Speeding up prediction performance of BDT-based models

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

[1] Moscow Institute of Physics and Technology, Institutskiy per. 9, Dolgoprudny, Moscow Region, 141700, Russia

[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

Abstract. The outcome of a machine learning algorithm is a prediction model. Typically, these models are computationally expensive, where improving of the quality the prediction leads to a decrease in the inference speed. However it is not always tradeoff between quality and speed. In this paper we show it is possible to speed up the model by using additional memory without losing significat prediction quality for a novel boosted trees algorithm called CatBoost. The idea is to combine two approaches: training fewer trees and merging trees into a kind of hashmaps called 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 the 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 10^{-3}.

1. Introduction

Today, Machine Learning (ML) approaches are used in various fields ranging from the information retrieval in the Internet, to speech recognition, to optimizatimg steel compositions, to the search for the New Physics. The resources (memory, CPU, network etc.) can be very limited when solving tasks in production settings.

For most ML algorithms, improving the quality of the model usually leads to higher levels of resource consumption. For example, when using tree-based boosting algorithms, the increase in the number of trees from tens to thousands usually leads to significant increases in the quality, the model size and inference time. Higher demands for the computational resources of the more accurate models are usually satisfied by various technical optimizations of the model storage mechanism. For example vectorisation of the tree weights storage can be used. Another approach is to modify the training process so that it creates models with certain properties that will accelerate their application without maintaining the required level of quality.

For example, the Topological Trigger in the CERN-based LHCb experiment has very strict application time restrictions. In [2], the original MatrixNet [1] model allowed for significant improvements in quality. However, to get acceptable applying time the model was coarsed and converted to a Bonsai Boosted Decision Tree format [4].

In this paper, we consider a modification to the training process in the decision tree-based algorithm CatBoost. CatBoost is a successor of MatrixNet [3]. Our approach is also applicable

to other boosted decision tree-based algorithms.

2. CatBoost

CatBoost is the implementation of a tree-based gradient boosting in which oblivious decision trees are used [3]. The values of each feature i are discretized into baskets using the boundaries $B_i = b_{i1}, ..., b_{ip_i}$, where p_i is the number of boundaries. The boundaries themselves are determined in advance using various approaches. The original feature vector with real values is replaced by a vector of binary values 0 and 1 ($false$ and $true$). The value of the i-th attribute f_i is replaced with a binary vector $G_i = g_{i1}, ..., g_{ip_i}$ where $g_{ij} = f_i > b_{ij}$. Finally, all G_i are concatenated into one vector. Consecutively, the resulting decision will a make bunch of boolean checks instead of comparing some real feature value with the thresholds.

The number of baskets used for the discretization can be controlled by specifying the appropriate parameter during the training procedure.

The learning rate is another important training characteristic in boosted decision-tree methods. 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 and vice versa. 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.

CatBoost shows already acceptable quality with default parameters, so we variate only number of trees (and learning rate to compensate) because number of trees affects time. The other parameters were left at their default values.

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 value combinations.

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 that the values of every feature belongs to the interval $[0; 1]$. In other words, every feature vectors lies in n-dimensional cube $[0; 1]$.

Let B_i be the a of basket boundaries for the i-th feature, which was used in T. Then we can split the hyper-cube $[0; 1]$ 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. If we pre-calculate the value for each piece, then we will accelerate the application of our model, effectively reducing it to the time required to search a piece that contains the point. We called this approach a Decision Tensor. The asymptotic complexity of the application of the tree-based mode is $(h \cdot t)$ where h is the height of the trees, t is the total number of the trees. Asymptotic complexity of Tensor is $(n \cdot \log(b))$ where n is the number of the features and b is the number of cuts per feature (this complexity is estimated using a binary search). It should be noted for a small b a simple linear search could be slightly faster that a binary search. With a small number of features the tensor-based prediction may be computed much faster than the tree-based model.

However, Decision Tensors have a 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 D can be estimated as follows:

$$D = (B_1, ..., B_n),$$

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

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

Consider the case for small n. We can get a small b in two main ways: by limiting the number of cuts for each feature and training the model with a large enough tree count or by severely limiting the tree count. In the latter case, 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 can be found in [2]: which used a trained MatrixNet (ancestor of CatBoost) model with a small number of cuts that has been precomputed into a Decision Tensor. The consumption of memory for a single tensor can be extremely huge provided that acceptable quality is achieved, so the natural extension of the idea is the construction of Decision Tensors ensemble but with some deterioration of applying time.

4. Tensor Similarity

Let's introduce the metric for the similarity of two tensors. Suppose, we have two tensors: $D^1 = (B_1^1, ..., B_n^1)$ and $D^2 = (B_1^2, ..., B_n^2)$. The sum $D^u = D^1 + D^2$ could be defined as $D^u = (f_1^u, ..., f_f^u)$, where $f_i^u = f_i^1 \cup f_i^2$. Now, let's define similarity as the difference between the sum of their sizes and the size of their union:

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

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

Suppose we have T trees and we want to combine them into M Decision Tensors. The efficiency metric of the partition is the resulting memory footprint of the tensor set. 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.

6. A complete algorithm for constructing an Decision Tensors Ensemble

There are two key approaches:

- We use a rough partitioning by a small number of cuts 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 time and size constraints
(2) Determine the desired number of tensors (M) based on time constraints
(3) Train the model with a limited number of buckets
(4) Merge the trees from the obtained model into the first Decision Tensor from the ensemble
(5) Choose another set of hyperparameters using a random search

- Train the model, using the previously trained model as a baseline

- Transform it into M-1 Decision Tensors

(6) Remove ensembles that do not fit the required size limits

(7) Choose the ensemble with the best quality on an additional data sample.

This algorithm identifies the Pareto-optimal ensemble of Decision Tensors.

7. Real problem illustration

We have used CatBoost as a main algorithm and also took the dataset and the model from [2]. Number of features in the dataset is 10, the model was trained with 5000 trees and 64 buckets. We compared our algorithm for constructing Decision Tensors Ensemble with the 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 tensors to 6 and the total size of the models to one gigabyte, so every prediction requires 4 bytes (float). We have scanned only through the key CatBoost hyperparameters, i.e. number of trees and the learning 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 one.

8. Discussion

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

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

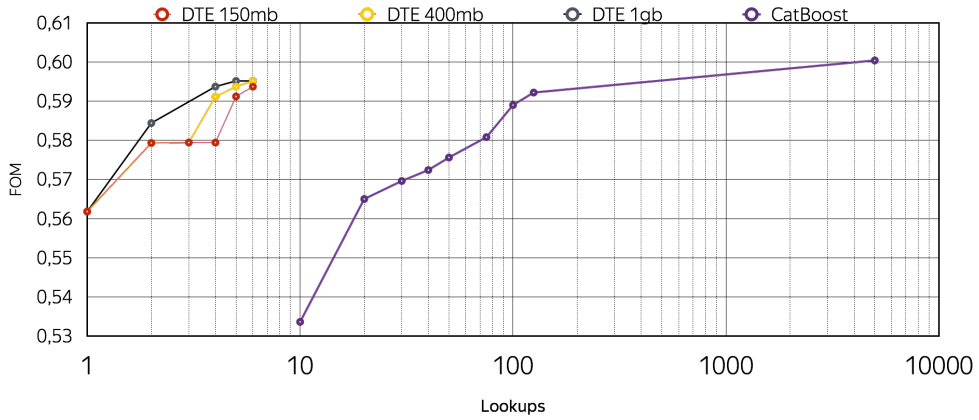

Figure 1. Lookups count versus FOM for different models sizes

It can be seen that at a comparable quality with fast CatBoost models (1-125 trees), DTE allows a 25 times reduction in the number of lookups. And in comparison with the best model (5000 trees), the number of lookups is 1000 times smaller with a small loss in quality (< 0.006 with the best quality of 0.6).

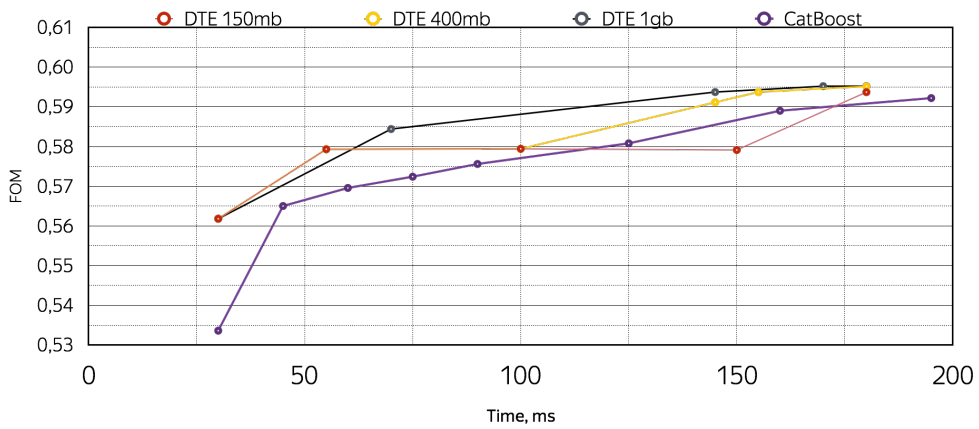

Figure 2. Application time versus FOM for different models sizes

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 reading from the RAM, which bottlenecks the working time. 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.

9. Conclusion

Decision Tensor Ensembles allow you to speed up the inference by complex Boosted Decision Trees algorithms (such as CatBoost) in real-time systems. The algorithm considered 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 and can be used in the LHCb experiment triggers.

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 (IOP Publishing) *Journal of Physics: Conference Series* Vol. 664, No. 8, p. 082025

[3] Dorogush A, Gulin A, Gusev G, Kazeev N, Ostroumova L, Vorobev A Fighting biases with dynamic boosting arXiv:1706.09516

[4] Gligorov V, Williams M 2013 Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree *Journal of Instrumentation* 8 P02013