CSCI-SHU 360 Machine Learning
Solution to homework 4

Yufeng Xu yx3038@nyu.edu

April 16, 2024

1 Programming Problem: Random Forests

1.1

rmse model data	RF	least square	ridge($\alpha = 0.5$)
train	3.737	4.821	4.822
test	4.197	5.209	5.187

Table 1: The training and test RMSE of random forest, least square regression, and ridge regression on Boston housing price dataset.**RF outperforms both least square and ridge regression on training and test RMSE.**

1.2

accuracy data split	credit risk	breast cancer
train	77.43%	98.74%
test	73.00%	95.91%

Table 2: The training and test accuracy of random forest on bad credit risk and breast cancer prediction.

2 Programming Problem: Gradient Boosting Decision Trees

2.1

For a tree of depth d, it has at most $2^d - 1$ nodes. For each node, there are m choices of feature dimension and n_j choices of threshold.

A naive way to find the best feature and the threshold (p_j, τ_j) is: we first sort all the data points by a candidate feature p'_j. and try all thresholds $\tau'_j = \frac{1}{2}\left(x_{p_j}^{(i)} + x_{p_j}^{(i+1)}\right)$. For each τ'_j, we have $G_L = \sum_{\{i|x_{p_j}^{(i)} \le \tau_j\}} g_i$ and $G_R = \sum_{\{i|x_{p_j}^{(i)} > \tau_j\}} g_i$; $H_L = \sum_{\{i|x_{p_j}^{(i)} \le \tau_j\}} h_i$ and $H_R = \sum_{\{i|x_{p_j}^{(i)} > \tau_j\}} h_i$. We compute the gain based on G_L, G_R, H_L, H_R and compare it with the best gain so far.

Therefore, for every candidate (p_j, τ_j) on node j, we need to compute the corresponding gain, which takes $O(n_j)$ time. There are $O(mn_j)$ combinations of (p_j, τ_j), so for one node the time complexity is $O(mn_j^2)$.

For all the possible depths $d' \in [0, d)$, we have $\sum_{j,\text{depth}(j)=d'} n_j = n$. We know $\frac{1}{2}n^2 \leq \sum_{j,\text{depth}(j)=d'} n_j^2 < n^2$. Therefore, $\sum_{d'=0}^{d-1} \cdot \sum_{j,\text{depth}(j)=d'} m \cdot n_j^2 = O(n^2md)$. Therefore, the computational complexity is $O(n^2md)$.

2.2

As stated in 2.1, the most computationally expensive part in GBDT training (and also in other decision tree algorithms) is the pick of (p_j, τ_j), which takes $O(n_j^2 m)$ time for a single node. While different p_j's can be tested in parallel, we suggest a method can improve the efficiency of choosing τ_j given p_j without parallelism.

Suppose data split D on node j has size n_j; the candidate feature is p_j. Then we have k possible thresholds where $k \leq n-1$. Suppose the thresholds satisfy $\tau_{p_j}^{(1)} < \cdots < \tau_{p_j}^{(k)}$, $G_L^{(t)} = \sum_{\{i | x_{p_j}^{(i)} \leq \tau_j^{(t)}\}} g_i$, $t = 1, \ldots, k$. We observe that

$$\begin{aligned}
G_L^{(t+1)} &= \sum_{\{i | x_{p_j}^{(i)} \leq \tau_j^{(t+1)}\}} g_i \\
&= \sum_{\{i | x_{p_j}^{(i)} \leq \tau_j^{(t)}\}} g_i + \sum_{\{i | \tau_j^{(t)} < x_{p_j}^{(i)} \leq \tau_j^{(t+1)}\}} g_i \\
&= G_L^{(t)} + \sum_{\{i | \tau_j^{(t)} < x_{p_j}^{(i)} \leq \tau_j^{(t+1)}\}} g_i
\end{aligned}$$

Similarly, $G_R^{(t+1)} = G_R^{(t)} - \sum_{\{i | \tau_j^{(t)} < x_{p_j}^{(i)} \leq \tau_j^{(t+1)}\}} g_i$. The same observation holds for H_L and H_R as well.

In practice, we sort the data split D by dimension p_j, and reorder g and h to match the sorted data points (takes $O(n_j \log n_j)$ time). As a result, each time we only need to compute the difference between $G_L^{(t)}$ and $G_L^{(t+1)}$ to obtain $G_L^{(t+1)}$, and examining k thresholds only takes $O(n_j)$ time.

Therefore, the new time complexity is $O(\sum_{d'=0}^{d-1} \sum_{j,\text{depth}(j)=d'} mn_j \log n_j)$. For a given layer d', we have $n(\log(n) - d') < \sum_{j,\text{depth}(j)=d'} n_j \log(n_j) < n \log(n)$, therefore, the new time complexity is $O(nmd \log n)$.

2.3

There are parts in GBDT we can compute in parallel: (1) the evaluation of different nodes on the same level; (2) the different candidate features dimensions given a node; (3) the different thresholds given a feature dimension.

In our practice, we parallelize the evaluation of different candidate features dimensions given a node. In Python, we take the evaluation of each feature dimension(decision rule) as an individual function, and use **multiprocess.Pool.starmap()** to execute the functions in parallel.

2.4

rmse \ model / data	RF	least square	ridge($\alpha = 0.5$)
train	2.154	4.821	4.822
test	3.785	5.209	5.187

Table 3: The training and test RMSE of GBDT, least square regression, and ridge regression on Boston housing price dataset. **GBDT outperforms both least square and ridge regression on training and test RMSE.**

2.5

accuracy data / split	credit risk	breast cancer
train	77.57%	99.50%
test	73.67%	95.91%

Table 4: The training and test accuracy of GBDT on bad credit risk and breast cancer prediction.

2.6

By comparing Table 1 and Table 3, Table 2 and Table 4, we observe that GBDT outperforms RF on all the datasets.

A possible explanation is that while GBDT corrects the errors made by previous trees along the direction of the gradient, RF corrects the error randomly by randomly sampling data points from the training set. As a result, directed correction brings better accuracy to the model compared to random correction.

Another important factor is that we conducted parameter-tuning for GBDT, including tuning of the number of trees to prevent over-correction, and tuning of the learning rate to obtain an optimal optimization result.