Accelerating Document Sorting with Pruned Decision Trees and Chunking (Postprint)

Authors: Li Weijiang, Chang Wei, Zhengtao Yu

Date: 2018-11-29T00:00:00+00:00

Abstract

In recent years, an increasing number of users have been retrieving information they require from the Internet. Retrieval systems utilize learning-to-rank algorithms to produce ranking models from training sets. The time required for data retrieval represents an important research direction for retrieval systems. To reduce retrieval time, pruning strategies and caching for ranking models have been studied. By leveraging the redundant characteristics of decision trees and cache memory, a pruned decision tree model and a chunking algorithm are proposed. Finally, experiments were conducted on two public datasets, focusing primarily on whether the efficiency of ranking models can be improved without affecting model effectiveness. Experimental results indicate that the pruned decision tree model and chunking algorithm can effectively reduce the ranking time per query.

Full Text

Preamble

Method of Pruning Decision Tree Model and Block-Wise to Speed-Up Ranking Candidate Documents

Li Weijiang, Chang Wei, Yu Zhengtao
(School of Information Engineering & Automation, Kunming University of Science & Technology, Kunming 650500, China)

Abstract: Recently, more and more people retrieve information from the Internet. Retrieval systems use learning-to-rank (LtR) algorithms to produce ranking models from training sets. The time required to retrieve data represents an important research direction for retrieval systems. To reduce retrieval time, we studied pruning strategies for ranking models and caching. Leveraging the redundancy characteristics of decision trees and cache memory, we propose a pruned decision tree model and a block-wise algorithm. Finally, we conducted

experiments on two publicly available datasets, focusing on whether we can improve ranking model efficiency without affecting model effectiveness. Experimental results demonstrate that the pruned decision tree model and block-wise algorithm can effectively reduce ranking time per query.

Key words: learning to rank; cache; efficiency; pruning

0 Introduction

In information retrieval, ranking relevant content according to relevance criteria is a fundamental and important problem. A research field called learning to rank [1,2] has shown that machine learning methods can effectively solve ranking problems. Machine learning algorithms train a ranking model from a dataset containing relevance-labeled query-document pairs. Tree-based ensemble ranking models are highly effective for ranking query results returned by Web search engines [3,4].

A complex ranking architecture typically consists of two components: retrieving candidate documents and re-ranking them [5]. The first stage retrieves a sufficiently large candidate document set (CDq) from an inverted index, which may or may not be relevant to the user' s query. |RDq| denotes the number of relevant documents displayed on the final page, and |CDq| » |RDq|. This stage aims to improve recall. The initial filter that retrieves the candidate document set is usually a simple and fast basic algorithm, such as BM25 [6], Boolean models, or extended Boolean models. The learning-to-rank model is used in the second stage to re-rank this candidate document set, with the goal of improving accuracy. Finally, after sorting the candidate document set, the top |RDq| documents are displayed on the final page. In this two-stage architecture, the time required to re-rank the candidate document set is critical due to large query volumes and user demands for response time.

Modern search engines have very strict time constraints for responding to user queries. Exploring new strategies and techniques to reduce the time for ranking query results is an urgent research topic. Consequently, researchers have begun exploring optimization techniques to reduce ranking time. For example, Lucchese et al. [7] proposed a ranking algorithm called QuickScorer that uses bit vectors to represent a decision tree ensemble and a new access pattern. Asadi et al. [8] transformed control dependencies into data dependencies to traverse the decision tree ensemble. Lucchese et al. [9] proposed a rank-based feature framework to expand the original feature set and reduce document ranking time. Lucchese et al. [10] proposed a V-QuickScorer algorithm that leverages modern CPU SIMD instruction sets to vectorize the processing of multiple document rankings. Zhou et al. [11] studied result re-ranking in personalized cross-language information retrieval.

To reduce ranking time per query, this paper proposes a method combining

pruned decision tree models and a block-wise algorithm. This method integrates pruning and caching techniques. On one hand, when computing a document' s score, all decision trees in the ensemble must be traversed, so document ranking time is proportional to the number of decision trees. We use pruning techniques to reduce the number of decision trees without affecting model effectiveness. On the other hand, the block-wise technique leverages the principle of cache temporal locality [12] to further reduce ranking query time. Our experiments show that this method outperforms state-of-the-art baseline ranking algorithms, such as StructPlus and VPRED, in terms of efficiency.

The main contributions of this paper are as follows:

a) Each document requires traversing all decision trees in the ensemble, and each document receives a score for ranking. The time to predict a document' s score is proportional to the number of trees. Therefore, using pruning techniques to reduce the number of decision trees can reduce document ranking time without affecting ranking model effectiveness.

b) Because memory access latency is several orders of magnitude slower than cache access latency, we use caching techniques to optimize document traversal. A candidate document set and a decision tree ensemble may exceed cache capacity, causing frequent cache content replacement. To fully utilize cache temporal locality, we use block-wise techniques to partition both the candidate document set and the decision tree ensemble into several blocks, further reducing ranking time.

---

1 Related Work and Background

With an increasing number of documents on the Internet, the initial filter algorithm retrieves more relevant documents for user queries, leading to longer candidate document ranking times.

Today, learning-to-rank algorithms are widely used to solve most ranking problems. The two most effective learning-to-rank algorithms are GBRT (Gradient Boosting Regression Trees) [13] and -MART (Lambda-Multiple Additive Regression Tree) [14]. Both algorithms produce an additive decision tree ensemble model. Below, we review several state-of-the-art decision tree traversal algorithms that serve as baselines for comparison in this paper.

a) **StructPlus [8]**. This is an improvement over the STRUCT algorithm. It uses a data structure to implement tree traversal that stores pointers to left and right child nodes, feature indices, and thresholds. The traversal process starts from a decision tree' s root node. Based on the Boolean condition result at each visited branch node, it moves from the root to a leaf node. The leaf node' s output value represents the tree' s potential contribution to the document score. The disadvantage of this algorithm is that StructPlus frequently introduces control dependencies—meaning the

next node to traverse can only be determined after the Boolean condition test. Therefore, the next instruction to execute depends on the condition result, making algorithm efficiency highly dependent on branch misprediction rates.

b) **PRED [8]**. Asadi et al. proposed a reordering method that transforms control dependencies into data dependencies. The PRED algorithm converts a tree into an array data structure (Node). Node[i] represents a branch or leaf node in the tree. It stores a feature index (fid), a threshold (theta), and an array (child) holding the indices of left and right child nodes. The first variable child[0] stores the index of the current node' s left child, while child[1] stores the index of the right child. To obtain the index of the next node to traverse, the output value from Equation (1) is used as the index for the child array:

The output value (0|1) from Equation (1) serves as the subscript for the child array. This paper uses self-looping to connect each leaf node to itself. A tree of depth d is expanded into d operations:

This leads to a situation where even if the traversal process reaches a leaf node early, it cannot exit prematurely and must complete all d operations. Additionally, it involves long expansion sequences.

c) **VPRED [8]**. This is an improvement over the PRED algorithm. It uses a vectorized method to compute scores for multiple documents simultaneously to reduce instruction branch misprediction rates and mask high memory access latency. The VPRED algorithm parallelizes the computation of scores for V documents. Literature [8] shows that the algorithm performs best when V = 16.

---

2 Related Definitions and Pruned Decision Tree Model

This chapter presents the decision tree-related definitions used in this paper and introduces the details of the pruned decision tree model.

2.1 Related Definitions

A query-document pair is represented by a ground-truth feature vector , where is the set of features describing the document. stores feature values. A tree contains a set of branch nodes and a set of leaf nodes . Each branch node consists of a Boolean condition, a feature index , and a threshold . The Boolean condition formula is . Each leaf node stores a prediction value , representing the tree' s potential contribution to document . Assume the partial feature values of document are . [Figure 1: see original paper] shows the tree traversal process.

Definition 1 (False Node and True Node). If a branch node' s Boolean condition test evaluates to FALSE, the node is a false node; otherwise, it is a

true node. In [Figure 1: see original paper], is a true node, while and are false nodes.

Definition 2 (Exit Node). The tree traversal process starts from the root node. If the visited node is a false node (Definition 1), the traversal follows the right branch; otherwise, it follows the left branch. This process recurses until reaching a leaf node. This leaf node is called the exit node, denoted as . In [Figure 1: see original paper], leaf node is the exit node of this tree.

Definition 3 (Sub-score of Query-Document Pair). This sub-score represents the potential contribution of a specific tree to document . The sub-score uses the exit node from Definition 2 and is calculated as:

where denotes the prediction value of the exit node in tree , and is the weight of tree .

Definition 4 (Final Document Score). This score is used to rank documents. All trees in ensemble undergo this traversal process. The final score is obtained by weighted summation of each tree ' s contribution to document using the sub-score from Definition 3:

2.2 Pruned Decision Tree Model

GBRT and -MART algorithms both produce an additive decision tree ensemble. There is an open-source implementation of these two algorithms [15]. Although these models can achieve high recall and accuracy for ranking documents, response time is very long because ranking time is proportional to the number of decision trees. Therefore, this paper proposes a pruned decision tree model to reduce the number of decision trees in the model without affecting the effectiveness of the original model.

This section introduces the details of the pruned decision tree model. Obtaining this pruned decision tree model involves two steps: a) using GBRT or -MART algorithms to produce a good model; b) using pruning strategies to reduce the number of decision trees in the original model.

Step 1: We first use the GBRT algorithm to generate the decision tree model. GBRT builds decision tree models by minimizing root mean square error. GBRT uses a training set , a loss function , and iteration count M. The algorithm first initializes with a constant:

Then it iterates M times to build M trees, improving the decision tree model quality. In each iteration, it calculates pseudo-residuals:

The training set is used to train the learner. The value is computed by solving the following one-dimensional optimization problem:

Finally, the decision tree ensemble is updated:

Step 2: We use a pruning technique to prune the obtained model. Given an ensemble , the pruning technique produces a smaller ensemble with at least

the same quality as the original ensemble but higher efficiency. The pruning technique has two steps: a) use a pruning strategy to prune into a subset ; b) use a linear search algorithm to update the weight values of the remaining trees to improve a specified loss function.

First, we use a pruning strategy to remove some trees from the ensemble. This step uses a quality-loss pruning strategy to reduce the number of decision trees.

Definition 5 (Quality Loss). This pruning strategy removes trees based on their impact on a specified loss function. For a tree , the average quality loss is calculated by:

where and are defined in Definitions 3 and 4. The n-p trees with the smallest impact are removed from the ensemble.

Let denote a specified loss function, and denote the tree weight values after pruning. We optimize the weight values using Equation (9):

Since most loss functions are non-differentiable, computing is infeasible, so we use a heuristic method to optimize the remaining tree weight values . First, we find a descent direction along the loss function . Then we compute a step size to minimize the value. This process iterates until the value falls below a specified threshold.

---

3 Pruning and Block-Wise Approach

This chapter presents the pruning and block-wise algorithm and describes its details. Researchers have proven that learning-to-rank models can effectively solve ranking problems, but simultaneously suffer from long response times. To apply these effective models to modern search engines, this paper proposes a method using both pruning and caching techniques to reduce ranking time, hoping to make ranking models applicable to modern search engines.

3.1 Pruning and Block-Wise Method

This section describes the structure of the proposed method. [Figure 2: see original paper] illustrates the overall process, which can be divided into three steps:

a) Use GBRT or -MART algorithms to train a decision tree model from the training set (corresponding to edge 1 in [Figure 2: see original paper]).

b) Use a pruning strategy to reduce the number of trees in the ranking model and update the weights of the remaining trees [16] (corresponding to edge 2 in [Figure 2: see original paper]).

c) Use caching techniques to compute all document scores and rank documents using these scores (corresponding to edge 3).

The proposed method can reduce ranking time for two main reasons: First, document scoring time is proportional to the number of decision trees. When pruning techniques reduce the number of trees, document scoring time decreases. Second, cache access latency is hundreds of times faster than memory. When the algorithm' s required documents and decision trees are in cache, document scoring time is further reduced.

The pruning and block-wise approach has two components: The pruning process (top-left in [Figure 2: see original paper]) reduces the number of decision trees in the model without affecting the original model' s effectiveness. Therefore, we use this pruned decision tree model as the final ranking model (introduced in Section 3.2). The caching scheme (bottom-left in [Figure 2: see original paper]) is described in the following sections, which focus on the block-wise algorithm.

3.2 Block-Wise Technique

This section describes the implementation details of the block-wise algorithm. We propose this block-wise scheme primarily because cache access latency is hundreds of times faster than memory, so we hope to fetch documents and rankers (decision trees) from cache. However, cache is typically very small and cannot contain all candidate documents and all rankers in the model, causing frequent cache content replacement and resulting in very low utilization of cache temporal locality. To improve cache temporal locality utilization, we propose a block-wise approach that partitions a large candidate document set and ranking model into several blocks, allowing a document block and a ranker block to fit simultaneously into cache.

Block-wise algorithm implementation details: The algorithm uses four nested loops. The outermost two loops handle interactions between document blocks and ranker blocks, while the innermost two loops handle interactions between documents within a document block and rankers within a ranker block. Assume the candidate document set contains D documents, and the decision tree ensemble contains S rankers. A document block contains d documents, and a ranker block contains s rankers. There are document blocks and ranker blocks. To simplify algorithm description, we assume and are integers. The traversal process of candidate documents and rankers is shown in the right half of [Figure 2: see original paper], where numbers on edges indicate the access order between document blocks and ranker blocks.

Based on the above analysis, we present Algorithm 1.

Algorithm 1: Block-Wise Algorithm

Input: documents: candidate document set; ensemble: decision tree ensemble
Output: scores: document score collection, one score per document

```
for i = 0 to - 1 do
    for j = 0 to - 1 do
        for ii = 0 to s - 1 do
```

```
            for jj = 0 to d-1 do
                (see Definition 3)
            end for
        end for
    end for
end for
(see Definition 4)
```

---

4 Experiments

We conducted a series of experiments on two publicly available datasets: Microsoft LETOR (MSLR-10K) (Microsoft Learning to rank dataset. http://research.microsoft.com/en-us/projects/mslr) and a new dataset provided by Istella (Istella-s) (Istella Blog, Istella Learning to rank dataset. http://blog.istella.it/istella-learning-to-rank-dataset/). The first dataset contains five folders; our experiments used only the first folder (MSN-1). Each query-document pair in the MSN-1 dataset is represented by 136 features. The Istella-s dataset contains 3,408,630 query-document pairs, with each document represented by 220 features. Each query-document pair has a relevance judgment level ranging from 0 (irrelevant) to 4 (highly relevant). All datasets were split into training, validation, and test sets in a 60%-20%-20% ratio.

All experiments ran on a Linux machine with a 2-core Intel I3-6100 CPU at 3.7 GHz and 8 GB of memory. CPU cache levels: L1 data cache of 32 KB per core; L2 cache of 256 KB per core; and L3 cache of 3 MB shared among all cores. We used NDCG@10 to evaluate model effectiveness in our experiments [17]:

where N-1 is a normalization factor representing the DCG score of an ideal ranking; rj denotes the relevance level of the j-th document; and g(rj) is an exponential function. In our experiments, d(j) is computed by Equation (12):

We compared the pruning and block-wise algorithm with the following algorithms in terms of efficiency and effectiveness:

a) Standard traversal algorithm: a document directly traverses all trees.

b) No cache technique.

c) StructPlus and VPRED [8].

We conducted three groups of experiments to analyze the efficiency of the pruning and block-wise algorithm.

4.1 Effectiveness Analysis

Our framework first uses pruning techniques to remove some trees from the ensemble and then updates the weights of the remaining trees. Therefore, we

first evaluate the effectiveness of the pruned model. This experiment includes two sub-experiments using GBRT and -MART algorithms to train ranking models on two public datasets (MSN-1 and Istella-s). To better analyze the impact of different parameter values on NDCG@10, we used different pruning rates and leaf node counts in each sub-experiment. We set pruning rate P to 0, 0.1, 0.2, ···, 0.9, and leaf node counts to 8, 16, 32, 64. When P = 0, the NDCG@10 value represents the effectiveness score of the original model (i.e., the evaluation value before pruning). Results from the two sub-experiments are shown in [Figure 3: see original paper].

[FIGURE:3(a)] shows NDCG@10 evaluation values with the GBRT algorithm and MSN-1 dataset under different leaf nodes and pruning rates. [FIGURE:3(b)] shows NDCG@10 evaluation values with the -MART algorithm and Istella-s dataset under different leaf nodes and pruning rates.

From these sub-experiments, we observe that under different pruning rates, models with fewer leaf nodes (8, 16) achieve better effectiveness evaluations than models with more leaf nodes (32, 64), and their effectiveness degrades more slowly. We select the smallest (i.e., most efficient) pruned model that still provides equal or greater effectiveness evaluation than the original model on the validation set. The final selected pruned models are shown in .

shows the selected pruned decision tree models under different leaf node counts and datasets. The highlighted and bolded sections represent the best pruned models in terms of pruning rate. We can see that using pruning techniques can significantly reduce 30% to 70% of decision trees in the ensemble without affecting the original model' s effectiveness. We also observe that models with fewer leaves have higher pruning rates. Therefore, we select the highlighted and bolded pruned decision tree models for the following experiments.

4.2 Cache Behavior Analysis

Our algorithm uses caching techniques to reduce document ranking time. The block-wise algorithm partitions a candidate document set and decision tree ensemble into several blocks. A document block containing d documents and a ranker block containing s rankers are placed into cache. We analyze the impact of different document counts d and ranker counts s on L3 cache miss rates and ranking time. We used Linux' s Perf tool to analyze L3 cache miss rates.

[FIGURE:4(a)] shows the L3 cache miss rate of the block-wise algorithm under different d and s values. This trend strongly correlates with the ranking time curve in [FIGURE:4(b)], which shows the time required to compute a document score under different d and s values. We can see that the optimal point occurs at d = 1000 and s = 500, where these documents and decision trees fit exactly into cache and loop counts are minimized. When s = 500, the L3 miss rate varies from 2.11% (d = 1000) to 82.22% (d = 10000), causing corresponding ranking time to vary from 3.0191 s to 15.2335 s. In subsequent experiments, we set d = 1000 and s = 500.

4.3 Time Complexity Analysis

This section formally analyzes the time complexity of each ranking model to observe ranking efficiency.

Our algorithm' s time complexity primarily comes from Algorithm 1 in Section 3.2, which uses caching techniques to partition documents and decision trees into blocks, leveraging fast cache speeds to improve ranking model efficiency.

Algorithm 1 has four nested loops. For simplicity, we assume s decision trees and d documents can fit simultaneously into cache. In the innermost loop, loading d documents from memory to cache takes time . In the third loop, loading s decision trees to cache takes time . The cache can hold documents and decision trees simultaneously. In the second loop, the s decision trees remain in cache, but blocks of documents must be loaded from memory, giving time complexity . In the outermost loop, blocks of decision trees are loaded into cache, resulting in total time complexity:

From Section 4.2, we have . In terms of time complexity, Algorithm 1 is lower than other algorithms.

In algorithms without caching, standard traversal, and StructPlus, they all traverse one document and one decision tree at a time, giving these algorithms time complexity . Our algorithm is s times faster than these three algorithms. The VPRED algorithm uses parallelization to compute scores for v documents simultaneously, masking low memory access latency. Literature [8] shows the algorithm performs best when v = 16, giving it time complexity . From time complexity analysis, Algorithm 1 has lower time complexity than other algorithms.

4.4 Ranking Time Analysis

Our goal is to reduce ranking time to apply effective ranking models to search engines. This section compares the pruning and block-wise algorithm with other ranking algorithms in terms of efficiency. We conducted four sub-experiments.

and show per-document scoring time (s) for different traversal algorithms under GBRT and -MART algorithms on MSN-1 and Istella-s datasets, respectively. The tables also show speedup ratios of our algorithm over others in parentheses. For example, in scoring one document, our algorithm is 1.17 to 2.24 times faster than VPRED (v = 16). On the MSN-1 dataset with 8 leaf nodes (), our algorithm and VPRED algorithm take 3.0191 s and 6.79 s per document, respectively.

From and , we can see that under different settings, the pruning and block-wise algorithm outperforms other algorithms in efficiency, demonstrating 1.17 to 10.35 times faster ranking time.

5 Conclusion

Modern search engines have strict time constraints for responding to user queries. This paper proposes a method to reduce candidate document ranking time, enabling high-quality ranking models for modern search engines. This method integrates a pruned decision tree model and a block-wise algorithm. Under the same time constraints, this method can rank more documents, thereby achieving better recall and accuracy. Experimental results demonstrate that the method can effectively reduce ranking time per query without affecting model quality.

For future work, we plan to apply this framework to other domains, such as classification.

References

[1] Liu Tieyan. Learning to rank for information retrieval [J]. Foundations and Trends in Information Retrieval, 2009, 3 (3): 225-331.

[2] Wang Yang, Huang Yalou, Liu jie, et al. Algorithm of active learning to rank based on PRank algorithm [J]. Computer Engineering, 2008, 34 (21): 38-40.

[3] Ganjisaffar Y, Caruana R, Lopes C V. Bagging gradient-boosted trees for high precision, low variance ranking models [C]// Proc of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2011: 85-94.

[4] Viola P, Jones M. Robust real-time face detection [J]. International Conference on Computer Vision, 2001, 57 (2): 137-154.

[5] Cambazoglu B B, Zaragoza H, Chapelle O, et al. Early exit optimizations for additive machine learned ranking systems [C]// Proc of the third ACM International Conference on Web Search and Data Mining. New York: ACM Press, 2010: 411-420.

[6] Robertson S, Zaragoza H. The probabilistic relevance framework: BM25 and beyond [J]. Foundations and Trends in Information Retrieval, 2009, 3 (4): 333-389.

[7] Lucchese C, Nardini F M, Orlando S, et al. QuickScorer: a fast algorithm to rank documents with additive ensembles of regression trees [C]// Proc of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2015: 73-82.

[8] Asadi N, Lin J J, De Vries A P. Runtime optimizations for tree-based machine learning models [J]. IEEE Trans on Knowledge and Data Engineering, 2014, 26 (9): 2281-2292.

[9] Lucchese C, Nardini F M, Orlando S, et al. Speeding up document ranking with rank-based features [C]// Proc of the 38th International ACM SIGIR

Conference on Research and Development in Information Retrieval. New York: ACM Press, 2015: 895-898.

[10] Lucchese C, Nardini F M, Orlando S, et al. Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles [C]// Proc of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2016: 513-522.

[11] Zhou Dong, Zhao Wenyu, Wu Xuan, et al. Result re-ranking in personalized cross-language information retrieval [J]. Computer Engineering and Science, 2017, 39 (10): 1922-1929.

[12] Tang Xun, Jin Xin, Yang Tao. Cache-conscious runtime optimization for ranking ensembles [C]// Proc of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2014: 1123-1126.

[13] Friedman J H. Greedy function approximation: a gradient boosting machine [J]. Annals of Statistics, 2001, 29 (5): 1189-1232.

[14] Wu Qiang, Burges C J, Svore K M, et al. Adapting boosting for information retrieval measures [J]. Information Retrieval, 2010, 13 (3): 254-270.

[15] Capannini G, Lucchese C, Nardini F M, et al. Quality versus efficiency in document scoring with learning-to-rank models [J]. Information Processing and Management, 2016, 52 (6): 1161-1177.

[16] Lucchese C, Nardini F M, Orlando S, et al. Post-learning optimization of tree ensembles for efficient ranking [C]// Proc of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2016: 949-952.

[17] Jarvelin K, Kekalainen J. Cumulated gain-based evaluation of IR techniques [J]. ACM Trans on Information Systems, 2002, 20 (4): 422-446.

Note: Figure translations are in progress. See original paper for figures.