January 2017.

XXX

Egor Khairullin

mikari.san@gmail.com

Abstract: The result of many machine learning algorithms are computational complex models. And further growth in the quality of the such models usually leads to a deterioration in the applying times. However, such high quality models are desirable to be used in the conditions of limited resources (memory or cpu time).

1. introduction

Today, Machine Learning (ML) is used in various fields ranging from information search in the Internet and speech recognition to optimization of the steel composition and the search for new physics. In almost all cases, resources (memory, CPU, network, etc.) are very limited to solve the task.

For the most ML algorithms, improving the quality of the model usually leads to higher levels of resource consumption. For example, with the use tree-based boosting algorithms, increasing the number of trees from tens to thousands usually leads to a significant increase in quality, while increasing the load on memory and CPU when applying such a model. Increased requirements on the hardware resources of more accurate models is usually solved by various technical optimizations of storage and application of the resulting model: for example, using vector calculations. Another approach is the modification of the learning process, withch makes it possible to obtain models with certain properties that make it possible to signigicantly accelerate their application without maintaining the required level of quality.

In this paper, we consider the modification of the learning process of predictive models by the algorithm CatBoost (Gulin, Kuralenok, and Pavlov 2011), which will accelerate the apply of the model while maintaining the required level of quality.

2. catboost

CatBoost is the implementation of a tree-based gradient boosting in which so-caled oblivious decision trees are used. In this algorithm the values of each feature i are discretized into baskets using the boundaries B_i. The boundaries themselves are determined in advance using various statistics. Source vector with real values is replaced by a vector binary values 0 and 1 ($false$ and $true$). The value of the i-th attribute f_i is replaced by the binary vector g_i where $g_{ij} = f_i > B_{ij}$. Finally, all g_i are concatenated into one vector. Consecutively, in the resuling trees instead of comparing some real feature value with some threshold there will be just boolean check.

Number of baskets could by controlled by passing the appropriate parameter during learning procedure.

Also another important parameter should be marked. It is the speed of learning. The smaller it is, the smaller the contribution of each new tree. However, this has to be compensated by a large number of trees. The speed of learning can be selected by cross-validation process with the fixed number of trees.

Usually, more trees means higher quality.

3. decision tensor

One of the simplest ways to accelerate is the preliminary calculation. The feature vector obtained after binarization takes a limited number of values. Therefore, we can precompute the prediction for all such values.

Let n be the number of features. Let us consider some model Z, trained by some algorithm, containing a set of trees T. For convenience, we assume the values of the features lie in thre interval $[0, 1]$.

Let f_i be the set of basket boundaries for the i-th feature, which was used in T. Then using the planes $X_i = b_{ij}$ for all i, j, we can split the space $[0; 1]$ int small hyperparallelepipeds. It is obvious that within each hyperparallelepiped the prediction of the model is constant. In fact, the model Z allows you to calculate the value at any point of our hypercube (or Tensor). However, if we pre-calculate the value in each piece, then we will accelerate the application of our model, reducin it only determining the desired piece. The asymptotic complexity of the application of the tree-based mode it $O(h \cdot t)$ where h is the height of the trees, t is the total number of trees. Asymptotic complexity of Tensor is $O(n \cdot log(b))$ where n is the number of features and b is the number of cuts per feature (this complexity is achieved by using binary search). It should be marked for small b simple linear search could be a bit faster that binary.

So, with a small number of features Tensor can be applied much faster than usual tree-based model.

However, Decision Tensor has large memory footpring. It needs $O(b^n)$ of memory and it may be too large for common computer. Therefore, the precomputed tensor can only be used for small b and n.

In the general, the size of one tensor can be defined as follows:

$$D = (f_1, ..., f_n)$$

$$f_i = \{b_{i1}, ..., b_{ip_i}\}$$

$$S(D) \propto \prod |f_i| = \prod p_i$$

Consider the case for small n. We can get a small b by two main ways: limit the number of cuts for each feature and train model with big enough tree count or severely limit the tree count, then the ottal number of really used cuts will be exactly $h \cdot t$ without taking into account repeated use of the same cuts.

The first method was used in (Likhomanenko et al. 2015): there was trained a CatBoost with a small number of cuts and turned into a Decision Tensor.

The consumption of memory in one hypercube can be extremely huge provded that acceptable quality is achieved, so the natural continuation is the construction of several Decision Tensors.

4. tensor similarity

Introduce the metric of the similarity of two tensors. Let's define it as the difference between the sum of their sizes and the size of their union:

$$D^1 = (f_1^1, ..., f_n^1)$$

$$D^2 = (f_1^2, ..., f_n^2)$$

$$D^u = D^1 + D^2$$

$$D^u = (f_1^u, ..., f_f^u), f_i^u = f_i^1 \cup f_i^2$$

$$Sim(D^1, D^2) = S(D^1) + S(D^2) - S(D^1 + D^2)$$

5. multicriteria optimization.

Target metrics are \$ Q \$ (model quality), \$ S \$ (size), and \$ t \$ (time of application). However, we can assume that the application time is proportional to the number of decision tensors \$ M \$. The input vector will need to describe the algorithm for obtaining trees and the algorithm for combining them into cubes. We denote it by \$ V \$. Then we solve the following problem:

$\min\{Q(V), S(V), M(V)\}$ over all possible \$ V \$.

Before we move to consider the contents of the vector \$ X \$, it should be noted that for practical application this problem can be solved using the constraint method. Usually in practice there are quite strict limitations on the size of the model and the available application time.

6. algorithm of combining a particular set of trees in a set of decision tensors.

Suppose we have N trees and we want to combine them in M of the decision tensors. The metric of optimality of the partition will be assumed to be occupied memory. In general, this is not an easy task and it requires a separate study. We used the following heuristic approach: 1) Initialize M tensors by placing in each one an arbitrarily chosen tree 2) Choose any of the remaining trees 3) Combine it with the most similar tensor 4) Repeat steps 2-3 until there are no unused trees left.

7. a complete algorithm for constructing an decision tensors ensemble with the help of catboost.

There are two key approaches:

1) We fix a very rough partition with a small number of cubes and train an arbitrarily large number of trees.

2) The partitioning is done quite detailed, but at the same time we significantly reduce the number of trained trees, thereby reducing the number of actually used buckets.

The algorithm actively uses both these ideas.

1) Determine the desired number of tensors (M) based on time constraints

2) Train the CatBoost model with a limited number of buckets

3) Unite the trees from the obtained model into the first decisive tensor from the ensemble

4) Use random search to sort through the key parameters of CatBoost

- Train the CatBoost model, using the model from 2) as the baseline
- Transform into M-1 Decision Tensors
- Remove ensembles that do not fit the required size limits.

5) We choose the ensemble with the best quality on an additional test set of data.

This algorithm allows to find the optimal Pareto ensemble of Decision Tensors.

8. application of the algorithm on the dataset

We have used the dataset and the model from [@ likhomanenko2015lhcb]. Number of features in the dataset is 10, the model was trained on 5000 trees with a number of the buckets 64. We compared our algorithm for constructing Decision Tensors Ensemble with trivial approach: training a small number of trees but with high value of learning speed. For this we divided the data set into three parts: teaching (50%), test (25%) and validation (25%). We trained the models on the training sample, then using the test we selected the best quality models under given constraints.

We have limited the number of tensor by 6 and the total size of models by one gigabyte, believing every prediction requires 4 bytes (float). We looked through only the key parameters of CatBoost about the number of trees and the speed of learning.

The area under the ROC curve where False Positive Rate less than 0.05 was taken as a metric. For convenience, we have adjusted this metric so that its maximum value (with an ideal model) is equal to 1.

9. discussion.

We compared the CatBoost model with 5000 trees ("best"), the fast CatBoost models for 1-125 trees ("fast") and the Decision Tensors Ensembles with different memory limits and different tensors numbers.

Chart 1 shows the dependence of quality on the number of lookups. Chart 2 shows the dependence of quality on the total time of application at the full validation dataset. Table 3 shows properties of the models trained in the process.

TODO: добавить графики из презентации

It can be seen that at a comparable quality with "fast" models, DTE allows a 25 times reduction in the number of lookups. And in comparison with the "best" model, the number of lookups is 1000 times smaller with a small loss in quality ($ <0.006 $ with the "best" quality of 0.6).

However, the difference in working time is not so significant - 2.5 times and 50 times, respectively. This is due to the principles of the cache and the specialty of our benchmark. Since the Decision Tensor takes up a very large amount of memory and does not fit into the cache because of this, almost every lookup results in reading from the RAM. CatBoost models are small enough, because of what can be entirely placed in the cache. Since our benchmark consists only of model applications, CatBoost models are not forced out of the cache. In real applications, the CatBoost model will most likely not own the cache alone and will be even slower because of this. And Decision Tensor Ensemble almost do not depend on the cache size.

10. conclusion.

Decision Tensor Ensbles allow you to simplify the application of complex Boosting Decision Trees algorithms (such as CatBoost) in real-time systems without the ability to use batching application. Considered algorithm for building ensembles allows you to exchange consumed memory for the speed of application with a small loss in quality. In addition, the use of the Ensemble can significantly improve the quality in comparison with the use of a single Decision Tensor.

Gulin, Andrey, Igor Kuralenok, and Dmitry Pavlov. 2011. "Winning the Transfer Learning Track of Yahoo!'s Learning to Rank Challenge with Yetirank." In Yahoo! Learning to Rank Challenge, 63–76.

Likhomanenko, Tatiana, Philip Ilten, Egor Khairullin, Alex Rogozhnikov, Andrey Ustyuzhanin, and Michael Williams. 2015. "LHCb Topological Trigger Reoptimization." In Journal of Physics: Conference Series, IOP Publishing, 082025.