BOAI: Fast Alternating Decision Tree Induction based on Bottom-up Evaluation*

Bishan Yang, Tengjiao Wang, Dongqing Yang, and Lei Chang

Key Laboratory of High Confidence Software Technologies (Peking University),
Ministry of Education, CHINA
School of Electronics Engineering and Computer Science,
Peking University, Beijing, 100871, CHINA
{bishan_yang, tjwang, dqyang, changlei}@pku.edu.cn

Abstract. Alternating Decision Tree (ADTree) is a successful classification model based on boosting and has a wide range of applications. The existing ADTree induction algorithms apply a "top-down" strategy to evaluate the best split at each boosting iteration, which is very time-consuming and thus is unsuitable for modeling on large data sets. This paper proposes a fast ADTree induction algorithm (BOAI) based on "bottom-up" evaluation, which offers high performance on massive data without sacrificing classification accuracy. BOAI uses a pre-sorting technique and dynamically evaluates splits by a bottom-up approach based on VW-group. With these techniques, huge redundancy in sorting and computation can be eliminated in the tree induction procedure. Experimental results on both real and synthetic data sets show that BOAI outperforms the best existing ADTree induction algorithm by a significant margin. In the real case study, BOAI also provides better performance than TreeNet and Random Forests, which are considered as efficient classification models.

Key words: classification, decision tree, ADTree, BOAI

1 Introduction

Boosting procedure has been proved to be very helpful to improve the accuracy of decision tree classifiers. AdaBoost, introduced by Freund and Schapire [1], is the most commonly used boosting procedure. It has been successfully used to combine with decision trees like C4.5 [2], and produces very good classifiers. However, the output classifiers are often large, complex and difficult to interpret. Freund and Mason solved this problem by proposing Alternating Decision Tree (ADTree) [3] and an induction algorithm based on AdaBoost. ADTrees can produce highly accurate classifiers while generating trees in small size which are easy to interpret. They can also provide a measure of classification which helps to rate prediction confidence. Based on these attractive features, ADTrees have

* This work is supported by the National '863' High-Tech Program of China under grant No. 2007AA01Z191, and the NSFC Grants 60473051, 60642004

a wide range of applications, such as customer churn prediction, fraud detection and disease trait modeling [4, 5].

Whereas ADTree is a very popular model in classification, it faces a problem of training efficiency on huge volumes of data. The original induction algorithm proposed by Freund and Mason performs split evaluation with a top-down strategy at each boosting round. The algorithm is very expensive to apply to large knowledge discovery tasks. Several techniques have been developed to tackle the efficiency problem. However, there still be a large space to improve.

For very large data sets, several techniques have been developed, mainly based on traditional decision trees. SLIQ [6] and Sprint [7] use new data structures and processing methods to scale decision trees to large data sets. PUBLIC [8] integrates the MDL "pruning" phase into the tree "building" phase. RainForest [9] uses AVC-groups which are sufficient for split evaluation to speed up tree construction. BOAT [10] provides techniques to build trees based on a subset of data and results in faster tree construction. All these algorithms are based on traditional decision trees, which compute the split criteria only based on the information of the current node, and thus can not directly apply to ADTree.

With regards to the scalability of ADTree, several optimizing methods are introduced in [11]: Z_{pure} cutoff, merging and three heuristic mechanisms. The former two methods gain little efficiency until reaching 50 boosting iterations. Although the heuristic methods reduce the induction complexity obviously, they generate trees that are different from the original trees. In [12], ADTree is upgraded to first order logic and three efficiency improvements are proposed. The caching optimization, which stores the success (failure) of each rule for each relevant instance in a bit-matrix, was shown to be most effective. Nevertheless, the additional memory consumption grows fast in the number of boosting rounds.

To address the efficiency challenges, we introduce a novel ADTree induction algorithm called BOAI[1] that gains great efficiency in handling large data sets without sacrificing classification accuracy. BOAI uses a pre-sorting technique and a bottom-up evaluation approach based on VW-group to avoid much redundancy of sorting and computation in the tree building process. To validate the efficiency of BOAI on large data sets, we conduct comprehensive experiments on both synthetic and real data sets. We also apply BOAI to a real data mining application to evaluate its performance. The results are very encouraging as BOAI offers significant performance improvements.

The remainder of this paper is organized as follows. Section 2 describes ADTree and its Induction algorithm. Section 3 introduces the new techniques used in BOAI and then describes the algorithm and implementation issues. Section 4 presents the experimental results on both real and synthetic data. Finally, section 5 concludes the paper.

[1] The acronym BOAI stands for BOttom-up evaluation for ADTree Induction

2 Preliminaries

2.1 Alternating Decision Tree

Unlike traditional decision trees, Alternating Decision Tree (ADTree) contains two kinds of nodes: decision nodes and prediction nodes. Each decision node involves a splitting test while each prediction node involves a real-valued number (Fig. 1 shows an example). A decision node splits sets of training instances into two parts with each part belonging to a prediction node. An instance defines a set of paths along the tree from the root to some of the leaves. The classification of an instance is the sign of the sum of the prediction values along the paths defined by this instance and the sum can be interpreted as a measure of confidence. For example, the classification of the instance (age, income) = (35, 1300) is sign(0.5-0.5+0.4+0.3) = sign(0.7) = +1. The prediction nodes in the instance's defined paths are shadowed in the figure.

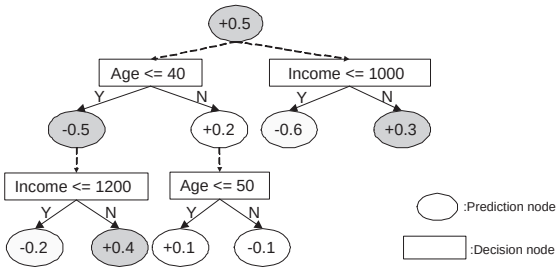

Fig. 1. an example of ADTree

2.2 ADTree Learning with AdaBoost

Freund and Mason presented the ADTree induction algorithm with the application of AdaBoost [3]. There are two sets maintained in the algorithm, a set of preconditions and a set of rules, denoted as $\mathcal{P}$ and $\mathcal{R}$ respectively. $\mathcal{C}$ denotes the set of base conditions. The algorithm is given as Algorithm 1. The induction procedure can be divided into two phases at each boosting iteration: *Split Evaluation* and *Node Partition*. In the evaluation phase (line 2-5), the algorithm evaluates all the splits basically by a top-down strategy. It traverses the tree by a depth-first search. For each prediction node, it scans the instances at the node to compute the total weight of the instances that satisfy each possible condition. Before the computation, the instances need to be sorted on each numeric attribute to obtain the possible splits of the attribute. The best split is found by minimizing Z-value of the function that measures the weighted error of the rules (Equation 1). In the partition phase (line 6-8), a new rule is added to set $\mathcal{R}$ and two prediction values are calculated. A decision node is created according to the

rule and two prediction nodes are created associated with the prediction values. Applying the rule, the instances are split into two parts with each part propagated to one of the prediction nodes. After each boosting round, the weights of the instances belonging to these two prediction nodes are updated, decreasing for correctly classified instances and increasing for incorrectly classified instances. As described above, the complexity of the algorithm mainly lies in the evaluation phase because of the huge sorting and computational cost. They will result in low-efficiency when training on massive data sets.

Algorithm 1 ADTree Learning with an application of AdaBoost

Input: $\mathcal{S} = \{(x_1,y_1), \ldots, (x_m,y_m)\} \mid x_i \in R^d$, $y_i \in \{-1,+1\}\}$
Initialize. Set each instance's weight $w_{i,0} = 1.0$, $1 \leq i \leq m$. Set the rule set $\mathcal{R}_1 = \{True\}$. Calculate the prediction value for the root node as $a = \frac{1}{2} \ln \frac{W_+(c)}{W_-(c)}$, $c = True$. $W_+(c)$ (resp. $W_-(c)$) is the total weight of the positive (resp. negative) instances that satisfying condition c. Adjust the weights of the instances at the root node as $w_{i,1} = w_{i,0} e^{-ay_i}$.

1: **for** $t = 1$ to T **do**
2: **for all** c_1 such that $c_1 \in \mathcal{P}_t$ **do**
3: **for all** c_2 such that $c_2 \in \mathcal{C}$ **do**
4: Calculate

$$Z_t(c_1, c_2) = 2(\sqrt{W_+(c_1 \wedge c_2) W_-(c_1 \wedge c_2)} + \sqrt{W_+(c_1 \wedge \neg c_2) W_-(c_1 \wedge \neg c_2)}) + W_+(\neg c_1) \quad (1)$$

5: Select c_1, c_2 which minimize $Z(c_1, c_2)$ and set $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_t$: precondition c_1, condition c_2, two prediction values $a = \frac{1}{2} \ln \frac{W_+(c_1 \wedge c_2)+1}{W_-(c_1 \wedge c_2)+1}$, $b = \frac{1}{2} \ln \frac{W_+(c_1 \wedge \neg c_2)+1}{W_-(c_1 \wedge \neg c_2)+1}$ $\}$
6: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{c_1 \wedge c_2, c_1 \wedge \neg c_2\}$
7: Update weights: $w_{i,t+1} = w_{i,t} e^{-r_t(x_i) y_i}$, $r_t(x_i)$ is the prediction value that the rule r_t associates with the instance x_i.

3 BOAI - Bottom-up Evaluation for ADTree Induction

In this section, we present BOAI, an efficient ADTree induction algorithm. Unlike the original top-down mechanism, BOAI performs split evaluation using a bottom-up approach. It gains significant efficiency of tree induction while maintaining the classification accuracy. In addition, it can easily combine with the optimizations in [11].

To bring down the large cost in the evaluation phase, BOAI uses a pre-sorting technique and applies a bottom-up evaluation based on VW-group for split evaluation. The pre-sorting technique aims at reducing the sorting cost to linear time. The bottom-up evaluation approach evaluates splits from the leaf nodes to the root node. On each prediction node, the evaluation is performed on

a VW-group, which stores sufficient statistics for split evaluation. The VW-group can be built up in linear time by a bottom-up merging process. The combination of these techniques enables BOAI to induce ADTree efficiently on large data sets. Following are details about these techniques.

3.1 Pre-Sorting technique

BOAI uses a special sorting technique as a preprocessing step. It works as follows. At the beginning of the algorithm, the values of each numeric column in the input database are sorted separately. Suppose for attribute A, the sorting space of its distinct values is $x_0, x_1, \ldots, x_{m-1}$. These values can be mapped into an integer value field $0, 1, \ldots, m-1$, which reflects the offset address of each value in the sorted space. Then the original values in the data are replaced with their mapped values in the value field. As the replaced values preserve the original value distribution on the attribute, it will not affect the following evaluation on the attribute. The benefit of this method is that we can easily use the actual attribute values to index into a sorted array. The detailed analysis is given in Sect. 3.3.

3.2 Data Structure

Note that for a prediction node p, the possible splits of an attribute A can be evaluated separately from other attributes. Besides, the total weight of the instances that satisfy each condition on each prediction node is needed to compute for split evaluation. Let $F(p)$ denote the instances projected onto node p. Similar to the AVC-set structure in [9], the VW-set (The acronym VW stands for Attribute-Value, Class-Weight) of a predictor attribute A at node p is defined to preserve the weight distribution of each class for each distinct value of A in $F(p)$. Each element in a VW-set contains an attribute value field and a class-weight field (operations on the class-weight are performed on weights of two classes (positive and negative) respectively). The class-weight field can be viewed as caching $W_+(A = v)$ and $W_-(A = v)$ for each distinct attribute value v of A. Suppose in $F(p)$, $v_0, ..., v_{m-1}$ are the distinct values of A. If A is a categorical attribute, the split test is of form $A = v_i$, where $0 \leq i \leq m-1$. If A is a numeric attribute, the split test is of form $A \leq (v_i + v_{i+1})/2$, where $0 \leq i \leq m-2$, and $v_0, ..., v_{m-1}$ are in sort order. For each possible condition c on A, $W_+(c)$ and $W_-(c)$ can be easily calculated by scanning the VW-set of A at p. The VW-group of node p is defined to be the set of all VW-sets at node p, and p can be evaluated based on its VW-group, whose result is the same as that of being evaluated via scanning $F(p)$. The size of the VW-set of an attribute A at node p is determined by the number of distinct values of A in $F(p)$ and is not proportional to the size of $F(p)$.

3.3 Bottom-up Evaluation

The great complexity in the split evaluation is due to the exhaustive exploring on all possible splits at each boosting round. Since the weights of instances

change after each round, we can not simply ignore evaluating any possible split in the following round. A fundamental observation is that there are recurrences of instances at the prediction nodes. When evaluating the prediction nodes recursively from the root to the leaves, the instances in fact have a great deal of computing and sorting overlap. To eliminate this crucial redundancy, we propose a bottom-up evaluation approach. The bottom-up approach evaluates splits from the leaf nodes to the root node based on the VW-group of each node. It uses the already computed VW-groups of the offspring nodes to construct the VW-groups of the ancestor nodes. The approach of VW-group construction is described as follows.

For a leaf prediction node p, it scans the instances at p to construct the VW-set of each attribute. For a categorical attribute A, a hash table is created to store the distinct values of A. As the attribute values in the VW-set of A are not required to be sorted, the VW-set can be constructed by collecting the distinct values of A from the hash table and computing the weight distributions on these values. For a numeric attribute A, the attribute values on A need to be sorted. With the pre-sorting technique, the sort takes linear time in most cases. Suppose there are N instances at node p and the mapped value field on A is range from 0 to $M-1$, where M is the number of distinct values of A. It takes one pass over N instances mapping their weights into the value field of A. Then the attribute values together with their corresponding weights will be compressed into the VW-set of A. The total time for getting sorted attribute values in the VW-set is $O(N+M)$. For most cases, M is smaller than N, in which case the running time is $O(N)$. If M is much larger than N, the algorithm switches to quick sort.

For an internal prediction node p, the VW-group is constructed through a merging process, with the VW-set of each attribute generated at a time. Each generation only require time $O(m)$ where m is the total number of elements in the two merging VW-sets. Suppose Z is the VW-set of attribute A at node p and X, Y are the VW-sets of A at node p_1 and p_2 which are two prediction nodes under a decision node of p. If A is a categorical attribute, as the attribute values in X and Y are not sorted, Z can be generated by performing hash join on the attribute values in one pass over X and Y. If A is a numeric attribute, we can perform the merge procedure similar to merge sort to generate Z, and the attribute values in Z are kept in order after merging. Fig. 2 shows this process pictorially.

Since the VW-group of each prediction node can be constructed by the bottom-up approach, and each prediction node can be correctly evaluated based on its VW-group, the global minimum Z-value found by evaluating all the prediction nodes is correct. Because the best split found at each boosting round is correct, the tree induced by the bottom-up evaluation approach is the same as that induced by the top-down evaluation approach.

The reduced cost by using the bottom-up evaluation is remarkable. In the top-down evaluation, instances are sorted on each numeric attribute on every prediction node, with each sort taking at least $O(n \log n)$ time (n is the number

of the considered instances) in the average case. While in bottom-up evaluation, we focus on gaining orders of distinct attribute values whose cost is much inexpensive. Additionally, the spectacular redundancy in computing weight distributions is eliminated since the statistics are cached in VW-group to prevent being recomputed. Moreover, the bottom-up technique will not affect the accuracy of the original algorithm.

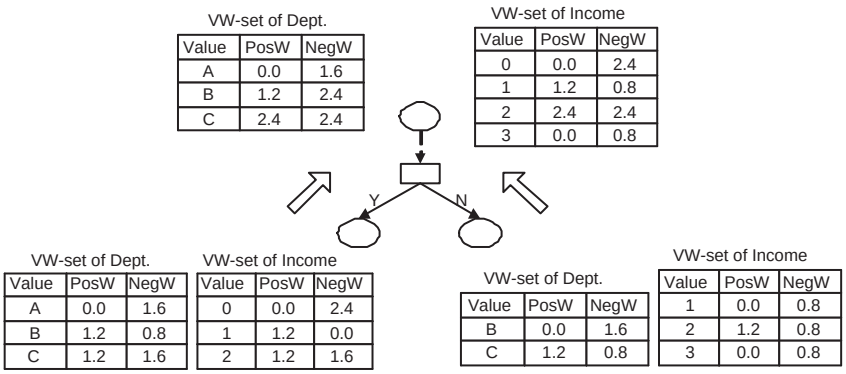

Fig. 2. Construct VW-set via merging: Example

3.4 Algorithm

In this section, we present BOAI algorithm. Note that the partition phase contributes a little to the complexity of tree induction. BOAI shares it with Algorithm 1. We just provide illustration about the evaluation phase here. Let p-VW-group denote the VW-group at prediction node p, and p-VW-set denote the VW-set contained in p-VW-group. The algorithm is given in Algorithm 2. The procedure is invoked at every boosting step, with the root node r as an input parameter. Since ADTree can have more than one decision node below a prediction node, the instances at the prediction node can be partitioned by different split criteria. We only consider partitions of one decision node for performing merging (we always choose the first decision node in BOAI). For other decision nodes, we view each of their prediction children as the root node of a subtree. The evaluating process will start from these root nodes individually. In this way, no redundant merging of VW-groups is performed. Note that when the combination is finished, the two VW-groups being merged can be deleted.

The optimizing techniques introduced in [11] can be easily integrated in BOAI and show better performance improvements. The Z_{pure} calculation can be sped up by merging the sum of the weights of the positive (negative) instances through the merging process of the bottom-up approach. The heuristic mechanisms can be performed by only evaluating the tree portion included in the heuristic path.

Algorithm 2 EvaluateSplits(Prediction Node p)

```
1: if p is a leaf then
2:    generate p-VW-group via scanning F(p)
3: else
4:    /*p1 and p2 are two children of p's first decision node*/
      p1-VW-group = EvaluateSplits(p1);
5:    p2-VW-group = EvaluateSplits(p2);
6:    p-VW-group ⇐ Merge p1-VW-group and p2-VW-group
7: for each attribute A do
8:    traverse p-VW-set of attribute A /* the value field stores v_0, ..., v_{m-1} */
9:    if A is a categorical attribute then
10:      for i = 0 to m − 1 do
11:         compute Z-value for test (A = v_i) using class-weight associated with v_i
12:   if A is a numeric attribute then
13:      for i = 0 to m − 2 do
14:         cumulate the sum of the class-weights associated with the former i values
            and v_i  /* the values are in sorted order */
15:         compute Z-value for test (A ≤ (v_i + v_{i+1})/2) using the cumulated sum
16: for each node s such that s is the child of p's other decision nodes do
17:    EvaluateSplits(s);
18: return p-VW-group;
```

4 Experiments

In this section, we perform comprehensive experiments on both synthetic and real data sets to study the performance of BOAI. In the first experiment, we compare efficiency of BOAI and ADT on synthetic data sets up to 500,000 instances. In the next, we use the real data sets contained 290,000 records with 92 variables to evaluate the efficiency of BOAI. At last, we apply BOAI to churn prediction application, comparing to ADT, Random Forests [13] and TreeNet [14], which are considered as accurate and efficient classification models. (TreeNet won the Duke/Teradata Churn modeling competition in 2003 and won the KDD2000 data mining competition.)

BOAI and ADT are written in C++. The software of TreeNet and Random forests are downloaded from the web site (http://www.salford-systems.com/churn .html) of Salford Systems. All our experiments were performed on AMD 3200+ CPU running Windows XP with 768MB main memory.

4.1 Synthetic Databases

In order to study the efficiency of BOAI, we used the well-known synthetic data generation system developed by the IBM Quest data mining group [15], which is often used to study the performance of decision tree construction [7–10]. Each record in this database consists of nine attributes. Among the attributes, six are numeric and the others are categorical. Ten classification functions are used to generate different data distributions. Function 1 involves two predictor attributes

with respect to the class label. Function 7 is linear depending on four predictor attributes. We only show results of these two functions due to space limitation, the results are similar for other functions.

First, we examined the modeling time of BOAI and ADT as the number of the instances increases from 100,000 to 500,000. The number of boosting iterations is set to 10. We consider the Z_{pure} cut-off, merging and heuristic search techniques [11] in the comparison. In the following experiments, ADT and BOAI are default with Z_{pure} and merging options. Fig. 3 and Fig. 4 show the results of the two algorithms for function 1 and 7. BOAI is faster by a factor of six. Fig. 5 and Fig. 6 show the results of employing heuristic options (the produced models are different from those of the original algorithm). BOAI also makes significant gains for each heuristic option. We further investigated the cost of sorting and computation in the split evaluation. Fig. 7 shows that the sorting cost in ADT rises eight times faster than BOAI in function 1. Fig. 8 shows that BOAI is about twenty-two times faster than ADT in comparison of Z-value computation cost in function 1. As the above two cost are dominant cost during tree induction, they can explain why BOAI outperforms ADT by a large margin.

We also examined the effect of boosting iterations on BOAI. We changed the number of boosting iterations from 10 to 50 while fixing the number of the instances at 200,000. Fig. 9 and Fig. 10 show the results for Function 1 and Function 7. The results are both encouraging as BOAI grows much smoother than ADT with the increasing number of boosting iterations.

Fig. 11 and Fig. 12 show the effect of adding extra attributes with random values to the instances in the input database. The number of the instances are kept at 200,000 and the number of boosting iterations is set at 10. The additional attributes need to be evaluated but they will never be chosen as the split attribute. Thus the extra attributes increase tree induction time while the final classifier keeps the same. BOAI exhibits much more steady performance with the increasing number of attributes.

4.2 Real Data sets

In order to study the performance of BOAI in real cases, we experimented with a real data set obtained from China Mobile Communication Company. The data refers to seven months of customer usage, from January 2005 through July 2005. The data set consists of 290,000 subscribers covering 92 variables, including customer demographic information, billing data, call detail data, service usage data and company interaction data. The churn indicator attribute is the class attribute. We first study the training time of BOAI on the real data sets. Then we apply BOAI to churn prediction, comparing its performance to ADT, TreeNet and Random Forests. To guarantee the prediction accuracy, we don't consider heuristic techniques in the comparison.

We first compare the training time of BOAI and ADT. Fig. 13 shows the overall running time of the algorithms as the number of the input instances increases from 20,083 to 219,644 and the number of boosting iterations sets at 10. BOAI is about fourteen times faster than ADT. Then we change the number

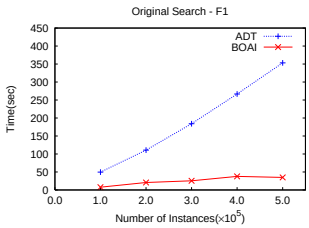

Fig. 3. Overall Time:F1

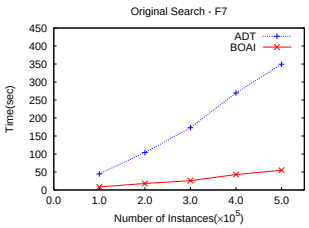

Fig. 4. Overall Time:F7

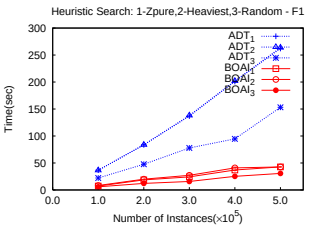

Fig. 5. Overall Time:F1

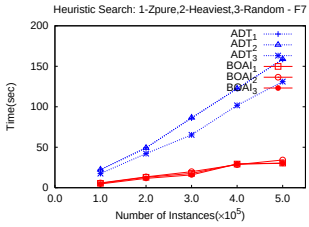

Fig. 6. Overall Time:F7

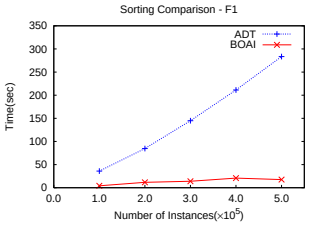

Fig. 7. Sorting Cost

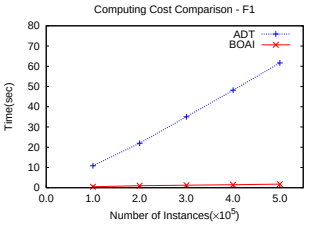

Fig. 8. Computing Cost

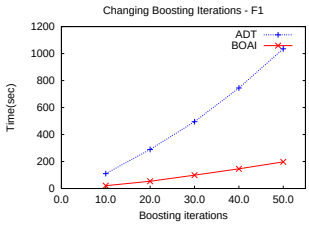

Fig. 9. Changing Iterations:F1

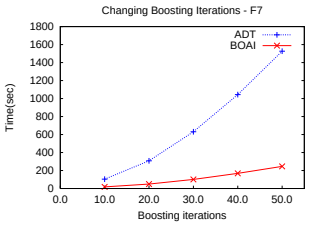

Fig. 10. Changing Iterations:F7

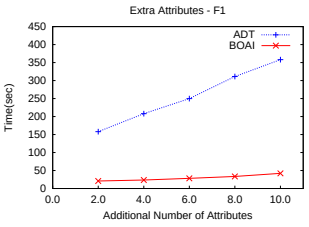

Fig. 11. Adding Attributes:F1

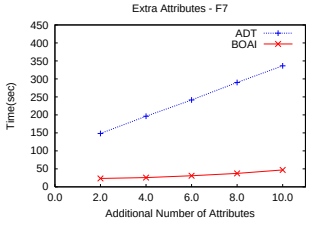

Fig. 12. Adding Attributes:F7

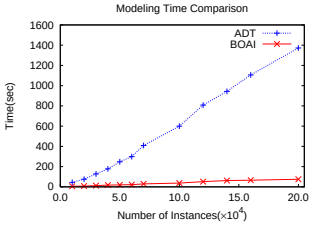

Fig. 13. Overall Time

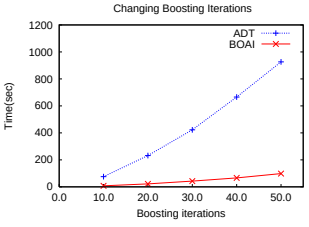

Fig. 14. Changing Iterations

of boosting iterations from 10 to 50 with 20,083 instances. Fig. 14 shows that BOAI offers more steady performance with the changing number of iterations. For memory usage, the largest size of the VW-group used in induction is only 10MB for 219,644 instances (with 92 attributes), which is also a small size to easily hold in memory.

In the next, we apply BOAI to churn prediction to study its performance. we sampled 20,083 examples from the original data set as a calibration set which has 2.1% churn rate, and 5,062 examples as a validation set which has 1.8% churn rate. Since the data is highly skewed, we take a re-balancing strategy to tackle the imbalanced problem. As a pre-processing step, we multiply the weight of each instance in the minority class by W_{maj}/W_{min}, where W_{maj} (resp. W_{min}) is the total weight of the majority (resp. minority) class instances. In this way, the total weights of the majority and minority instances are balanced. Unlike sampling [16], re-balancing weights has little information loss and does not introduce more computing power on average.

Table 1. Performance comparison on churn analysis

Models	F-measure	G-mean	W-accuracy	Modeling Time (sec)
ADT (w/o re-balancing)	56.04	65.65	44.53	75.56
Random Forests	19.21	84.04	84.71	960.00
TreeNet	72.81	79.61	64.40	30.00
BOAI	50.62	90.81	85.84	7.625

We compare the predicted accuracy of BOAI, ADT (without re-balancing), TreeNet and Random Forests, with measures of F-Measure, G-Mean and Weighted Accuracy [17], which are commonly used to evaluate performance on skewed class problem. The modeling time of these algorithms is also given. The results, shown in Table 1, indicate that BOAI outperforms ADT, TreeNet and RF when evaluated in terms of G-mean and Weighted-Accuracy. More importantly, BOAI uses the least modeling time.

5 Conclusion

In this paper, we have developed a novel approach for ADTree induction, called BOAI, to speed up ADTree construction on large training data sets. The key insight is to eliminate the great redundancy of sorting and computation in the tree induction by using a bottom-up evaluation approach based on VW-group. In experiments on both synthetic and real databases, BOAI offers significant performance improvements over the best existing algorithm while constructing exactly the same ADTree. We also study the performance of BOAI for churn prediction. With the re-balancing technique, BOAI offers good prediction accuracy while spends much less modeling time compared with ADT, TreeNet and

Random Forests, which are reported as efficient classification models. Therefore, BOAI is an attractive algorithm for modeling on large data sets. It has been successfully used for real-life churn prediction in telecommunication.

Acknowledgments. We gratefully thank Prof. Jian Pei in Simon Fraser University for his insightful suggestions.

References

1. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119–139 (1997)
2. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
3. Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In: 16th International Conference on Machine Learning, pp. 124–133 (1999)
4. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition. Morgan Kaufmann (2005)
5. Liu, K.Y., Lin, J., Zhou, X., Wong, S.: Boosting Alternating Decision Trees Modeling of Disease Trait Information. BMC Genetics, 6(1) (2005)
6. Mehta, M., Agrawal, R., Rissanen J.: SLIQ: A fast scalable classifier for data mining. In: 5th International Conference on Extending Database Technology, pp. 18–32 (1996)
7. Shafer, J., Agrawal, R., Mehta, M.: SPRINT: A scalable parallel classifier for data mining. In: 22nd International Conference on Very Large Databases, pp. 544–555 (1996)
8. Rastogi, R., Shim, K.: PUBLIC: A Decision Tree Classifier that Integrates Pruning and Building. In: 24th International Conference on Very Large Database, pp. 315–344 (1998)
9. Gehrke, J., Ramakrishnan, R., Ganti, V.: Rainforest: A framework for fast decision tree construction of large data sets. In: 24th International Conference on Very Large Database, pp. 127–162 (1998)
10. Gehrke, J., Ganti, V., Ramakrishnan, R., Loh, W.Y.: BOAT—optimistic decision tree construction. In: ACM SIGMOD International Conference on Management of Data, pp. 169–180 (1999)
11. Pfahringer, B.,Holmes, G., Kirkby, R.: Optimizing the Induction of Alternating Decision Trees. In: 5th Pasific-Asia Conference on Knowledge Discovery and Data Mining, pp. 477–487 (2001)
12. Vanassche, A., Krzywania, D., Vaneyghen, J., Struyf, J., Blockeel, H.: First order alternating decision trees. In: 13th International Conference on Inductive Logic Programming, pp. 116–125 (2003)
13. Breiman, L.: Random forests. Machine Learning Journal, 45, 5–32 (2001)
14. `http://www.salford-systems.com/products-treenet.html`
15. IBM Intelligent information systems, `http://www.almaden.ibm.com/software/quest/resources/`
16. Maloof, M.: Learning when data sets are imbalanced and when costs are unequal and unknown. In: ICML Workshop on Learning from Imbalanced Data Sets (2003)
17. Chen, C., Liaw, A., Breiman, L.: Using Random Forest to Learn Imbalanced Data. Technical Report 666, Statistics Department, University of California at Berkeley (2004)