xgBoost

Victor Kitov

Introduction

- xgBoost - one of many open source realizations of gradient boosting.
- Very successful:
  - among 29 kaggle competitions on Kaggle in 2015 17 winning solutions used xgBoost and among these 8 used only xgBoost.
- Success reasons:
  1. optimization criteria is flexible enough to fit any loss function
  2. optimization criteria has regularization.
  3. optimized for big data
  4. optimized for sparse data
- We will consider only optimization criteria here.

Boosting reminder

- Boosting prediction is performed with sum of M predictor functions:

$$\widehat{y}_i = \sum_{m=1}^{M} f_m(x_i),$$

where each f_m is a regression tree:

  - $f_m \in \{f(x) = w_{q(x)}\}$,
  - $q : \mathbb{R}^D \to T$, $w \in \mathbb{R}^T$
  - T is the number of leaves.

- Each tree f_m:
  - has independent tree structure $q(x)$ and weights w
  - is built greedily after optimizing $f_1, f_2, ...f_{m-1}$ to achieve greatest score improvement.

Optimization score

At step m we optimize:

$$L^{(m)}(f_m) = \sum_{n=1}^{N} \mathcal{L}(y_n, \widehat{y}_n^{(m-1)} + f_m(x_n)) + R(f_m) \qquad (1)$$

Here:

- $\mathcal{L}(y_n, \widehat{y}_n^{(m)})$ is the loss induced by predicting y_n with $\widehat{y}_n$
- $R(f_m) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ is the regularization term, penalizing f_m for complexity.
  - γT penalizes the number of leaves
  - $\|w\|^2 = \sum_t^T w_t^2$ penalizes the magnitude of leaf predictions.

Taylor expansion

- Using Taylor expansion expand $\mathcal{L}(y_n, \widehat{y}_n^{(m)})$ into

$$\mathcal{L}(y_n, \widehat{y}_n^{(m)}) \approx \mathcal{L}(y_n, \widehat{y}_n^{(m-1)}) + g_n f_m(x_n) + \frac{1}{2} h_n f_m^2(x_n) \qquad (2)$$

where

$$g_n = \frac{\partial}{\partial \widehat{y}^{(m-1)}} \mathcal{L}\left(y_n, \widehat{y}_n^{(m-1)}\right), \qquad h_n = \frac{\partial^2}{\partial^2 \widehat{y}^{(m-1)}} \mathcal{L}\left(y_n, \widehat{y}_n^{(m-1)}\right)$$

- Plugging (2) into (1), obtain:

$$L^{(m)}(f_m) \approx \sum_{n=1}^{N} \left[\mathcal{L}(y_n, \widehat{y}_n^{(m-1)}) + g_n f_m(x_n) + \frac{1}{2} h_n f_m^2(x_n) \right] + R(f_m) \qquad (3)$$

Taylor expansion

- We get approximate criterion for (3):

$$\widehat{L}^{(m)}(f_m) = \sum_{n=1}^{N}\left[\mathcal{L}(y_n,\widehat{y}_n^{(m-1)}) + g_n f_m(x_n) + \frac{1}{2}h_n f_m^2(x_n)\right] + \gamma T + \frac{1}{2}\lambda\sum_{t=1}^{T} w_t^2 \quad (4)$$

- Define $I_t = \{n:\ q(x_n) = t\}$. Then (4) can be rewritten as:

$$\widehat{L}^{(m)}(f_m) = \sum_{t=1}^{T}\left[\left(\sum_{n\in I_t} g_n\right) w_t + \frac{1}{2}\left(\sum_{n\in I_t} h_n + \lambda\right) w_t^2\right] + \gamma T + const(f_m) \quad (5)$$

Optimized loss

- Optimizing (5) with respect to w_t, we obtain:

$$w_t^* = -\frac{\sum_{n \in I_t} g_n}{\sum_{n \in I_t} h_n + \lambda}$$

- Plugging w_t^* into (5) gives

$$L^* = -\frac{1}{2} \sum_{t=1}^{T} \frac{\left(\sum_{n \in I_t} g_n\right)^2}{\sum_{n \in I_t} h_n + \lambda} + \gamma T + const(f_m)$$

Split finding

- In optimized loss we have fixed optimal weight w_t^*
- Optimized loss can also be used as impurity function in greedy one-step-ahead tree building.
- define I - indexes of objects in the node, being split into left and right node
    - define I_L, I_R - indexes of objects inside left and right node
    - using L^* the split is found to maximize the gain:

$$gain = -L^*_{left} - L^*_{right} + L^*_{initial} \to \max_{feature,\, threshold}$$

which is equal to

$$\frac{1}{2}\left[\frac{\left(\sum_{n\in I_L} g_n\right)^2}{\sum_{n\in I_L} h_n + \lambda} + \frac{\left(\sum_{n\in I_R} g_n\right)^2}{\sum_{n\in I_R} h_n + \lambda} - \frac{\left(\sum_{n\in I} g_n\right)^2}{\sum_{n\in I} h_n + \lambda}\right] - \gamma$$

Additional extensions of xgBoost

- Shrinkage in xgBoost is the same as in usual boosting
- Subsampling is possible:
    - over objects
    - over features
- Approximate split finding possible
    - suppose N (number of objects) is large.
    - for continuous feature there may be up to N unique feature values.
    - instead of looking through all unique values, it is possible to look through fixed number of percentiles:
        - found once and for all nodes
        - or recalculated at each node

Conclusion

- xgBoost is very successful gradient boosting open source implementation
- tree construction is not tied to specific criteria (entropy, gini) but is adapted to final user loss function
- optimized loss function has regularization, penalizing complex base learner trees.
- it is possible to optimize through a representative subset of feature values instead of all feature values by looping through percentiles.