Beam Search Induction and Similarity Constraints for Predictive Clustering Trees

Dragi Kocev[1], Jan Struyf[2], and Sašo Džeroski[1]

[1] Dept. of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Dragi.Kocev@ijs.si, Saso.Dzeroski@ijs.si
[2] Dept. of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
Jan.Struyf@cs.kuleuven.be

Abstract. Much research on inductive databases (IDBs) focuses on local models, such as item sets and association rules. In this work, we investigate how IDBs can support global models, such as decision trees. Our focus is on predictive clustering trees (PCTs). PCTs generalize decision trees and can be used for prediction and clustering, two of the most common data mining tasks. Regular PCT induction builds PCTs top-down, using a greedy algorithm, similar to that of C4.5. We propose a new induction algorithm for PCTs based on beam search. This has three advantages over the regular method: (a) it returns a set of PCTs satisfying the user constraints instead of just one PCT; (b) it better allows for pushing of user constraints into the induction algorithm; and (c) it is less susceptible to myopia. In addition, we propose similarity constraints for PCTs, which improve the diversity of the resulting PCT set.

1 Introduction

Inductive databases (IDBs) [9, 5] represent a database view on data mining and knowledge discovery. IDBs contain not only data, but also models. In an IDB, ordinary queries can be used to access and manipulate data, while inductive queries can be used to generate, manipulate, and apply models. For example, "find a set of accurate decision trees that have at most ten nodes" is an inductive query.

IDBs are closely related to constraint-based mining [3]. Because the inductive queries can include particular constraints, the IDB needs constraint-based mining algorithms that can be called to construct the models that satisfy these constraints. The above example query includes, for instance, the constraint that the trees can contain at most ten nodes.

Much research on IDBs focuses on local models, i.e., models that apply to only a subset of the examples, such as item sets and association rules. We investigate how IDBs can support global models. In particular, we consider predictive clustering trees (PCTs) [1]. PCTs generalize decision trees and can be used for both prediction and clustering tasks. We define PCTs in Section 2.

Regular PCT induction builds PCTs top-down using a greedy algorithm similar to that of C4.5 [12] or Cart [4]. This approach has three main disadvantages w.r.t. inductive databases: (a) it returns only one PCT. This is incompatible with the IDB view that inductive queries should (if possible) return the set of all models satisfying the constraints in the query; (b) many useful constraints are not easy to push into the induction algorithm. Size constraints, such as the one in our example query, must be handled mostly during post-pruning [7]; and (c) because the algorithm is greedy, it is susceptible to myopia: It may not find any tree satisfying the constraints, even though several exist in the hypothesis space.

In this paper, we propose a new induction algorithm for PCTs that addresses these three problems to a certain extent, while maintaining an acceptable computational cost. The algorithm employs beam search. Beam search considers at each step of the search the k best models according to a particular evaluation score. Therefore, it returns a set of models instead of just one model. Beam search also supports pushing of size constraints into the induction algorithm, as we will show in Section 4. Finally, beam search is known to be less susceptible to myopia than regular greedy search.

Preliminary experiments have revealed a disadvantage of using beam search for constructing PCTs. Namely, the beam tends to fill up with small variations of the same PCT, such as trees that differ only in one node. To alleviate this, we propose similarity constraints for PCTs. We show that these constraints improve beam diversity.

The remainder of this paper is organized as follows. In Section 2 we present PCTs. The beam search algorithm is explained in Section 3. In Sections 4 and 5 we discuss anti-monotonic and similarity constraints that can be pushed in the beam search induction process. Section 6 presents the experimental setup, and Section 7 discusses the experimental results. We conclude and discuss further work in Section 8.

2 Predictive Clustering Trees

PCTs [1] generalize classification and regression trees and can be used for a variety of learning tasks including different types of prediction and clustering. The PCT framework views a decision tree as a hierarchy of clusters (Fig. 1): the top-node of a PCT corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The leaves represent the clusters at the lowest level of the hierarchy and each leaf is labeled with its cluster's centroid. PCTs are constructed such that each split maximally improves average cluster homogeneity.

PCTs can be built with a greedy recursive top-down induction algorithm (PCT-TDI, Table 1), similar to that of C4.5 [12] or Cart [4]. The algorithm takes as input a set of training instances I. The main loop searches for the best acceptable attribute-value test that can be put in a node (BestTest, Table 1). If such a test t^* can be found then the algorithm creates a new internal node labeled

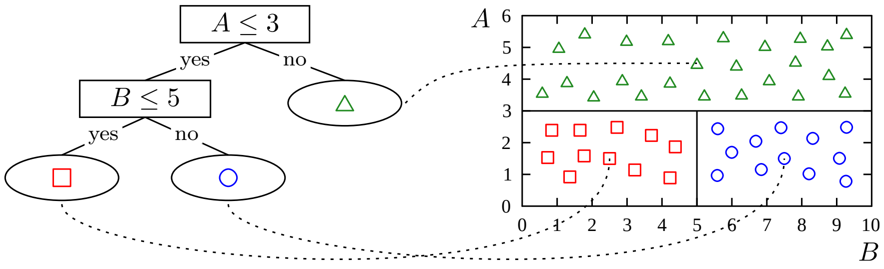


Fig. 1. A classification tree (left) is a special case of a PCT. It hierarchically partitions the instances into homogeneous clusters (right).

Table 1. The top-down induction (TDI) algorithm for PCTs.

```
procedure PCT-TDI($I$)
1: $(t^*, h^*, \mathcal{P}^*) = \text{BestTest}(I)$
2: if $t^* \neq none$ then
3:   for each $I_k \in \mathcal{P}^*$ do
4:     $tree_k = \text{PCT}(I_k)$
5:   return node($t^*$, $\bigcup_k \{tree_k\}$)
6: else
7:   return leaf(centroid($I$))
```

```
procedure BestTest($I$)
1: $(t^*, h^*, \mathcal{P}^*) = (none, 0, \emptyset)$
2: for each possible test $t$ do
3:   $\mathcal{P} = \text{Partition}(t, I)$
4:   $h = Var(I) - \sum_{I_k \in \mathcal{P}} \frac{|I_k|}{|I|} Var(I_k)$
5:   if $(h > h^*) \wedge \text{Acceptable}(t, \mathcal{P})$ then
6:     $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: return $(t^*, h^*, \mathcal{P}^*)$
```

t^* and calls itself recursively to construct a subtree for each subset in the partition $\mathcal{P}^*$ induced by t^* on the training instances. If no acceptable test can be found, then the algorithm creates a leaf. (A test is unacceptable if $\min_{I \in \mathcal{P}} |I| < m$, with m a parameter that lower-bounds the number of instances in a leaf.) The heuristic that ranks the tests is computed as the reduction in variance caused by partitioning the instances (Line 4 of BestTest).

The difference with standard decision tree learners is that PCTs treat the variance function and the centroid function that computes a label for each leaf as parameters that can be instantiated for a given learning task. For a clustering task, the variance function takes all the attributes into account, while for a prediction task it focuses on the target attribute that is to be predicted. The same holds for the centroid function.

PCTs include classification and regression trees [4] as a special case. To construct a regression tree, for example, the regular definition of variance is used and the centroid is computed as the mean of the target values in the node. To construct a classification tree, the variance is computed either as the entropy of the class attribute (then the heuristic is equivalent to information gain [12]), or by converting the target attribute to a set of binary attributes (one for each class) and using the regular definition of variance over the resulting 0/1 vectors (then the heuristic reduces to the Gini index [4]). The centroid function labels a leaf with the majority class of the examples. These definitions can be trivially

Table 2. The beam search algorithm Clus-BS.

```
procedure Clus-BS($I$,$k$)
1: $i = 0$
2: $T_{\text{leaf}}$ =leaf(centroid($I$))
3: $h$ = Heuristic($T_{\text{leaf}}, I$)
4: $\text{beam}_0 = \{(h, T_{\text{leaf}})\}$
5: repeat
6:   $i = i + 1$
7:   $\text{beam}_i = \text{beam}_{i-1}$
8:   for each $T \in \text{beam}_{i-1}$ do
9:     $R$ = Refine($T, I$)
10:    for each $T_{\text{cand}} \in R$ do
11:      $h$ = Heuristic($T_{\text{cand}}, I$)
12:      $h_{\text{worst}} = \max_{\text{T} \in \text{beam}_\text{i}}$Heuristic(T, I)
13:      $T_{\text{worst}} = \text{argmax}_{\text{T} \in \text{beam}_\text{i}}$Heuristic(T, I)
14:      if $h < h_{\text{worst}}$ or $|\text{beam}_i| < k$ then
15:        $\text{beam}_i = \text{beam}_i \cup \{(h, T_{\text{cand}})\}$
16:      if $|\text{beam}_i| > k$ then
17:        $\text{beam}_i = \text{beam}_i \setminus \{(h_{\text{worst}}, T_{\text{worst}})\}$
18: until $\text{beam}_i = \text{beam}_{i-1}$
19: return $\text{beam}_i$
```

```
procedure Refine($T, I$)
1: $R = \emptyset$
2: for each leaf $l \in T$ do
3:   $I_l$ = Instances($I$,$l$)
4:   for each attribute $a$ do
5:     $t$ = best test on $a$
6:     $\{I_1, I_2\}$ = Partition($t$, $I_l$)
7:     $l_1$ = leaf(centroid($I_1$))
8:     $l_2$ = leaf(centroid($I_2$))
9:     $n$ = node($t$,$\{l_1, l_2\}$)
10:    $T_r$ = replace $l$ by $n$ in $T$
11:    $R = R \cup \{T_r\}$
12: return $R$
```

extended to the multi-objective case (more than one target attribute) and to less trivial learning tasks, such as multi-label and hierarchical classification [2], or clustering of time series [6].

PCTs are implemented in the Clus system. Clus implements syntactic constraints and constraints on the size and/or accuracy of the trees [13]. It also implements various pruning methods, which are commonly used by decision tree learners to avoid over-fitting. More information about PCTs and Clus can be found at `http://www.cs.kuleuven.be/~dtai/clus` and in reference [1].

3 Beam Search

We propose the beam search algorithm Clus-BS, shown in Table 2. The beam is a set of PCTs ordered by their heuristic value. The algorithm starts with a beam that contains precisely one PCT: a leaf covering all the training data I.

Each iteration of the main loop creates a new beam by refining the PCTs in the current beam. That is, the algorithm iterates over the trees in the current beam and computes for each PCT its set of refinements (Fig. 2). A refinement is a copy of the given PCT in which one particular leaf is replaced by a depth one sub-tree (i.e., an internal node with a particular attribute-value test and two leaves). Note that a PCT can have many refinements: a PCT with L leaves yields $L \cdot M$ refined trees, with M the number of possible tests that can be put in a new node. In Clus-BS, M is equal to the number of attributes. That is, Clus-BS considers for each attribute only the test with the best heuristic value.

Note that the number of possible tests on a numeric attribute A is typically huge: one test $A < a_i$, for each possible split point a_i. CLUS-BS only constructs one refined tree for the split that yields the best heuristic value. This approach limits the number of refinements of a given PCT and increases the diversity of the trees in the beam.

CLUS-BS computes for each generated refinement its heuristic value. The heuristic function differs from the heuristic used in the top-down induction algorithm (TDI) from Section 2. The heuristic in the latter is local, i.e., it only depends on the instances local to the node that is being constructed. In CLUS-BS, the heuristic is global and measures the quality of the entire tree. The reason is that beam search needs to compare different trees, whereas TDI only needs to rank different tests for the same tree node. The heuristic that we propose to use is:

$$h(T, I) = \left(\sum_{\text{leaf} \,\in\, T} \frac{|I_{\text{leaf}}|}{|I|} \, Var(I_{\text{leaf}}) \right) + \alpha \cdot \text{size}(T) \, , \qquad (1)$$

with I all training data and I_{leaf} the examples sorted into leaf. It has two components: the first one is the average variance of the leaves of the PCT weighted by size, and the second one is a size penalty. The latter biases the search to smaller trees and can be seen as a soft version of a size constraint. The size function that we use throughout the paper counts the total number of nodes in the PCT (internal nodes + leaves).

After the heuristic value of a tree is computed, CLUS-BS compares it to the value of the worst tree in the beam. If the new tree is better, or if there are fewer than k trees (k is the beam width), then CLUS-BS adds the new PCT to the beam, and if this exceeds the beam width, then it removes the worst tree from the beam. The algorithm ends when the beam no longer changes. This either occurs if none of the refinements of a tree in the beam is better than the current worst tree, or if none of the trees in the beam yields any valid refinements. This is the point in the algorithm where the user constraints from the inductive query can be used to prune the search: a refinement is valid in CLUS-BS if it does not violate any of these constraints. Section 4 discusses this in detail.

Note that (1) is similar to the heuristic used in the TDI algorithm from Section 2. Assume that there are no constraints, $\alpha = 0$ and $k = 1$. In this case, the tree computed by CLUS-BS will be identical to the tree constructed with TDI. The only difference with TDI is the order in which the leaves are refined: TDI refines depth-first, whereas CLUS-BS with a beam width of one refines best-first.

The computational cost of CLUS-BS is as follows. Computing the best test for one attribute for the instances in a given leaf costs $O(|I_{\text{leaf}}| \log |I_{\text{leaf}}|)$ (to find the best split point for a numeric attribute, the instances must be sorted; after sorting, finding the best split can be done in $O(|I_{\text{leaf}}|)$ time [12]). If the score of the best test is better than that of the worst tree in the beam, then the refined tree must be constructed $O(\text{size}(T))$ and inserted into the beam $O(\log k)$ (if the beam is implemented as a balanced binary search tree). Repeating this for all

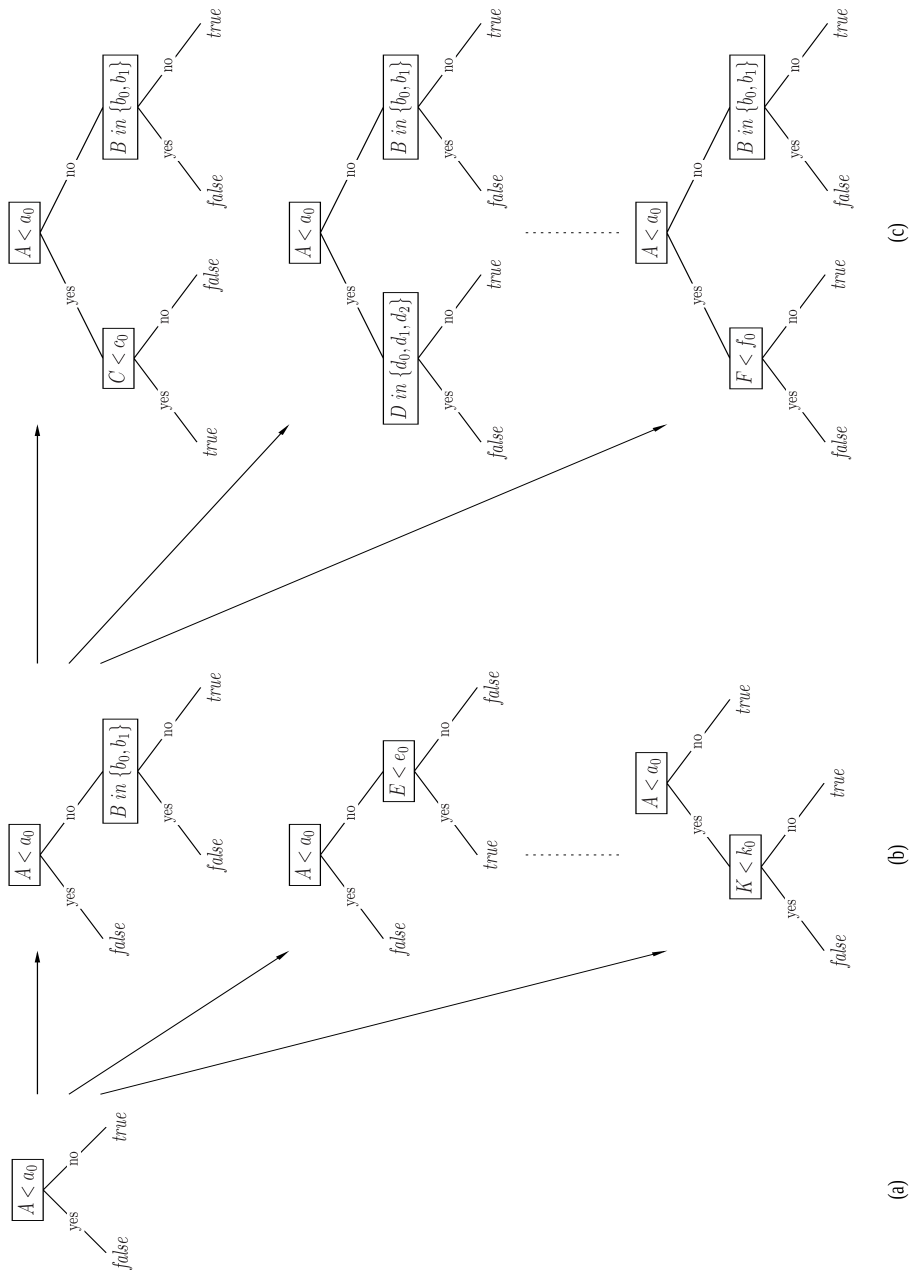


Fig. 2. Refining the trees in the beam. (a) A tree in the beam; (b) the refinements of tree (a); (c) the refinements of the top-most tree in (b). Note that the refinements (c) are only computed in a subsequent iteration of the search after the top-most tree of (b) has entered the beam.

attributes and all leaves yields $O(|A| \cdot |I| \log |I| + |A| \cdot |\text{leaves}(T)| \cdot (\text{size}(T) + \log k))$ because each instance occurs in at most one leaf. If s upper-bounds the size of the trees in the beam, then the cost of refining the entire beam is $O(k \cdot |A| \cdot |I| \log |I| + s^2 k \cdot |A| + s \cdot |A| \cdot k \log k)$. Finally, the cost of running n iterations of CLUS-BS is $O(nk \cdot |A| \cdot |I| \log |I| + ns^2 k \cdot |A| + ns \cdot |A| \cdot k \log k)$. For comparison, the computational cost of TDI is $O(D \cdot |A| \cdot |I| \log |I|)$, with D the depth of the tree. Assuming that the first term dominates the complexity of CLUS-BS, it follows that CLUS-BS is $O(kn/D)$ times slower. Note that n is in the best case equal to the number of leaves in the largest tree because each iteration can add at most one leaf.

4 Anti-Monotonic Constraints

CLUS-BS supports any conjunction or disjunction of constraints that are anti-monotonic with regard to the refinement order. We define this more precisely with the following definitions.

Definition 1 (Refinement set). *The refinement set $\rho^*(T)$ of a tree T given a refinement operator ρ is the set that is obtained by recursively applying ρ, that is,* $\lim_{n \to \infty} \rho^n(T)$, *with $\rho^0(T) = \{T\}$ and $\rho^i(T) = \rho^{i-1}(T) \cup \left(\bigcup_{T_r \in \rho^{i-1}(T)} \rho(T_r) \right)$ if $i > 0$.*

In CLUS-BS, $\rho(T)$ is implemented by the Refine procedure in Table 2. Consider again Fig. 2. All trees shown in this figure are part of $\rho^*(T)$, with T the tree in Fig. 2.a.

Definition 2 (Refinement order). *The refinement order $\geq_{\text{ref}}$ is a partial order defined on trees as $T_1 \geq_{\text{ref}} T_2$ if and only if $T_1 \in \rho^*(T_2)$.*

Note that $T_1 \geq_{\text{ref}} T_2$ can be thought of as "T_2 is a subtree of T_1 sharing the same root".

Definition 3 (Anti-monotonic constraint). *A constraint is a Boolean function over trees. A constraint c is anti-monotonic with respect to $\geq_{\text{ref}}$ if and only if $\forall T_1, T_2 : (T_1 \geq_{\text{ref}} T_2 \land c(T_1)) \rightarrow c(T_2)$.*

If one considers an increasing sequence of trees according to the refinement order (E.g., going from (a) to (c) in Fig. 2) then the value of an anti-monotonic constraint can only decrease along the sequence, that is, change from true to false. This observation is exploited by CLUS-BS. If a given refinement violates one of the anti-monotonic constraints, then the search can be pruned (by not adding the refinement to the beam) because any subsequent refinement will also violate the constraint (because of its anti-monotonicity).

We list a number of useful anti-monotonic constraints for trees.

1. The maximum size constraint $c_s(T) = (\text{size}(T) \leq s)$ upper-bounds the size of the tree. This constraint is useful for decision trees because domain experts are typically interested in small trees for interpretability reasons.

2. The minimum cluster size constraint $c_m(T) = (\min_{\text{leaf} \in T} |I_{\text{leaf}}| \geq m)$ lower-bounds the number of instances in each leaf of the tree. This constraint is implemented by most decision tree learners.
3. The maximum depth constraint upper-bounds the maximum depth of the tree. Sometimes it is useful to constrain tree depth, for example, because the resulting tree will be more balanced.
4. The maximum prediction cost constraint. This is an extension of the maximum depth constraint, where each test is weighted by its prediction cost. Prediction cost constraints are useful in medical applications where the attributes correspond to expensive lab tests. In such applications, it is useful to upper-bound the prediction cost of the tree.

All the above constraints are anti-monotonic and can be handled efficiently by Clus-BS. In the experimental evaluation, we investigate the effect of using maximum size constraints (Section 6).

So far, we assumed that the user is interested in obtaining trees that are as accurate as possible. For decision trees, this is accomplished algorithmically by using a heuristic function that is known to yield accurate trees. Another possibility is that the user explicitly provides a constraint on the accuracy of the tree, e.g., $\text{acc}(T, I) \geq 85\%$. (Note that this constraint is not anti-monotonic.) The goal is then to find, e.g., the smallest tree that satisfies this constraint. To this end, the algorithm is run repeatedly with increasing values for the size constraint until the accuracy constraint is satisfied [13].

5 Similarity Constraints

The heuristic value defined in Section 3 only takes the variance and the size of the PCT into account. In this section, we define a soft similarity constraint, which can be included in the heuristic computation and biases the search towards a diverse set of trees (which are dissimilar to each other, as much as possible). We will call Clus-BS with these (dis)similarity constraints included Clus-BS-S.

To quantify the (dis)similarity of two trees, Clus-BS-S requires a distance function between trees. Two obvious approaches are: (a) define the distance based on the syntactic representation of the trees, or (b) define the distance based on the predictions that the trees make on the training instances. One problem with (a) is that many syntactically different trees can represent the same concept, for example, the trees in Fig. 3 both represent the concept $A \wedge B$. If our goal is to find trees that are semantically different, then (b) is to be preferred and we therefore focus on this approach here.

We propose to compute the distance between two trees (T_1 and T_2) as the normalized root mean squared distance between their predictions on the data set at hand, that is

$$d(T_1, T_2, I) = \frac{1}{\eta} \cdot \sqrt{\frac{\sum_{t \in I} d_p(p(T_1, t), p(T_2, t))^2}{|I|}}, \qquad (2)$$

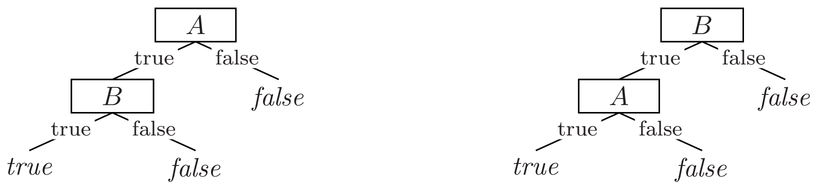


Fig. 3. Syntactically different trees representing the concept $A \wedge B$.

with η a normalization factor, $|I|$ the number of training instances, $p(T_j, t)$ the prediction of tree T_j for instance t, and d_p a distance function between predictions. In (2), η and d_p depend on the learning task. For regression tasks, d_p is the absolute difference between the predictions, and $\eta = M - m$, with $M = \max_{t \in I, j \in \{1,2\}} p(T_j, t)$ and $m = \min_{t \in I, j \in \{1,2\}} p(T_j, t)$. This choice of η ensures that $d(T_1, T_2, I)$ is in the interval $(0, 1)$. For classification tasks, $d_p = \delta$ with

$$\delta(a, b) = \begin{cases} 1 \text{ if } a \neq b \\ 0 \text{ if } a = b \end{cases}, \tag{3}$$

and η is 1. These distance functions can be easily adapted to the more general PCT types mentioned in Section 2 (e.g., for multi-target prediction, multi-label and hierarchical classification and clustering of time series).

The heuristic value of a tree can now be modified by adding a term that penalizes trees that are similar to the other trees in the beam.

$$h(T, \text{beam}, I) = \left(\sum_{\text{leaf} \in T} \frac{|I_{\text{leaf}}|}{|I|} \, Var(I_{\text{leaf}}) \right) + \alpha \cdot \text{size}(T) + \beta \cdot \text{sim}(T, \text{beam}, I) \tag{4}$$

Because the heuristic value of a tree now also depends on the other trees in the beam, it changes when a new tree is added. Therefore, each time that Clus-BS-S considers a new candidate tree, it recomputes the heuristic value of all trees already in the beam using (4), which incorporates the similarity to the new candidate tree (T_{cand}) by using (5).

$$\text{sim}(T, \text{beam}, I) = 1 - \frac{d(T, T_{cand}, I) + \sum_{T_i \in \text{beam}} d(T, T_i, I)}{|\text{beam}|} \tag{5}$$

Note that (5) is in the interval $(0, 1)$ because the numerator has $|\text{beam}|$ non-zero terms that are in $(0, 1)$. Clus-BS-S computes the heuristic value of T_{cand} using (4). If the heuristic value of the candidate tree is better than that of the worst tree in the beam, the candidate tree enters the beam and the worst tree is removed.

Table 3. The data sets and their properties: number of instances ($|I|$), number of discrete/continuous input attributes (D/C), number of classes (*Cls*), probability of majority class (*Maj*), entropy of the class distribution (*Ent*), mean value of the target (*Mean*), and standard deviation of the target (*St.dev.*).

(a) Classification data sets.

Data set	$\lvert I\rvert$	D/C	*Cls*	*Maj*	*Ent*
car	1728	6/0	4	0.70	1.21
mushroom	8124	22/0	2	0.52	1.00
segment	2310	0/19	7	0.14	2.81
vowel	990	3/10	11	0.09	3.46
vehicle	846	0/19	4	0.26	2.00
iris	150	0/4	3	0.33	1.58
ionosphere	351	0/34	2	0.64	0.94
chess	3196	36/0	2	0.52	1.00

(b) Regression data sets.

Data set	$\lvert I\rvert$	D/C	*Mean*	*St.dev.*
autoPrice	159	0/15	11445.73	5877.86
bodyfat	252	0/14	19.15	8.37
cpu	209	1/6	99.33	154.76
housing	506	1/12	22.53	9.20
pollution	60	0/15	940.36	62.21
servo	167	4/0	1.39	1.56
pyrim	74	0/27	0.66	0.13
machine_cpu	209	0/6	105.62	160.83

6 Experiments

6.1 Aims

We compare Clus with the regular recursive top-down induction algorithm (TDI, Table 1) to Clus with beam search (BS, Table 2), and beam search with similarity constraints (BS-S). Our aim is to test the following hypotheses.

1. Hill-climbing search, which is used by TDI, suffers from shortsightedness. TDI may return a suboptimal model due to its limited exploration of the hypothesis space. Beam search is known to be less susceptible to this problem. We therefore expect that on average BS will yield models that are more accurate or at least as accurate as the models built by TDI.
2. Similarity constraints improve the diversity of the beam, possibly at the cost of some predictive accuracy. Diversity is important for the domain expert, who is typically interested in looking at different models, for example because one PCT is easier to interpret than the other PCTs. Note that if we consider all models in the beam to make up the answer to the inductive query, then all these PCTs should be reasonably accurate.

6.2 Setup

We perform experiments on 8 regression and 8 classification data sets from the UCI machine learning repository [10]. Table 3 lists the properties of the data sets. We set the parameters of the beam search algorithms ad-hoc to the following values: $k = 10$, $\alpha = 10^{-5}$, and $\beta = 1$, where k is the beam width, α is the size penalty and β is the influence of the similarity constraint (Equations (1) and (4)). For the classification data sets, we use the version of the heuristic that employs class entropy to estimate the variance of the target attribute. All experiments are performed with the Clus system (Section 2), in which the algorithms BS and BS-S have been implemented.

We measure the predictive performance of each algorithm using 10 fold cross-validation. For the classification data sets, we report accuracy and for the regression data sets the Pearson correlation coefficient. Because the beam search algorithms yield not one but k trees, we have to select one of these k trees to compare to TDI. We decided to use the tree that performs best on the training data (T_{train}) for this purpose.

To test if all trees in the beam are sufficiently accurate, we measure their average predictive accuracy (or correlation coefficient). We also measure the minimum and maximum accuracy (or correlation coefficient) of the trees in the beam and use these to calculate the difference in performance between the worst tree and T_{train} and the best tree and T_{train}. That is, we compute $D_{\text{worst}} = A_t - A_w$ and $D_{\text{best}} = A_b - A_t$, with A_t the test set performance of T_{train}, and A_w the minimum and A_b the maximum test set performance of the trees in the beam. If $D_{\text{worst}} = 0$, then T_{train} is the worst tree in the beam, and if $D_{\text{best}} = 0$, then it is the best tree in the beam. We report the average of D_{worst} and D_{best} over the 10 cross-validation folds.

To quantify the effect of the similarity constraints, we calculate for the two beam search algorithms beam similarity, which we define as the average similarity of the trees in the beam. $\text{Similarity}(\text{beam}, I) = \frac{1}{|\text{beam}|} \sum_{T \,\in\, \text{beam}} \text{sim}(T, \text{beam}, I)$, with $\text{sim}(T, \text{beam}, I) = 1 - \frac{1}{|\text{beam}|} \sum_{T_i \,\in\, \text{beam}} d(T, T_i, I)$, the similarity of tree T w.r.t. the other trees in the beam, and $d(T, T_i, I)$ the distance between trees T and T_i as defined in Section 5. We report beam similarity on the test set averaged over the 10 cross-validation folds.

We perform experiments for different values of the size constraint. Recall that in the beam search algorithm, this type of constraints can be enforced during the search (Section 4). For TDI this is not possible and therefore we use a two step approach that first builds one large tree and subsequently prunes it back to satisfy the size constraint [7]. We also report results without any size constraint. For these results we use, both for TDI and BS, the same pruning algorithm that is also used in C4.5 [12] (for classification data sets) and in M5 [11] (for regression data sets).

7 Results and Discussion

7.1 Predictive Performance

Table 4 compares the cross-validated accuracy of TDI, BS, and BS-S on the classification data and Table 5 the cross-validated correlation coefficient for the regression data. The tables contain results for different values of the size constraint: maximum size ranging from 5 (SC5) to 51 (SC51) nodes, and no size constraint (NoSC). Each column includes the number of statistically significant wins ($p \leq 0.05$), which are obtained by a 10 fold cross-validated paired t-test and indicated in bold face.

The results confirm our first hypothesis. BS yields models of comparable accuracy to TDI. BS wins on 5 classification and 3 regression tasks. TDI wins on 2 classification and no regression tasks. This confirms that BS yields more accurate models, which can be explained because it is less susceptible to myopia. There is no clear correlation between the number of wins and the value of the size constraint.

BS-S wins on 6 classification and 4 regression tasks and loses on 13 classification and 1 regression tasks. BS-S performs, when compared to BS, worse on classification data than on regression data. This is because the heuristic (used in BS-S) trades off accuracy for diversity. If a given tree in the beam is accurate, then new trees will be biased to be less accurate because the similarity score favors trees with different predictions. For classification problems this effect is more pronounced because a 0/1 distance between predictions is used, whereas in the regression case a continuous distance function is used. The latter makes it "easier" to have different predictions that are still reasonably accurate. Also, this effect is stronger for bigger size constraints (the majority of the losses of BS-S are for SC31, SC51 and NoSC) because the relative contribution of the similarity score to the heuristic is greater for bigger size constraints. Note that the losses are in the range of 1-2% accuracy, so for the majority of domains this is not a serious problem.

Our second hypothesis was that BS-S trades off accuracy for beam diversity. Table 6 lists the beam similarity for BS and BS-S for the classification data and SC7. The beam similarity of BS-S is always smaller than that of BS. Fig. 4 shows the trees in the final beam for the "vehicle" data for BS and BS-S. The trees of BS all have the same test in the top node and include tests on 5 attributes. The BS-S trees have tests on 3 different attributes in the top node and include tests on 6 attributes in total. This shows that the trees produced by BS-S not only produce different predictions, but are also syntactically different from the trees constructed with BS.

Table 6 lists the average accuracy of the trees in the beam and shows how much worse (better) the worst (best) tree is compared to the result reported in Table 4. Consider first the results for BS. For the data sets "mushroom", "segment", and "vehicle", all trees are of comparable accuracy. For "car", "vowel", "iris", "ionosphere", and "chess", the differences in accuracy become larger. For most of these, T_{train} is on average among the best trees. This is most obvious for "chess" where $D_{\text{best}} = 0$. Only for 2 out of 8 data sets ("car" and "ionosphere") $D_{\text{best}} > D_{\text{worst}}$. Note that the differences are larger for BS-S than for BS. This shows that the variance in accuracy increases with the beam diversity.

7.2 Induction Time

Table 7 compares the running times of all algorithms and the number of models evaluated by BS and BS-S. Observe that BS-S is (much) slower than BS and TDI. The longer running time of BS-S is due to two reasons. First, it evaluates more PCTs because of the similarity measure that is included in the heuristic score. In BS, the score of the "worst" tree in the beam monotonically improves with the iteration number. In BS-S, this is no longer the case because the score of the trees in the beam needs to be recomputed when a new tree enters the

Table 4. Comparison of beam search (BS) and BS with similarity constraints (BS-S) to top-down induction (TDI) on classification data (accuracy).

	TDI	BS	TDI	BS	TDI	BS	TDI	BS
Data set		SC5		SC7		SC11		SC17
car	77.8	77.8	**79.2**	77.1	82.2	81.8	87.0	85.6
mushroom	99.4	99.4	99.4	**99.6**	99.9	**100.0**	100.0	100.0
segment	40.0	40.7	55.6	55.6	80.9	81.1	90.2	90.4
vowel	20.6	20.7	25.2	27.3	31.6	33.6	38.9	**42.3**
vehicle	48.7	51.2	51.2	**60.2**	64.5	64.5	68.9	66.4
iris	92.0	92.0	94.0	96.0	93.3	93.3	93.3	92.7
ionosphere	89.5	89.2	88.6	88.3	88.9	90.6	88.6	88.9
chess	75.5	76.9	90.4	90.4	94.1	93.8	96.5	96.9
Wins	0	0	1	2	0	1	0	1
		SC31		SC51		NoSC (Acc)		NoSC (Size)
car	92.8	92.6	**95.0**	94.0	97.5	97.6	113	117
mushroom	100.0	100.0	100.0	100.0	100.0	100.0	15	11
segment	94.9	94.2	96.2	96.0	96.7	96.8	85	85
vowel	49.2	51.8	55.7	**61.2**	79.2	80.8	191	179
vehicle	70.0	72.5	71.7	72.7	73.9	72.0	167	179
iris	93.3	92.7	93.3	92.7	92.7	92.7	9	11
ionosphere	88.9	88.9	88.9	88.9	88.6	89.5	29	27
chess	97.8	97.7	99.3	99.4	99.4	99.5	53	53
Wins	0	0	1	1	0	0		

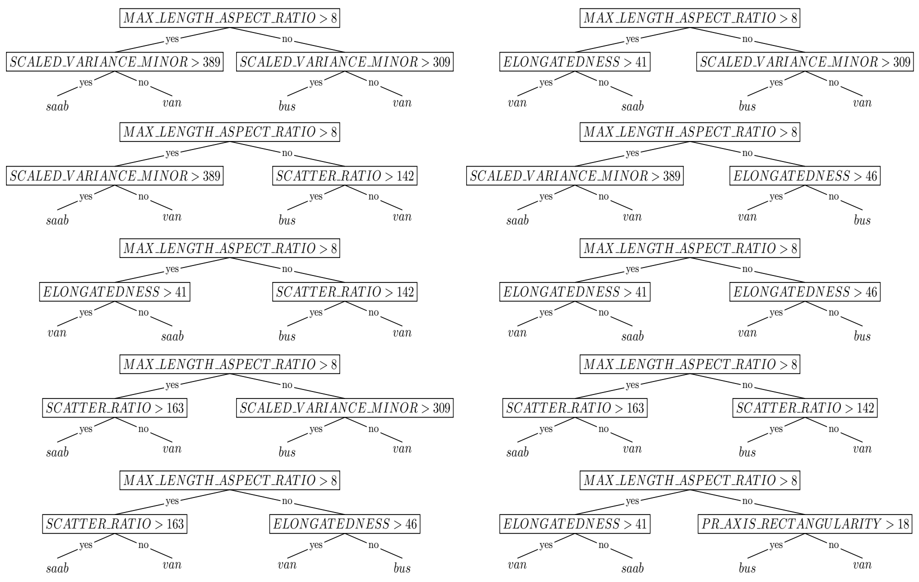
	TDI	BS-S	TDI	BS-S	TDI	BS-S	TDI	BS-S
Data set		SC5		SC7		SC11		SC17
car	77.8	77.8	**79.2**	77.1	82.2	81.1	87.0	86.0
mushroom	99.4	99.4	99.4	**99.6**	99.9	**100.0**	**100.0**	99.6
segment	40.0	39.6	55.6	55.1	80.9	81.0	90.2	**91.6**
vowel	20.6	20.7	25.2	27.3	31.6	**36.0**	38.9	**42.0**
vehicle	48.7	51.2	51.2	**60.2**	64.5	64.3	68.9	68.7
iris	92.0	92.0	94.0	96.0	93.3	93.3	93.3	92.7
ionosphere	89.5	89.2	88.6	88.6	88.9	92.0	88.6	91.5
chess	75.5	76.9	90.4	90.4	94.1	93.8	**96.5**	95.6
Wins	0	0	1	2	0	2	2	2
		SC31		SC51		NoSC (Acc)		NoSC (Size)
car	**92.8**	90.9	**95.0**	93.3	97.5	97.2	113	95
mushroom	**100.0**	99.6	**100.0**	99.6	**100.0**	99.6	15	11
segment	**94.9**	94.3	**96.2**	94.8	**96.7**	95.4	85	81
vowel	49.2	50.3	55.7	57.9	79.2	81.7	191	187
vehicle	70.0	67.4	71.7	71.4	73.9	71.6	167	189
iris	93.3	92.7	93.3	92.7	92.7	94.7	9	13
ionosphere	88.9	90.6	88.9	90.3	88.6	92.0	29	25
chess	97.8	97.6	**99.3**	98.3	**99.4**	98.3	53	43
Wins	3	0	4	0	3	0		

Table 5. Comparison of beam search (BS) and BS with similarity constraints (BS-S) to top-down induction (TDI) on regression data (correlation coefficient).

	TDI	BS	TDI	BS	TDI	BS	TDI	BS
Data set		SC5		SC7		SC11		SC17
autoPrice	0.86	0.88	0.88	0.90	0.87	0.89	0.88	0.89
bodyfat	0.87	0.87	0.94	0.94	0.95	0.95	0.97	0.96
cpu	0.92	0.92	0.92	0.92	0.93	0.94	0.95	0.95
housing	0.76	0.76	0.80	0.78	0.86	0.85	0.89	0.88
pollution	0.44	0.44	0.50	0.53	0.48	0.41	0.55	0.51
servo	0.82	0.82	0.89	0.91	0.90	**0.94**	0.91	0.93
pyrim	0.64	0.49	0.68	0.54	0.72	0.65	0.73	0.74
machine_cpu	0.80	0.79	0.84	0.83	0.87	0.86	0.88	0.87
Wins	0	0	0	0	0	1	0	0
		SC31		SC51	NoSC	(Acc)	NoSC	(Size)
autoPrice	0.88	0.90	0.88	0.91	0.88	0.91	17	17
bodyfat	0.98	0.97	0.97	0.97	0.96	0.97	65	77
cpu	0.95	0.95	0.95	0.95	0.94	0.94	23	51
housing	0.91	0.90	0.90	0.89	0.90	0.89	63	55
pollution	0.52	**0.62**	0.53	0.59	0.49	0.52	13	13
servo	0.92	0.95	0.92	**0.95**	0.91	0.91	21	17
pyrim	0.73	0.74	0.73	0.68	0.58	0.56	11	11
machine_cpu	0.89	0.89	0.90	0.89	0.89	0.87	33	27
Wins	0	1	0	1	0	0		

	TDI	BS-S	TDI	BS-S	TDI	BS-S	TDI	BS-S
Data set		SC5		SC7		SC11		SC17
autoPrice	0.86	0.88	0.88	0.90	0.87	0.86	0.88	0.91
bodyfat	0.87	0.87	0.94	0.94	0.95	0.95	0.97	0.97
cpu	0.92	0.92	0.92	0.92	0.93	0.93	0.95	0.95
housing	0.76	0.76	0.80	0.78	0.86	0.85	0.89	0.89
pollution	0.44	0.44	0.50	0.50	0.48	0.47	0.55	0.60
servo	0.82	0.82	0.89	0.91	0.90	**0.94**	0.91	0.93
pyrim	0.64	0.34	0.68	0.63	0.72	0.53	0.73	0.68
machine_cpu	0.80	0.79	0.84	0.83	0.87	0.85	0.88	0.88
Wins	0	0	0	0	0	1	0	0
		SC31		SC51	NoSC	(Acc)	NoSC	(Size)
autoPrice	0.88	**0.90**	0.88	**0.91**	0.88	0.90	17	29
bodyfat	0.98	0.97	0.97	0.97	0.96	0.98	65	71
cpu	0.95	0.95	0.95	0.95	0.94	0.95	23	41
housing	0.91	0.90	**0.90**	0.89	0.90	0.90	63	75
pollution	0.52	0.62	0.53	0.51	0.49	0.52	13	13
servo	0.92	0.95	0.92	**0.96**	0.91	0.91	21	19
pyrim	0.73	0.65	0.73	0.64	0.58	0.54	11	13
machine_cpu	0.89	0.90	0.90	0.90	0.89	0.88	33	25
Wins	0	1	1	2	0	0		

(a) Without similarity constraint (BS):


MAX_LENGTH_ASPECT_RATIO > 8
yes
no
SCALED_VARIANCE_MINOR > 389
SCALED_VARIANCE_MINOR > 309
saab
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
SCALED_VARIANCE_MINOR > 309
van
saab
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
SCALED_VARIANCE_MINOR > 389
SCATTER_RATIO > 142
saab
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
SCALED_VARIANCE_MINOR > 389
ELONGATEDNESS > 46
saab
van
van
bus
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
SCATTER_RATIO > 142
van
saab
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
ELONGATEDNESS > 46
van
saab
van
bus
MAX_LENGTH_ASPECT_RATIO > 8
SCATTER_RATIO > 163
SCALED_VARIANCE_MINOR > 309
saab
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
SCATTER_RATIO > 163
SCATTER_RATIO > 142
saab
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
SCATTER_RATIO > 163
ELONGATEDNESS > 46
saab
van
van
bus
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
PR_AXIS_RECTANGULARITY > 18
van
saab
bus
van

(b) With similarity constraint (BS-S):

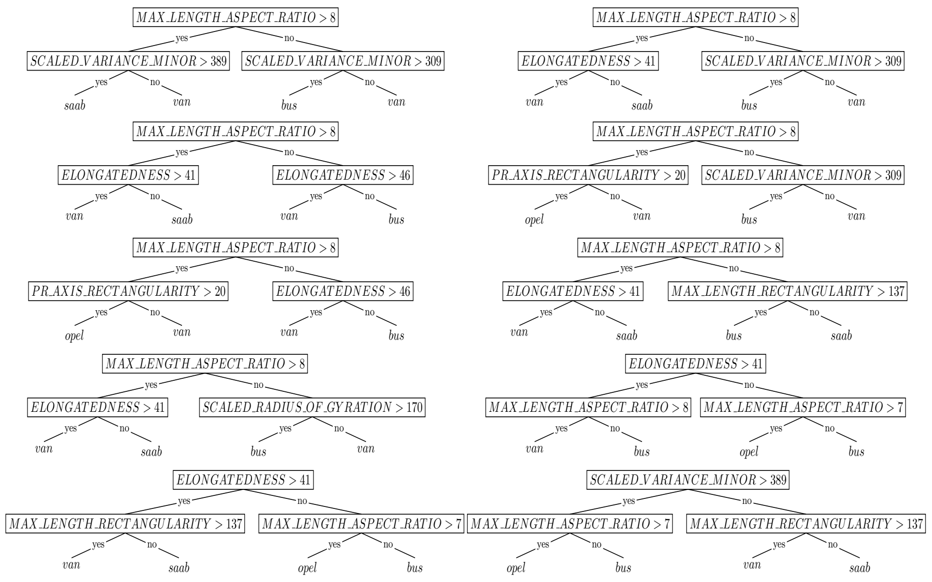

MAX_LENGTH_ASPECT_RATIO > 8
yes
no
SCALED_VARIANCE_MINOR > 389
SCALED_VARIANCE_MINOR > 309
saab
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
SCALED_VARIANCE_MINOR > 309
van
saab
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
ELONGATEDNESS > 46
van
saab
van
bus
MAX_LENGTH_ASPECT_RATIO > 8
PR_AXIS_RECTANGULARITY > 20
SCALED_VARIANCE_MINOR > 309
opel
van
bus
van
MAX_LENGTH_ASPECT_RATIO > 8
PR_AXIS_RECTANGULARITY > 20
ELONGATEDNESS > 46
opel
van
van
bus
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
MAX_LENGTH_RECTANGULARITY > 137
van
saab
bus
saab
MAX_LENGTH_ASPECT_RATIO > 8
ELONGATEDNESS > 41
SCALED_RADIUS_OF_GYRATION > 170
van
saab
bus
van
ELONGATEDNESS > 41
MAX_LENGTH_ASPECT_RATIO > 8
MAX_LENGTH_ASPECT_RATIO > 7
van
bus
opel
bus
ELONGATEDNESS > 41
MAX_LENGTH_RECTANGULARITY > 137
MAX_LENGTH_ASPECT_RATIO > 7
van
saab
opel
bus
SCALED_VARIANCE_MINOR > 389
MAX_LENGTH_ASPECT_RATIO > 7
MAX_LENGTH_RECTANGULARITY > 137
opel
bus
van
saab

Fig. 4. Trees in the final beam for the "vehicle" data (BS and BS-S), SC7.

Table 6. Average cross-validated accuracy of all trees in the beam, comparison of the worst tree in the beam to the reported result of T_{train} (D_{worst}), comparison of the best tree in the beam to T_{train} (D_{best}), and beam similarity. Results for trees constrained to have at most 7 nodes (SC7).

	Avg. test perf.		D_{worst}		D_{best}		Beam similarity	
Data set	BS	BS-S	BS	BS-S	BS	BS-S	BS	BS-S
car	78.8	77.4	1.1	8.5	4.3	4.3	0.67	0.61
mushroom	99.4	98.8	0.2	3.6	0.0	0.0	0.99	0.90
segment	55.6	55.1	1.0	4.2	0.2	2.6	0.82	0.38
vowel	25.1	25.3	7.2	7.9	2.5	3.8	0.43	0.27
vehicle	59.6	55.8	1.5	13.2	0.1	1.2	0.89	0.47
iris	93.0	93.3	5.3	10.7	1.3	2.0	0.91	0.86
ionosphere	88.6	88.1	2.0	8.6	4.0	5.4	0.86	0.72
chess	82.4	81.3	13.8	17.3	0.0	0.0	0.67	0.55

Table 7. Run times and number of evaluated models. (The experiments were run on an AMD Opteron Processor 250 2.4GHz system with 8GB of RAM running Linux.)

	Run time [s]			Evaluated models	
Classification data set	TDI	BS	BS-S	BS	BS-S
car	0.06	0.80	31.25	2638	57879
mushroom	0.08	1.38	20.17	217	4761
segment	0.28	10.17	168.05	2168	151868
vowel	0.34	9.52	249.62	6567	483294
vehicle	0.14	6.18	264.18	6723	590628
iris	0.04	0.04	0.09	198	809
ionosphere	0.10	1.01	5.35	779	29926
chess	0.08	2.35	111.55	1800	92256
Regression data set					
autoPrice	0.06	0.37	40.03	2856	69158
bodyfat	0.06	0.48	84.05	2471	94185
cpu	0.08	0.21	9.67	1333	12415
housing	0.09	3.30	1009.27	10821	549292
pollution	0.04	0.09	3.09	1288	13049
servo	0.06	0.16	6.67	2114	11239
pyrim	0.03	0.16	2.65	1195	9357
machine_cpu	0.06	0.25	21.44	2178	28131

beam (because of the similarity component). As a result, it becomes harder for BS-S to satisfy the stopping criterion (the beam no longer changes). Second, in BS-S, evaluating a single model takes a factor $O(k^2 \cdot |I|)$ longer than in BS, with k the beam width and $|I|$ the number of instances. (We exploit properties of the distance measure ($d(T_a, T_b, I) = d(T_b, T_a, I)$ and $d(T_a, T_a, I) = 0$) to make the evaluation of the similarity component efficient.)

8 Conclusion and Further Work

We propose a new algorithm for inducing predictive clustering trees (PCTs) that uses beam search. The main advantages of this algorithm are that: it induces a set of PCTs instead of just one PCT; it supports pushing of anti-monotonic user constraints, such as maximum tree size, into the induction algorithm; and it is less susceptible to myopia. In order to improve beam diversity, we introduce soft similarity constraints based on the predictions of the PCTs.

Our current set of experiments takes into account fixed values for the parameters k (the beam width), and α and β (the contribution of tree size and similarity score to the heuristic value). In future work, we plan to perform experiments for different values of β to gain more insight in the trade-off between predictive performance and beam similarity. Also, the influence of the beam width will be investigated.

We plan to investigate the use of alternative distance functions for the similarity score. Recall that we hypothesized that the reason for having less accurate trees in the classification case is that the distance function is less "continuous" than in the regression case. We plan to investigate smoother distance functions for classification. Such functions could, for example, take the predicted class distribution into account instead of just the predicted majority class.

Model diversity, which can be controlled by means of the heuristic proposed in Section 5, has been shown to increase the predictive performance of classifier ensembles [8]. Therefore, we plan to investigate if beam search with similarity constraints can be used to construct an accurate ensemble of PCTs. That is, instead of selecting from the beam the one PCT that performs best on the training data, the PCT ensemble will combine all PCTs in the beam by means of a combination function, such as majority voting. We also plan to investigate other alternatives for introducing diversity in the beam.

The experimental evaluation of this paper focuses on classification and regression trees. In future work, we plan to test beam search for more general PCT types. Note that this is, from an algorithmic point of view, trivial: the only component that changes is the definition of the variance and distance functions. In this context, we plan to investigate the use of beam search for multi-target prediction tasks [1], where non-trivial interactions between the target attributes may exist.

Acknowledgments. This work was supported by the IQ project (IST-FET FP6-516169). Jan Struyf is a postdoctoral fellow of the Fund for Scientific Research of Flanders (FWO-Vlaanderen).

References

1. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *15th Int'l Conf. on Machine Learning*, pages 55–63, 1998.
2. H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare. Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, volume 4213 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2006.
3. J-F. Boulicaut and B. Jeudy. Constraint-based data mining. In O. Maimon and L. Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 399–416. Springer, 2005.
4. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees.* Wadsworth, Belmont, 1984.
5. L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, 2002.
6. S. Džeroski, I. Slavkov, V. Gjorgjioski, and J. Struyf. Analysis of time series data with predictive clustering trees. In *5th Int'l Workshop on Knowledge Discovery in Inductive Databases*, pages 47–58, 2006.
7. M. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. Building decision trees with constraints. *Data Mining and Knowledge Discovery*, 7(2):187–214, 2003.
8. L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.
9. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
10. C.J. Merz and P.M. Murphy. UCI repository of machine learning databases [`http://www.ics.uci.edu/~mlearn/mlrepository.html`], 1996. Irvine, CA: University of California, Department of Information and Computer Science.
11. J.R. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348. World Scientific, 1992.
12. J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann series in Machine Learning. Morgan Kaufmann, 1993.
13. J. Struyf and S. Džeroski. Constraint based induction of multi-objective regression trees. In *Knowledge Discovery in Inductive Databases, 4th Int'l Workshop, KDID'05, Revised, Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*, pages 222–233, 2006.