Speeding up prediction performance of the BDT-based models

E. Khairullin[1,3] **and A. Ustyuzhanin**[1,2,3]

[1] Moscow Institute of Physics and Technology
[2] National Research University Higher School of Economics, 20 Myasnitskaya st., Moscow 101000, Russia
[3] Yandex School of Data Analysis, 11/2, Timura Frunze st., Moscow 119021, Russia

E-mail: mikari@yandex-team.ru, anaderi@yandex-team.ru

Abstract. The outcome of a machine learning algorithm is a prediction model. It is quite usual such models are computationaly expensive and growth of the quality of such models leads to the deterioration in the inference speed. However it is not always tradeoff between quality vs speed. In this paper we show it is possible to speed up model by using additional memory without loosing prediction quality significantly for a novel boosted trees algorithm called Catboost. The idea is to combine two approaches: training fewer trees and merging trees into kind of hash maps (DecisionTensors). The proposed method allows for pareto-optimal reduction of the computational complexity of the decision tree model with regard to the quality of the model. In the considered example number of lookups was decreased from 5000 to only 6 (speedup factor of 1000) while AUC score of the model was reduced by less than per mil.

1. Introduction

Today, Machine Learning (ML) approach is used in various fields ranging from the information retrieval in the Internet and speech recognition to optimization of the steel composition and the search for the New Physics. The resources (memory, CPU, network, etc.) are very limited to solve the task in production settings.

For the most of the ML algorithms, improving the quality of the model usually leads to a higher levels of resource consumption. For example, with the use tree-based boosting algorithms, the increase of the number of trees from tens to thousands usually leads to the significant increase in the quality and at the same time the model size and inference time also amplify. The higher demands for the hardware resources for the more accurate models are usually solved by various technical optimizations model storage mechanism. For example vectoriation of the tree weights storage can be used. Another approach is the modification of the training process that 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 training process of a predictive models by decision tree-based algorithm CatBoost [1]. Such modification will accelerate the prediction time of the model while maintaining the required level of the quality.

2. CatBoost

CatBoost is the implementation of a tree-based gradient boosting in which so-caled oblivious decision trees are used. The values of each feature i are discretized into baskets using the boundaries B_i. The boundaries themselves are determined in advance using various heuristic. Original feature 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 a binary vector g_i where $g_{ij} = f_i > B_{ij}$. Finally, all g_i are concatenated into one vector. Consecutively, the resuling decision will make bunch of boolean checks instead of comparing some real feature value with the thresholds.

Number of baskets used for the discretization could be controlled by specifying the appropriate parameter during the training procedure.

Let's consider another important training characteristic, namely the learning rate. The smaller it is, the smaller the contribution of each next tree in the ensemble. However, this has to be compensated by a large number of trees. The learning rate can be chosen by the cross-validation process given the fixed number of trees. Usually, the more trees the higher the model quality.

3. Decision Tensor

One of the simplest ways to accelerate computational performance is the preliminary calculation. The feature vector obtained after the discretization takes a limited number of values. Therefore, we can precompute the predictions for all possible values combination.

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

Let f_i be the set of basket boundaries for the i-th feature, which was used in T. Then we can split the interval $[0; 1]$ into small hyper-parallelepipeds. The prediction of the model is constant within each hyperparallelepiped. In fact, the model Z allows you to calculate the value at any point of our hypercube that we call Tensor. However, if we pre-calculate the value for each piece, then we will accelerate the application of our model, effectively reducing it to the time requited for the determination of 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 the trees. Asymptotic complexity of Tensor is $O(n \cdot log(b))$ where n is the number of the features and b is the number of cuts per feature (this complexity is achieved by using binary search). (It should be noted for small b simple linear search could be a bit faster that binary). So, with a small number of features the Tensor-based prediction can be computed much faster than usual tree-based model.

However, Decision Tensor has larger memory footprint. It needs $O(b^n)$ bytes 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 estimated 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 in two main ways: limit the number of cuts for each feature and train the model with big enough tree count or severely limit the tree count, then the total number of effective cuts will be exactly $h \cdot t$ without taking into account repeated use of the same cuts.

The first method was used in [2]: there was a trained CatBoost model with a small number of cuts that has been precomputed into a Decision Tensor.

The consumption of memory for a single hypercube can be extremely huge provded that acceptable quality is achieved, so the natural extension of the idea is the construction of Decision Tensors ensemble.

4. Tensor Similarity

Let's introduce the metric for 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

The target metrics are Q (model quality), S (size), and t (time of inference). However, we can assume that the inference time is proportional to the number of decision tensors M. The input feature vectors will define the algorithm for obtaining the trees and the algorithm for combining them into the Tensors. We denote single vector by v and let V be the set of all input vectors. Then we need to solve the following problem:

$$min(Q(V), S(V), M(V)), v \in V$$

It should be noted that for practical application this problem can be solved using the constrained optimization method. But in practice there are quite strict limitations on the size of the model and the available application time (TODO - what did you mean?).

6. Algorithm for combining set of Decision Trees to a set of Decision Tensors

Suppose we have N trees and we want to combine them in M of the Decision Tensors. The efficiency metric of the partition is the result Tensor set memory footprint. 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.

Such reduction of the set of trees to a set of Tensors we call partitioning. (TODO: what is rough and what is detailed partition?)

7. A complete algorithm for constructing an Decision Tensors Ensemble with the help of CatBoost
There are two key approaches:

- We fix a very rough partition by a small number of Tensors and train an arbitrarily large number of trees.
- 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 of 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) Merge the trees from the obtained model into the first decisive tensor from the ensemble
(4) Chose another set of CatBoost hyperparameters using Random Search

- Train the CatBoost model, using the previously trained model as the baseline
- Transform it into M-1 Decision Tensors
- Remove ensembles that do not fit the required size limits (TODO: specify limits at the beginning?).

(5) We choose the ensemble with the best quality on an additional data sample.

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

8. Real problem illustration
We have used the dataset and the model from [2]. 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 the training speed. For this we divided the data set into three parts: training (50%), test (25%) and validation (25%). We trained the models on the training sample, then used the test for the selection of the best models under given constraints.

We have limited the number of tensor by 6 and the total size of models by one gigabyte, so every prediction requires 4 bytes (float). We have scanned only through the key CatBoost hyperparameters: number of trees and the learing rate.

The figure of merit we've taken also from [2], which is the area under the ROC curve where False Positive Rate less than 0.05. For the convenience, we have normalized 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 (DTE) with different memory limits and different tensors numbers.

Figure 1 shows the dependence of the quality on the number of lookups. Figure 2 shows the dependence of the quality vs the total inference time at the full validation dataset. Table 1 shows properties of the models trained in the process.

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).

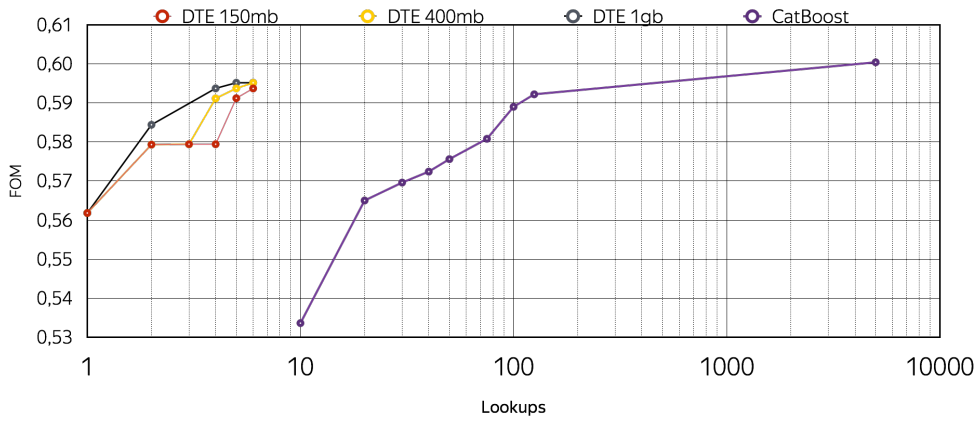

Figure 1. Lookups count versus FOM for different models sizes

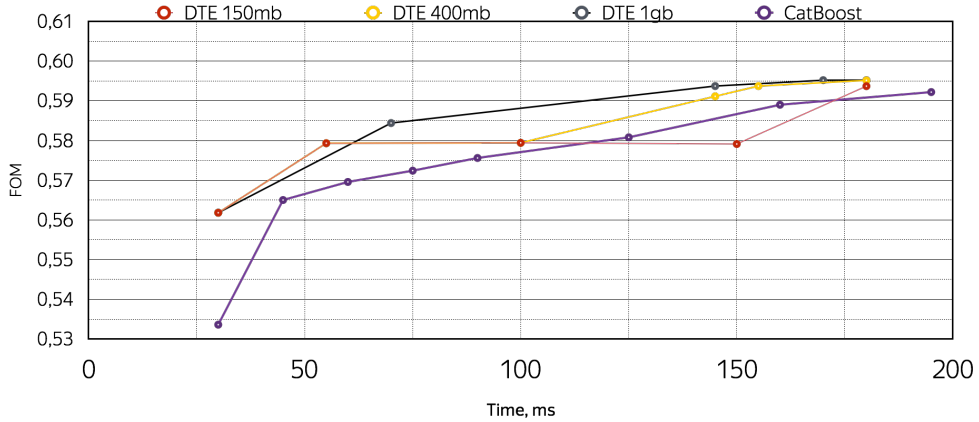

Figure 2. Application time versus FOM for different models sizes

Table 1. Properties of the some models trained in the process

Model	Roc	Time, ms	Size, Mb
catboost-5000	0.6004	10000	< 5
catboost-100	0.589	160	< 5
catboost-75	0.5808	125	< 5
catboost-10	0.5336	30	< 5
big dte-6	0.5952	180	1000
big dte-3	0.5938	145	1000
big dte-2	0.5844	70	1000
small dte-2	0.5794	55	150
dte-1	0.5618	30	150

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 CPU caching and the size of our dataset.

Since the Decision Tensor occupies significant amount of memory and does not fit into the CPU cache almost every lookup results in (slow) reading from the RAM. CatBoost models are small enough to fit into the cache. There was a single operation type of model application in our tests, so the CatBoost models stayed persistently in the cache. In real applications, CatBoost model will most likely not stay in the cache alone and will be even slower because of this. On the contrary the performance of the Decision Tensor Ensemble will not degrade due to that fact.

10. Conclusion

Decision Tensor Ensbles allow you to speed up the inference by complex Boosting Decision Trees algorithms (such as CatBoost) in real-time systems. Considered algorithm for building Tensor ensembles allows for additional tradeoff between memory and the inference speed without significant loss of the model quality. In addition, the use of the Ensemble can significantly improve the model quality in comparison with a single Decision Tensor.

References

[1] Gulin A, Kuralenok I, Pavlov D 2011 *Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank.* (Yahoo! Learning to Rank Challenge) p 63–76

[2] Likhomanenko T, Ilten P, Khairullin E, Rogozhnikov A, Ustyuzhanin A, Williams M 2015 *LHCb Topological Trigger Reoptimization* (Journal of Physics: Conference Series)