ML and Numerical Software Development

Machine Learning-III

Organon Analytics

January 7, 2020

Agenda

- Model Averaging
- Gradient Boosting Machines: The Algorithm

Model Averaging

- The goal: To obtain the best predictions by combining different models
- Train several different models separately
- Use different algorithms, different levels of model complexity
- Build an ensemble model: Some kind of averaging should be employed

The idea/hope: If the errors produced by each model are independent, averaging the predictions will reduce the error

Model Averaging: Bagging

Bagging means boot-strapped aggregation. It works as follows:

- Given a dataset $\mathcal{D}$ of sample size N
- Generate bootstrapped samples of size N by sampling from D uniformly and with replacement: $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_M$
- Build model for each data-set
- Generate the final prediction as a simple average of all predictions produced:

$$\hat{Y}_{\text{bagged}} \equiv \frac{1}{M}(\hat{Y}_1 + \hat{Y}_2 + \cdots, \hat{Y}_M)$$

- Works well for high variance models
- Use it for small datasets, and large-parameter models

Model Averaging: Random Forests

Pseudo-code

1. Given a dataset $\mathcal{D}$ of sample size N

- Generate bootstrapped samples of size N by sampling from D uniformly and with replacement: $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_K$
- Randomly sample m columns out of M columns. The optimal value for m could be found with hyper-parameter optimization
- Build a CRT model for each data-set
- Generate the final prediction as a simple average of all predictions produced:

$$\hat{Y}_{\mathrm{RF}} \equiv \frac{1}{K}(\hat{Y}_1 + \hat{Y}_2 + \cdots, \hat{Y}_K)$$

Model Averaging: Boosting

As in Random Forests, the final model is an additive function of individual models:

$$F(X) \equiv \sum_i F_i(X)$$

The way $F(X)$ is built is different from the way a Random Forest is built:

1. $F_i(X)$ is built sequentially: each F_i is built after F_{i-1}
2. Each F_i learns the residual after F_{i-1} added to the ensemble
3. Each F_i could be a tree, a linear regression model, ANN, etc. Trees are chosen for their simplicity and (ease&speed) of build

Let's derive the algorithm (The XGBoost version) $\Rightarrow$

XGBoost derivation

$$
\begin{aligned}
y_i &: \quad \text{output for the sample i} \\
\hat{y}_i^t &: \quad \text{estimate for the sample i at iteration t} \\
\hat{y}_i^t &\equiv \sum_{j=1}^{t} f_j(x_i) \\
\hat{y}_i^t &= \hat{y}_i^{t-1} + f_t(x_i)
\end{aligned}
$$

The total-loss at the iteration-t is given as follows

$$\mathcal{L}^t \equiv \sum_i \mathcal{L}(y_i, \hat{y}_i^t) = \sum_i \mathcal{L}(y_i, \hat{y}_i^{t-1} + f_t(x_i))$$

The goal: How can we produce $f_t(\cdot)$ as a function of x so as to minimize $\mathcal{L}^t$

Let's try to approximate $\mathcal{L}^t$ to the second order $\Rightarrow$

XGBoost derivation

Remember Taylor's theorem from the calculus

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + O((x - x_0)^3)$$

This is a good approximation to the 3rd order as as long as $f^3(x_0)$ is "small"
Now, expand the loss function at iteration-t around $x_0 = \hat{y}^{t-1}$, with $(x - x_0) \equiv f_t \Rightarrow$

$$
\begin{aligned}
\mathcal{L}^t &\approx \sum_i \left[\mathcal{L}(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] \\
g_i &= \frac{\partial \mathcal{L}(y, \hat{y})}{\partial \hat{y}} \,|_{\hat{y}=\hat{y}_i^{t-1}} \\
h_i &= \frac{\partial^2 \mathcal{L}(y, \hat{y})}{\partial \hat{y}^2} \,|_{\hat{y}=\hat{y}_i^{t-1}}
\end{aligned}
$$

Hence g_i is the gradient(1st derivative), and h_i is the hessian(2nd derivative) of the loss with respect to the approximating function $\hat{y}$ evaluated at its latest value $\hat{y}^{t-1}$.

Notes:

- Instead of finding a minima for $\mathcal{L}^t$, we find a minimum of its approximation. If the approximation is good, the minima of both will be close
- We have to parameterize $f_t(\cdot)$, and set the first derivative of $\mathcal{L}^t$ wrt to these parameters to find the minima

Now, the first term $\mathcal{L}(y_i, \hat{y}_i^{t-1})$ does not depend on f_t. So, we need to minimize the quantity

$$\tilde{\mathcal{L}}^t = \sum_i \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right]$$

Let's assume $f_t(\cdot)$ is a regression tree. Hence

$$f_t(x_i) = \sum_{j=1}^{T} w_j I_{A_j}(x_i)$$

where w_j is the prediction at node j, with the rule A_j. Re-arrange the loss $\tilde{\mathcal{L}}^t$ as follows wrt this representation:

$$\begin{aligned}
\tilde{\mathcal{L}}^t &= \sum_{j=1}^{T} \left[\sum_{i \in A_j} (g_i w_j + \frac{1}{2} h_i w_j^2) \right] \\
\tilde{\mathcal{L}}^t &= \sum_{j=1}^{T} \left[(\sum_{i \in A_j} g_i) w_j + \frac{1}{2} (\sum_{i \in A_j} h_i) w_j^2 \right]
\end{aligned}$$

$$
\begin{aligned}
\tilde{\mathcal{L}}^t &= \sum_{j=1}^{T}\left[(\sum_{i\in A_j} g_i)w_j + \frac{1}{2}(\sum_{i\in A_j} h_i)w_j^2\right] \\
\tilde{\mathcal{L}}^t &= \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}H_j w_j^2\right] \\
G_j &= \sum_{i\in A_j} g_i,\ \ H_j = \sum_{i\in A_j} h_i
\end{aligned}
$$

Setting derivative of $\tilde{\mathcal{L}}^t$ wrt w_j to zero, we get:

$$
\begin{aligned}
w_j^{\text{min}} &= -\frac{G_j}{H_j} \\
\tilde{\mathcal{L}}^t_{\text{min}} &= -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j}
\end{aligned}
$$

Where's the catch? We still don't know the partitions A_j created by our imaginary tree $\Rightarrow$

Build the tree by CRT with the split criterion:

$$-\frac{1}{2}\frac{G_j^2}{H_j}$$

Hence, the best split for a given input maximizes

$$\begin{aligned}
\mathcal{L}_{\text{split}} &= \left(\frac{G_{\text{Left}}^2}{H_{\text{Left}}} + \frac{G_{\text{Right}}^2}{H_{\text{Right}}}\right) - \frac{G_{\text{Parent}}^2}{H_{\text{Parent}}} \\
G_{\text{Left}} &= \sum_{i \in \text{Left}} g_i \\
H_{\text{Left}} &= \sum_{i \in \text{Left}} h_i
\end{aligned}$$

Now, let's workout $\{g_i, h_i\}$ for common loss functions $\Rightarrow$

Gradients and Hessians for Least Squares Loss

$$
\begin{aligned}
\mathcal{L}(y, \hat{y}) &\equiv \frac{1}{2}(y - \hat{y})^2 \\
g_i &= \frac{\partial \mathcal{L}(y, \hat{y})}{\partial \hat{y}} \Big|_{\hat{y}=\hat{y}_i^{t-1}} \\
g_i &= -(y - \hat{y}) = -\epsilon_i \\
h_i &= 1
\end{aligned}
$$

Hence, for the least squares error, $f_t(x_i)$ is the result of fitting a function to the residuals generated by $y_{t-1}(x_i)$

Gradients&Hessians for binary classification with log-likelihood loss

The output takes binary values in the set $\{0, 1\}$

$$
\begin{aligned}
p &\equiv \frac{1}{1 + e^{-\hat{y}}} \\
\mathcal{L}(y, \hat{y}) &\equiv -(y \log p + (1 - y) \log (1 - p)) \\
g_i &= p_i - y_i = -\epsilon_i \\
h_i &= p_i(1 - p_i)
\end{aligned}
$$

The pseudo-code for XGBoost could be written as $\Rightarrow$

1. Initialize f_0 as a constant. e.g., for least squares loss, $f_0 = \bar{y}$
2. Calculate gradients, and hessians $\{g_i, h_i\}$ for each sample
3. Build a binary CRT with the following split criterion at each node:

$$\mathcal{L}_{\text{split}} = \left(\frac{G^2_{\text{Left}}}{H_{\text{Left}}} + \frac{G^2_{\text{Right}}}{H_{\text{Right}}} \right) - \frac{G^2_{\text{Parent}}}{H_{\text{Parent}}}$$

4. Update $\hat{y}$ with the new update f_t as follows:

$$\hat{y} = \sum_{i=1}^{t-1} f_i + \gamma f_t$$

5. Go back to Step-2, and iterate until the loss in validation set starts to increase

Notes, and some important implementation details follow $\Rightarrow$

- The original GBM as formulated by Friedman used only gradient information g_i. Each f_t is an approximation of the gradient. This is a first order approximation for the update to $\hat{y}_{t-1}$. Remember the Gradient Descent update for finding the minimum of univariate function $f(x)$

$$x_t = x_{t-1} - \gamma f'(x_{t-1})$$

- A better approximation for the update is provided by incorporating second derivative of the function. This is a second order method for optimization.

$$x_t = x_{t-1} - \gamma \frac{f'(x_{t-1})}{f''(x_{t-1})}$$

- Hence, XGBoost tries to approximate the Newton-step (g_i/h_i)

- γ is called the *learning rate*. Remember that we had a lot of approximations and they hold if f_t is small. f_t is the right direction to take, but we do not take the full step so as not to violate the approximations
- γ is the most important parameter of the GBM algorithm. How is γ determined? It is a hyper-parameter and determine via a separate validation set
- γ does not need to be a constant
- Introducing *stochasticity* to ML algorithms almost always improve the results. Two kinds of stochasticity are used:
- Columns and rows are randomly sampled at each model-build step. These 2 sampling parameters are hyper-parameters of the algorithm
- If a CRT is used as the base learner, the depth of the tree, or alternatively, the number of nodes is a hyper-parameter of the algorithm

- How many individual models f_t should be fitted? This is also a hyper-parameter of the algorithm. A large GBM model might consist of thousands of individual models
- Regularization could be employed to reduce model complexity. Putting an L_2 constraint on the weights in each node changes the split criterion as follows:

$$
\begin{aligned}
\text{Penalty} &= \lambda_2 \sum_j w_j^2 \\
\mathcal{L}_{\text{split}} &= \left(\frac{G_{\text{Left}}^2}{H_{\text{Left}} + \lambda_2} + \frac{G_{\text{Right}}^2}{H_{\text{Right}} + \lambda_2} \right) - \frac{G_{\text{Parent}}^2}{H_{\text{Parent}} + \lambda_2}
\end{aligned}
$$

- Regularization parameters are also hyper-parameters

- Hyper-parameter optimization is necessary to get high accuracy GBM models
- Note that, the reduced loss function could also be written as follows:

$$\tilde{\mathcal{L}}^t = \sum_i \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right]$$

$$\tilde{\mathcal{L}}^t = \sum_i h_i \left(f_t(x_i) + \frac{g_i}{h_i} \right)^2$$

Hence, any regression algorithm that models $\frac{g_i}{h_i}$ with case-weights h_i could be used as the base learner

Primary requirements for GBM software development

- The algorithm should be able to process data residing in SQL database tables. The first SQL database will be Postgre-SQL. The addition of new SQL databases should have minimum over-head and should not exceed 2-3 man-day effort
- The algorithm should be fast. Hyper-parameter optimization must be done in order to get a good model. Typically, one needs to run hundreds of models with different hyper-parameters. The algorithm should be able to finish modelling for a 1m row, 1000 column data-set in a couple of hours
- A random search methodology should be used for hyper-parameter optimization
- An initial list of hyper-parameters is given as follows:
    - The learning rate
    - Number of nodes in each tree
    - Row sampling rate
    - Column sampling rate
    - Number of models that will be built

- The software should conform with the decription of the XGBoost algorithm as outlined in the paper "XGBoost.pdf"
- The modelling and scoring services should be separate
- Both in-database scoring and in-memory scoring functionality should be provided
- The software should be able to build regression, binary classification, and multinomial classification models
- All configuration parameters should be able to provided externally in JSON format
- Logging should be provided. Logging should provide detailed information about run-time events. Logging should be parametrically directed to either a file or a database table
- The algorithm should accept an optional metadata table that gives further information about data fields, e.g. missing values

- The software should validate the contents of the configuration file(s) and should log about the errors before the computation begins
- The software should be able to handle a maximal dataset of 1-million rows and 5,000 columns in-memory
- A model documentation must provide model performance, variables used, importance of each individual models
- Progress of the algorithm should be documented as the computation proceeds. Specifically, model performance over various datasets should be logged as the computation proceeds
- The software should accept as degree-of-parallelism parameter that will dictate the amount of in-memory parallelism that will be used
- The software should allow for L_1 and L_2 regularization