Movie Rate Prediction Based on XGBoost

Hui Cai
Pei Gu
Wu Xia

January 8, 2019

0.1 Model and algorithm

0.1.1 Model

Here we are dealing with a supervised learning problem, where we could use the training data which may contain several features to predict the target variable. The objective function includes the loss function estimating the error and regularization term to avoid overfitting.
For n samples each with d features, we define $\phi(\boldsymbol{x})$ below as our prediction function, each f_k represents a decision tree in forest $\mathcal{F}$.

$$\phi(\boldsymbol{x}_i) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i), \quad f_k \in \mathcal{F}$$

Here $\mathcal{F} = \{f(\boldsymbol{x}) = w_{q(\boldsymbol{x})}\}$ is the set of trees (i.e. forest), K is the number of trees we use to predict; for the expression $f_k(\boldsymbol{x}) = w_{q_k(\boldsymbol{x})}$, $w_k \in \mathbb{R}^{T_k}$ is the leaf weight of tree k; T_k is the number of leaves in tree k and $q_k : \mathbb{R}^d \to T_k$ maps feature $\boldsymbol{x}$ to a specific leaf.
For the next step, we can choose reasonable loss function to minimize the objective function $\mathcal{L}$ which has form as follows:

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

where $\Omega(f_k) = \gamma T_k + \frac{1}{2}\lambda ||w||^2$.
Here we use MSE loss function, where Ω represents the regularization term, which helps to smooth the learned parameters and thus avoid overfitting. Note that $\mathcal{L}(\phi)$ includes functions as parameters and cannot be optimized using traditional convex optimization, hence we used the following iterative approach to update the parameters:
Let $\hat{y}_i^{(t)}$ be the prediction of the i - th instance at the t - th iteration. Then we greedily add f_t that most improves the model at next iteration:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

According to Tianqi[13], we could use the second order approximation to quickly acquire an optimal to the objective above

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t)$$

where,

$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}, h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2}$$

Let $I_j = \{i|q(x_i) = j\}$ be the sample set of leaf j, we would rewrite $\tilde{\mathcal{L}}^{(t)}$ as the following

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Then for each structure $q(x)$, we could compute the optimal weight w_j^* of leaf j

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

and the corresponding optimal value

$$\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2}\frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda}$$

This optimal value would be used as the score function for the algorithm to measure the quality of a tree structure q, which is similar as the impurity score for a simple decision tree.

0.1.2 Algorithm

Here is the detailed process for XGBoost model algorithm: The idea for XGBoost algorithm is firstly we use only one tree to learn a regression predictor, then we compute the error residual and add an additional tree to learn to predict the residual. The error rate is calculated using the parameters mentioned below. We repeat the steps above until error estimate is small enough.
One of the key problems in tree learning is to find the best split for one tree. In order to do so, a split finding algorithm enumerates over all the possible splits on all the features. We call this the exact greedy algorithm. It is computationally demanding to enumerate all the possible splits for continuous features. Here we attach the pseudo code for it:

With exact greedy algorithm for finding best split methods for each tree, we can construct the whole XGBoost algorithm. Here we attach pseudo code for XGBoost algorithm:

Note here γ is the minimum required structure score, L is the maximum number of levels in the tree, l is the current level of the tree, λ is the learning rate, N is the number of training steps.

Algorithm 1: Exact greedy algorithm for split finding used in our price prediction model.

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0, H_L \leftarrow 0$
  for j *in sorted (I, by* $\boldsymbol{x_{jk}}$*)* **do**
    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
    $score \leftarrow max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: eXtreme Gradient Boosting

Input: $(x_i, y_i)_1^n$ /*the labeled training data*/
Input: $(\gamma, L, l, \lambda, N)$
Output: A tree which is configured to predict the class label of a test sample
$l \leftarrow l + 1$;
if $l \leq L$ **then**
  $t \leftarrow 0$; $f \leftarrow 0$;
  while $t < N$ **do**
    estimate f_t as a regressor function, density, or distribution;
    $f \leftarrow f + f_t$
  end
  initialize array of scores S[:];
  ***ApplyAlgorithm* 1 :** greedy algorithm for computing max score;
  $C_L, C_R \leftarrow \boldsymbol{MaxGain\,((x_i, y_i)_1^n, S)}$;
  /*C_L, C_R are the left and right children of this node respectively*/
  if C_L *does not satisfy the desired level of purity* **then**
    GradientBoostedTree$(C_L, \gamma, L, l, \lambda, N)$;
  end
  if C_R *does not satisfy the desired level of purity* **then**
    GradientBoostedTree$(C_R, \gamma, L, l, \lambda, N)$;
  end
end