Learning Optimal Decision Trees

Siegfried Nijssen SIEGFRIED.NIJSSEN@CS.KULEUVEN.BE
K.U.Leuven, Celestijnenlaan 200 A, Leuven, Belgium

Élisa Fromont ELISA.FROMONT@CS.KULEUVEN.BE
K.U.Leuven, Celestijnenlaan 200 A, Leuven, Belgium

Abstract

All currently known algorithms for learning decision trees are based on the paradigm of heuristic top-down induction. Although the results of these algorithms are usually good, there is no guarantee that the resulting trees are really as small, accurate or shallow as possible. In this paper, we introduce an algorithm for inducing the smallest most accurate decision tree on training data. This algorithm allows us to find out how well heuristic algorithms approximate truly optimal decision trees.

1. Introduction

Among the most popular prediction models in machine learning are the decision trees, because there are efficient, relatively easily understandable learning algorithms, and the models are easy to interpret. From this perspective, it is surprising that learning decision trees under constraints has not been given much attention. For the problems listed below, currently no broadly applicable algorithm exists even if some attempts have been made in [6] for the last two problems:

- given a dataset D, find the most accurate tree on this training data in which each leaf covers at least n examples;
- given a dataset D, find the k most accurate trees on this training data in which the majority class in each leaf covers at least n examples more than any of the minority classes;
- given a dataset D, find the most accurate tree on this training data in which each leaf has a high statistical correlation with the target class according to a χ^2 test;
- given a dataset D, find the smallest decision tree in which each leaf contains at least n examples, and the expected accuracy is maximized;
- given a dataset D, find the shallowest decision tree which has an accuracy higher than *minacc*;
- given a dataset D, find the smallest decision tree which has an accuracy higher than *minacc*;

Clearly, in the interactive process that knowledge discovery is, the ability to pose *queries* that answer these questions can be very valuable.

In this paper, we will restrict ourselves to one of these optimization problems, although extensions to deal with the other queries are possible.

Assume given a training data set and a constraint which should be satisfied for every leaf in a decision tree. Then the optimization problem that we are interested in, is to find the smallest tree for which accuracy is maximized on the training data. An example of a constraint is that every leaf contains at least a certain minimum number of examples.

Most known algorithms for building decision trees use a top-down induction paradigm, in which a good split is chosen heuristically. If such algorithms do not find a tree that satisfies the specified constraints, this does not mean that a tree with given constraints does not exist—it only means that the chosen heuristic is not good enough to find it. To assess the quality of heuristic learners, we would need to know for a sufficiently large number of datasets what their true optimum under given constraints is. Until now, no attempts have been made to compute truly optimal trees for many datasets; most people have not seriously considered

Proceedings of the 18th Benelearn
P. Adriaans, M. van Someren, S. Katrenko (eds.)

the problem as it is known to be NP hard [14], and therefore, an efficient algorithm can most likely not exist.

Nevertheless, in this paper, we propose an algorithm for learning optimal decision trees under constraints. Given the high complexity of decision tree optimization, we do not expect that this algorithm will work in all conceivable settings. We will show, however, that it works on a significant number of UCI datasets that are commonly used to assess machine learning algorithms.

The paper is organized as follows. Section 2 introduces the necessary terminology. Section 3 formalizes the optimization problem that we are solving. Section 4 gives examples of decision trees that cannot be found by heuristic learning algorithms. Section 5 introduces our algorithm. Section 6 provides experiments; Section 7 discusses related work and section 8 concludes. More details can be found in [16].

2. Itemset Lattices for Decision Tree Mining

Let us first introduce some background information about *decision trees* and *frequent itemsets*.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items and let $D = \{T_1, T_2, \ldots, T_n\}$ be a bag of transactions, where each transaction T_k is an itemset such that $T_k \subseteq \mathcal{I}$. A transaction T_k contains a set of items $I \subseteq \mathcal{I}$ iff $I \subseteq T_k$. The transaction identifier set (TID-set) $t(I) \subseteq \{1, 2, \ldots n\}$ of an itemset $I \subseteq \mathcal{I}$ is the set of all identifiers of transactions that contain itemset I.

The frequency of an itemset $I \subseteq \mathcal{I}$ is defined to be the number of transactions that contain the itemset, i.e. $\mathit{freq}(I) = |t(I)|$; the support of an itemset is $\mathit{support}(I) = \mathit{freq}(I)/|D|$. An itemset I is said to be frequent if its support is higher than a given threshold *minsup*; this is written as $\mathit{support}(I) \geq \mathit{minsup}$ (or, equivalently, $\mathit{freq}(I) \geq \mathit{minfreq}$).

In this work, we are interested in finding frequent itemsets for databases that contain examples labeled with classes. If we compute the frequency $\mathit{freq}_c(I)$ of an itemset I for each class c separately, we can associate to each itemset the class label for which its frequency is highest. The resulting rule $I \rightarrow c(I)$, where $c(I) = argmax_{c'} \mathit{freq}_{c'}(I)$, is called a *class association rule*.

A decision tree aims at classifying examples by sorting them down a tree. The leaves of the tree provide the classifications of examples [13]. Each node of the tree specifies a test of one attribute of the example, and each branch of the node corresponds to one of the possible values of the attribute. We assume that all tests are boolean; nominal attributes are transformed into boolean attributes by mapping each possible value to a separate attribute. The input of a decision tree learner is then a binary matrix B, where B_{ij} contains the value of attribute i of example j.

Our results are based on the following observation.

Observation 1 *Let us transform a binary table B into transactional form D such that $T_j = \{i | B_{ij} = 1\} \cup \{\neg i | B_{ij} = 0\}$. Then the examples that are sorted down every node of a decision tree for B are characterized by an itemset of items occurring in D.*

For example, consider the decision tree in Figure 2. Then we can determine of which leaf an example is part by checking if it includes the itemset $\{B\}$, $\{\neg B, C\}$ or $\{\neg B, \neg C\}$. We denote the set of these itemsets with $\mathit{leaves}(T)$.

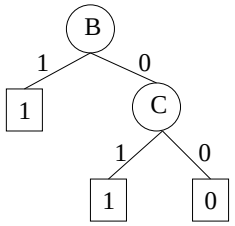


Figure 2. An example tree

The leaves of a decision tree correspond to class association rules, as leaves have associated classes. In decision tree learning it is common to specify a minimum number of examples that should be covered by each leaf. For association rules, this would correspond to giving a support threshold.

The accuracy of a decision tree is derived from the number of misclassified examples in the leaves: $\mathit{accuracy}(T) = \frac{|D| - e(T)}{|D|}$, where

$$e(T) = \sum_{I \in \mathit{leaves}(T)} e(I) \text{ and } e(I) = \mathit{freq}(I) - \mathit{freq}_{c(I)}(I).$$

A further illustration of the relation between itemsets and decision trees is given in Figure 1. In this figure, every node represents an itemset; an edge denotes a subset relation. Highlighted is one possible decision tree, which is nothing else than a set of itemsets. The branches of the decision tree correspond to subset relations.

From the theory of frequent itemset mining, it is known that itemsets form a *lattice* (these are typically

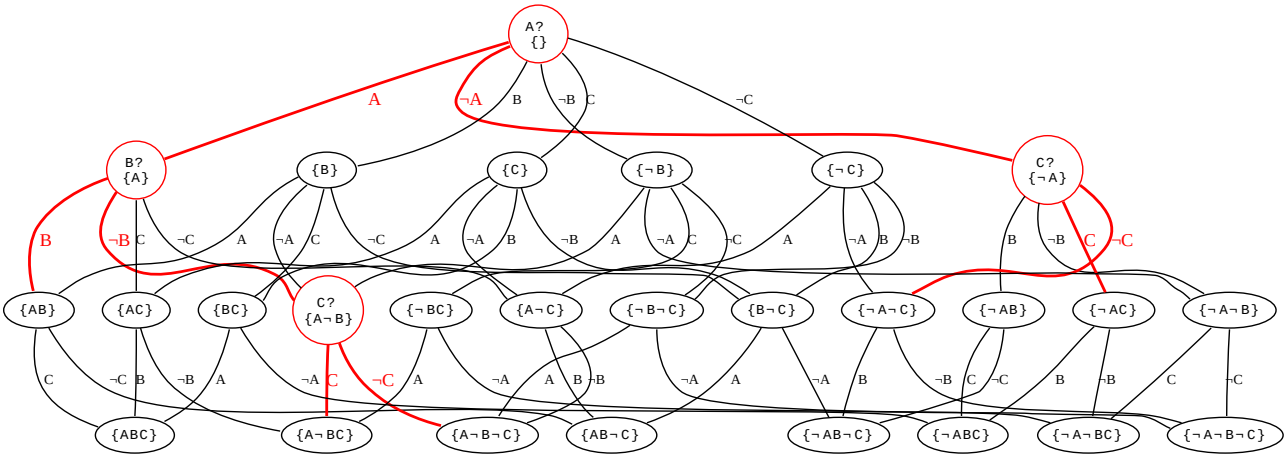


Figure 1. An itemset lattice for items $\{A, \neg A, B, \neg B, C, \neg C\}$; binary decision tree A(B(C(l,l),l),C(l,l)) is hidden in this lattice

depicted as in Figure 1). In this paper we present DL8, an algorithm for mining **D**ecision trees from **L**attices.

3. Queries for Decision Trees

The optimization problem that we study can be seen as a *query* to a database. This query consists of two parts. The first part specifies the constraints on the leaves of the decision trees.

1. $\mathcal{T} := \{T | T \in \mathit{DecisionTrees}, \forall I \in \mathit{leaves}(T), p(I)\}$

The set $\mathcal{T}$ is called the set of *locally constrained decision trees* and *DecisionTrees* is the set of all possible decision trees. Predicate $p(I)$ expresses a constraint on paths. In the simplest setting, $p(I) := (\mathit{freq}(I) \geq \mathit{minfreq})$. In general, p can be a formula constructed from the following atoms.

- $\mathit{freq}_i(I) \geq \mathit{minfreq}_i$, to express a constraint on the minimum number of examples for a class;
- $\mathit{freq}(I) \geq \mathit{minfreq}$, to express a constraint on the total number of examples in each leaf;
- $\chi^2(I) \geq \mathit{mincorr}$, to express that every leaf should have a χ^2 correlation of at least *mincorr* with the target attribute of the prediction problem;
- $\mathit{diff}(I) \geq \mathit{mindiff}$, where

$$\mathit{diff}(I) = \mathit{freq}_m(I) - \max_{c \neq m} \mathit{freq}_c(I),$$

and $m = \mathit{argmax}_c \mathit{freq}_c(I)$; this constraint expresses that in every leaf there should be more examples for the majority class than for any of the minority classes.

These atoms may be used in disjunctions and conjunctions, but may not be negated.

The first two predicates are well-known in data mining, as they are *anti-monotonic*. A predicate $p(I)$ on itemsets $I \subseteq \mathcal{I}$ is called *anti-monotonic* iff $p(I) \wedge I' \subseteq I \Rightarrow p(I')$.

The second two predicates are not anti-monotonic, but it is known that one can *bound* these atoms [17] with anti-monotonic formulas. Let us illustrate this for the atom $\mathit{diff}(I) \geq \mathit{mindiff}$. For a leaf to have $\mathit{diff}(I) \geq \mathit{mindiff}$, it should at least have one class c for which $\mathit{freq}_c(I) \geq \mathit{mindiff}$, or, in other words, a disjunction of minimum frequency constraints must be satisfied. Before testing the constraint $\mathit{diff}(I) \geq \mathit{mindiff}$, we can thus already ignore all itemsets for which the bound does not apply.

It is known that disjunctions and conjunctions of anti-monotonic predicates are also anti-monotonic [4]. We will denote the local constraint p in which χ^2 and *diff* have been replaced by their anti-monotonic bounds with p_b.

In the second step, we express a preference for a tree in the set $\mathcal{T}$.

2. output $\mathit{argmin}_{T \in \mathcal{T}}[e(T), \mathit{size}(T)]$

The tuples $[e(T), \mathit{size}(T)]$ are compared lexicographically. Instead of the standard error function e, we also allow a function to be used which computes the

expected error.

Several algorithms for estimating test set accuracy have been presented in the literature. One such estimate is at the basis of the *error based pruning* algorithm of C4.5. Essentially, C4.5 computes an additional penalty term $x(freq_1(I), \ldots freq_n(I))$ for each leaf I of the decision tree, from which we can derive a new estimated number of errors

$$ex(T) = \sum_{I \in leaves(T)} e(I) + x(freq_1(I), \ldots freq_n(I)).$$

By using $ex(T)$ instead of $e(T)$, we can find the most accurate tree after pruning such as done by C4.5. Effectively, the penalty terms make sure that trees with less leaves are sometimes preferable even if they are less accurate.

To illustrate our querying mechanism we will now give two examples.

Query 1 *Small Accurate Trees with Frequent leaves.*

$$\begin{aligned} \mathcal{T} := \{T \quad | \quad & T \in DecisionTrees, \\ & \forall I \in leaves(T), freq(I) \geq 10\} \\ output\ & argmin_{T \in \mathcal{T}}[e(T), size(T)]. \end{aligned}$$

In other words, we have $p(T) := (freq(I) \geq minfreq)$. This query investigates all decision trees in which each leaf covers at least 10 examples of the training data D. Among these trees, we find the smallest most accurate one.

Query 2 *Small Accurate Trees with Correlated leaves.*

$$\begin{aligned} \mathcal{T} := \{T \quad | \quad & T \in DecisionTrees, \\ & \forall I \in leaves(T), \chi^2(I) \geq 10\} \\ output\ & argmin_{T \in \mathcal{T}}[ex(T), size(T)]. \end{aligned}$$

In this query, we restrict ourselves to trees that perform well in terms of C4.5's pruning measure. Each leaf in the tree should have a significant correlation with the target class.

4. Motivating Examples

To motivate our work, it is useful to briefly consider two examples that illustrate what kind of trees cannot be found if the well-known information gain (ratio) heuristic of C4.5 is used to answer example Query 1.

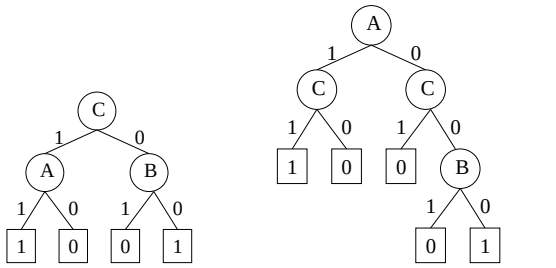
As a first example consider the database in Figure 3, in which we have 2 target classes. Assume that we are interested in answering Query 1 with $minfreq = 10$. An optimal tree exists (see Figure 2), but a heuristic learner will not find it, as it prefers attribute A in the root: A has information gain 0.33 (cq. ratio 0.54), while B only has information gain 0.26 (cq. ratio 0.37). The tree that is found by C4.5 contains a single test, as the examples that contain $\{\neg A\}$ cannot be split further without violating the constraints.

A	B	C	Class	#
1	1	0	1	40×
1	1	1	1	40×
1	0	1	1	5×
0	0	0	0	10×
0	0	1	1	5×

Figure 3.
Example database 1

A	B	C	Class	#
1	1	1	1	30×
1	1	0	0	20×
0	1	0	0	8×
0	1	1	0	12×
0	0	0	1	12×
0	0	1	0	18×

Figure 4.
Example database 2



(a) Smallest (b) Learned using heuristics

Figure 5. Two accurate trees for example database 2

As a second example consider the database in Figure 4, which is a variation of the XOR problem. Then the correct answer to Query 1 with $minfreq = 1$ is given in Figure 5(a), but the use of information gain (ratio) would yield the tree in Figure 5(b), as the information gain (cq. ratio) of A is 0.098 (cq. 0.098), while the information gain of C is 0.029 (cq. 0.030).

These examples learn us that the proportions of examples can 'fool' heuristic decision trees into a suboptimal shape. Optimal learners are less sensitive to such behavior.

We can also see in these examples that the smallest most accurate tree is not necessarily smaller or larger than the tree found by C4.5.

5. The DL8 Algorithm

We will now present the DL8 algorithm for answering the previously introduced queries. Pseudo-code of the algorithm is given in Algorithm 1. DL8 is initially called with $I = \emptyset$. Predicate p_b is the anti-monotonic bound of the local constraint p.

Given an input itemset I, DL8(I, p_b, p) computes the

Algorithm 1 DL8(I,p_b, p)

```
 1: if DL8(I,p_b,p) was computed before then
 2:     return stored result
 3: if p(I) then
 4:     C ← {l(c(I))}
 5: else
 6:     C ← ∅
 7: if ¬pure(I) then
 8:     for all i ∈ 𝓘 do
 9:         if p_b(I ∪ {i}) ∧ p_b(I ∪ {¬i}) then
10:             T_1 ← DL8(I ∪ {i}, p_b, p)
11:             T_2 ← DL8(I ∪ {¬i}, p_b, p)
12:             if T_1 and T_2 are not undef then
13:                 C ← C ∪ {n(i, T_1, T_2)}
14:         end if
15:     end for
16: if C = ∅ then
17:     T ← undef
18: else
19:     T ← argmin_{T∈C}[e_{t(I)}(T), size(T)]
20: store T as the result for I and return T
```

smallest most accurate decision tree for the transactions $t(I)$ that contain the itemset. In DL8 we use several functions: $l(c)$, which returns a tree consisting of a single leaf with class label c; $n(i, T_1, T_2)$, which returns a tree that contains test i in the root, and has T_1 and T_2 as lefthand and righthand branches; $e_t(T)$, which computes the error of tree T when only the transactions in TID-set t are considered; and finally, we use a predicate *pure*(I); predicate *pure* blocks the recursion if all examples $t(I)$ belong to the same class.

The correctness of DL8 is essentially based on the fact that the lefthand branch and the righthand branch of a node in a decision tree can be optimized independently. In more detail, the correctness follows from the following observations.

(line 4) for any set of transactions, one candidate decision tree for classifying these examples consists of a single leaf; for the leaf the original constraint $p(I)$ should be satisfied, otherwise a leaf is not a candidate for the transactions $t(I)$.

(line 7) in some cases, we can prove that continuing the recursion is not necessary. The simplest such case occurs if all examples in a set of transactions belong to the same class (after all, any larger tree will not be more accurate than a leaf).

(line 9) only tests that result in a split in which both branches fulfill the anti-monotonic constraints, need to be considered.

(line 8–15) let tree T be the smallest most accurate tree that can be constructed for the transactions $t(I)$, and assume that tree T is not a leaf. Then the lefthand and righthand branch of the root i of T must also be among the smallest most accurate trees that can be constructed for $t(I \cup \{i\})$ and $t(I \cup \{\neg i\})$, respectively. Given that for every i, we can assume that such trees are found in the recursive calls of DL8, at the end of the for-loop we must have found the most accurate tree for $t(I)$.

A key feature of DL8 is that in line 20 it stores every result that it computes. Consequently, DL8 avoids that an optimal decision tree for any itemset is computed more than once; furthermore, we do not have to store each resulting decision tree entirely; it is sufficient to store its root and statistics (error and size); lefthand and righthand subtrees can be recovered from the stored results for the lefthand and righthand itemsets if necessary.

As with most algorithms, the most time consuming operations are those that access the data. Several strategies are possible. In this paper, we will discuss two.

Direct Data Access The most straightforward approach is to compute the itemset frequencies while DL8 is executing. In this case, once DL8 is called for an itemset I, we obtain the frequencies of I in a scan over the data, and store the result to avoid later recomputations.

Frequent Itemset Mining An alternative approach is based on the observation that every itemset that occurs in a decision tree that answers a query, must satisfy the local anti-monotonic constraint p_b. In the data mining literature several algorithms for finding itemsets under anti-monotonic constraints have been proposed, among others: Apriori [1], Eclat [23] and LCM [21]. We can use these algorithms in a preprocessing step to obtain all statistics of the itemsets that fulfil the local constraints.

If we assume that the output of the frequent itemset miner consists of a directed graph structure such as Figure 1, then DL8 operates in time linear in the number of edges of this graph.

Several optimizations of both approaches are possible, but are beyond the scope of this paper.

6. Experiments

In this section we compare the decision trees learned by DL8 with the trees learned by J48. J48 is the Java

Datasets	#Ex	#Test	Datasets	#Ex	#Test
anneal	812	36	tumor	336	18
a-credit	653	56	segment	2310	55
balance	625	13	soybean	630	45
breast	683	28	splice	3190	3466
chess	3196	41	thyroid	3247	36
diabetes	768	25	vehicle	846	55
g-credit	1000	77	vote	435	49
heart	296	35	vowel	990	48
ionosphere	351	99	yeast	1484	23
mushroom	8124	116	pendigits	7494	49

Figure 6. Datasets description

implementation of *C4.5* [19] in Weka [22]. Note that further experiments including runtime report can be found in [16].

Our aim is to show how optimal decision trees compare to decision trees that are learned heuristically. It is not our aim to convince the reader that our algorithm produces classifiers with higher test-set accuracies than any other algorithm. It is well-known that some algorithms can produce classifiers with higher accuracies than decision trees. After all, decision trees can only represent a particular type of decision boundaries, no matter how they are learned.

The experiments were performed on a large number UCI datasets [15]. Figure 6 gives a brief description of the datasets in terms of the number of examples and the number of tests after binarization. Numerical data were discretized before applying both J48 and DL8. We used Weka's unsupervised discretization method with a number of bins equal to 4. We limited the number of bins in order to reduce the number of created attributes.

We used a stratified 10-fold cross-validation to compute the training and test accuracies of both systems. The bottleneck of our algorithm is the in-memory construction of the itemsets, and, consequently, the application of our algorithm is limited by the amount of memory available.

Our main results are given in Figure 7. In these experiments, we used minimum freqency as the local constraint. We lowered the minimum frequency to the lowest value that still allowed the computation to be performed within the memory of our computers. For J48, results are provided for pruned trees and unpruned trees; for DL8 results are provided in which the e (unpruned) and ex (pruned) error functions are optimized. Both algorithms were applied with the same minimum frequency setting. We used a corrected two-tailed t-test [3] with a significance threshold of 5% to compare the test set accuracies of both systems. The test set accuracy results are in bold when the result is significantly better than its counterpart result on the other system.In the last three columns, we also give results for J48 with its default $minfreq = 2$ setting. The test accuracies of J48 are compared to the test accuracies given by DL8 using pruning. The results of the significance test are given in the "S" column : "+" means that J48 is significantly better, "-" that it is significantly worse and "0" not significantly different.

The experiments show that both with and without pruning the optimal trees computed by DL8 have a better training accuracy than the trees computed by J48 with the same frequency values. Furthermore, on the test data, in both cases DL8 is significantly better than J48 on 9 of the 20 datasets and only significantly worse on one dataset. When pruned trees are compared to unpruned ones, the sizes of the trees are on average 1.75 times smaller for J48 and 1.5 time smaller for DL8. After pruning, DL8's trees are still 1.5 times larger than J48's ones. A closer inspection of these trees reveils a similar phenomenon as in the data of Figure 3: C4.5's trees are smaller as it creates trees with small numbers of incorrectly classified examples in the leaves, which cannot be split off without violating the constraints. In cases where DL8 accuracy is significantly better, the pruned trees of DL8 are only 3 to 9 nodes larger those of J48. If we compare the best results of DL8 with those given by J48 with minimum frequency 2, we see that J48's testing accuracy is significantly better on 8 of the 20 datasets used. However, for all datasets the sizes of the trees are a lot smaller for DL8.

These experiments show that DL8 is not only a useful tool to estimate the best accuracy that can be obtained on some datasets, it can also compute trees that have a better size-accuracy trade-off than the trees computed by renown systems such as J48.

7. Related work

The search for optimal decision trees dates back to the 70s, when several algorithms for building such trees using dynamic programming were proposed [7, 12, 18, 20, 10]. All this work concentrated on finding small summarizations of input data, and did not study the prediction of new examples. Optimization criteria were based on the cost of attributes, and the size or the depth of the tree.

Research in this direction was almost abandoned in the last two decades. One of the reasons was that for prediction purposes, heuristic tree learners were believed to be nearly optimal.

More recently, pruning algorithms for decision trees have been studied [8]. The DL8 algorithm can be con-

			Unpruned						Pruned						Pruned Freq = 2		
	Freq		Train acc		Test acc		Size		Train acc		Test acc		Size		Testa	S	size
Datasets	#	%	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48		J48
anneal	10	1.2	0.85	0.87	0.82	0.81	43.2	56.6	0.83	0.85	0.81	0.82	22.2	31.4	”	0	”
anneal	2	0.2	0.89	0.89	0.82	0.82	106.6	87.8	0.86	0.87	0.82	0.82	44.4	45.6	”	0	”
a-credit	45	6.8	0.87	0.88	0.86	0.87	6.2	11	0.86	0.88	0.86	0.88	3.6	11	0.84	-	36.4
balance	15	2.4	0.81	0.85	0.76	**0.79**	23.8	40.8	0.81	0.85	0.79	0.80	17.2	31	”	0	”
balance	2	0.3	0.90	0.90	0.82	0.81	99.0	114.4	0.89	0.89	0.80	0.80	72.4	65.4	”	0	”
breast-w	40	5.8	0.93	0.97	0.93	**0.95**	3.4	9.6	0.93	0.97	0.93	**0.95**	3.4	7.6	0.96	0	15.6
breast-w	30	4.3	0.96	0.96	0.95	0.95	6.8	7.0	0.96	0.97	0.95	0.95	6.8	9.4	”	0	”
chess	200	6.2	0.91	0.91	0.91	0.90	9.0	13	0.91	0.95	0.90	**0.95**	8.6	13.0	0.99	+	54.4
diabetes	15	1.9	0.79	0.83	0.75	0.72	26.4	55.4	0.79	0.82	0.74	0.74	20.4	32.4	”	0	”
diabetes	2	0.2	0.90	0.99	0.68	0.66	200.2	288.4	0.84	0.92	0.74	0.71	69	135.2	”	0	”
g-credit	100	10	0.73	0.75	0.70	0.70	6.4	11.6	0.73	0.75	0.70	0.71	6.2	9.6	”	0	”
heart-c	30	10.1	0.77	0.84	0.74	0.77	4.4	11.8	0.77	0.84	0.73	**0.78**	3.6	11.8	0.78	0	31.6
heart-c	5	1.6	0.88	0.94	0.77	0.75	30.8	70	0.87	0.94	0.76	0.80	16.8	35.4	”	0	”
heart-c	2	0.6	0.94	1	0.76	0.74	67.6	74.4	0.90	0.97	0.78	0.77	31.6	50.2	”	0	”
ionosph	50	14.2	0.83	0.86	0.79	**0.84**	4.0	7.4	0.83	0.86	0.79	**0.84**	4	6.8	0.86	0	34.6
ionosph	40	11.3	0.89	0.89	0.88	0.88	5	6.8	0.89	0.89	0.88	0.88	5	5.6	”	0	”
mushro	800	9.8	0.92	0.97	0.92	**0.97**	5.0	11.0	0.92	0.97	0.92	**0.97**	5.0	11.0	1.0	+	16.8
pendigits	600	8.0	0.58	0.72	0.58	**0.72**	13.6	17	0.58	0.72	0.58	0.72	13.4	15.3	0.95	+	340
p-tumor	15	4.4	0.44	0.49	0.38	0.37	22.6	26.8	0.44	0.49	0.39	0.37	19.2	22.2	”	0	”
p-tumor	2	0.5	0.63	0.71	0.40	0.36	116.4	152.2	0.60	0.67	0.40	0.40	81.2	105.2	”	0	”
segment	200	8.6	0.72	0.83	0.73	**0.83**	12.6	15.0	0.73	0.83	0.73	**0.83**	12.6	15.0	”	0	”
soybean	60	9.5	0.51	0.55	0.50	**0.55**	13.0	15.0	0.51	0.55	0.50	**0.55**	11.2	15.0	”	+	”
soybean	50	7.9	0.55	0.59	0.52	**0.59**	14.6	16.8	0.55	0.59	0.51	**0.58**	14.2	16.8	”	+	”
splice	700	21.9	0.74	0.74	0.74	0.73	5.0	5.0	0.74	0.74	0.74	0.73	5.0	5.0	0.94	+	126.8
thyroid	80	2.4	0.91	0.92	0.91	0.91	1.0	13.4	0.91	0.91	0.91	0.91	1.0	3.0	0.91	+	34.2
vehicle	50	5.9	0.63	0.71	0.59	**0.67**	17	22.4	0.63	0.71	0.59	**0.67**	14.8	22.4	”	0	”
vote	20	4.5	0.96	0.97	**0.96**	0.94	3.0	15.0	0.96	0.96	0.96	0.94	3.0	7.6	0.96	0	12.6
vote	15	3.4	0.96	0.97	0.95	0.94	3.4	18.0	0.96	0.97	0.96	0.95	3.0	9.2	”	0	”
vowel	100	10.1	0.36	0.39	0.34	0.33	11.0	14.4	0.36	0.39	0.34	0.33	11.0	14.4	0.78	+	290
vowel	70	7.0	0.39	0.46	0.35	0.40	17.0	20.6	0.39	0.45	0.35	0.40	16.8	20.2	”	+	”
yeast	100	6.7	0.53	0.55	0.50	**0.53**	13.8	15.4	0.53	0.55	0.51	0.53	11.4	13.8	0.53	0	186.0
yeast	2	0.1	0.74	0.82	0.49	0.48	501.2	724.2	0.68	0.75	**0.53**	0.50	186.0	307.2	”	+	”

Figure 7. Comparison of J48 and DL8, with and without pruning

ceived as an extension of these algorithms, that applies the pruning strategy on a lattice instead of on a tree.

A popular topic in data mining is currently the selection of interesting itemsets from a large set of itemsets found by frequent itemset mining algorithms [2, 5]. DL8 can be seen as one such algorithm for selecting itemsets. Furthermore, several algorithms have been presented for classification using itemsets, of which CBA is the most well-known example [11]. DL8 is however the first algorithm that outputs a well-known type of model, and provides optimality guarantees for this model.

8. Conclusions

In this paper we presented DL8, an algorithm for finding optimal decision trees. We were able to apply this algorithm on many UCI datasets. The experiments showed that the accuracies of optimal trees compete with C4.5 accuracies. For equal frequency constraints, however, optimal decision trees perform better more often. This is particularly the case if high minimum frequency constraints are used.

There are many opportunities for improving DL8. The relation between DL8 and frequent itemset mining algorithms deserves further study, as this may enable the discovery of optimal decision trees for lower minimum support values, with lower runtimes and less memory. We concentrated on the simple minimum frequency constraint in this paper, but more sophisticated constraints, including constraints on the size of decision trees, are also worth consideration. As the runtime of DL8 on a set of itemsets is low once this set is constructed, DL8 may be used for interactive data mining as long as the local constraint is not generalized. Such interactive investigations allow users to pick their preferred trade-off between size and accuracy. This topic is important, as typically users will not immediately have a good idea what desirable constraints are.

As DL8 makes intensive reuse of patterns, and allows many types of queries to be answered, we believe that it is a very important addition to the framework of *inductive databases*. Inductive databases [9] have been proposed as a view on data mining in which every data mining operation is a query on a database. The interaction between DL8 and other operations in such databases —frequency queries in particular— is an interesting topic for further studies.

Acknowledgments

Siegfried Nijssen was supported by the EU FET IST project IQ ("Inductive Querying"), contract number FP6-516169. Élisa Fromont was supported through the GOA project 2003/8, "Inductive Knowledge bases", and the FWO project "Foundations for inductive databases". The authors thank Luc De Raedt and Hendrik Blockeel for many interesting discussions. We also wish to thank Daan Fierens for preprocessing the data.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. 1996.

[2] R. Bathoorn, A. Koopman, and A. Siebes. Reducing the frequent pattern set. In *ICDM Workshops*, pages 55–59, 2006.

[3] R. R. Bouckaert and E. Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2004.

[4] L. De Raedt. Towards query evaluation in inductive databases using version spaces. In *Database Support for Data Mining Applications*, pages 117–134, 2004.

[5] L. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *SIAM International Conference on Data Mining*, 2007.

[6] E. Fromont, H. Blockeel, and J. Struyf. Integrating decision tree learning into inductive databases. In *Revised selected papers of the workshop KDID'06*, LNCS, to appear. Springer, 2007.

[7] M. R. Garey. Optimal binary identification procedures. *SIAM Journal of Applied Mathematics*, 23(2):173–186, September 1972.

[8] M. N. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. Building decision trees with constraints. *Data Min. Knowl. Discov.*, 7(2):187–214, 2003.

[9] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The Acm*, 39:58–64, 1996.

[10] A. Lew. Optimal conversion of extended-entry decision tables with general cost criteria. *Commun. ACM*, 21(4):269–279, 1978.

[11] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD)*, New York, USA, 1998.

[12] W. S. Meisel and D. Michalopoulos. A partitioning algorithm with application in pattern classification and the optimization of decision tree. *IEEE Trans. Comput.*, C-22:93–103, 1973.

[13] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[14] B. M. Moret, M. G.Thomason, and R. C. Gonzalez. Optimization criteria for decision trees. Technical Report CS81-6, Dep. of Computer Science, Univ. of New Mexico, Albuquerque, 1981.

[15] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.

[16] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. Technical Report CW476, KU Leuven, 2007.

[17] S. Nijssen and J. N. Kok. Multi-class correlated pattern mining. In *KDID*, volume 3933 of *Lecture Notes in Computer Science*, pages 165–187. Springer, 2006.

[18] H. J. Payne and W. S. Meisel. An algorithm for constructing optimal binary decision trees. *IEEE Trans. Computers*, 26(9):905–916, 1977.

[19] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[20] H. Schumacher and K. C. Sevcik. The synthetic approach to decision table conversion. *Commun. ACM*, 19(6):343–351, 1976.

[21] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the Workshop on Frequent Itemset Mining Implementations*, volume 126 of *CEUR Workshop Proceedings*,2004.

[22] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition, 2005.

[23] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. Technical Report TR651, 1997.