Parallel Data Mining Revisited. Better, Not Faster

Zaenal Akbar, Violeta N. Ivanova, and Michael R. Berthold

Nycomed-Chair for Bioinformatics and Information Mining
Dept. of Computer and Information Science, University of Konstanz
Box 712, D-78457 Konstanz, Germany
firstname.lastname@uni-konstanz.de

Abstract. In this paper we argue that parallel and/or distributed compute resources can be used differently: instead of focusing on speeding up algorithms, we propose to focus on improving accuracy. In a nutshell, the goal is to tune data mining algorithms to produce better results in the same time rather than producing similar results a lot faster. We discuss a number of generic ways of tuning data mining algorithms and elaborate on two prominent examples in more detail. A series of exemplary experiments is used to illustrate the effect such use of parallel resources can have.

1 Introduction

Research in Parallel Data Mining traditionally is focused on accelerating the analysis process - understandable in times of limited compute power and increasingly complex analysis algorithms. More recently, however, data mining research has split into two main themes: "big data" type analyses, where the goal is still the efficient mining of insights from increasingly large datasets on the one hand and more elaborate mining algorithms in order to find better and more representative patterns in not necessarily such large data repositories, on the other.

With regard to the latter, usage of parallel compute engines has not gained much attention as the compute time tends to be less critical – especially in times when computational resources even on desktop computers are available in such abundance. Nevertheless many if not all relevant algorithms rely on heuristics or user supplied parameters to somewhat reduce the otherwise entirely infeasible hypothesis space.

In this paper, we argue that modern architectures, which provide access to numerous parallel computing resources, emphasized by the recent advance of multi-core architectures, can also be utilized to reduce the effect of these user parameters or other algorithmic heuristics. Instead of trying to provide the same results faster then conventional algorithms we claim that interest in this area of analysis methods should also consider using parallel resources to provide **better** results in **similar time**. In the following we refer to this use of parallel resources

J. Hollmén, F. Klawonn, and A. Tucker (Eds.): IDA 2012, LNCS 7619, pp. 23–34, 2012.

as *tuning* models (in parallel) to distinguish it from the more common attempts to simply accelerate algorithms.

Obviously a number of algorithms can very easily be extended in such a manner. Most prominent are, of course, simple ensemble methods. Instead of sequentially training a bag of models it is embarassingly simple to train those in parallel. However, for many well known methods this type of straight forward parallelization is not applicable. Boosting already requires information about other models predictions and many other, presumably easy algorithms are not as easily parallelizable. Just try to parallelize a decision tree induction or a clustering algorithm. In addition, early experiments indicate that simply randomizing the collection of models is suboptimal – steering the parallel search by controlling diversity offers substantial promise here.

In this paper we propose two generic approaches for this type of parallel search algorithm. The resulting framework is demonstrated on two well-known algorithms that form the basis for many data mining methods.

Please note that this paper is meant as a demonstration of how parallel resources can be used for improving the quality of solutions of heuristic data mining algorithms in general. Therefore the parallel approaches discussed in this paper are simple and intuitive and do not aim to outperform optimized algorithms of the same type in practice. We strongly believe that the increasing amount of available parallel commodity hardware will lead to a rethinking of the design of heuristic data mining algorithms and the aim of this paper lies on this line of research.

2 Generic Approaches to Parallel Tuning

In order to better describe the generic approaches we are proposing in order to tune the accuracy of data mining algorithms let us first look at standard algorithms and how they search for a matching hypothesis in a given set of training data. Many of these algorithms (and these are the ones we are interested in here) follow some iterative scheme, where at each iteration a given model is refined. We can formalize this as follows:

$$m' = s\left(r(m)\right)$$

where m is the original model, for instance a neural network or a decision tree under construction, and m' is the next intermediate model, e.g. the neural network after one more gradient descent step or after another branch has been added to the decision tree. The two functions $s(\cdot)$ and $r(\cdot)$ describe a selection resp. refinement operation. In the case of the neural network, the refinement operator represents the gradient descent step and there is no real selection operator. For the decision tree, the refinement operator would actually return a set of expanded decision trees and the selection operator picks the one with the highest information gain.

As mentioned in the introduction, most existing algorithms employ some sort of heuristic optimization. Gradient descent performs a local optimization during

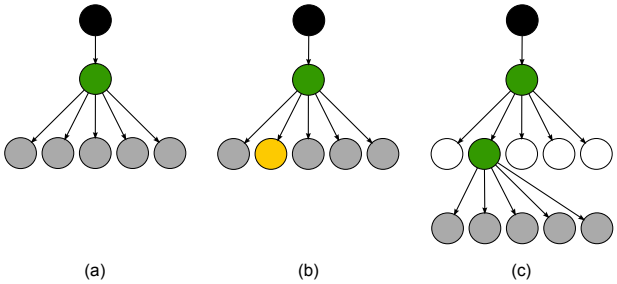

Fig. 1. The classic heuristic (often greedy) search algorithm. On the left (a), the current model m is depicted in green, the refinement options $r(m)$ are shown grey. The selection operator s picks the yellow refinement (b) and the next level then continues the search based on this choice.

the refinement operator. Greedy searches usually embed heuristics in both operators, they only generate a subset of all possible refinements and the selection operator has usually no way of estimating absolute quality but has to rely on a local heuristic, i.e. a greedy heuristic, as shown in Figure 1.

The formalization shown above now allows us to motivate the two general parallelization tuning methods - we can reduce (or – in extreme cases – even eliminate) the influence of the heuristics affecting either the selection function $s(\cdot)$ or the refinement operator $r(\cdot)$ or both.

2.1 Widening

This first approach, called *Widening* invests parallel resources into reducing the impact of the refinement heuristic by investigating more alternatives in parallel. In essence, this is similar to a beam search. Using the formalization from above, we can express widening as follows:

$$\{m'_1, \ldots, m'_{k'}\} = s\left(\{r(m_1), \ldots, r(m_k)\}\right).$$

The refinement operator $r(\cdot)$ does not necessarily change in this context and it can, as above, also return an entire set of refined models. The main point here is that we invest our available k parallel resources into running $r(\cdot)$ in parallel k times resulting in a much larger set of refined models. The selection function $s(\cdot)$ is now not forced to pick one locally optimal model but instead picks k' which are then refined again in parallel. In many cases $k = k'$, i.e. the number of explored models (or the width of the beam) will remain constant.

It is worth noting that this approach allows us not only to perform a beam search and explore the currently best k models but also to use a selection function which rewards diversity of the models and hence supports broader exploration of the hypothesis space. In recent work on the use of beam search in data mining algorithms evidence has arisen that such diverse space exploration is actually beneficial.

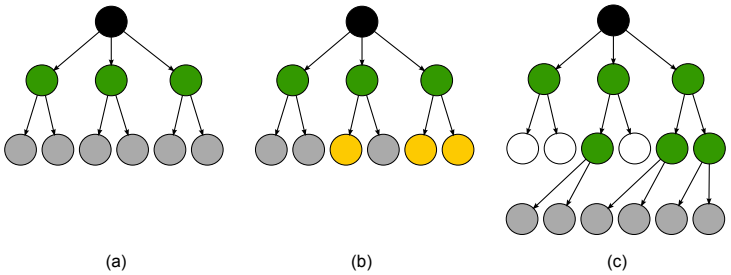

Fig. 2. Widening. From a set of models m (green circles), the refinement operator creates (in parallel) several sets of models (grey), shown on the left (a). The selection now picks a subset of the refined models (yellow circles in (b) and the search continues from those on the right (c).

Figure 2 shows how the widening approach works. At each step, the algorithm selects the k best models to be explored in the next step.

2.2 Deepening

The second approach, called *Deepening* focuses on reducing the impact of the selection heuristic:

$$m' = s_{\text{deep}}\left(r(m)\right).$$

In contrast to the widening approach, here only the selection function changes. For example, with regard to a decision tree: in order to determine the best split at each step only the immediate splits are considered. As we will describe in the next section, one could imagine looking further ahead to better estimate the quality of each split. In effect, the idea is to look forward and subsequently investigate how future steps of the refinement and selection process will be affected by the current choice. One could formalize this as follows:

$$s_{\text{deep}}\left(r(m)\right) = f\left(s\left(r(m)\right), s\left(r(s(r(m)))\right), \dots\right)$$

so the deeper selection operator is a function f of the normal, one-step-look-ahead selection criteria and the quality of further refinements of the original model. Figure 3 shows how the deepening approach works. At each step, the algorithm explores additional future models, selects the currently best model which, in turn, will lead to the best model in the future.

3 Example One: Tuned Set Covering

To illustrate the potential of the approaches discussed in the previous section, we chose the set cover problem, a problem that underlies quite a few data mining algorithms, for instance when trying to find the minimum number of rules or item sets.

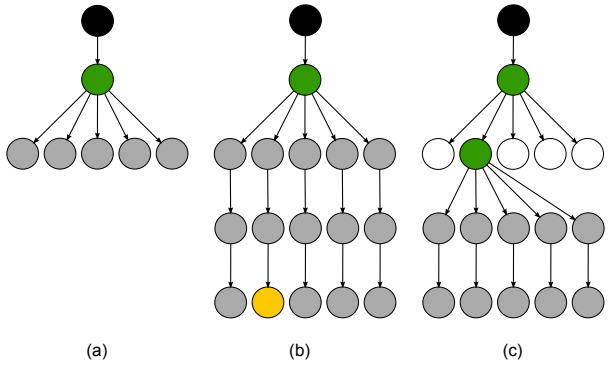

Fig. 3. Deepening. From one model m (green circles), the refinement operator creates (in parallel) several sets of models (grey), shown on the left (a). Note how, in order to apply the selection operator each of those choices is now expanded at a much greater depth (in parallel) using the classic heuristic search process. The selection then picks the refined models (yellow circles in (b)) that indicate the best performance at the deeper level, however the search continues from the models at the first level (c).

Although finding an optimal solution for the set cover problem is an NP-complete problem [1], a greedy algorithm, which at each step selects the subset with the largest number of uncovered elements, is widely used. Regardless of its simplicity, this classic greedy algorithm performs surprisingly well [2]. It can be shown that it achieves an approximation ratio of $H(n)$, where n is the size of the set to be covered.

However, please bear in mind that we are not interested in presenting a competitive new parallel set covering algorithm but that we are using this problem to illustrate how the tuning approaches presented above can be utilised. Hence we skip reviewing the state of the art in algorithmic improvements for the set covering problem here.

3.1 The Set Cover Problem

We consider the standard (unweighted) Set Cover problem. Given a universe X of n items and a collection $\mathcal{S}$ of m subsets of X : $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$. We assume that the union of all of the sets in $\mathcal{S}$ is X, with $|X| = n$: $\bigcup_{S_i \in \mathcal{S}} S_i = X$. The aim is to find a sub-collection of sets in $\mathcal{S}$, of minimum size, that covers all elements of X.

3.2 Greedy Set Covering

The greedy algorithm [2] attempts to construct the minimal set cover in the following way. It starts with the empty set being the temporary cover and at each

step selects and adds a single subset to it. The subset selected is the one, which contains the most elements that are not yet covered by the temporary cover. To be consistent with the terminology previously defined: if C is the temporary cover, a refinement generated by $r(C)$ represents the addition of a single subset, not yet part of C, to C. From all the possible refinements, generated by $r(\cdot)$, the one with the largest number of elements is chosen as the new temporary cover.

3.3 Tuning of Set Covering

Two possible tuning possibilities of the Set Cover algorithm can be derived from the simple algorithm described above.

Widened Set Covering. In contrast to the greedy algorithm, the widening of the greedy algorithm builds k temporary covers in parallel. The focus in this algorithm is to use resources to explore (possibly very) large number of refinements in parallel, depending on the available resources. Here k is referred to as "widening" parameter.

A single iteration of the widened algorithm then operates as follows. Let $C_1, \cdots C_k$ represent the k temporary covers. A refinement of C_i is created by adding a new subset to C_i. For each C_i, the k refinements which contain the largest number of elements, are selected. This results in $k*k$ refinements in total. From those, the top k refinements are selected, resulting in k new temporary covers $C_1^{'}, \cdots, C_k^{'}$. The choice of parameter k depends on the available compute resources. If k parallel resources (cores, computers) are available the running time of the algorithm will remain the same as the one for the simple greedy algorithm. As we will see later, the quality of the solutions will increase with larger k, due to more options being explored.

Deepened Set Covering. Unlike the widening of the greedy algorithm, the focus of the deepening is to not explore several options in parallel but use the additional computing resources to evaluate the quality of the candidates in more depth. This results in a less greedy algorithm as the choice at each step is based on more knowledge about the "future" quality of that solution.

At a given step, the algorithm explores the k refinements, which cover the largest number of elements. From them the algorithm selects only one as a new temporary cover. To select from the k refinements in the selection stage, the algorithm builds "deeper covers" for each of them l steps ahead in parallel. The deeper cover for a refinement i is built by performing the simple greedy selection l times starting from refinement i. We will refer to this deeper cover as *l-deep cover*. The refinement with the largest l-deep cover is selected as a temporary cover for the next iteration of the algorithm. A breadth parameter k determines how many refinements will be explored at each stage, and a depth parameter l determines how many steps "ahead" will the algorithm do to build the future covers.

3.4 Experimental Evaluation

In order to compare the greedy heuristic and the two tuning approaches – widening and deepening, we used a number of publicly available benchmarks. We explicitly selected data sets, which pose a challenge for the simple greedy algorithm in order to demonstrate the merits of the tuned approaches. The greedy set covering algorithm is usually at a disadvantage when at many steps it has to select among many equally good subsets. So we picked data sets exhibiting this property.

The two data sets discussed here, $Rail - 507$ and $Rail - 582$, stem from the OR library [3] and arose from a real-life railway-crew scheduling problem. $Rail - 507$ is based on $|X_{507}| = 507$ elements and $|\mathcal{S}_{507}| = 63,009$ sets whereas $Rail - 582$ consists of $|X_{582}| = 582$ elements and $|\mathcal{S}_{582}| = 55,515$ sets. For both data benchmarks the sets themselves are fairly small none of them has a cardinality of more than 12. So the greedy algorithm quite naturally often encounters cases where different alternatives have exactly the same benefit. In addition, we ignored the cost information as we are dealing with the unicost version of the set cover problem here.

To assess a possible improvement achieved by parallelization, we compared the sizes of the obtained solutions, e.g. the number of sets in the final cover that each of the algorithm returns. The focus of our experiments lies on how the quality of the solutions changes, as we vary the number of used parallel resources. To evaluate the average performance of the algorithms with respect to the parallelization parameter, for each data set each algorithm was run randomized 50 times and the mean and the standard deviation of the size of the cover was reported. If our initial hypothesis is correct, the average performance should go up (here: the size of the cover should go down) when more parallel resources are used. Note that for the deepening the depth parameter is set to be maximal for all experiments. That is, during the selection the look ahead is conducted until the complete cover is reached.

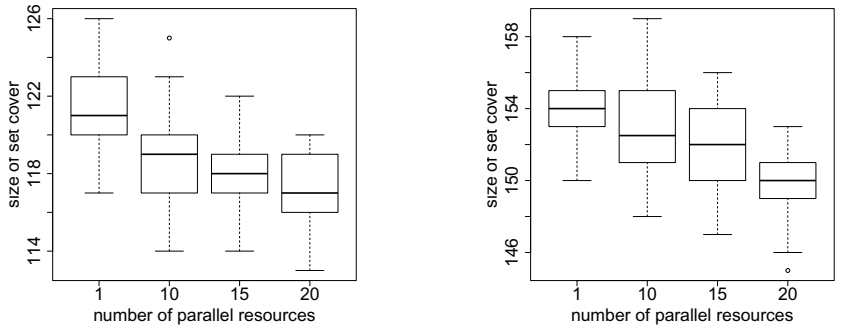

Fig. 4. Results for Widening on two data sets from the OR library (see text for details). As expected, with increasing widening of the search, the performance gets better and the standard deviation goes down.

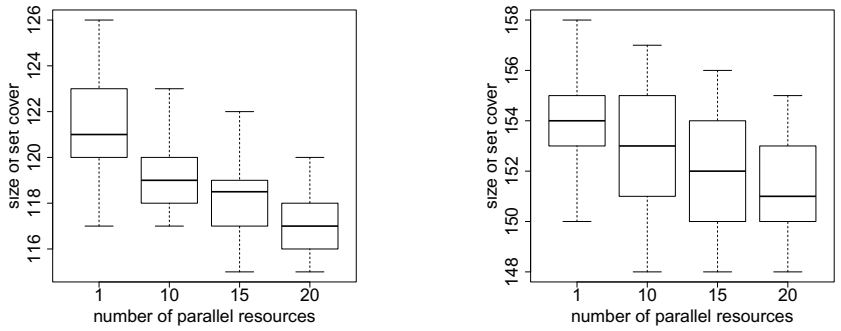

Fig. 5. Results for Deepening on two data sets from the OR library showing a similar trend to the widening approach

Figures 4 and 5 summarize the results. Note that number of parallel resource = 1 corresponds to the simple greedy heuristic. From the results one can conclude that when increasing the parallel resources, the mean of the obtained set cover decreases. A decrease in standard deviation also indicates that we are indeed converging towards a optimal solution.

4 Example Two: Tuned Decision Tree Induction

As a second example we investigated the well known decision tree models. The most prominent examples for decision tree learning are CART [4], ID3 [5], C4.5 [6] but many variations exist. Typically, however, they all start from a root node and grow the tree by splitting the data set recursively until a given stopping criterion is satisfied. The partitioning (or splitting) criterion is usually based on a greedy approach which picks the locally best attributes at each node. To determine the quality of a potential split, a measure for the expected information gain is derived from the available training data.

Mapping this to our representation, the refinement operator would create a series of possible splits at each node and the selection operator picks the split with the highest information gain (or another, similar measure of split quality).

4.1 Widening Decision Tree

From the discussion above it is quite obvious that random forests [7] are an incarnation of our widening approach by creating k trees on random data (and feature) subsets. However, in the end, the entire ensemble is used which is not the focus of our work – classic widening will pick the best model from the k models that were generated.

A straightforward extension would be to simply start with random starting points but poll the resulting trees (and the various possible refinements from each model in the current set) and use a selection criteria which rewards both tree size as well as accuracy estimates.

4.2 Deepening Decision Tree

Deepening decision tree induction requires more advance algorithmic modifications. The refinement operator expands candidate refinement solutions in parallel deeper until a given level k. A greedy selection operator is then employed to select the best solution to explored at the next iteration based on this deeper split quality estimate.

This seems an obvious extension of the standard decision tree learning methods which one would expect intuitively to perform well. However, the impact of the local, greedy choices on larger decision trees is not quite as dramatic as one expects. "Missed" opportunities at higher levels can easily be fixed by splitting (then just in neighboring branches) on that same attribute further down. In [8] such effects of "look ahead strategies" have been mentioned before and in [9] it was even reported that applying look-ahead in decision trees can produce trees that are both larger and less accurate than trees built by the normal algorithm.

In order to emphasize the potential benefit of tuning on this type of model learning we focused on decision stumps [10] which are essentially trees limited to a certain depth. Instead of constructing the tree until a certain criterion is met, these algorithms stop when a certain depth is reached. These types of models should be more sensitive to suboptimal local greedy choices when selecting splits and one should expect to see an effect of the deepening procedure described above.

4.3 Experimental Evaluation

Figure 6 shows results from widening (left) and deepening (right) on the Libras-Movement Dataset[1] with decision stumps limited to a depth of 6. We again ran 50 experiments on randomly chosen subsets of the training data (picking 90% of the data randomly each time). Also here we can see how the quality of the decision tree (here measured by the performance on the test data) improves when we conduct deeper refinements. Also the standard deviation of the accuracy goes down, indicating that we are converging towards an optimal solution.

5 Related Work

A wealth of literature exists relating to parallelized approaches for data mining and other algorithms. However to the best of our knowledge, no attempt has been made to employ parallel resources to improve accuracy in a similarly structured way. Nevertheless, many of the ideas presented elsewhere can be cast into the framework presented here one way or the other.

Some strategies, similar to the deepening strategy we discussed here, have already been proposed as a means to improve the accuracy of existing heuristics on sequential algorithms. For instance in [11] a "lookahead-search" approach to improve some greedy algorithms was discussed. With regard to decision trees in

[1] UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml/index.html`

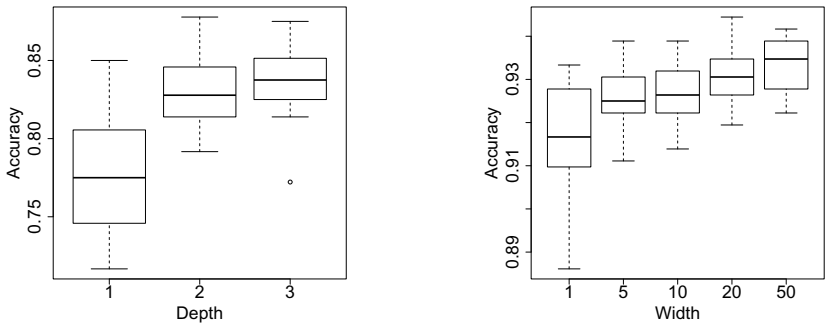

Fig. 6. Widening and Deepening decision tree induction (see text for details)

particular a number of sequential approaches have employed look-ahead strategies. In [9] the authors compare no lookahead and one-level lookahead and show the benefit of this type of lookahead were small or even none which confirms our observations with unstumped decision trees discussed earlier.

In [12] the lookahead approach is used differently. Instead of trying to find the split that produces the optimal tree, a greedy measure is used to find the locally optimal binary split. The subtree that is grown after this split is considered for the split decision. Their results show the algorithm to be useful in many cases. And finally in [13] two other lookahead-based algorithms for anytime induction of decision trees are presented. The depth-k look-ahead algorithm tests the effect of a split further down the tree until level k and in the second algorithm, each candidate split is evaluated by summing up the estimated size of each subtree, taking the smallest subtree as the optimal subtree at this stage.

Even more literature exists for the Set Cover problem as this is one of the most fundamental and best studied problems in optimization. A large amount of heuristics to solve it are also available, some of which focus on improving the accuracy of the result, others on the efficiency of obtaining a solution. Many of those ideas could also be resued for widening or deeping the search. The performance of the simple greedy algorithm comes very close to the best possible result for a polynomial-time algorithm, with optimality guarantees of $\ln(n)$. Various sequential algorithms have been developed to improve the bound and achieve $\log c$ factor from the optimal solution, such as [14], where the authors propose deterministic polynomial time method for finding a set cover in a set system (X, R) of VC-dimension d such that the size of the solution is at most $O(d \log(dc))$ of the optimal size, c.

Multiple parallel approaches exist for the set covering problem to improve the efficiency of the existing sequential algorithms. In [15] parallelization has been used to improve efficiency by "bucketing" utility values (the number of new elements covered) by factors of $(1 + \epsilon)$ and processing sets within a bucket in parallel. The bucketing approach ensures diversity of the explored hypothesis space, which could be particularly interesting for the purposes discussed in this

paper. In [16], a similar approach has been employed, again resulting in a linear-work RNC $(1+\epsilon)H_n$-approximation algorithm.

A veritable flood of publications is available on various aspects of other data mining algorithms for various types of distributed and parallel architectures. The focus of those papers is almost exclusively on reproducing the same results as the sequential algorithm but in a much shorter time. However the scope of this paper is not to present yet another parallel algorithm that outspeeds the sequential version. Instead, we present a general approach of how increasing computing power can be used to increase accuracy and reduce the impact of the underlying heuristics. Nevertheless in order to develop truly useful tuned algorithms it is worth investigating this literature and learning from the lessons presented there.

6 Conclusion

We have introduced a generic approach to make use of parallel resources to improve the accuracy and reduce the heuristic bias of common data mining algorithms. We demonstrated this concept on two greedy heuristics – the greedy algorithm underlying the standard solution for the Set Cover problem and the induction of decision tree stumps. We presented two approaches for this type of "algorithm tuning": *deepening* and *widening*, both based on relaxing the greedy criterion for hypothesis refinement and selection.

Again, we emphasize that this type of work is not all that novel – research in the parallel computing community has already focused quite extensively on optimizing all sorts of search strategies. However, in the context of data mining, such approaches have not yet been utilized. In contrast to ensemble or other set-of-model approaches, we believe it is often still desirable to find a single (interpretable) model. Utilizing parallel resources to find a better quality model in similar time to sequential approaches offers potential here, especially when the tuning approaches presented above are combined with diversity criteria to enforce a thorough exploration of the hypothesis space.

We believe that this type of approach can also be used to estimate the quality of a solution, i.e. how close to the optimum the current solution actually is. This may enable us to build systems that, after a certain time, can provide hints as to how much more time (and/or computing resources) would be needed to achieve a certain model quality.

References

1. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
2. Johnson, D.S.: Approximation algorithms for combinatorial problems. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC 1973, pp. 38–49. ACM, New York (1973)

3. Beasley, J.E.: Or-library: Distributing test problems by electronic mail. The Journal of the Operational Research Society 41(11), 1069–1072 (1990)
4. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth and Brooks, Monterey (1984)
5. Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1(1), 81–106 (1986)
6. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
7. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
8. Quinlan, J.R., Cameron-Jones, R.M.: Oversearching and layered search in empirical learning. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1019–1024 (1995)
9. Murthy, S., Salzberg, S.: Lookahead and pathology in decision tree induction. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1025–1031 (1995)
10. Iba, W., Langley, P.: Induction of one-level decision trees. In: Proceedings of the Ninth International Workshop on Machine Learning, pp. 233–240 (1992)
11. Sarkar, U., Chakrabarti, P., Ghose, S., Desarkar, S.: Improving Greedy Algorithms by Lookahead-Search. Journal of Algorithms 16(1), 1–23 (1994)
12. Elomaa, T., Malinen, T.: On Lookahead Heuristics in Decision Tree Learning. In: Zhong, N., Raś, Z.W., Tsumoto, S., Suzuki, E. (eds.) ISMIS 2003. LNCS (LNAI), vol. 2871, pp. 445–453. Springer, Heidelberg (2003)
13. Esmeir, S., Markovitch, S.: Lookahead-based algorithms for anytime induction of decision trees. In: Twenty-First International Conference on Machine Learning, ICML 2004, pp. 33–40. ACM Press, New York (2004)
14. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite vc-dimension (preliminary version). In: Proceedings of the Tenth Annual Symposium on Computational Geometry, SCG 1994, pp. 293–302. ACM, New York (1994)
15. Berger, B., Rompel, J., Shor, P.W.: Efficient nc algorithms for set cover with applications to learning and geometry. Journal of Computer and System Sciences 49(3), 454–477 (1994)
16. Blelloch, G.E., Peng, R., Tangwongsan, K.: Linear-work greedy parallel approximate set cover and variants. In: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011, pp. 23–32. ACM, New York (2011)