Multi-Agent Reinforcement Learning Trees

1129503
College of Science and Technology, Wenzhou-Kean University

Abstract

Finding the best classifier in function space is an NP-Hard Problem. In this paper, we combine the trees-based algorithm with Reinforcement Learning, called Multi-Agent Reinforcement Learning Trees. RLT treats the trees as agents, giving them more opportunities to explore better classifiers in the function space with efficient guidance. We also introduce the experience playback pool to increase the consistency of training. Compared with Decision Tree, Random Forest, Gradient Boosting Decision Trees, and AdaBoost, RLT shows excellent performance on sci-kit learn basic datasets [Pedregosa et al.2011] and UCI datasets.

introduction

Given a dataset, the accurate classification of tags by features has been a puzzle in data science for decades. Decision trees [Quinlan1986], as the basic learning machine of bagging and boosting algorithms, have been widely improved and applied in the field of data analysis. The famous improved algorithm contains ID3, C4.5, and CART. [Freund, Schapire, and Abe1999] Those algorithms aim to generate smaller classifier with the better property. However, when they use the improved algorithm to avoid overfitting, a single decision tree tends to have a large error in fitting the datasets. Therefore, Breiman in 2001, generates the random forest algorithm, which combines all of the trees, deciding by voting. To increase the generalization ability of the classifier, each decision tree adopts the random sampling with placement back and the tree construction scheme selected from the feature subset. Since each tree is built independently, the random forest can use the parallelization scheme to establish, which makes the running speed of random forest much faster than other schemes that also use multiple decision trees. Another famous multiple decision trees algorithm uses boosting method. In Boosting method, the establishment of each decision tree depends on the residual of the previous one, that is forward stage-wise additive model. For example, Gradient Boosting Decision Trees uses the negative gradient of the loss function to learn the deviation part of the existing learner. AdaBoost [Freund, Schapire, and Abe1999] algorithm increases the weight of misclassified points. In 2016, Tianqi, proposed XGBoost [Chen and Guestrin2016] algorithm. It introduces the regular term into the objective function and uses Taylor's second-order expansion to approximate the objective function. The article also proposed a decision tree generation strategy based on a greedy algorithm and uses shrinkage and column sampling methods to prevent overfitting. This algorithm shines in industrial applications.

Observing the existing tree algorithms, their tree-building strategies are all one-off. In other words, these building strategies can not be learned repeatedly. Therefore, these learners often fall into a locally optimal solution. Since finding the optimal classifier is an NP-Hard problem, it also shows that the learning method based on a fixed strategy can not escape the local optimal solution. Therefore, to give the classifier more exploration possibilities, we introduce reinforcement learning.

Reinforcement learning has five basic elements, including environment, agent, state, action, and reward. It aims to train the agent to perform more effective actions in a specific environment. Deepmind [Mnih et al.2013] at 2013 using Deep reinforcement learning exceeded the best human level. Observing the construction process of the decision tree, we found that it has the basic elements of reinforcement learning. Therefore, we proposed a multi-agent reinforcement learning algorithm.

We mainly used the Q-learning [Watkins and Dayan1992] to explore a better classifier. To address the overestimation of this algorithm, we introduced the experience replay to give the agents, decision tree, efficient guidance. At the beginning of the training of the reinforcement learning tree, the feature with the largest information gain is selected as the split point with a high probability, and then as the exploration value gradually decays, it turns to select features from the Q-table. And finally, this algorithm shows excellent performance compared with the traditional trees algorithm. The principal contributions are as follows:

- We proposed the Multi-Agents Reinforcement Learning algorithm, trying to find a better classifier for given datasets in the field of classification. And we increased about 2%-5% accuracy in sci-kit learned basic datasets.
- We design an efficient system, which treats the tree as an agent, the selection of features as actions, the Gini information gain [Raileanu and Stoffel2004] as the reward for

each step, the optimal tree in the experience playback pool as the reward for each stage, the datasets as the environment, and the number of times each feature is selected as the state.

- We introduced the experience replay pool to increase the speed of convergence.

Related Work

In this part, we will mainly analyze the development of various algorithms based on decision trees. Firstly, we will introduce the basic algorithms including ID3, C4.5, and CART. Based on a single decision tree model, follow-up development algorithms include random forest [Breiman2001], GBDT [Friedman2001], AdaBoost and XGBoost

Decision Tree ID3, C4.5 and CART ID3 is a greedy algorithm, which builds a decision tree that can completely classify samples. Therefore, it has a high degree of fitting to the original datasets, which leads to worse performance in the case of generalization ability.

Consider a original simple$\{(x_i, y_i)\}_1^N$.The total entropy is $-\sum_{k=1}^{K} a_k \log a_k$, where $a_k = \frac{|C_k|}{|D|}$, and $|C_k|$ Indicates the number of samples belonging to the k-th category in all samples D, $|D|$ indicates the number of total samples. Therefore, the entropy of the total sample can be written as:

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|} \tag{1}$$

where $|D_i|$ indicates the number of samples in the i-th subset, $|D_{ik}|$ indicates the number of samples belonging to the k-th category in the i-th subset. Therefore, the formula for information gain can be written as:

$$g(D, A) = H(D) - H(D \mid A) \tag{2}$$

where $H(D \mid A) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log \frac{|D_{ik}|}{|D_i|}$

In C4.5 algorithm, in order to eliminate the information gain brought by the sample distribution after the feature, the algorithm introduces the information gain ratio:

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)} \tag{3}$$

where $H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}$, A is the selected feature. Finally C4.5 chooses the feature with the largest information gain ratio.

CART algorithm is the most popular one not only for decision tree but also for the bagging [Breiman1996] ans boosting algorithm [Freund, Schapire, and Abe1999]. Compared with two algorithm above, CART algorithm builds the binary tree, which reduced the problem of overfitting. CART uses the Gini gain, the approximation of information gain formula:

$$Gini(D) = \sum_{k=1}^{K} p_k \left(1 - p_k\right) \tag{4}$$

After choosing the feature A, the original datasets are splited into two new datasets, so that the total Gini index is:

$$Gini(D, A) = \frac{|D_1|}{|D|} Gini\left(D_1\right) + \frac{|D_2|}{|D|} Gini(D2) \tag{5}$$

where D1 and D2 is two new datasets.

Random Forest Random forest uses decision trees as the base classifier, tries to increases the difference between decision trees through random replacement sampling and feature subset selection. This algorithm ultimately enhances the generalization ability. Breiman also proves that the generalization error tends to a finite upper limit.

Consider k weak classifiers,
$h = \{h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_K(\mathbf{x})\}$.
Define the margin function:

$$mg(\mathbf{x}, y) = av_k I\left(h_k(\mathbf{x}) = y\right) - \max_{j \neq Y} av_k I\left(h_k(\mathbf{x}) = j\right) \tag{6}$$

where I is indicator function and av_k means average.

Then define the generalization error:

$$PE^* = P_{X,Y}(mg(\mathbf{x}, y) < 0) \tag{7}$$

Finally, it proves that given a datasets x and , as the number of decision trees become larger, the generalization error will converge:

$$\frac{1}{M} \sum_{m=1}^{M} I\left(h\left(\mathbf{x}, \mathbf{\Theta}_m\right) = j\right)$$

$$\rightarrow \textstyle\sum_k P_\Theta\left(\phi\left(\Theta_m\right) = k\right) I\left(\mathbf{x} \in S_k\right) \tag{8}$$

$$= F_\Theta(h(\mathbf{x}, \Theta) = j)$$

AdaBoost Adaptive Boosting algorithm generates the strong classifier for 0-1 classification by combining the several weak classifier.
Given a training datasets:
$\left\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\right\}, y_i \in \{-1, +1\}$
The weight of m-th classifier is defined as follows:

$$\alpha_m = \frac{1}{2} \log \frac{1 - r_m}{r_m} \tag{9}$$

where $r_m = \sum_{i=1}^{n} P\left(f_m\left(x_i\right) \neq y_i\right) =$
$\sum_{i=1}^{n} w_{mi} I\left(f_m\left(x_i\right) \neq y_i\right)$ and $f_m(x)$ is the basic classifier. This algorithm Give more weight to misclassified points.

GBDT Gradient Boosting Decision Tree aims to fit the residual of the previous tree. The loss function of the gradient boosting tree is defined as follows:

$$\tilde{y}_i = -\left.\frac{\partial L\left(y_i, F(x)\right)}{\partial F(x)}\right|_{F(x) = F_{m-1}(x)} \tag{10}$$

And then we use the m-th tree to fit the $\tilde{y}_i$. Therefore, we can get the model:

$$F_m(x) = F_{m-1}(x) + f_m^\star(x) = F_{m-1}(x) + T_m\left(x; \theta_m^\star\right) \tag{11}$$

XGBoost Compared with the gradient boosting tree, XGBoost adds a regular term to the objective function to balance the complexity of the model. Its expression is as follows:

$$obj = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k=1} \Omega\left(f_k\right) \tag{12}$$

The regular term improves the generalization ability of the model by giving an additional penalty to the tree size. In order to get the efficient feedback from objective function. This algorithm uses Taylor second-order expression to approximate the objective function:

$$\text{Minimize }_{obj} = \sum_{i=1}^{n} \left[g_i \cdot f_k\left(x_i\right) + \frac{1}{2} h_i \cdot f_k^2\left(x_i\right) \right] + \Omega\left(f_k\right) \tag{13}$$

where g_i and h_i respectively represent the first and second derivatives of the loss function of the $k-1th$ decision tree.

Through the above algorithm, we can find that the existing trees-based algorithm performs a greedy search on a fixed strategy, which inevitably leads to the final classifier falling into the local optimal solution. This is why we introduced reinforcement learning.

Reinforcement Learning Tree Algorithm

Multi-Agent Reinforcement Learning Algorithm combines mainly the random forest algorithm. It builds a binary tree according to the Q-table.

RLT Framework

The multi-agent reinforcement learning tree algorithm allows each classifier, that is, the agent, to continuously learn the tree-building strategy in several rounds to achieve better classification results. Meanwhile, the strategy of the algorithm is temporal differential learning.

$States$: The combination of the number of times each feature is selected.Each tree is independent. Assuming there are a total of n optional features, then each number in parentheses represents the number of times each feature has been selected. It can be written as $(0_1, ..., 0_n)$.

$Actions$: The characteristics of each tree to be selected in the next step.

$Transition$: After selecting feature m in state n, the state changes from $S_n(s_0, ..., s_m, ..., s_n)$ to $S_{n+1}(s_0, ...,\text{s+1}_m, ..., s_n)$

$Rewards$: The Gini gain of the data set after selecting a feature for each step. After every 50 episodes, a reward of 100 will be given to the optimal path from the experience replay pool.

Initial the State-Action Function

The action value function is called Q-table. Action and value determine a specific Q value, we express it as $Q(s, a)$. When initializing Q-table, we take a random number between $(0, 2)$ as the initial value.

Reinforced algorithm

In this article, we use the learning function of Q-learning to update the Q-table. This is a time-series difference algorithm that uses an offline learning strategy. The function of Q-learning is expressed as follows:

$$Q\left(s_t, a_t\right) = (1-\lambda) \cdot Q\left(s_t, a_t\right) +$$

$$\lambda \cdot \left[r\left(s_t, a_t\right) + \gamma \max_{a'} Q\left(s_{t+1}, a'\right)\right] \tag{14}$$

where λ is the learning rate. $r\left(s_t, a_t\right)$ is the reward at (s_t, a_t), γ is the discount for future.

When selecting features in reinforcement learning, there will be a certain probability for random selection to ensure that the generated learner will not fall into the local optimum. And with the iteration of episode, the probability of exploration δ will gradually decay. The dacay rate is β.

Experience Replay Pool [Schaul et al.2015]

After the experience replay pool stores a certain number of trees and establishment paths, the Q-table is updated by testing the learning effect in the experience replay pool. Because not every exploration is successful, if the Q-table is updated every episode Trees are established, its order will be destroyed, and the convergence speed and learning effect will be affected. Therefore, we introduce the experience replay pool to avoid this problem.

Experimental Result

Experiment Setup

The experiments environment is Inter® i9-10850K 10-core CPU and 32 GB RAM. The parameters of Q-learning in Reinforcement Learning Trees are set as follows: λ = 0.1, γ = 0.9, δ = 0.9, β = 0.998, episode = 2000.

Results and Comparison

This experiment compares random forest, GBDT, decision tree, and AdaBoost. The training set for this test is the scikit Learning datasets including iris, wine, and breast cancer. In addition, we also used a relative larger UCI dataset. The results show that the accuracy of the reinforcement learning tree is 2%-5% higher than that of other traditional trees based algorithms.

Conclusion and Future Work

This article proposes a new algorithm called a multi-agent reinforcement learning tree. This algorithm gives the classifier more exploration possibilities in the function space. And we use the Q-learning function to effectively guide the agents. In the end, this algorithm beats traditional machine learning on a given data set in terms of accuracy. Compared with the traditional decision tree algorithm, this algorithm allows the classifier to build the decision tree multiple times so that the classifier has more learning experience. But the multi-agent reinforcement learning tree algorithm is

Algorithm 1: Multi-Agent Reinforcement Learning Trees

Data: λ,γ,δ,β, episode.
The datasets $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$
Result: The best performance trees $f_b(x)$, update the Q-table.

initialization;
for $i = 1 : episode$ **do**
  for $j = 1 : n_estimators$ **do**
    train test split by 10:3;
    t = 0;
    while *the trees can split* **do**
      if $randomNumber(0, 1) < \delta$ **then**
        choose the max Gini gain in all of the features
      else
        choose the feature $argmaxQ(S_t, A)$
      end
      Reward = Gini gain of two datasets
      update the Q-table by:

$$Q(s_t, a_t) = (1 - \lambda) \cdot Q(s_t, a_t) + \lambda \cdot [r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')]$$

    end
    Add trees and path to remember replay pool;
    t = t+1;
  end
  if $i\%50$ = *0* **then**
    for *trees in experience replay pool* **do**
      test the accuracy of the trees;
      receive the best trees $f_b(m)$;
    end
    give the best trees the reward of 100 to the building path;
    clear the experience replay pool;
  else
    continue;
  end
  δ = δ*β
end
return $f_b(m)$

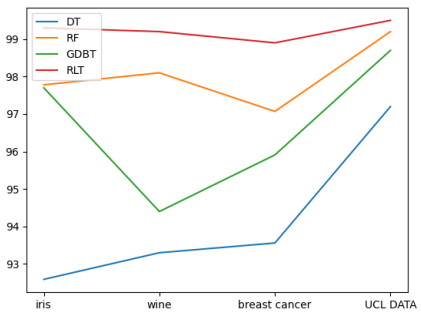

Figure 1: Result Comparison of Decision Tree, Random Forest, GBDT and Reinforcement Learning Tree

only applied to classification problems. For regression problems, when the data set is too large or there are too many features, we need to introduce deep reinforcement learning to approximate Q-value. And compared with traditional algorithms, this algorithm requires more memory space and longer learning time. Therefore, this algorithm is more practical for data sets in a specific field. People only need to learn it once to continue to use Q-table to update the establishment strategy. This also shows that its advantages in dynamic data sets cannot be replaced. And researchers can obtain the reasons for the changes in the establishment strategy, so as to make better strategic adjustments. For future work, we can make building strategies more diversified and increase the information interaction between agents. In terms of algorithms, this kind of reinforcement learning algorithm is more of an improvement on random forest. We can continue to do integration on the algorithm. Introduce the objective function strategy of XGBoost to conduct joint learning of multiple agent trees.

References

Breiman, L. 1996. Bagging predictors. *Machine learning* 24(2):123–140.

Breiman, L. 2001. Random forests. *Machine learning* 45(1):5–32.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Freund, Y.; Schapire, R.; and Abe, N. 1999. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence* 14(771-780):1612.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* 1189–1232.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in python. *the Journal of machine Learning research* 12:2825–2830.

Quinlan, J. R. 1986. Induction of decision trees. *Machine learning* 1(1):81–106.

Raileanu, L. E., and Stoffel, K. 2004. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence* 41(1):77–93.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.