Intelligent Wordprocessor Using Tree-Based Machine Learning Ranking Models

Naveena.R[1], Saigeetha.S[2], Rajalakshmi.S[3],
Department of Computer Science and Engineering
Parisutham Institute of Technology and Science, Thanjavur.

***Abstract*- Machine learning is a type of Artificial Intelligence which aggrandize the Computers to learn themselves from a given set of data.This paper focuses on generating some automatic keywords using Best-First trees which is used for ranking.Most recursive words are given a higher order priority and ranking using the technique called prediction.The direction of the respective text is usually obtained using a technique called vectorization. Using these techniques the efficiency and performance of the system is enhanced.**

Keywords*- *Machinelearning, prediction, Vectorization, Best-First trees, Automatic keywords.

I. INTRODUCTION

Tree based models have become an efficient method for the process of ranking.When combined with Machine learning the Tree based models becomes more efficient and provides results more effectively. The document ranking is also better exploited using the machine learning technique called "learning to rank" approach[1].Using the tree-based models,runtime optimization is performed by making predictions,specially using gradient-boosted regression trees for learning to rank[2]. Our experimentation focus on an individual tree, the runtime execution of which involves checking a predicate in an interior node, following the left or right branch depending on the result of the predicate, and repeating until a leaf node is reached. We assume that the predicate at each node involves a feature and a threshold: if the feature value is less than the threshold, the left branch is taken; otherwise, the right branch is taken. Of course, trees with greater branching factors and more complex predicate checks can be converted into an equivalent binary tree, so our formulation is entirely general. Note that our discussion is agnostic with respect to the prediction at the leaf node. This implementation has two advantages: simplicity and flexibility. However, we have no control over the physical layout of the tree nodes in memory, and hence no guarantee that the data structures exhibit good reference locality. Prediction with this implementation essentially boils down to pointer 1chasing across the heap: when following either the left or the right pointer to the next tree node, the processor is likely to be stalled by a cache miss. The contribution of this work lies in novel implementations of tree-based models that are highly-tuned to modern processor architectures, taking advantage of cache hierarchies and superscalar processors. We illustrate our techniques on three separate learning-to-rank datasets and show significant performance improvements over standard implementations.

II. ARCHITECTURE

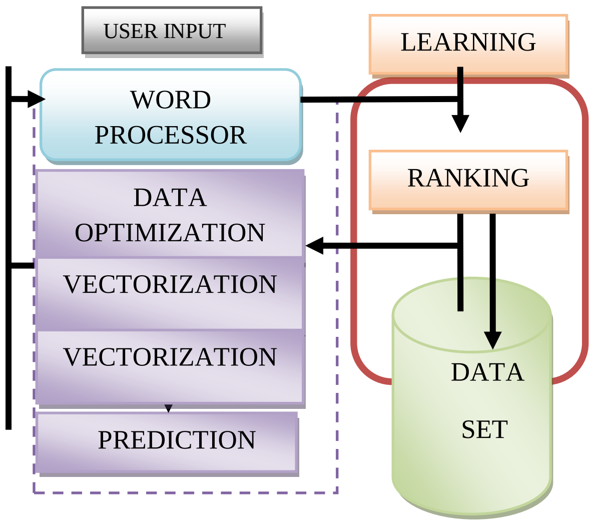

Fig.1.System Architecture Diagram

Here,the user input is given to the wordprocessor,then the process of data optimization takes place where logical schema is formed from the given data schema.Data optimization is very important in database management and also in data warehouse management.The next process is the vectorization,which generally used to exploit the modern processor architecture in a better way and using this process,the direction of the words that is needed to be displayed id obtained.The final block is the process of prediction,which is a supervised learning task where the data are used directly to predict the words.This process helps in finding the related words that are needed to be displayed.If the word is a new one,it is fed to the block of learning and ranking,where the new word is learnt and the ranked,then the usual process takes place.

III. RELATED WORK

In this section, we present a sample of previous work on parallel machine learning most related to our work. The related work falls into three categories: i)parallel decision trees, ii)parallelization of boosting, and iii)parallelization of web search ranking using other approaches such as bagging.[3] Parallel decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modelling approaches used in statistics, data mining and machine learning.The process is done parallely. Most of the previous work on parallelizing boosting focuses on parallel construction of the weak learners [4] or on the original AdaBoost algorithm [5, 6] instead of gradient boosting. MultiBoost [7] combines wagging with AdaBoost, which can be performed in parallel, but inherits AdaBoost's sensitivity to noise.

Our initial goal in building PLANET was to develop a scalable tree learner with accuracy comparable to a traditional in-memory algorithm, but capable of handling much more training data. We believe our experience in building and deploying PLANET provides lessons in using MapReduce for other non-trivial mining and data processing tasks. The strategies we developed for handling tree learning should be applicable to other problems requiring multiple iterations, each requiring one or more applications of MapReduce. CSS-trees can improve searching performance by making good use of cache lines. As the gap between CPU and memory speed is widening, we expect the improvement that can be achieved by exploiting the cache will be even more significant.Cache conscious searching behavior is just one step towards efficiently utilizing the cache in database systems.[8]

The two optimization techniques central to our approach borrow from previous work. Using a technique called predication [9], [10], originally from compilers, we can convert control dependencies into data dependencies. However, as compiler researchers know well, predication does not always help under what circumstances it is worthwhile for our machine learning application is an empirical question we examine. Another optimization that we adopt, vectorization, was pioneered by database researchers [11], [12] the basic idea is that instead of processing a tuple at a time, a relational query engine should process a vector (i.e., batch) of tuples at a time to take advantage of pipelining and to mask memory latencies. We apply this idea to prediction with tree-based models and are able to obtain many of the same benefits.[1].

Essen *et al*. [13] compared multi-core, GPU, and FPGA implementations of compact random forests. They also take advantage of predication, but there are minor differences that make our implementation more optimized. Furthermore, neither of these two papers take advantage of vectorization, although it is unclear how vectorization applies to GPUs, since they are organized using very different architectural principles.

IV. PROPOSED WORK:

Our system focus on an individual tree, the runtime execution of which involves checking a predicate in an interior node, following the left or right branch depending on the result of the predicate, and repeating until a leaf node is reached. We assume that the predicate at each node involves a feature and a threshold: if the feature value is less than the threshold, the left branch is taken; otherwise, the right branch is taken. Of course, trees with greater branching factors and more complex predicate checks can be converted into an equivalent binary tree, so our formulation is entirely general. Note that our discussion is agnostic with respect to the prediction at the leaf node. This implementation has two advantages: simplicity and flexibility. However, we have no control over the physical layout of the tree nodes in memory, and hence no guarantee that the data structures exhibit good reference locality. Prediction with this implementation essentially boils down to pointer chasing across the heap: when following either the left or the right pointer to the next tree node, the processor is likely to be stalled by a cache miss.

A. algorithm

a. back propagation

A Back Propagation network learns by example.You give the algorithm examples of what you want the network to do and it changes the network's weights so that, when training is finished, it will give you the required output for a particular input.

In decision trees, the overfitting can occur when the size of the tree is too large compared to the number of training data. Many methods for decision tree pruning have been proposed, and all of them remove some nodes from the tree to reduce its size. However, some removed nodes may have a significance level or some contribution in classifying new data. Therefore, instead of absolutely removing nodes, our proposed method employs a back propagation neural network to give weights to nodes according to their significance.

Backpropagation, an abbreviation for "backward propagation of errors", is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respects to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

Back propagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method, although it is also used in some unsupervised networks such as auto encoders. It is a generalization of the delta rule to multi-layered feed forward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Back propagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.

B. *Methodology*

a. *Machine Learning*

Machine learning is a scientific discipline that explores the construction and study of algorithms that can learn from data. Such algorithms operate by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions. In particular, we define machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty (such as planning how to collect more data!).

Machine learning is usually divided into two main types. In the predictive or supervised learning approach, the goal is to learn a mapping from inputs x to outputs y, given a labeled set of input-output pairs D = {(xi, yi)}N i=1. Here D is called the training set, and N is the number of training examples.

There is a third type of machine learning, known as reinforcement learning, which is somewhat less commonly used. This is useful for learning how to act or behave when given occasional reward or punishment signals. Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

V. PERFORMANCE EVALUATION

The focus of our work is on efficiency. Our primary evaluation metric is 1) Data Optimization 2) Vectorization 3) Prediction 4) Learning & Ranking

A. *Data Optimization*

The synthetic data consist of randomly generated trees and randomly generated feature vectors. Each intermediate node in a tree has two fields: a feature id and a threshold on which the decision is made. Each leaf is associated with a regression value. Construction of a random tree of depth d begins with the root node. We pick a feature id at random and generate a random threshold to split the tree into left and right subtrees.This process is recursively performed to build each subtree until we reach the desired tree depth. When we reach a leaf node, we generate a regression value at random.These data are optimized and given to our system as input.

B. *Vectorization*

The idea is to work on *v* instances (feature vectors) at the same time, so that while the processor is waiting for memory access for one instance, useful computation can happen on another. This takes advantage of pipelining and multiple dispatch in modern superscalar processors. The effectiveness of vectorization depends on the relationship between time spent in actual computation and memory latencies. For example, if memory fetches take only one clock cycle, then vectorization cannot possibly help. The longer the memory latencies, the more we would expect vectorization (larger batch sizes) to help. However, beyond a certain point, once memory latencies are effectively masked by vectorization, we would expect larger values of *v* to have little impact. In fact, values that are too large start to become a bottleneck on memory bandwidth and cache size.

C. *Prediction*

VPRED:Predication eliminates branches but at the cost of introducing data hazards. Each statement in PRED requires an indirect memory reference. Subsequent instructions cannot execute until the contents of the memory location are fetched—in other words, the processor will simply stall waiting for memory references to resolve. Therefore, predication is entirely bottlenecked on memory access latencies. This takes advantage of multiple dispatch and pipelining in modern processors (provided that there are no dependencies between dispatched instructions, which is true in our case). Thus, while the processor is waiting for the memory access from the predication step on the first instance, it can start working on the second instance. In fact, we can work on *v* instances in parallel.

D. Learning & Ranking

We are given training, validation, and test data as well as a tree-based learning-to-rank model. Using the training and validation sets we learn a complete tree ensemble. Evaluation is then carried out on test instances to determine the speed of the various algorithms. These end-to-end process gives us insight on how different implementations compare in a real-world application. We used three standard learning-to-rank datasets: LETOR-MQ2007, MSLR-WEB10K,4 and the Yahoo! Webscope Learning-to-Rank Challenge dataset. All three datasets are pre-folded, providing training, validation, and test instances. This can be incorporated into the learning algorithm as a penalty on tree topologies, much in the same way that regularization is performed on the objective in standard machine learning. Thus, it is to jointly learn models that are both fast and good, as in the "learning to *efficiently* rank" framework.

VI. CONCLUSION

:

In this paper we show how the efficiency can be improved with data optimization, vectorization, predication, learning and ranking as a whole in one paper which gives more efficiency in the datasets environment rather than emerged papers . The process of optimizing data can shorten development, elimination of redundant data reduce costs and ensure data security. Vectorization process is like parallelism inside a single CPU core, achieved by applying a CPU instruction to multiple data elements at once. Vectorizing a loop can deliver a significant performance boost and it also improves the scalability. The main purpose of predication is to avoid jumps over very small sections of program code, increasing the effectiveness of pipelined execution and avoiding problems with the cache. Predication and vectorization, coupled with a more compact memory layout, can significantly accelerate the runtime performance for tree-based models, both on synthetic data and on real-world learning-to-rank datasets. The learning and ranking is based on the priority of the data. That is, the frequently searched words will be ranked high and vice versa. We have given the current trend towards machine learning in larger data sets, we expect our algorithm to increase in both relevance and utility in the foreseeable future.

REFERENCES

[1] N. Asadi and J. Lin, "Training efficient tree-based models for document ranking," in *Proc.34th ECIR* , Moscow, Russia, 2013.

[2] Nima Asadi, Jimmy Lin, and Arjen P. de Vries "Runtime optimizations for tree based machine learning models", in 2014.

[3] S. Tyree, K. Q. Weinberger, and K.Agrawal, "Parallel boosted regression trees for web search ranking," in *Proc. 20th Int. Conf. WWW*, Hyderabad, India, 2011, pp. 387–396.

[4] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "PLANET: Massively parallel learning of tree ensembles with mapreduce," in *Proc. 35th Int. Conf. VLDB*, Lyon, France, 2009, pp. 1426–1437.

[5] A. Lazarevic and Z. Obradovic. Boosting algorithms for parallel and distributed learning. Distributed and Parallel Databases, 11(2):203–229, 2002.

[6] N. Uyen and T. Chung. A new framework for distributed boosting algorithm. Future Generation Communication and Networking, 1:420–423, 2007.

[7] G. Webb. Multiboosting: A technique for combining boosting and wagging. Machine learning, 40(2):159–196, 2000.

[8] J. Rao and K. A. Ross, "Cache conscious indexing for decisionsupport in main memory," in *Proc. 25th Int. Conf. VLDB*, Edinburgh, U.K., 1999, pp. 78–89.

[9] D. I. August, W. W. Hwu, and S. A. Mahlke, "A framework for balancing control flow and predication," in *Proc. 30th MICRO*, North Carolina, NC, USA, 1997, pp. 92–103.

[10] H. Kim, O. Mutlu, Y. N. Patt, and J. Stark, "Wish branches: Enabling adaptive and aggressive predicated execution," *IEEE Micro*, vol. 26, no. 1, pp. 48–58, Jan./Feb. 2006.

[11] M. Zukowski, P. Boncz, N. Nes, and S. Héman, "MonetDB/X100—A DBMS in the CPU cache," *IEEE Data Eng. Bull.*, vol. 28, no. 2, pp. 17–22, Jun. 2005

[12] P. A. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyperpipelining query execution," in *Proc. 2nd Biennial CIDR*, Pacific Grove, CA, USA, 2005.

[13] B. V. Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?" in *Proc. IEEE 20th Annu. Int. Symp. FCCM*, Toronto, ON, Canada, 2012, pp. 232–239.

Naveena.R **Saigeetha.S** **Rajalakshmi.S**