STA141C: Big Data & High Performance Statistical Computing
Lecture 10: Tree-based Algorithms

Cho-Jui Hsieh
UC Davis

May 22, 2018
Outline

- Decision Tree
- Random Forest
- Gradient Boosted Decision Tree (GBDT)
Each node checks one feature $x_i$:
- Go left if $x_i < \text{threshold}$
- Go right if $x_i \geq \text{threshold}$
A real example

Play tennis or not

Outlook
  Sunny
  Rain
    Overcast
      Yes
  Wind
    Strong
      No
      Weak
        Yes
  Humidity
    High
      No
    Normal
      Yes
Decision Tree

- **Strength:**
  - It’s a *nonlinear* classifier
  - Better *interpretability*
  - Can naturally handle *categorical* features
Strength:
- It’s a *nonlinear* classifier
- Better *interpretability*
- Can naturally handle *categorical* features

Computation:
- Training: *slow*
- Prediction: *fast*

\[ h \text{ operations (} h: \text{ depth of the tree, usually } \leq 15) \]
Splitting the node

- **Classification tree:** Split the node to maximize entropy
- Let $S$ be set of data points in a node, $c = 1, \cdots, C$ are labels:

  $$H(S) = - \sum_{c=1}^{C} p(c) \log p(c),$$

  where $p(c)$ is the proportion of the data belong to class $c$.

  - Entropy=0 if all samples are in the same class
  - Entropy is large if $p(1) = \cdots = p(C)$

![Entropy Example](image)

**Bad split**

Entropy: 
\[-(1/3)\log(1/3) - (1/3)\log(1/3) - (1/3) \log(1/3) = 1.58\]

**Good split**

Entropy: 
\[-1\log*(1) = 0\]
Information Gain

- The averaged entropy of a split $S \rightarrow S_1, S_2$

\[
\frac{|S_1|}{|S|} H(S_1) + \frac{|S_2|}{|S|} H(S_2)
\]

- Information gain: measure how good is the split

\[
H(S) - \left( \left( \frac{|S_1|}{|S|} \right) H(S_1) + \left( \frac{|S_2|}{|S|} \right) H(S_2) \right)
\]
Information Gain

Entropy = 1.58

Averaged entropy: \( \frac{2}{3} \times 1 + \frac{1}{3} \times 0 = 0.67 \)

Information gain: 1.58 - 0.67 = 0.91
Information Gain

Entropy = 1.58

Entropy = 1.52
Entropy = 1.5

Averaged entropy: 1.51
Information gain: 1.58 – 1.51 = 0.07
Splitting the node

Given the current note, how to find the best split?

For all the features and all the threshold
Compute the information gain after the split
Choose the best one (maximal information gain)
For \( n \) samples and \( d \) features: need \( O(nd) \) time
Splitting the node

- Given the current note, how to find the best split?
- For all the features and all the threshold
  - Compute the information gain after the split
  - Choose the best one (maximal information gain)
Splitting the node

- Given the current note, how to find the **best split**?
- For all the **features** and all the **threshold**
  - Compute the information gain after the split
  - Choose the best one (**maximal information gain**)
- For *n* samples and *d* features: need $O(nd)$ time
Regression Tree

- Assign a real number for each leaf
- Usually **averaged y values** for each leaf
  (minimize square error)

\[
\begin{align*}
\text{y}_1 &= 1 & \text{y}_5 &= 2 & \text{y}_6 &= 3 \\
\text{y}_2 &= 4 \\
\text{y}_3 &= 100 & \text{y}_7 &= 200 \\
\text{y}_4 &= 1
\end{align*}
\]
Objective function:

$$\min_F \frac{1}{n} \sum_{i=1}^{n} (y_i - F(x_i))^2 + \text{(Regularization)}$$

The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:

$$\sum_{i \in S_1} (y_i - y^{(1)})^2 + \sum_{i \in S_2} (y_i - y^{(2)})^2,$$

where $y^{(1)} = \frac{1}{|S_1|} \sum_{i \in S_1} y_i$, $y^{(2)} = \frac{1}{|S_2|} \sum_{i \in S_2} y_i$
Regression Tree

- **Objective function:**

  $$\min_{F} \frac{1}{n} \sum_{i=1}^{n} (y_i - F(x_i))^2 + \text{(Regularization)}$$

- The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:

  $$\sum_{i \in S_1} (y_i - y^{(1)})^2 + \sum_{i \in S_2} (y_i - y^{(2)})^2,$$

  where $y^{(1)} = \frac{1}{|S_1|} \sum_{i \in S_1} y_i$, $y^{(2)} = \frac{1}{|S_2|} \sum_{i \in S_2} y_i$

- **Find the best split:**

  Try all the features & thresholds and find the one with **minimal objective function**
Parameters

- Maximum depth: (usually \(\sim 10\))
- Minimum number of nodes in each node: (10, 50, 100)
Parameters

- Maximum depth: (usually ~ 10)
- Minimum number of nodes in each node: (10, 50, 100)
- Single decision tree is not very powerful... 
- Can we build multiple decision trees and ensemble them together?
Random Forest
Random Forest (Bootstrap ensemble for decision trees):
- Create $T$ trees
- Learn each tree using a subsampled dataset $S_i$ and subsampled feature set $D_i$
- Prediction: Average the results from all the $T$ trees

Benefit:
- Avoid over-fitting
- Improve stability and accuracy

Good software available:
- R: “randomForest” package
- Python: sklearn
Random Forest
Gradient Boosted Decision Tree
Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i)) \quad \text{with} \quad F(x) = \sum_{m=1}^{T} f_m(x)$$

(each $f_m$ is a decision tree)
Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i)) \quad \text{with} \quad F(x) = \sum_{m=1}^{T} f_m(x)$$

(each $f_m$ is a decision tree)

- Direct loss minimization: at each stage $m$, find the best function to minimize loss
  - solve $f_m = \arg\min_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + f_m(x_i))$
  - update $F_m \leftarrow F_{m-1} + f_m$

$F_m(x) = \sum_{j=1}^{m} f_j(x)$ is the prediction of $x$ after $m$ iterations.

Two problems:
- Hard to implement for general loss
- Tend to overfit training data
Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i)) \quad \text{with} \quad F(x) = \sum_{m=1}^{T} f_m(x)$$

(each $f_m$ is a decision tree)

- Direct loss minimization: at each stage $m$, find the best function to minimize loss
  - solve $f_m = \arg\min_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(x_i) + f_m(x_i))$
  - update $F_m \leftarrow F_{m-1} + f_m$

$F_m(x) = \sum_{j=1}^{m} f_j(x)$ is the prediction of $x$ after $m$ iterations.

- Two problems:
  - Hard to implement for general loss
  - Tend to overfit training data
Gradient Boosted Decision Tree (GBDT)

- Approximate the current loss function by a quadratic approximation:

$$
\sum_{i=1}^{n} \ell_i(\hat{y}_i + f_m(x_i)) \approx \sum_{i=1}^{n} \left( \ell_i(\hat{y}_i) + g_i f_m(x_i) + \frac{1}{2} h_i f_m(x_i)^2 \right)
$$

$$
= \sum_{i=1}^{n} \frac{h_i}{2} \| f_m(x_i) - g_i / h_i \|^2 + \text{constant}
$$

where $g_i = \partial_{\hat{y}_i} \ell_i(\hat{y}_i)$ is gradient, $h_i = \partial^2_{\hat{y}_i} \ell_i(\hat{y}_i)$ is second order derivative
Gradient Boosted Decision Tree

- Finding $f_m(x, \theta_m)$ by minimizing the loss function:

$$\arg\min_{f_m} \sum_{i=1}^{N} \left[ f_m(x_i, \theta) - g_i/h_i \right]^2 + R(f_m)$$

- Reduce the training of any loss function to regression tree (just need to compute $g_i$ for different functions)
- $h_i = \alpha$ (fixed step size) for original GBDT.
- XGboost shows computing second order derivative yields better performance
Gradient Boosted Decision Tree

- Finding $f_m(x, \theta_m)$ by minimizing the loss function:

$$\argmin_{f_m} \sum_{i=1}^{N} [f_m(x_i, \theta) - g_i/h_i]^2 + R(f_m)$$

- Reduce the training of any loss function to regression tree (just need to compute $g_i$ for different functions)
- $h_i = \alpha$ (fixed step size) for original GBDT.
- XGboost shows computing second order derivative yields better performance

Algorithm:
- Computing the current gradient for each $\hat{y}_i$.
- Building a base learner (decision tree) to fit the gradient.
- Updating current prediction $\hat{y}_i = F_m(x_i)$ for all $i$. 
Key idea:
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$
Gradient Boosted Decision Trees (GBDT)

Key idea:
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$

$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{m-1}(x_i)}$$
Gradient Boosted Decision Trees (GBDT)

- **Key idea:**
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial F} \)

\[
F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad \text{and} \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i) = F_{m-1}(x_i)}
\]
Gradient Boosted Decision Trees (GBDT)

**Key idea:**
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial F} \)

\[
F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i) = F_{m-1}(x_i)}
\]
Gradient Boosted Decision Trees (GBDT)

- **Key idea:**
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient \( \frac{\partial \ell}{\partial f} \)

Final prediction:

\[
F(x_i) = \sum_{j=1}^{T} f_j(x_i)
\]
Coming up

- Clustering

Questions?