Gradient Boosting and related methods. Analysis and applications.

Angel Dominguez
angel.dominguez@cimat.mx

CIMAT

Abstract

In this work, I synthesize some papers regarding to gradient boosting and related methods like xgboost, a powerful machine learning technique for efficiently solving some classic problems with state of the art performance.

Theory

Consider a training set of N pairs of data (x_i, y_i) for $1 \leq i \leq N$. Our aim is to find a function F such that it minimizes

$$E_{y,\mathbf{x}} L(y, F(\mathbf{x})) \tag{1}$$

where the loss function L represents the quality of the prediction.

As it is proposed in [1] , we can restringe the space of functions for F by using a sum of functions $h(\mathbf{x}; \mathbf{a}_m)$. Estimating the joint distribution of x and y only with the training set, the following approximation of (1)

$$\sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m h(x_i; a_m)) \tag{2}$$

Now, using the "greedy-stagewise" strategy proposed in [1] , which is taking $F_0(x)$ as the best constant predictor that minimizes $\sum_{i=i}^{N} L(y_i, \rho)$, and defining for $m = 1, 2, \ldots, M$

$$(\beta_m, \mathbf{a}_m) = \underset{\beta, \mathbf{a}}{\text{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i; \mathbf{a})) \tag{3}$$

and take $F_m(x) = F_{m-1}(x) + \beta_m h(\mathbf{x}, \mathbf{a}_m)$.
In equation (2) , since y_i are fixed, the predictor vector $(F_m(x_1), \ldots, F_m(x_n))$ determine the value of the loss function. Then, using the gradient descent strategy we want to move in the direction of the gradient vector. This define the gradient boosting algorithm (1). Using trees, we have $F_m(x) = F_{m-1}(x) + \rho_m \sum_{j=1}^{J} b_{jm} 1(x \in R_{jm})$, then by defining $\gamma_{jm} = \rho_m b_{jm}$ it implies

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm}) \tag{4}$$

And we have then J functions to add, every one of them can be maximized individually, as we can see in algorithm (2). The stochastic version of this algorithm uses only a proportion of the data for the calculations at each iteration, see [2].
XGBoost is a regularized version of gradient boosting, all the details of its development are in [3]. Let the tree obtained in the m-th iteration be f_m. The penalization of this tree with J_m leafs and weights vector w_m is $\gamma J_m + \frac{1}{2}\lambda ||\mathbf{w}_m||^2$. Now, let $I_j = \{i | x_i \in R_{jm}\}$ and

$$g_i = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{5}$$

$$h_i = \left[\frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2}\right]_{F(x)=F_{m-1}(x)} \tag{6}$$

$H_j = \sum_{i \in I_j} h_i$ and $G_j = \sum_{i \in I_j} g_i$ we can choose the weights vector by minimizing the second order taylor approximation of the regularized loss function, which is

$$\sum_{j=1}^{J_m} [G_j w_{jm} + \frac{1}{2}(H_j + \lambda) w_{jm}^2] + \gamma J_m \tag{7}$$

by taking $w_{jm} = -\frac{G_j}{H_j + \lambda}$. And the optimal value for (7) is

$$-\frac{1}{2}\sum_{j=1}^{J_m} (\frac{G_j^2}{H_j + \lambda}) + \gamma J_m \tag{8}$$

Let $I = I_j$ for some j, let's make a criteria for doing a split over I. If the split of I makes I_L, I_R and defining G_L, G_R, H_L, H_R, G, H analogously to (5) and (6) then, the reduction of (8) by doing that split is

$$\frac{1}{2}[\frac{G_L^2}{H_L^2 + \lambda} + \frac{G_R^2}{H_R^2 + \lambda} - \frac{G^2}{H^2 + \lambda}] - \gamma \tag{9}$$

Finally taking into account sparse data cases, we can determine the best direction for the missing values as we can see in algorithm (3)

Algorithms

Algorithm 1 Gradient Boosting

$F_0(x) = \text{argmin}_\rho \sum_{i=i}^{N} L(y_i, \rho)$
for $m = 1$ to M do: **do**
  $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, 1 \leq i \leq N$
  $\mathbf{a}_m = \text{argmin}_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(x_i; \mathbf{a})]^2$
  $\rho_m = \text{argmin}_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i, \mathbf{a}_m))$
  $F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m)$
end for

Algorithm 2 Gradient Tree Boosting

$F_0(x) = \text{argmin}_\rho \sum_{i=i}^{N} L(y_i, \rho)$
for $m = 1$ to M do: **do**
  $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, 1 \leq i \leq N$
  Build regression tree for the values $\tilde{y}_i$. C Creating the regions R_{jm} for $j = 1 \ldots, J_m$
  For $j = 1 \ldots, J_m$, calculate $\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$
  $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$
end for

Algorithm 3 Sparsity-aware Split Finding

Input: I, indexes of the observations in the current node.
Input: I_k, indexes of the observations in the current node without missing values in the feature k.
Input:d, feature dimension
score $\leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to d do: **do**
  //missing value goes to right
  $G_L \leftarrow 0, H_L \leftarrow 0$
  for j in $ascentSorted(I_k$,by $x_{jk})$ **do**
    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
    $score \leftarrow max(score, \frac{G_L^2}{H_L^2+\lambda} + \frac{G_R^2}{H_R^2+\lambda} - \frac{G^2}{H^2+\lambda})$
  end for
  //missing value goes to left
  $G_R \leftarrow 0, H_R \leftarrow 0$
  for j in $descentSorted(I_k$,by $x_{jk})$ **do**
    $G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$
    $score \leftarrow max(score, \frac{G_L^2}{H_L^2+\lambda} + \frac{G_R^2}{H_R^2+\lambda} - \frac{G^2}{H^2+\lambda})$
  end for
end for
Output: split with maximum score and directions of the missing values

Applications

From [1]
Least-squares regression:

$$L(y, F) = \frac{(y - F)^2}{2} \tag{10}$$

Least-absolute-deviation regression:

$$L(y, F) = |y - F| \tag{11}$$

M-regression

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \frac{\delta}{2})|y - F| > \delta \end{cases} \tag{12}$$

Two-class logistic regression and classification

$$L(y, F) = log(1 + exp(-2yF)) \tag{13}$$

Multi-class logistic regression and classification

$$L(\{y_k, F_k(x)\}_1^K) = -\sum_{k=1}^{K} y_k log p_k(x) \tag{14}$$

Applications (ranking)

From [4]
Pointwise approach.

$$L^r(\{(x_i, y_i)\}_1^N, F) = \sum_{i=1}^{N} (F(x_i) - y_i) \tag{15}$$

Pairwise approach.

$$L^p(\{(x_i, y_i)\}_1^N, F) = \sum_{i,s | y_i < y_s} (\phi(F(x_i) - y_i)) \tag{16}$$

Listwise approach.

$$L^l(;F) = \sum_{s=1}^{N-1} [-F(s_{p_s}) + log(\sum_{i=s}^{N} exp(F(x_{p_i})))] \tag{17}$$

XGBoost library

xgboost is a library available in R, some useful parameters are:

- booster: The default is gbtree, boosting using trees.
- eta: It's the shrinkage rate η (or ν in notation of 2).
- gamma: Penalization for the number of leafs.
- lambda: Penalization of the sum of weights of the leafs λ. By taking gamma=0 and lambda=0 we are using gradient boosting or stochastic gradient boosting.
- subsample: Proportion of the data used for adjusting the model at each iteration. If subsample< 1 we have stochastic gradient boosting.

Some loss functions are

- reg:squarederror for Least-squares regression
- binary:logistic obtains the probabilities for the two-class logistic regression and classification
- multi:softmax for multiclass classification. It returns the most probable class.
- multi:softprob similar to softmax, it returns the matrix of probabilities.
- rank:pairwise for ranking using the pairwise approach.

As referred in [5] xgboost can automatically do parallel computation on Windows and Linux and supports customized objective functions. A full list of parameters can be found in [6]

References

[1] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of statistics, 1189-1232.

[2] Friedman, J. H. (2002). Stochastic gradient boosting. Computational statistics & data analysis, 38(4), 367-378.

[3] 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). ACM.

[4] Chen, W., Liu, T. Y., Lan, Y., Ma, Z. M., & Li, H. (2009). Ranking measures and loss functions in learning to rank. In Advances in Neural Information Processing Systems (pp. 315-323).

[5] Chen, T., He, T., Benesty, M., Khotilovich, V., & Tang, Y. (2015). Xgboost: extreme gradient boosting. R package version 0.4-2, 1-4.

[6] https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst
Last accessed July 30, 2019

See full report (in spanish) at: