Introduction to Machine Learning

XBoosting Trees and Random Forests

Andres Mendez-Vazquez

August 4, 2020

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Partition

Tree partition of the space

- They partition the space of all joint predictor variable values into disjoint regions:

$$R_j,\ j = 1, 2, ..., J$$

Thus, a constant γ_j is assigned to each such region

$$x \in R_j \Rightarrow f(x) = \gamma_j$$

Partition

Tree partition of the space

- They partition the space of all joint predictor variable values into disjoint regions:

$$R_j,\ j = 1, 2, ..., J$$

Thus, a constant γ_j is assigned to each such region

$$\boldsymbol{x} \in R_j \Rightarrow f(\boldsymbol{x}) = \gamma_j$$

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Finally, we can see a tree as

Formal, Equation

$$T(\boldsymbol{x}|\Theta) = \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x} \in R_j)$$

- $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$

Then, we have the following Loss function for Θ

$$L(\boldsymbol{x}_i, \gamma_j|\Theta) = I[y_i \neq \gamma_j]$$

Finally, we can see a tree as

Formal, Equation

$$T(\boldsymbol{x}|\Theta) = \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x} \in R_j)$$

- $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$

Then, we have the following Loss function for Θ

$$L(\boldsymbol{x}_i, \gamma_j|\Theta) = I[y_i \neq \gamma_j]$$

This is a problem

We have an Empirical Risk used to obtain the parameters

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\boldsymbol{x}_i \in R_j} L\left(\boldsymbol{x}_i, \gamma_j | \Theta\right)$$

This is a combinatorial problem

- This can be quite difficult to solve

This is a problem

We have an Empirical Risk used to obtain the parameters

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\boldsymbol{x}_i \in R_j} L\left(\boldsymbol{x}_i, \gamma_j | \Theta\right)$$

This is a combinatorial problem

- This can be quite difficult to solve

We can solve it, if ...

Finding R_j

- Note also that finding the R_j entails estimating also γ_j.

Normally, for this type of problems we use given that they are NP-Complete

- Recursive Branch and Bound algorithms

We can solve it, if ...

Finding R_j

- Note also that finding the R_j entails estimating also γ_j.

Normally, for this type of problems we use given that they are NP-Complete

- Recursive Branch and Bound algorithms

Pseudo-code for Branch-and-Bound

We have

BRANCH-AND-BOUND(P_0)

1. *Start with some problem* P_0
2. *Let* $\mathcal{S} = \{P_0\}$, *the set if active subproblems*
3. *bestsofar*$=\infty$
4. *While* $\mathcal{S} \neq \emptyset$
5. *choose a subproblem (Partial Solution)* $P \in \mathcal{S}$ *and remove it from* $\mathcal{S}$
6. *expand it into smaller subproblems* $P_1, P_2, ..., P_k$
7. *For each* P_i
8. *if* P_i *is a complete solution:*
9. *update bestsofar*
10. *else*
11. *if lowerbound*$(P_i) <$*bestsofar: add* P_i *to* $\mathcal{S}$
12. *return bestsofar*

Pseudo-code for Branch-and-Bound

We have

BRANCH-AND-BOUND(P_0)

1. *Start with some problem* P_0
2. *Let* $\mathcal{S} = \{P_0\}$, *the set if active subproblems*
3. *bestsofar*$=\infty$
4. *While* $\mathcal{S} \neq \emptyset$
5. ***choose*** *a subproblem (Partial Solution)* $P \in \mathcal{S}$ *and remove it from* $\mathcal{S}$
6. ***expand*** *it into smaller subproblems* $P_1, P_2, ..., P_k$
7. *For each* P_i
8. *if* P_i *is a complete solution:*
9. *update bestsofar*
10. *else*
11. *if lowerbound*$(P_i) <$*bestsofar: add* P_i *to* $\mathcal{S}$
12. *return bestsofar*

Pseudo-code for Branch-and-Bound

We have

BRANCH-AND-BOUND(P_0)

1. *Start with some problem* P_0
2. *Let* $\mathcal{S} = \{P_0\}$, *the set if active subproblems*
3. *bestsofar*$=\infty$
4. *While* $\mathcal{S} \neq \emptyset$
5. ***choose*** *a subproblem (Partial Solution)* $P \in \mathcal{S}$ *and remove it from* $\mathcal{S}$
6. ***expand*** *it into smaller subproblems* $P_1, P_2, ..., P_k$
7. *For each* P_i
8. *if* P_i *is a complete solution:*
9. *update bestsofar*
10. *else*
11. *if lowerbound*$(P_i) <$ *bestsofar: add* P_i *to* $\mathcal{S}$
12. *return bestsofar*

Pseudo-code for Branch-and-Bound

We have

BRANCH-AND-BOUND(P_0)

1. *Start with some problem* P_0
2. *Let* $\mathcal{S} = \{P_0\}$, *the set if active subproblems*
3. *bestsofar*$=\infty$
4. *While* $\mathcal{S} \neq \emptyset$
5. ***choose*** *a subproblem (Partial Solution)* $P \in \mathcal{S}$ *and remove it from* $\mathcal{S}$
6. ***expand*** *it into smaller subproblems* $P_1, P_2, ..., P_k$
7. *For each* P_i
8. *if* P_i *is a complete solution:*
9. *update bestsofar*
10. *else*
11. *if lowerbound*(P_i) $<$*bestsofar: add* P_i *to* $\mathcal{S}$
12. *return bestsofar*

Pseudo-code for Branch-and-Bound

We have

BRANCH-AND-BOUND(P_0)

1. *Start with some problem* P_0
2. *Let* $\mathcal{S} = \{P_0\}$, *the set if active subproblems*
3. *bestsofar*$=\infty$
4. *While* $\mathcal{S} \neq \emptyset$
5. ***choose*** *a subproblem (Partial Solution)* $P \in \mathcal{S}$ *and remove it from* $\mathcal{S}$
6. ***expand*** *it into smaller subproblems* $P_1, P_2, ..., P_k$
7. *For each* P_i
8. *if* P_i *is a complete solution:*
9. *update bestsofar*
10. *else*
11. *if lowerbound*(P_i) $<$*bestsofar: add* P_i *to* $\mathcal{S}$
12. *return bestsofar*

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Therefore

We use a smoother criterion that the one by $I[y_i \neq \gamma_j]$

$$\widetilde{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \widetilde{L}\left(T\left(\boldsymbol{x}_i|\Theta\right), y_i|\Theta\right)$$

Here, we encounter a problem

- Given R_j, How do we estimate γ_j?

Here, we do the following

- $\widehat{\gamma}_j = \overline{y}_j$, the mean of the y_i falling in the region R_j.

Therefore

We use a smoother criterion that the one by $I[y_i \neq \gamma_j]$

$$\widetilde{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \widetilde{L}\left(T\left(\boldsymbol{x}_i|\Theta\right), y_i|\Theta\right)$$

Here, we encounter a problem

- Given R_j, How do we estimate γ_j?

Here, we do the following

- $\widehat{\gamma}_j = \overline{y}_j$, the mean of the y_i falling in the region R_j.

Therefore

We use a smoother criterion that the one by $I[y_i \neq \gamma_j]$

$$\widetilde{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \widetilde{L}\left(T\left(\boldsymbol{x}_i|\Theta\right), y_i|\Theta\right)$$

Here, we encounter a problem

- Given R_j, How do we estimate γ_j?

Here, we do the following

- $\widehat{\gamma}_j = \overline{y}_j$, the mean of the y_i falling in the region R_j.

Therefore

For misclassification loss

- $\widehat{\gamma}_j$ is the modal class of the observations falling in R_j.

How do we estimate R_j

- We can use Gini or Shannon Entropy...

Therefore

For misclassification loss

- $\widehat{\gamma}_j$ is the modal class of the observations falling in R_j.

How do we estimate R_j

- We can use Gini or Shannon Entropy...

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

We are ready to define

The Boosted tree model is a sum of such trees

$$f_M(\boldsymbol{x}) = \sum_{i=1}^{N} T(\boldsymbol{x}|\Theta_m)$$

This comes from the Boosting classic cost function

$$C(\boldsymbol{x}_i) = \alpha_1 y_1(\boldsymbol{x}_i) + \alpha_2 y_2(\boldsymbol{x}_i) + \ldots + \alpha_M y_M(\boldsymbol{x}_i) \quad (1)$$

We are ready to define

The Boosted tree model is a sum of such trees

$$f_M(\boldsymbol{x}) = \sum_{i=1}^{N} T(\boldsymbol{x}|\Theta_m)$$

This comes from the Boosting classic cost function

$$C(\boldsymbol{x}_i) = \alpha_1 y_1(\boldsymbol{x}_i) + \alpha_2 y_2(\boldsymbol{x}_i) + ... + \alpha_M y_M(\boldsymbol{x}_i) \quad (1)$$

Thus, at each stage

We need to solve the following cost function

$$\widehat{\Theta} = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + T\left(\boldsymbol{x}_i|\Theta_m\right)\right)$$

For the region set and constants $\Theta_m = \left\{R_{jm}, \gamma_{jm}\right\}_{j=1}^{J_m}$

- Of the next tree give the previous model $f_{m-1}\left(\boldsymbol{x}_i\right)$

Thus, at each stage

We need to solve the following cost function

$$\widehat{\Theta} = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + T\left(\boldsymbol{x}_i|\Theta_m\right)\right)$$

For the region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_{j=1}^{J_m}$

- Of the next tree give the previous model $f_{m-1}\left(\boldsymbol{x}_i\right)$

This can be solved by

Forward Stage-wise Additive Modeling.

1. Init $f_0 = 0$
2. For $m = 1$ to M:
3. Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \beta b\left(\boldsymbol{x}_i|\gamma\right)\right)$$

4. Set $f_m\left(\boldsymbol{x}\right) = f_m\left(\boldsymbol{x}\right)$

- Here $b\left(\boldsymbol{x}_i|\gamma\right)$ simple functions of the multivariate argument $\boldsymbol{x}$.

Now

Given the regions R_{jm}

$$\widehat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{\boldsymbol{x}_i \in R_{jm}} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \gamma_{jm}\right)$$

Nevertheless, finding the regions can be difficult

- For a few special cases, the problem simplifies.

Now

Given the regions R_{jm}

$$\widehat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{\boldsymbol{x}_i \in R_{jm}} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \gamma_{jm}\right)$$

Nevertheless, finding the regions can be difficult

- For a few special cases, the problem simplifies.

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

We can use AdaBoost

We can use the exponential Loss

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp\left\{-y_i T\left(|\Theta_m\right)\right\}$$

Now, we have a conundrum

- We can decide to use a Robust Loss function
  - Absolute Error, the Huber loss

This will be make our life quite difficult

- Therefore, we opt for loss functions that can simplify our algorithms

We can use AdaBoost

We can use the exponential Loss

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp\left\{-y_i T\left(|\Theta_m\right)\right\}$$

Now, we have a conundrum

- We can decide to use a Robust Loss function
  - Absolute Error, the Huber loss

This will be make our life quite difficult

- Therefore, we opt for loss functions that can simplify our algorithms

We can use AdaBoost

We can use the exponential Loss

$$\widehat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp\left\{-y_i T\left(|\Theta_m\right)\right\}$$

Now, we have a conundrum

- We can decide to use a Robust Loss function
  - Absolute Error, the Huber loss

This will be make our life quite difficult

- Therefore, we opt for loss functions that can simplify our algorithms

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Therefore

We have the following loss function

$$L(f) = \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$$

Minimizing can be viewed as a numerical optimization

$$\hat{f} = \arg\min_{f} L(f)$$

Where

$$f = \{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_N)\}$$

Therefore

We have the following loss function

$$L(f) = \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$$

Minimizing can be viewed as a numerical optimization

$$\widehat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} L(\boldsymbol{f})$$

Where

$$f = \{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_N)\}$$

Therefore

We have the following loss function

$$L(f) = \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$$

Minimizing can be viewed as a numerical optimization

$$\widehat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} L(\boldsymbol{f})$$

Where

$$\boldsymbol{f} = \{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_N)\}$$

Thus, we have

As a Solution, we have a sum of component vectors

$$\boldsymbol{f}_M = \sum_{m=0}^{M} \boldsymbol{h}_m, \ \boldsymbol{h}_m \in \mathbb{R}^N$$

Thus, we select

- $\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m$ where ρ_m is a scalar and $\boldsymbol{g}_m \in \mathbb{R}^N$ is the gradient of

$$L(\boldsymbol{f}) = \sum_{i=1}^{N} L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))$$

  - Evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$

Thus, we have

As a Solution, we have a sum of component vectors

$$\boldsymbol{f}_M = \sum_{m=0}^{M} \boldsymbol{h}_m, \ \boldsymbol{h}_m \in \mathbb{R}^N$$

Thus, we select

- $\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m$ where ρ_m is a scalar and $\boldsymbol{g}_m \in \mathbb{R}^N$ is the gradient of

$$L(\boldsymbol{f}) = \sum_{i=1}^{N} L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))$$

  - Evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$

Then

The components

$$\boldsymbol{g}_{im} = \frac{\partial L\left(y_i, \boldsymbol{f}\left(\boldsymbol{x}_i\right)\right)}{\partial \boldsymbol{f}\left(\boldsymbol{x}_i\right)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

Where

$$\rho_m = \arg\min_{\rho} L\left(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m\right)$$

Then, we have the classic Gradient Descent

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m$$

Then

The components

$$\boldsymbol{g}_{im} = \frac{\partial L\left(y_i, \boldsymbol{f}\left(\boldsymbol{x}_i\right)\right)}{\partial \boldsymbol{f}\left(\boldsymbol{x}_i\right)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

Where

$$\rho_m = \arg\min_{\rho} L\left(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m\right)$$

Then, we have the classic Gradient Descent

$$f_m = f_{m-1} - \rho_m g_m$$

Then

The components

$$\boldsymbol{g}_{im} = \frac{\partial L\left(y_i, \boldsymbol{f}\left(\boldsymbol{x}_i\right)\right)}{\partial \boldsymbol{f}\left(\boldsymbol{x}_i\right)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

Where

$$\rho_m = \arg\min_{\rho} L\left(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m\right)$$

Then, we have the classic Gradient Descent

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m$$

Therefore

We have the following Gradients for some common Loss functions

Setting	Loss Function	Gradient $-\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))/\partial \boldsymbol{f}(\boldsymbol{x}_i)$
Regression	$\frac{1}{2}\left[y_i - f\left(\boldsymbol{x}_i\right)\right]^2$	$y_i - f\left(\boldsymbol{x}_i\right)$
Regression	$\left\|y_i - f\left(\boldsymbol{x}_i\right)\right\|$	$sign\left[y_i - f\left(\boldsymbol{x}_i\right)\right]$
Classification	$-\sum_{k=1}^{K} \log p_k\left(\boldsymbol{x}_i\right)$	k^{th} component $I\left(y = G_k\right) - p_k\left(\boldsymbol{x}_i\right)$

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:
   $$r_{im} = \left[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x}_i)=f_{m-1}(\boldsymbol{x}_i)}$$
   - Fit a regression tree to the targets r_{im} giving terminal regions R_{mj}, $j = 1, 2, ..., J_m$
   - For $j = 1, 2, ..., J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma)$$
   - Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$
3. Output $\widehat{f}(\boldsymbol{x}) = f_M(\boldsymbol{x})$

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:
   $$r_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$
   - Fit a regression tree to the targets r_{im} giving terminal regions R_{mj}, $j = 1, 2, ..., J_m$
   - For $j = 1, 2, ..., J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\widehat{f}(x) = f_M(x)$

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:

$$r_{im} = \frac{\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))}{\partial \boldsymbol{f}(\boldsymbol{x}_i)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

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

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

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

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:

$$r_{im} = \frac{\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))}{\partial \boldsymbol{f}(\boldsymbol{x}_i)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

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

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

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

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:

$$r_{im} = \frac{\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))}{\partial \boldsymbol{f}(\boldsymbol{x}_i)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

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

$$\gamma_{jm} = \arg\min_\gamma \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma)$$

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

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:

$$r_{im} = \frac{\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))}{\partial \boldsymbol{f}(\boldsymbol{x}_i)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

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

$$\gamma_{jm} = \arg\min_\gamma \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma)$$

   - Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$

3. Output $\widehat{f}(\boldsymbol{x}) = f_M(\boldsymbol{x})$

Final Algorithm

Gradient Tree Boosting Algorithm

1. $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, ..., N$ compute:

$$r_{im} = \frac{\partial L(y_i, \boldsymbol{f}(\boldsymbol{x}_i))}{\partial \boldsymbol{f}(\boldsymbol{x}_i)}\Big|_{\boldsymbol{f}(\boldsymbol{x}_i)=\boldsymbol{f}_{m-1}(\boldsymbol{x}_i)}$$

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

$$\gamma_{jm} = \arg\min_\gamma \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma)$$

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

How do we get the Right size for the Trees

We could see this as a separated procedure

- A very large (oversized) tree is first induced,
  - A bottom-up procedure is employed to prune it to the estimated optimal number of terminal nodes.

Problem

- The first trees are too Large, reducing performance...

How do we get the Right size for the Trees

We could see this as a separated procedure

- A very large (oversized) tree is first induced,
  - A bottom-up procedure is employed to prune it to the estimated optimal number of terminal nodes.

Problem

- The first trees are too Large, reducing performance...

We can do better

We can restrict the trees to have the same size on the number of Terminal Regions

$$J_m = J \ \forall m$$

- At each iteration a J-terminal node regression tree is induced.

Therefore

- Thus J becomes a meta-parameter of the entire boosting procedure.

We can do better

We can restrict the trees to have the same size on the number of Terminal Regions

$$J_m = J \ \forall m$$

- At each iteration a J-terminal node regression tree is induced.

Therefore

- Thus J becomes a meta-parameter of the entire boosting procedure.

What about M the number of trees

Another parameter to estimate

- The other meta-parameter of gradient boosting is the number of boosting iterations M.

Here, a problem is that a Large M

- It is clear that the Empirical Risk is reduced at each iteration.

A Large M can lead to Overfitting

- A convenient way to estimate M^* is to monitor prediction risk as a function of M on a validation sample.
  - Other Techniques are Shrinkage and Subsampling

What about M the number of trees

Another parameter to estimate

- The other meta-parameter of gradient boosting is the number of boosting iterations M.

Here, a problem is that a Large M

- It is clear that the Empirical Risk is reduced at each iteration.

A Large M can lead to Overfitting

- A convenient way to estimate M^* is to monitor prediction risk as a function of M on a validation sample.
  - Other Techniques are Shrinkage and Subsampling

What about M the number of trees

Another parameter to estimate

- The other meta-parameter of gradient boosting is the number of boosting iterations M.

Here, a problem is that a Large M

- It is clear that the Empirical Risk is reduced at each iteration.

A Large M can lead to Overfitting

- A convenient way to estimate M^* is to monitor prediction risk as a function of M on a validation sample.
  - Other Techniques are Shrinkage and Subsampling

For More on this

Take a Look at

- The Elements of Statistical Learning by Hastie et al. Chapter 10.11 and 10.12

In the Case of Shrinkage

Instead of using

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

We modify by a parameter ν

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

The parameter ν is controlling the learning rate of the boosting procedure.

- Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M.

In the Case of Shrinkage

Instead of using

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

We modify by a parameter ν

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

The parameter ν is controlling the learning rate of the boosting procedure.

- Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M.

In the Case of Shrinkage

Instead of using

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

We modify by a parameter ν

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$$

The parameter ν is controlling the learning rate of the boosting procedure.

- Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M.

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

A Popular Algorithm

It has been a winner 29 Kaggle challenges (2015)

- 17 solutions used XGBoost.

As solely algorithm

- Or with a combination of neural network algorithms as ensembles method.

A Popular Algorithm

It has been a winner 29 Kaggle challenges (2015)

- 17 solutions used XGBoost.

As solely algorithm

- Or with a combination of neural network algorithms as ensembles method.

Ensemble Learning

Definition

- In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain

Basically

1. Bootstrap aggregating (bagging)
2. Boosting
3. Bayesian parameter averaging
4. Bayesian model combination
5. etc

Ensemble Learning

Definition

- In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain

Basically

1. Bootstrap aggregating (bagging)
2. Boosting
3. Bayesian parameter averaging
4. Bayesian model combination
5. etc

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Cost Function Ensemble

For a given data set

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|\, |\mathcal{D}| = N, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$$

A Tree Ensemble model

$$\widehat{y}_i = \phi\left(\boldsymbol{x}_i\right) = \sum_{k=1}^{K} f_k\left(\boldsymbol{x}_i\right)$$

Where, the space of regression trees (CART)

$$\mathcal{F} = \left\{ f_k\left(\boldsymbol{x}\right) = w_{q(\boldsymbol{x})} \right\} \left(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T \right)$$

Cost Function Ensemble

For a given data set

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|\, |\mathcal{D}| = N, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$$

A Tree Ensemble model

$$\widehat{y}_i = \phi(\boldsymbol{x}_i) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i)$$

Where, the space of regression trees (CART)

$$\mathcal{F} = \left\{ f_k(\boldsymbol{x}) = w_{q(\boldsymbol{x})} \right\} \left(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T \right)$$

Cost Function Ensemble

For a given data set

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|\, |\mathcal{D}| = N, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$$

A Tree Ensemble model

$$\widehat{y}_i = \phi\left(\boldsymbol{x}_i\right) = \sum_{k=1}^{K} f_k\left(\boldsymbol{x}_i\right)$$

Where, the space of regression trees (CART)

$$\mathcal{F} = \left\{f_k\left(\boldsymbol{x}\right) = w_{q(\boldsymbol{x})}\right\}\left(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T\right)$$

Remarks

$q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T$

- q represents the structure of a tree that maps an example to the corresponding leaf index.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q and leaf weights w.

Remarks

$q : \mathbb{R}^m \to T, w \in \mathbb{R}^T$

- q represents the structure of a tree that maps an example to the corresponding leaf index.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q and leaf weights w.

Something Notable

- Unlike decision trees, each regression tree contains a continuous rank on each of the leaf.

Remarks

$q : \mathbb{R}^m \to T, w \in \mathbb{R}^T$

- q represents the structure of a tree that maps an example to the corresponding leaf index.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q and leaf weights w.

Something Notable

- Unlike decision trees, each regression tree contains a continuous rank on each of the leaf.

For this

- we use w_i to represent score on i^{th} leaf.

Remarks

$q : \mathbb{R}^m \to T, w \in \mathbb{R}^T$

- q represents the structure of a tree that maps an example to the corresponding leaf index.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q and leaf weights w.

Something Notable

- Unlike decision trees, each regression tree contains a continuous rank on each of the leaf.

For this

- we use w_i to represent score on i^{th} leaf.

Remarks

$q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T$

- q represents the structure of a tree that maps an example to the corresponding leaf index.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q and leaf weights w.

Something Notable

- Unlike decision trees, each regression tree contains a continuous rank on each of the leaf.

For this

- we use w_i to represent score on i^{th} leaf.

Final Cost Function

XGBoost minimize the following function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{whre } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Remarks

- l is a differentiable convex loss function.
- Ω penalize the complexity of the regression tree.
- $\frac{1}{2}\lambda \|w\|^2$ helps to smooth the final learned weights to avoid over-fitting.

Final Cost Function

XGBoost minimize the following function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{whre } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Remarks

- l is a differentiable convex loss function.
- Ω penalize the complexity of the regression tree.
- $\frac{1}{2}\lambda \|w\|^2$ helps to smooth the final learned weights to avoid over-fitting.

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Optimizing in an Additive Manner

For this, the model is trained in an additive manner

- Given $\widehat{y}_i^{(t)}$ be the prediction of the i^{th} instance at the t^{th} iteration,

We rewrite the cost function as

$$\mathcal{L}^{(t)}(\phi) = \sum_i l\left(\widehat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i), y_i\right) + \Omega(f_t)$$

- This means we greedily add the f_t that most improves our model.

Optimizing in an Additive Manner

For this, the model is trained in an additive manner

- Given $\widehat{y}_i^{(t)}$ be the prediction of the i^{th} instance at the t^{th} iteration,

We rewrite the cost function as

$$\mathcal{L}^{(t)}(\phi) = \sum_i l\left(\widehat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i), y_i\right) + \Omega(f_t)$$

- This means we greedily add the f_t that most improves our model.

Then, we can use the Taylor Second Optimization

Second-order approximation

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{N} \left[l\left(\widehat{y}_i^{(t-1)}, y_i\right) + g_i f_t\left(\boldsymbol{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\boldsymbol{x}_i\right) \right] + \Omega\left(f_t\right)$$

Where

- $g_i = \partial_{\widehat{y}^{(t-1)}} l\left(\widehat{y}_i^{(t-1)}, y_i\right)$ and $h_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(\widehat{y}_i^{(t-1)}, y_i\right)$

Then, we can use the Taylor Second Optimization

Second-order approximation

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{N} \left[l\left(\widehat{y}_i^{(t-1)}, y_i\right) + g_i f_t\left(\boldsymbol{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\boldsymbol{x}_i\right) \right] + \Omega\left(f_t\right)$$

Where

- $g_i = \partial_{\widehat{y}^{(t-1)}} l\left(\widehat{y}_i^{(t-1)}, y_i\right)$ and $h_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(\widehat{y}_i^{(t-1)}, y_i\right)$

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Furthermore

We have the following cost function after removing constant terms

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{N} \left[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \Omega(f_t)$$

Which can be expanded by defining $I_j = \{i | q(\boldsymbol{x}_i) = j\}$

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} \left[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \lambda T$$

Furthermore

We have the following cost function after removing constant terms

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{N} \left[g_i f_t\left(\boldsymbol{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\boldsymbol{x}_i\right) \right] + \Omega\left(f_t\right)$$

Which can be expanded by defining $I_j = \{i | q\left(\boldsymbol{x}_i\right) = j\}$

$$\begin{aligned}
\mathcal{L}^{(t)} &= \sum_{i=1}^{N} \left[g_i f_t\left(\boldsymbol{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\boldsymbol{x}_i\right) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \lambda T
\end{aligned}$$

Then, for a fixed structure $q(\boldsymbol{x})$

we can compute the optimal weight for a leaf

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Additionally, we can use the following function to score the structure of q

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

Then, for a fixed structure $q(\boldsymbol{x})$

we can compute the optimal weight for a leaf

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Additionally, we can use the following function to score the structure of q

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

Remarks

The previous equations can be used

- As a scoring function to measure the quality of a tree structure q

Something Notable

- This score is like the impurity score for evaluating decision trees

Remarks

The previous equations can be used

- As a scoring function to measure the quality of a tree structure q

Something Notable

- This score is like the impurity score for evaluating decision trees

However

Something Notable

- Normally, it is impossible to enumerate all the possible tree structures q.

Therefore

- A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.

Letting $I = I_L \cup I_R$, then the reduction is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

However

Something Notable

- Normally, it is impossible to enumerate all the possible tree structures q.

Therefore

- A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.

Letting $I = I_L \cup I_R$, then the reduction is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

However

Something Notable

- Normally, it is impossible to enumerate all the possible tree structures q.

Therefore

- A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.

Letting $I = I_L \cup I_R$, then the reduction is given by

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Basic Exact Greedy Algorithm

A Big Problem

- One of the key problems in tree learning is to find the best split by

$$\mathcal{L}_{split}$$

In order to do generate these splits

- A split finding algorithm enumerates over all the possible splits on all the features

Basic Exact Greedy Algorithm

A Big Problem

- One of the key problems in tree learning is to find the best split by

$$\mathcal{L}_{split}$$

In order to do generate these splits

- A split finding algorithm enumerates over all the possible splits on all the features

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $G_L = 0$ and $H_L = 0$
7. for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $G_L = G_L + g_j$, $H_L = H_L + h_j$
9. $G_R = G - G_L$, $H_R = H - H_L$
10. $score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output:** Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $G_L = 0$ and $H_L = 0$
7. for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $G_L = G_L + g_j$, $H_L = H_L + h_j$
9. $G_R = G - G_L$, $H_R = H - H_L$
10. $score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output:** Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $G_L = 0$ and $H_L = 0$
7. for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $G_L = G_L + g_j$, $H_L = H_L + h_j$
9. $G_R = G - G_L$, $H_R = H - H_L$
10. $score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output:** Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $G_L = 0$ and $H_L = 0$
7. for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $G_L = G_L + g_j$, $H_L = H_L + h_j$
9. $G_R = G - G_L$, $H_R = H - H_L$
10. $score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output:** Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $G_L = 0$ and $H_L = 0$
7. for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $G_L = G_L + g_j$, $H_L = H_L + h_j$
9. $G_R = G - G_L$, $H_R = H - H_L$
10. $score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output:** Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $\quad G_L = 0$ and $H_L = 0$
7. $\quad$ for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $\quad\quad G_L = G_L + g_j,\ H_L = H_L + h_j.$
9. $\quad\quad G_R = G - G_L,\ H_R = H - H_L.$
10. $\quad\quad score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. Output: Split with Max Score

Example, Exact Greedy Algorithm

Something Notable

1. **Input:** I, instance set of current node
2. **Input:** m, feature dimension
3. $gain = 0$
4. $G = \sum_{i \in I} g_i$ and $H = \sum_{i \in I} h_i$
5. for $k = 1$ to m do:
6. $\quad G_L = 0$ and $H_L = 0$
7. $\quad$ for j in $sorted\left(I, \text{by } x_{jk}\right)$ do
8. $\quad\quad G_L = G_L + g_j,\ H_L = H_L + h_j.$
9. $\quad\quad G_R = G - G_L,\ H_R = H - H_L.$
10. $\quad\quad score = \max\left\{score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right\}$
11. **Output**: Split with Max Score

Problem with this Algorithm

Quite computationally demanding

- This can be improved!!!

Problem with this Algorithm

Quite computationally demanding

- This can be improved!!!

For this

- The algorithm must first sort the data according to feature values.
- Then, it visits the data in sorted order to accumulate the gradient statistics.

Problem with this Algorithm

Quite computationally demanding

- This can be improved!!!

For this

- The algorithm must first sort the data according to feature values.
- Then, it visits the data in sorted order to accumulate the gradient statistics.

Therefore

Better to have an approximation

- Thus, people proposed the use the percentiles of feature distributions
  - To find the splitting points or candidate points

Then, it maps the continuous features into buckets split by these candidate points

- Basically you could use homogeneity via the Shannon Entropy
  - Or any other possible one

Aggregates the statistics on the buckets

- Then, It finds the best solution based on this statistics

Therefore

Better to have an approximation

- Thus, people proposed the use the percentiles of feature distributions
  - To find the splitting points or candidate points

Then, it maps the continuous features into buckets split by these candidate points

- Basically you could use homogeneity via the Shannon Entropy
  - Or any other possible one

Aggregates the statistics on the buckets

- Then, It finds the best solution based on this statistics

Therefore

Better to have an approximation

- Thus, people proposed the use the percentiles of feature distributions
  - To find the splitting points or candidate points

Then, it maps the continuous features into buckets split by these candidate points

- Basically you could use homogeneity via the Shannon Entropy
  - Or any other possible one

Aggregates the statistics on the buckets

- Then, It finds the best solution based on this statistics

The Two Variants for Splitting

The global variant

- It proposes all the candidate splits during the initial phase of tree construction

The local variant

- The local variant re-proposes after each split!!!

The Two Variants for Splitting

The global variant

- It proposes all the candidate splits during the initial phase of tree construction

The local variant

- The local variant re-proposes after each split!!!

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Approximate Algorithm for Split Finding

Algorithm

1. for $k = 1$ to m:
2. Propose $S_k =$ by using weighted percentiles at the feature k
3. Proposal can be done per tree (global) or per split
4. for $k = 1$ to m:
5. $G_{kv} = \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_j$
6. $H_{kv} = \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h_j$

However

An important subject

- How the Weighted Quantile Sketch works?

Weighted Quantile Sketch

- To understand the method in XGBoost

It is part of the original implementation

- Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 785-794. 2016.

However

An important subject

- How the Weighted Quantile Sketch works?

Weighted Quantile Sketch

- To understand the method in XGBoost

It is part of the original implementation

- Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 785-794. 2016.

However

An important subject

- How the Weighted Quantile Sketch works?

Weighted Quantile Sketch

- To understand the method in XGBoost

It is part of the original implementation

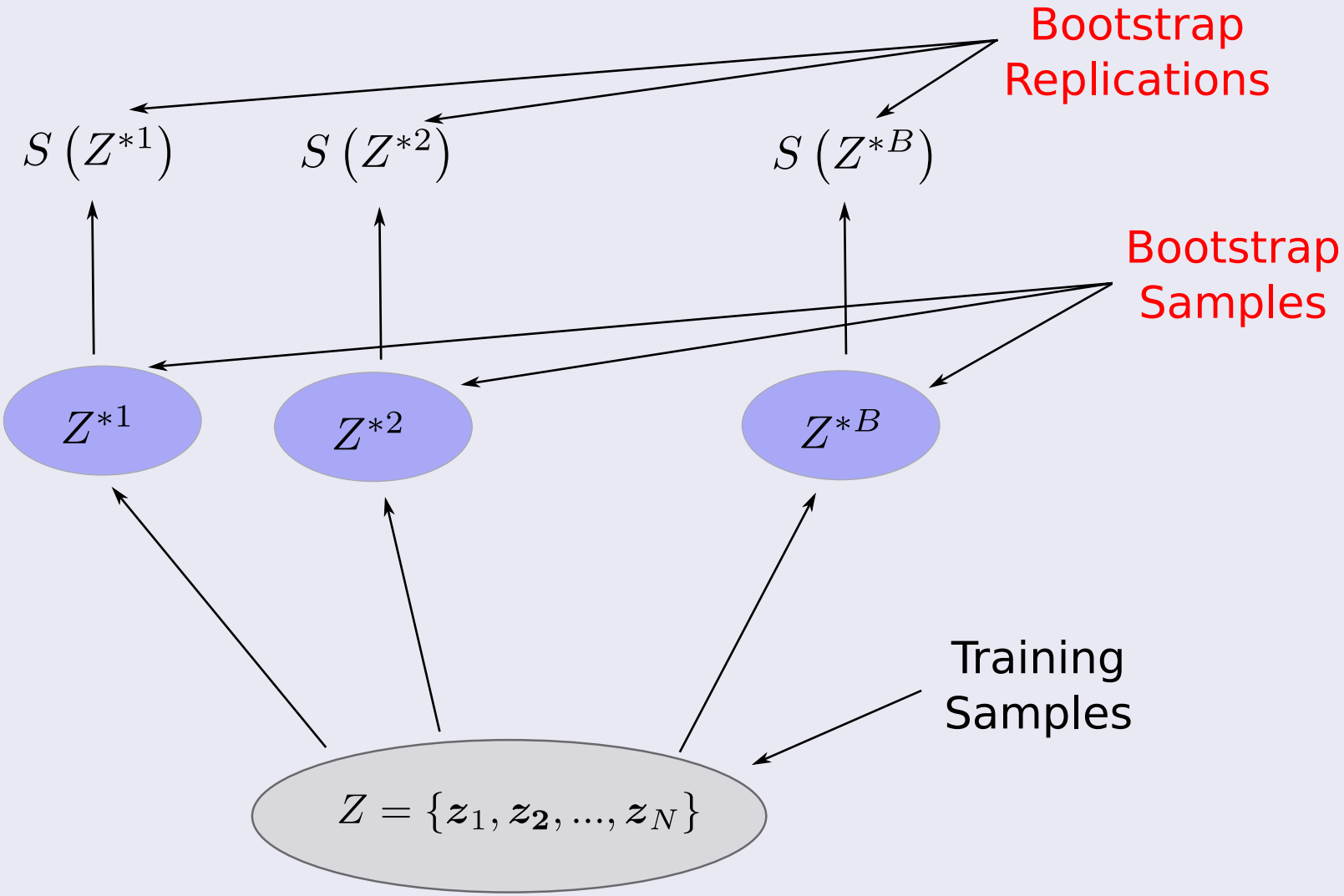
- Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 785-794. 2016.

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

Reminder

Bootstrap aggregating/ bagging



Main Idea

We have then

- The essential idea in bagging is to average many noisy but approximately unbiased models.

Thus, you reduce the variance

- And given that trees capture complex interactions

This is perfect given

- If we can decrease the variance of the decision trees
  - We obtain a more precise classifier.

Main Idea

We have then

- The essential idea in bagging is to average many noisy but approximately unbiased models.

Thus, you reduce the variance

- And given that trees capture complex interactions

This is perfect given

- If we can decrease the variance of the decision trees
  - We obtain a more precise classifier.

Main Idea

We have then

- The essential idea in bagging is to average many noisy but approximately unbiased models.

Thus, you reduce the variance

- And given that trees capture complex interactions

This is perfect given

- If we can decrease the variance of the decision trees
  - We obtain a more precise classifier.

Outline

1. Boosting Trees
   - Introduction
   - Cost Functions for Trees
   - Using a Smoother Version
   - Boosted Tree Model
   - AdaBoost for Classification Trees
   - Numerical Optimization via Gradient Boosting
2. XGBoost
   - Introduction
   - Cost Function
   - Solving some Issues
   - Taylor Expansion
   - Split Finding Algorithms
     - Generic Approximated Version
3. Random Forest
   - Introduction
   - From Bootstrap to Random Forest

The Model

In a series of papers and technical reports

- In a series of papers and technical reports - Leo Breiman demonstrated the substantial gains in classification and regression

By using ensembles of trees

- In Breiman's approach, each tree in the collection is formed by first selecting at random
  - At each node, a small of input coordinates/features

Then, we use such features to obtain the best split

- For the subsets at the nodes...

The Model

In a series of papers and technical reports

- In a series of papers and technical reports - Leo Breiman demonstrated the substantial gains in classification and regression

By using ensembles of trees

- In Breiman's approach, each tree in the collection is formed by first selecting at random
  - At each node, a small of input coordinates/features

Then, we use such features to obtain the best split

- For the subsets at the nodes...

The Model

In a series of papers and technical reports

- In a series of papers and technical reports - Leo Breiman demonstrated the substantial gains in classification and regression

By using ensembles of trees

- In Breiman's approach, each tree in the collection is formed by first selecting at random
  - At each node, a small of input coordinates/features

Then, we use such features to obtain the best split

- For the subsets at the nodes...

For example, using the idea of Bootstrap

Draw a bootstrap sample Z of size N from the training data

- Grow a random-forest tree T_b

For example, using the idea of Bootstrap

Draw a bootstrap sample Z of size N from the training data

- Grow a random-forest tree T_b

Using a stopping criteria of minimum node size $n_{\min}$

1. Select m variables at random from the d variables.
2. Pick the best variable/split-point among the m
3. Split the node into two daughter nodes

For example, using the idea of Bootstrap

Draw a bootstrap sample Z of size N from the training data

- Grow a random-forest tree T_b

Using a stopping criteria of minimum node size $n_{\min}$

1. Select m variables at random from the d variables.
2. Pick the best variable/split-point among the m
3. Split the node into two daughter nodes

Finally

Output the ensemble of trees $\{T_b\}_{b=1}^{B}$

For example, using the idea of Bootstrap

Draw a bootstrap sample Z of size N from the training data

- Grow a random-forest tree T_b

Using a stopping criteria of minimum node size $n_{\min}$

1. Select m variables at random from the d variables.
2. Pick the best variable/split-point among the m
3. Split the node into two daughter nodes

Finally

Output the ensemble of trees $\{T_b\}_{b=1}^{B}$

For example, using the idea of Bootstrap

Draw a bootstrap sample Z of size N from the training data

- Grow a random-forest tree T_b

Using a stopping criteria of minimum node size $n_{\min}$

1. Select m variables at random from the d variables.
2. Pick the best variable/split-point among the m
3. Split the node into two daughter nodes

Finally

Output the ensemble of trees $\{T_b\}_{b=1}^{B}$

In another example

The following procedure is then repeated $\lceil \log_2 k_n \rceil$

1. At each node, a feature of $\boldsymbol{x} = (x_1, x_2, ..., x_d)^T$ is selected, with the jth feature having a probability $p_{nj} \in (0, 1)$ of being selected.
2. At each node, after feature selection, the split is at the midpoint of the chosen side.

Therefore

A Random Forest

- It is a predictor consisting of a collection of randomized base trees

$$\{T_b(\boldsymbol{x}, \Theta_m, \mathcal{D}_n) | m > 1\}$$

where $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$

Here, $\Theta_1, \Theta_2, \ldots$ are i.i.d. outputs of a randomizing variable Θ

$$\hat{y}(X, \mathcal{D}_n) = E_\Theta [T_b(X, \Theta, \mathcal{D}_n)]$$

Therefore

A Random Forest

- It is a predictor consisting of a collection of randomized base trees

$$\{T_b(\boldsymbol{x}, \Theta_m, \mathcal{D}_n) | m > 1\}$$

where $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$

Here, $\Theta_1, \Theta_2, \ldots$ are i.i.d. outputs of a randomizing variable Θ

$$\widehat{y}(X, \mathcal{D}_n) = E_\Theta [T_b(X, \Theta, \mathcal{D}_n) |]$$

We tend to use the sample mean

Regression

$$\widehat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Classification, given $C_b(x)$ the classification prediction of the T_b tree

$$\widehat{C}_b(x) = \text{majority vote}\left\{C_b(x)\right\}_{b=1}^{B}$$

We tend to use the sample mean

Regression

$$\widehat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Classification, given $C_b(x)$ the classification prediction of the T_b tree

$$\widehat{C}_b(x) = \text{majority vote}\left\{C_b(x)\right\}_{b=1}^{B}$$

The nice part is that

Given that trees are notoriously noisy

- When we average over them, we obtained better accurate predictions

For More

Take a Look at

- The Elements of Statistical Learning by Hastie et al. Chapter 15