Gene expression Value prediction based on XGBoost Algorithm

Abstract

Gene expression profiling has been widely used to characterize cell status to reflect the health of the body, to diagnose genetic diseases, etc. In recent years, although the cost of genome-wide expression profiling is gradually decreasing, the cost of collecting expression profiles for thousands of genes is still very high. Considering gene expressions are usually highly correlated in humans, the expression values of the remaining target genes can be predicted by analyzing the values of 943 landmark genes. Hence, we designed an algorithm for predicting gene expression values based on XGBoost, which integrates multiple tree models and has stronger interpretability. We tested the performance of XGBoost model on the GEO dataset and RNA-seq dataset and compared the result with other existing models. Experiments showed that the XGBoost model achieved a significantly lower overall error than the existing D-GEX algorithm, linear regression, and KNN methods. In conclusion, the XGBoost algorithm outperforms existing models and will be a significant contribution to the toolbox for gene expression value prediction.

Introduction

Characterizing gene expression patterns in cells under various conditions is an important problem. Gene expression profiling is a vital biological tool commonly used to capture the response of cells to disease or drug treatments Although the cost of gene expression profiling is steadily decreasing in recent years, it is still very expensive when dozens or hundreds of samples need to processed. Inspired by this, many scholars have suggested that the expression value of the landmark gene can be used to predict the expression value of the target gene, which will

greatly reduce the cost of the gene expression profiling. In 2016, Yifei Chen et al. proposed the D-GEX algorithm based on Back Propagation neural network (Chen et al., 2016), in which 943 landmark genes correspond to 943 input units, and 9,520 target genes correspond to 9,520 output units. However, the prediction accuracy of this algorithm still has a large room for improvement. Besides, deep network has poor interpretability, and for each target gene, we cannot know which landmark genes have much greater impact on its expression. Last but not the least, deep network needs to read all the data into the memory at the time of training, and therefore, the algorithm is prone to occupy excessive memory in actual use, and has high demand for GPU too. In addition to deep network, some researchers also used linear regression, KNN and other classical algorithms for target gene expression prediction (Chen, 2014), but the prediction results of these algorithms were less accurate. Among the Boosting Tree models, XGBoost (Chen and Guestrin, 2016) has a very strong expansion and flexibility. It integrates multiple tree models to build a stronger learner model. Furthermore, XGBoost is characterized by its ability to automatically use the multithreading of the CPU for parallel computing, which can speed up the calculation. Based on the above research background, we proposed a new gene expression value prediction algorithm based on XGBoost, and established a regression prediction model for each target gene independently. The results showed that the XGBoost algorithm significantly improved the prediction accuracy, which is superior to D-GEX, LR, KNN, and other algorithms. It also had better predictive ability and generalization ability. Lastly, the XGBoost algorithm had stronger interpretability than other algorithms.

XG-Boost

XGBoost (Extreme Gradient Boosting) is a model that was first proposed by Tianqi Chen and Carlos Guestrin in 2011 and has been continuously optimized and improved in the follow-up study of many scientists (Chen and Guestrin, 2016). The model is a learning framework based on Boosting Tree models.

The traditional Boosting Tree models uses only the first derivative information. When training the nth tree, it is difficult to implement distributed training because the residual of the former n-1 trees is used. XGBoost performs a second-order Taylor expansion on the loss function and it can automatically use the multithreading of the CPU for parallel computing. Besides, XGBoost uses a variety of methods to avoid overfitting. The XGBoost algorithm is briefly introduced as follows (Chen and Guestrin, 2016), and the details are given in the Supplementary Material. Integrate the tree model with addition method, assuming a total of K trees, and use F to represent the basic tree model, then: y f ˆi k ( ) x f i k , F k K = ∈ = ∑1 (1) The objective function is: L l y y f i i k i k = + ∑ (ˆ ,) ∑Ω( ) (2) where l is the loss function, which represents the error between the predictive value and the true value; Ω is the function used for regularization to prevent overfitting: Ω( )f T = + γ λ w 1 2 2 (3) where T represents the number of leaves per tree, and w represents the weight of the leaves of each tree. After the second-order Taylor expansion of the objective function and other calculations which are detailed in Supplementary Material, we can finally get the information gain of the objective function after each split is: Gain g h g h i i I i i I i i I i i I L L R R = + + + + ∈ ∈ ∈ ∈ ∑ ∑ ∑ ∑ 1 2 2 2 ( ) ( ) λ λ ( ) g h i i I i i I 2 ∈ ∈ ∑ ∑ + − λ γ (4) As can be seen from (4), in order to suppress the growth of the tree and prevent the model from overfitting, a splitting threshold γ is added. The leaf node is allowed to split if and only if the information gain is greater than γ. This is equivalent to pre-pricing the tree while optimizing the objective function. In addition, we also used the following two excellent techniques of XGBoost to avoid overfitting in the experiment: 1. If all sample weights on the leaf nodes are less than the threshold, the splitting is stopped. This prevents the model from learning special training samples. 2. Sample features randomly when building each tree. These methods all make XGBoost more generalizable and get better performance in practical applications. In the experiment, the regression model based on XGBoost was independently trained for each target gene, and the number of input landmark genes was 943, which means the input feature dimension was 943, and this dimension is very high. However, many techniques in XGBoost for avoiding overfitting can help reduce the degree of overfitting and improve the accuracy of regression prediction. When the XGBoost model was actually used in the experiment, the following parameters were adjusted to make the model perform its best

performance: 1. n_estimators n_estimators is the number of iterations in training. A too small n_estimators can lead to underfitting, which makes the model not fully perform its learning ability. However, a too large n_estimators is usually not good either, because it will cause overfitting. 2. min_child_weight As we mentioned earlier, min_child_weight defines the sum of sample weight of the smallest leaf nodes to prevent overfitting. 3. max_depth It is the maximum depth of the tree. The greater the depth of the tree, the more complex the tree model is, and the stronger the fitting ability is, but at the same time, the model is much easier to overfit. 4. subsample This parameter means the sampling rate of all training samples. 5. colsample_bytree The last parameter that we need to config is colsample_bytree. It is the feature sampling rate when constructing each tree. In this task, this is equivalent to the sampling rate of the landmark gene. 6. learning_rate In most algorithms, learning rate is a very important parameter that needs to adjust, as well as in XGBoost. It greatly affects the performance of the model. We can reduce the weight of each step to make the model more robust. The details of parameters configuration were introduced in Section

Results

In this section, we firstly introduced the process of parameters configuration of XGBoost algorithm and its high interpretability. Then, we showed the results of XGBoost model on both the GEO data and the GTEx data, and compared it with the previous methods. Tuning Model Parameters GridSearchCV, a sub-module of the sklearn module in Python (Pedregosa et al., 2011), was used in the experiment to conduct grid search on all parameters to find the optimal parameters. The details of the tuning parameters are shown in Table 1: Take the target gene CHAD for example, we established its XGBoost regression model. We initialized all the parameters of the model as shown in the above Table 1, and adjusted them in order. Firstly, we adjusted n_estimators, and the absolute error of CHAD gene changes with n_estimators as shown in Figure 2 below: It can be seen that the absolute error of the validation set did not decrease after 350 iterations, and in order to prevent overfitting, the optimal value of n_estimators was set as 350. Update the value of n_estimators to 350 and adjust the next parameter γ, Table 2 shows the absolute error of validation set corresponding to different γ values. As can be seen from Table 2, 0.1 is the optimal value of γ. Then, we adjust the remaining parameters in turn, and we can finally get optimal values of all the parameters as shown in Table 3. Using the optimal parameters in Table 3, the absolute error of CHAD on validation set is 0.1513 and is 0.1518 on test set. It can be seen that after the

configuration of parameters, performance of the model was improved. Therefore, parameter adjustment is helpful for improving the accuracy. In addition, XGBoost is highly interpretable. After the tree model is created, the importance score for each feature can be obtained directly.

TABLE 1 | Detailed parameters configuration.

Parameters	Initialization value	Search space
n_estimators	300	[300, 330, 350, 370, 400]
γ	0	[0, 0.1, 0.2, 0.3, 0.4]
min_child_weight	1	[1, 2, 3, 4, 5, 6]
max_depth	5	[6, 7, 8, 9, 10, 11]
subsample	0.6	[0.6, 0.7, 0.8, 0.9]
colsample_bytree	0.8	[0.6, 0.7, 0.8, 0.9]
learning_rate	0.1	[0.01, 0.05, 0.08, 0.1]

TABLE 3 | Optimal values of all parameters.

Parameters	Optimal value
n_estimators	350
γ	0.1
min_child_weight	1
max_depth	8
subsample	0.8
colsample_bytree	0.8
learning_rate	0.1

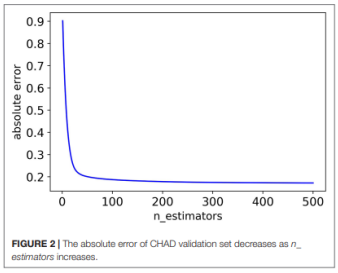

FIGURE 2 | The absolute error of CHAD validation set decreases as *n_estimators* increases.

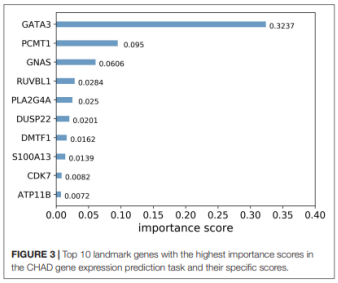

FIGURE 3 | Top 10 landmark genes with the highest importance scores in the CHAD gene expression prediction task and their specific scores.

TABLE 2 | Absolute errors of validation set corresponding to different γ.

γ	Absolute error
0	0.1712
0.1	**0.1701**
0.2	0.1709
0.3	0.1718
0.4	0.1709
0.5	0.1714

The figure in bold represents the lowest absolute error.

and optimized parameters according to the performance on the validation set. Finally, we evaluated the prediction ability of various models according to their performance on the test set.

For each target gene t, we define the Mean Absolute Error as follows:

$$MAE_{(t)} = \frac{1}{N}\sum_{i=1}^{N}\left|y_{i(t)} - \hat{y}_{i(t)}\right| \quad (8)$$

TABLE 3 | Optimal values of all parameters.

Parameters	Optimal value
n_estimators	350
γ	0.1
min_child_weight	1
max_depth	8
subsample	0.8
colsample_bytree	0.8
learning_rate	0.1

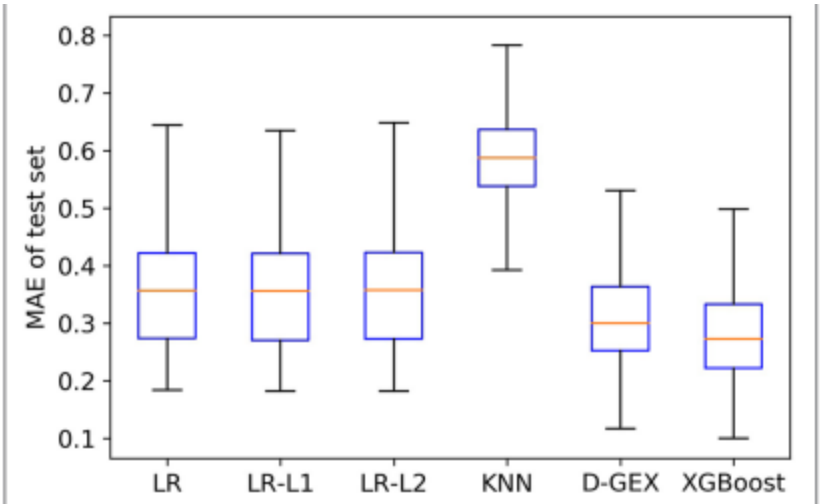

FIGURE 4 | The Mean Absolute Error (MAE) distribution boxplot of the six algorithms on the test set.