On the Training and Result Analysis of the Classification Algorithm XGBoost

36282705, 36202620, 36347797, 36228559

***Abstract*—In this paper, we have decided to go beyond the tree and boosting techniques we have learned in the course and combine these two techniques to explore XGBoost and gradient boosting decision tree (GBDT) for the analysis of COVID datasets for comparison. These two techniques have been widely used in many machine learning domains and have achieved good results, with XGBoost being a regular winner in many competitions. For the COVID dataset, XGBoost slightly outperformed GBDT in terms of average performance and model training time, while GBDT outperformed XGBoost in terms of accuracy and precision in the binary category of life and death prediction.**

Key Words: Classification, Boosting, XGBoosting, Gradient boosting decision tree

I. Introduction

Tree method is a powerful recursive construction method. There are a number of algorithms based on this method, for example, classification tree (decision tree), which is a very important and widely used classification tool [1]. And boosting, another tree based model, is a way of combining many weak models by iteratively modeling the residuals obtained from fitting a model and provides an alternative to bagging. Such as XGBoost (extreme gradient boosting tree) [2], which is a very popular method nowadays. In this project, we first studied the mathematical principle of XGBoost (Basic learner, objective function and its optimization, regularization coefficient, etc.), then implemented it by code (fitting and reasoning Covid-19 dataset with XGBoost model), and compared and analyzed the results obtained from the same operation on the same dataset with the GBDT algorithm which has been learned in the course, so as to obtain the advantages and disadvantages of the two methods and the applicability in different scenarios.

II. Methods

For this essay dataset, two methods, GBDT and XGBoost, are used in this paper.

A. Gradient Boosting Decision Tree

There are two main types of decision trees, classification trees and regression trees. Classification trees are used to classify tagged values; regression trees are used to predict real values. The difference between the two is that the result of a classification tree cannot be added or subtracted, while the result of a regression tree is a prediction of a value and can be added or subtracted. The decision tree in GBDT is a regression tree and the predicted outcome is a numerical value

Boosting is a family of algorithms that boosts weak learners into strong learners, and is part of the ensemble learning category, based on the idea that for a complex task, an appropriate combination of multiple experts' judgments is better than any one of them alone. The Gradient Boosting Machine (GBM) is a learning machine based on the gradient boosting algorithm. In theory, a GBM can choose from a variety of learning algorithms as its base learner, and GBDT is actually a case of GBM. Liang et al [3] mentioned GBDT as one of the ensemble learning algorithms, which combines multiple decision trees into one powerful classifier. A decision tree can be thought of as a collection of if-then rules that are easy to understand, highly interpretable and fast to predict. At the same time, decision tree algorithms require less feature engineering than other algorithms, e.g. they can be used without feature normalization, they can handle data with missing fields well, and they can be used without caring whether features are interdependent or not. Decision trees are able to combine multiple features automatically. However, the decision tree algorithm has the disadvantage of being prone to overfitting when used alone. Fortunately, the problem of overfitting can be solved by various methods that suppress the complexity of the decision tree, reduce the fitting ability of a single decision tree, and then integrate multiple decision trees by means of gradient boosting. This shows that the gradient boosting method and the decision tree learning algorithm can complement each other's strengths and are a perfect match.

Based on the research from Cheng et al [4], the algorithm for GBDT is roughly as below:

1) Initialize the residuals to form weak learner 1.
2) Calculate the residuals.
3) Find the best division point (threshold) for the regression tree.
4) Update the dichotomised residuals to the actual values and calculate the average of the actual values as the residuals. This constitutes the weak learner 2.
5) Merge the strong learners. (Weak learner 1 + weak learner 2)
6) Iterate to stop when the condition is met.

According to Liang et al [3], the final mathematical equation of the model can be written as follows:

$$f_n(x) = f_{(n-1)}(x) + r_n T(x_i; \alpha_n) \tag{1}$$

Where r_n represents the learning rate and $T(x_i; \alpha_n)$ represents a single CART regression tree.

B. XGBoost

1) Introduction of XGBoost:

XGBoost (extreme gradient boosting tree) is one of the boosting algorithms [5]. The basic idea of XGBoost is consistent with what has been mentioned previously, it integrates

many weak classifiers together to form a strong classifier. In XGBoost, the weak classifiers or so-called basic learners are CARTs.

a) CART regression: CART regression trees are assumed to be binary trees [6], by continually splitting the features. For example, the current tree node is split based on the j-th feature value and let the samples with that feature value less than s be divided into left subtrees and those greater than s be divided into right subtrees. We simply traverse all cut points of all features to find the optimal cut feature and cut point. The result is a regression tree. In XGBoost, it is assumed that, there are parametres:

1) Samples: $x_1, x_2, ..., x_M$.
2) Features of samples: $X_1, X_2, ..., X_m$
3) Values of nodes: $w_1, w_2, ..., w_n$
4) Sample collection in nodes: $I_1, I_2, ..., I_n$

The mathematical model of the tree, $q(x_i)$, links the sample with the serial number of the node, for example, if $q(x_i) = k$, the $w_{q_{(x_i)}} = w_k$, and $I_j = \{i|q(x_i) = j\}$

b) XGBoost model expression: In XGBoost, the model can be described as below, $\hat{y_i}^{(M)}$ here is the predicted value of the model, which is formed by the addition of M base learners, in which a base learner $f_j(x)$ is a CART, or it can also be understood as an additive model, that is, the addition of M-1 base learners plus the M-th base learner:

$$\hat{y_i}^{(M)} = \sum_{j=1}^{M} f_j(x_i) = \sum_{j=1}^{M-1} f_j(x_i) + f_i^{(M)}(x_i) \tag{2}$$

$$f_i^{(M)}(x_i) \ \ is \ just \ the \ w_{q_{(x_i)}} \tag{3}$$

c) XGBoost objective function : The XGBoost objective function is defined as below [2]:

$$Obj^{(t)} = \sum_{i=1}^{N} L\left(y_i, \hat{y}^{(t)}\right) + \sum_{j=1}^{t} \Omega(f_j) \tag{4}$$

The goal is to find argmin Obj, which is a list of values of nodes.

$$(w_1, w_2, w_3, \ldots\ldots, w_j) = argmin \ Obj^{(t)} \tag{5}$$

What makes the XGBoost special is that in its objective function, it has two parts, the first part measuring the difference between the predicted score and the true score, and the other part being the regularization term, Ω, which is defined as below.

$$\Omega(f_j) = \gamma T + \frac{1}{2}\lambda \|\omega\|^2 \tag{6}$$

γ is used to punish the number of leaf nodes to avoid the tree being too deep, λ is used to punish the node value to avoid too high a value, and both regular terms are used to avoid over fitting. The regularization term also contains two components, T for the number of leaf nodes and for the fraction of leaf nodes. controls the number of leaf nodes and controls the fraction of leaf nodes from being too large to prevent overfitting.

d) Taylor Expansion: In order to simplify and facilitate the calculation, second-order Taylor expansion is used to process the objective function, the loss function is transformed as below:

$$\begin{aligned} L\left(y_i, \hat{y_i}^{(t-1)} + w_j\right) \approx & L\left(y_i, \hat{y_i}^{(t-1)}\right) \\ & + L^{'}\left(y_i, \hat{y_i}^{(t-1)}\right) w_j \\ & + \frac{1}{2} L^{''}\left(y_i, \hat{y_i}^{(t-1)}\right) w_j^2 \end{aligned} \tag{7}$$

Since the L, $L^{'}$, and $L^{''}$ are constant, g_i and h_i are used to replace $L^{'}$, and $L^{''}$, the function is further adjusted as:

$$L\left(y_i, \hat{y_i}^{(t-1)} + w_j\right) \approx g_i w_j + h_i w_j^2 \tag{8}$$

Therefore, the final function of objective function is:

$$Obj^{(t)} \approx \gamma T + \sum_{j=1}^{T}\left(w_j G_j + \frac{1}{2} w_j^{\ 2}(\lambda + H_j)\right) \tag{9}$$

$$G_j = \sum g_j, H_j = \sum h_j \tag{10}$$

e) Optimal solution: As has been mentioned previously, the goal is:

$$(w_1, w_2, w_3, \ldots\ldots, w_j) = argmin \ Obj^{(t)} \tag{11}$$

Consider what has been obtained with Taylor expansion:

$$w_j = argmin \ \left(w_j G_j + \frac{1}{2} w_j^2 (\lambda + H_2)\right) \tag{12}$$

To obtain optimal solution, take the derivative of w_j and make the derivative equal to 0, then the optimal solution is obtained:

$$Obj^{(t)} = \min Obj^{(t)} \approx \gamma T - \frac{1}{2}\sum_{j=1}^{T} \frac{G_j}{H_{j+}\lambda} \tag{13}$$

2) Flow of XGBoost:

a) greedy algorithm: The XGboost algorithm involved in this paper involves the idea of greedy algorithm when splitting tree nodes. Greedy algorithm refers to always try to choose the best choice at present when solving the problem. What the algorithm gets is the local optimal solution of the current part of the problem. The key is to choose a measurement standard according to the requirements. The calculation results obtained by the greedy algorithm may not be able to obtain the optimal solution of the whole problem, and the greedy strategy used requires no aftereffect (that is, the process after a certain state will not affect the previous state, but only related to the current state) [7].

b) Description of the process of XGBoost:

$$\Omega(f_j) = \gamma T + \frac{1}{2}\lambda \|\omega\|^2 \tag{14}$$

$$minObj^{(t)} \approx \gamma T - \frac{1}{2}\frac{G_j^2}{H_j + \lambda} \tag{15}$$

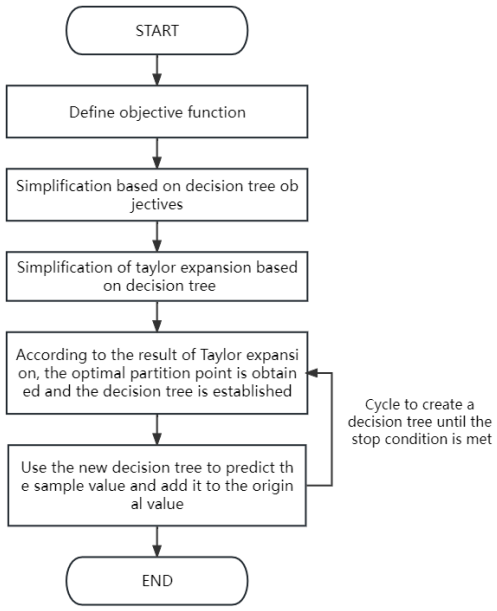

Fig. 1. XGBoost Flowchart

c) Pseudocode of Greedy Algorithm: The generation process of a tree can be seen as a process of continuous splitting of stages. For each node waiting for splitting, $g_i$is the constant of the first power term in the second order expansion of the objective function, h_i is the constant of the quadratic term in the second order expansion of the objective function, so the Obj (before splitting) of the node can be calculated. First, according to the feature order, X_1, X_2....., X_n, and then traverse the values x_{11}, x_{12}, x_{13}..... in each feature.Next, split them by value, and its Obj (after splitting) can also be calculated. Then, according to gain = Obj (before splitting – Obj (after splitting)), calculate the gain under all splitting conditions, and then select the largest one as the final splitting result. This idea of focusing on the local optimal solution uses the idea of greedy algorithm. The node splitting will stop when:

1) When the max gain obtained by this split is less than a very small number
2) Number of samples included in leaf node ¡= 1
3) Too high level (risk of over fitting)

The basic idea of greedy algorithm is:

1) Construct a mathematical model to explain the problems to be solved
2) The problem of the required solution is decomposed into several subproblems
3) Solve all subproblems of the problem to obtain the local optimal solution of a single part
4) The combination of these single local optimal solutions is one of the solutions of the initial problem

3) Optimization:

a) Shrinkage: According to Dhaliwal et al. [8]presented that the regularization of XGBoost is very important to help mitigate the problem of overfitting the data. XGBoost proposes this methods to prevent overfitting: Shrinkage. shrinkage

Algorithm 1 Exact Greedy Algorithm for Split Finding

```
input:I,instance set of current nodes
input: m, feature dimension
gain = 0
G = Σ_{i∈I} g_i,  H = Σ_{i∈I} h_i
for k = 1 to m do
    G_L = 0, H_L = 0
    for j in sorted (I, by x_jk) do
        G_L = G_L + g_j,  H_L = H_L + h_j
        G_R = G + G_L,  H_R = G + H_L
        Score = max (score, G_L^2/(H_L+λ) + G_R^2/(H_R+λ) − G^2/(H+λ) )
    end for
end for
output:split with max score
```

multiplies the fraction of each leaf node of the tree by a reduced weight η in each iteration, which makes each tree less influential, leaving more room for later generated trees to optimize the model. The formula of shrinkage as follow:

$$\hat{y}_i^t = \hat{y}_i^{(t-1)} + \eta f_t\left(x_i\right) \tag{16}$$

b) Optimization of sampling: There are three main optimizations for sampling:

1) Column sampling, random selection of several features rather than all features.
2) The eigenvalues are divided into buckets. Randomly sample the eigenvalues of each feature, that is, buckets. We use the method of evenly dividing buckets to ensure that the number of samples in each bucket is basically the same, so as to ensure that the weights are as consistent as possible.
3) Weighted quantile method: Transform the loss function. Note that a constant is added in the third step, but it does not affect the operation of the loss function. Finally, an expression is obtained, which has a weight, that is h_i:

$$\begin{aligned}
&\sum_{i=1}^{N}\left(g_i f_t\left(x_i\right)+\frac{1}{2}h_{\ i} f_t^2\left(x_i\right)\right)+\Omega\left(f_t\right) \\
&=\sum_{i=1}^{N}\frac{1}{2}h_{\ i}\left(2\frac{g_i}{h_i}f_{\ t}\left(x_i\right)+f_t^2\left(x_i\right)\right)+\Omega\left(f_t\right) \\
&=\sum_{i=1}^{N}\frac{1}{2}h_{\ i}\left(2\frac{g_i}{h_i}f_{\ t}\left(x_i\right)+f_t^2\left(x_i\right)\right)+\Omega\left(f_t\right) \\
&=\sum_{i=1}^{N}\frac{1}{2}h_{\ i}\left(2\frac{g_i}{h_i}f_{\ t}\left(x_i\right)+f_t^2\left(x_i\right)\right)+\Omega\left(f_t\right)
\end{aligned} \tag{17}$$

To prevent over fitting, η is usually taken as 0.1.

Column Subsampling is like selecting some of the features for tree building in a random forest. It can be divided into two types, one is random sampling by layer, where a random portion of features are selected before splitting each node within the same layer, and then only this portion of features need to be traversed to determine the optimal splitting point. The other is random selection of features, where a random

selection of features is made before the tree is built and then only those features are traversed before the split. In general, the former works better.

III. Experiment/Results

To test the performance of XGBoost classifier and learn more about XGboost classifier, Kaggle's Covid-19 dataset is used for training prediction in this report. Not only that, but this report also uses GBDT classifier to compare with XGBoost classifier. The dataset consists of 1,048,576 unique patients and contains 21 features, including sex, age, covid classification, patient type, pneumonia, pregnancy, diabetes, chronic obstructive pulmonary disease, asthma, immunosuppressed, hypertension, cardiovascular, renal chronic, other disease, obesity, tobacco, medical units of the first, second or third level, medical unit, intubed, ICU and date died. This report uses these features to predict whether the patient is in high risk or not.

The data set needs to be preprocessed before classification and prediction. First, this report needs to deal with the null values in the dataset. More than 80 percent of the ICU and intubed features in the dataset were null, and this report decided to remove these two features. The pregnant features data is 50% null. After searching, most of the null values are male, so this report decides to set the null value of this feature as not pregnant. Other characteristic hollow values are less than 5 percent. Data containing null values is selected to be deleted in this report. Finally, the age feature of the data set is standardized to complete the data set preprocessing.

Ten-fold cross validation is used to evaluate the classifier performance in this report. And the accuracy, recall, precision, f1 score and training time is used as the evaluation index. The accuracy, recall, precision, f1 score and training time of GBDT and XGBoost are shown in Table 1.

TABLE I
Average Evaluation Index of GBDT and XGBoost

Table Head	**Evaluation Index**				
	Accuracy	***Recall***	***Precision***	***F1***	***Training Time***
GBDT	92.71	50.00	57.30	48.11	108.
XGBoost	87.45	59.58	49.00	57.85	30.30

As shown in Table 1, GBDT has higher accuracy and precision than XGBoost, but XGBoost's recall and F1 score are higher than GBDT. Meanwhile, XGBoost has less training time.

GBDT versus XGBoost:

- While GBDT uses CART as the base classifier, XGBoost supports multiple types of base classifiers, such as linear classifiers.
- When using CART as the base classifier, XGBoost explicitly adds regular terms to control the complexity of the model, which helps to prevent overfitting and improve the generalization ability of the model.
- GBDT only uses the first derivative information of the cost function in model training, and XGBoost performs second order Taylor expansion of the cost function, which can use the first and second derivatives at the same time
- While traditional GBDT uses all data in each round of iteration, XGBoost adopts a strategy similar to random forest to support data sampling and column sampling, which can not only reduce overfitting, but also reduce calculation, which is also a feature of XGBoost different from traditional GBDT.
- XGBoost supports parallelism
- Traditional GBDT is not designed to handle missing values, and XGBoost can automatically learn its split direction. XGBoost learns a default split direction up front for missing values.

IV. Discussion

During the project, we found that most of the articles tend to compare bagging and boosting, and the model used in this study is based on boosting. In the future, this study can be used to further explore bagging, which is also one of the ensemble learning methods. After this experiment, XGBoost and GBDT, which are also Ensemble learning methods, are very different from each other. According to the results, GBDT slightly outperformed XGBoost in terms of accuracy, however, the average performance of the latter was better than that of the former.

Through the study of XGBoost, our team has further understood and mastered the boosting technology. In the process of research, we encountered the problem that XGBoost algorithm formula is difficult and slow to understand. After discussion among team members and continuous study of relevant papers, we overcame this difficulty.

REFERENCES

[1] Y.-Y. Song and L. Ying, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, no. 2, p. 130, 2015.

[2] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, no. 5, 2016, pp. 785–794.

[3] W. Liang, S. Luo, G. Zhao, and H. Wu, "Predicting hard rock pillar stability using gbdt, xgboost, and lightgbm algorithms," *Mathematics*, vol. 8, no. 1, p. 765, 2020.

[4] J. Cheng, G. Li, and X. Chen, "Research on travel time prediction model of freeway based on gradient boosting decision tree," *IEEE access*, vol. 7, no. 2, pp. 7466–7480, 2018.

[5] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 3, pp. 1–4, 2015.

[6] B. Li, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees (cart)," *Biometrics*, vol. 40, no. 4, pp. 358–361, 1984.

[7] D. Geling and D. Geling, "Detailed explanation of csp-j2019 simulation certification test 2," *Fujian Computer*, vol. 35, no. 6, p. 6, 2019.

[8] S. S. Dhaliwal, A.-A. Nahid, and R. Abbas, "Effective intrusion detection system using xgboost," *Information*, vol. 9, no. 7, p. 149, 2018.