \sum_{m=1}^{M} T(x, \Omega_m)$
- Numerical optimization procedures solve $f^{\star}$ as a sum of component vectors

$$f_M^{\star} = \sum_{m=0}^{M} h_m \tag{13}$$

- $h_m \in \mathbb{R}^N$ where $f_0 = h_0$ is an initial guess and each successive f_m is induced based on the current parameter vector f_{m-1}, which is the sum of the previously induced updates.
- Numerical optimization methods differ in their prescriptions for computing each increment vector h_m ("step").

Boosting Trees

Implementations of Gradient Boosting

- Gradient Tree Boosting Algorithm
  1. Initialize $f_0(x) = \underset{c}{argmin} \sum_{i=1}^{N} L(y_i, c)$
  2. for $m = 1$ to M:
     1. For $i = 1, 2, \ldots N$ compute

$$r_{im} = - \left. \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right|_{f(\mathbf{x}_i) = f^{(m-1)}(\mathbf{x}_i)} \tag{14}$$

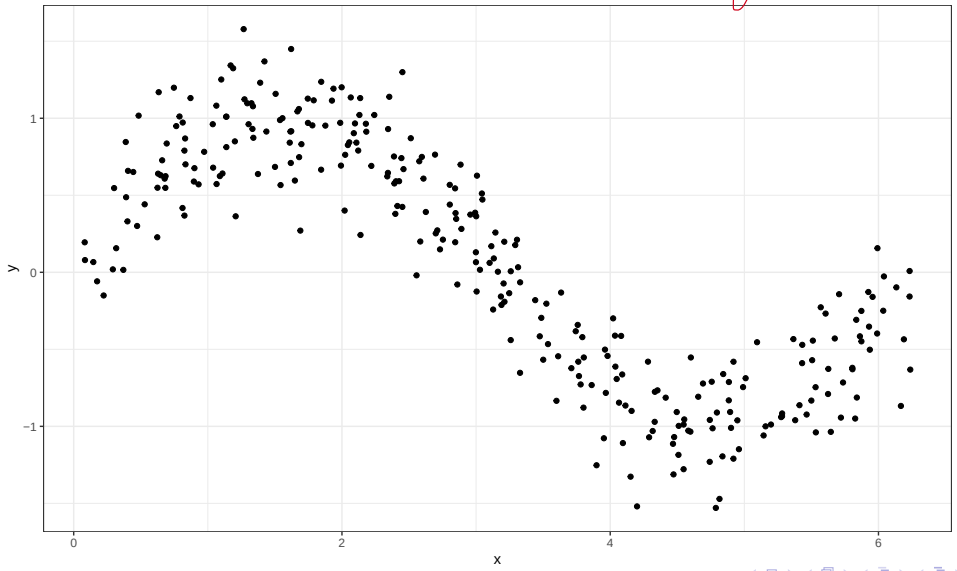
     2. Fit a regression tree to the targets r_{im} giving terminal regions R_{jm} $j = 1, 2, \ldots J_m$
     3. For $j = 1, 2, \ldots, J_m$ compute

$$c_{jm} = \underset{c}{argmin} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + c) \tag{15}$$

     4. Update $f_m(x) = f_{(m-1)}(x) + \sum_{j=1}^{J_m} c_{jm} I(x \in R_{jm})$
  3. Output $\hat{f}(x) = f_M(x)$

Boosting Trees: Demo

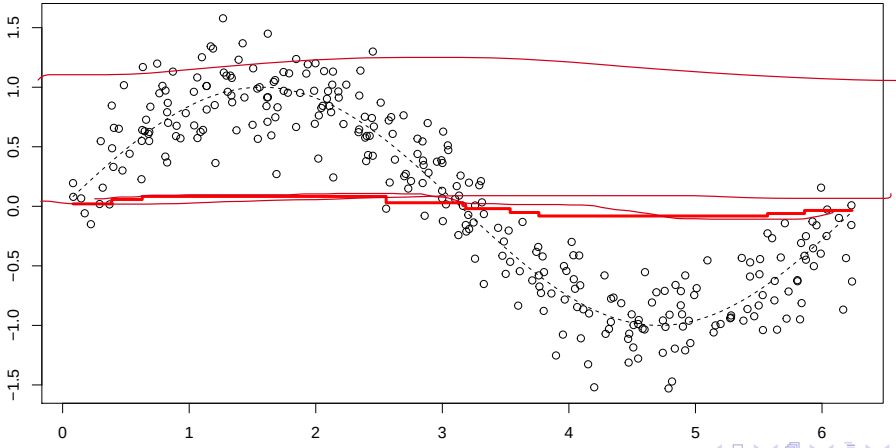
$f(x)$



Boosting Trees: Example

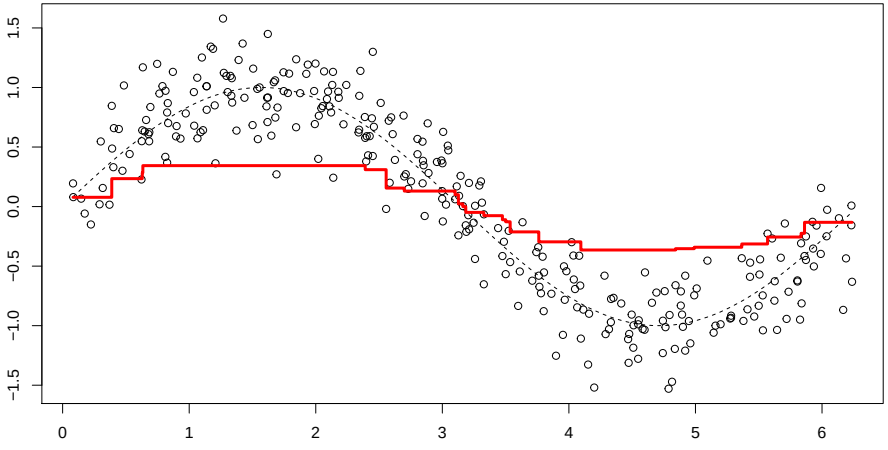
- Algorithm:

```
M<-2
```



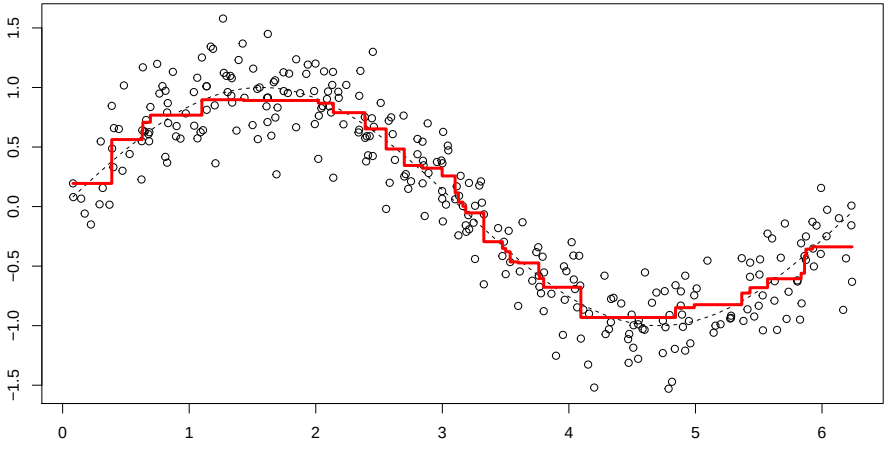
Boosting Trees: Example

```
M<-10
```



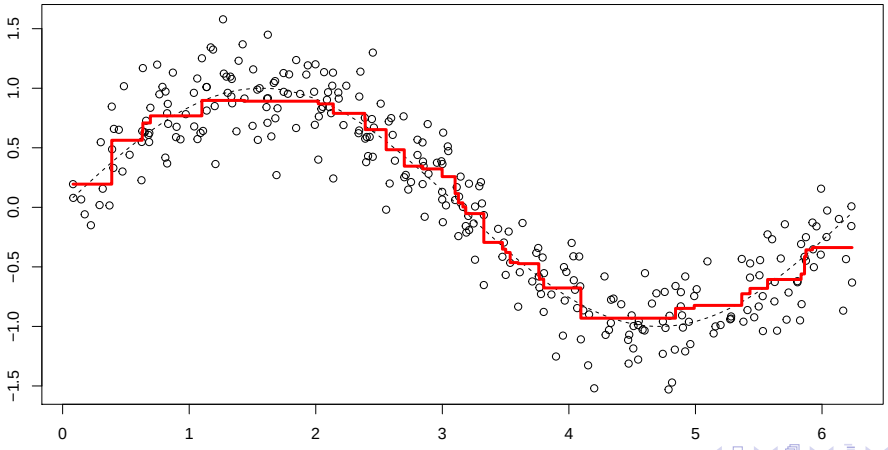
Boosting Trees: Example

```
M<-100
```



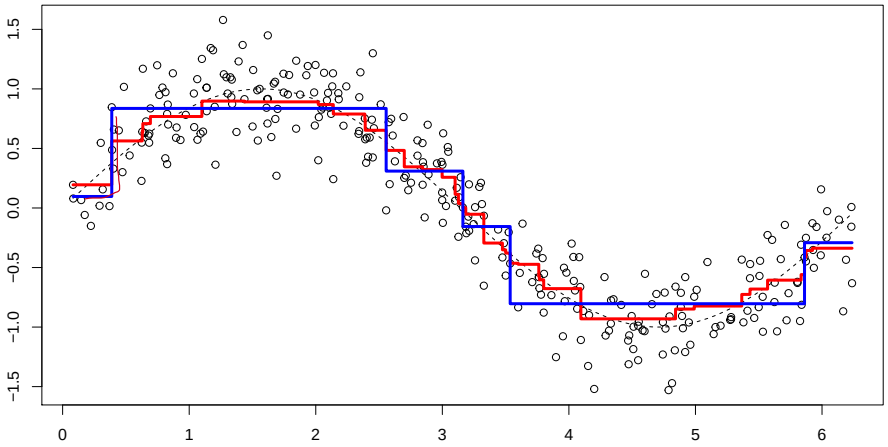
Boosting Trees: Example

```
M<-300
```



Boosting Trees: Example

- Simple tree (blue), boosted tree (red)



Boosting Trees: Algorithm explained

- Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient.
- It is intractable to learn all the trees at once.
- Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time. We write the prediction value at step m as $\hat{y}_i^m$.
- Then we have

$$\begin{aligned}
\hat{y}_i^0 &= 0 \\
\hat{y}_i^1 &= \hat{y}_i^0 + f_1(x_i) \\
\hat{y}_i^2 &= \hat{y}_i^1 + f_2(x_i) \\
&\dots \\
\hat{y}_i^M &= \sum_{m=1}^{M} f_m(x_i) = \hat{y}_i^{m-1} + f_m(x_i)
\end{aligned} \tag{16}$$

Boosting Trees: Algorithm explained

- Which tree do we want at each step?
- Add the one that optimizes our objective.

$$obj^m = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(m)}) \tag{17}$$

$$= \sum_{i=1}^{N} L(y_i, \hat{y}_i^{m-1} + f_m(x_i)) \tag{18}$$

- If we consider using mean squared error (MSE) as our loss function, the objective becomes

$$\sum_{i=1}^{N} (y_i - \hat{y}_i^{m-1} + f_m(x_i))^2 \tag{19}$$

- For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form.

Boosting Trees: Algorithm explained

- So in the general case, we take the Taylor expansion of the loss function:

$$obj^m = \sum_{i=1}^{N} \left[L(y_i, \hat{y}_i^{(m)}) + r_{im} f_m(x_i) \right] \tag{20}$$

where

$$r_{im} = - \left. \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right|_{f(\mathbf{x}_i)=f^{(m-1)}(\mathbf{x}_i)} \tag{21}$$

- After we remove all the constants, the specific objective at step m becomes

$$obj^m = \sum_{i=1}^{N} \left[r_{im} f_m(x_i) \right] \tag{22}$$

- This becomes our optimization goal for the new tree. One important advantage of this definition is that the value of the objective function only depends on r_i

Boosting Trees: Regularization

- How many iterations (M)?
  - Each iteration usually reduces the training risk $L(.)$, so that for M large enough this risk can be made arbitrarily small.
  - However, fitting the training data too well can lead to overfitting, which degrades the risk on future predictions.
  - Thus, there is an optimal number $M^\star$ minimizing future risk that is application dependent.
  - A convenient way to estimate $M^\star$ is to monitor prediction risk as a function of M on a validation sample. The value of M that minimizes this risk is taken to be an estimate of M .

Boosting Trees: Regularization

- Shrinkage
  - The simplest implementation of shrinkage in the context of boosting is to scale the contribution of each tree by a factor $\nu \in (0, 1)$ when it is added to the current approximation. That is, we replace step

$$\hat{y}_m(x) = f_{(m-1)}(x) + \nu f_m(x_i) \tag{23}$$

  - Empirically it has been found that smaller values of ν favor better test error, and require correspondingly larger values of M.
  - the best strategy appears to be to set ν to be very small ($\nu < 0.1$)
  - This yields dramatic improvements (over no shrinkage $\nu = 1$)
- Subsampling
  - With stochastic gradient boosting, at each iteration we sample a fraction η of the training observations (without replacement), and grow the next tree using that subsample.
  - Reduces the computing time by the same fraction η, and some cases improves prediction

XGBoost: Next Class Preview

- Now that you understand decision trees and gradient boosting, understanding XGBoost becomes easy: it is a gradient boosting algorithm that uses decision trees
- Beyond that, its implementation was specifically engineered for optimal performance and speed.
- Why talk about XGBoost?
  - Among the 29 challenge winning solutions published in Kaggle's blog during 2015, 17 solutions used XGBoost.
  - Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. (The second most popular method, deep neural nets, was used in 11 solutions)
  - The success of the system was also witnessed in 2015 Data Mining and Knowledge Discovery competition organized by ACM (KDD Cup) , where XGBoost was used by every winning team in the top-10.
  - Historically, XGBoost has performed quite well for structured, tabular data. But, if you are dealing with non-structured data such as images, neural networks are usually a better option (more on this later)

Review & Next Steps

- Gradient Based Optimization
- Boosting Trees
  - How it works
- Preview next class: XGBOOST
- Questions? Questions about software?

Further Readings

- Charpentier, Arthur (2018). Classification from scratch, boosting. https://freakonometrics.hypotheses.org/tag/xgboost
- Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).
- Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning (Vol. 1, No. 10). New York: Springer series in statistics Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning. Cambridge: MIT press.