Multi-Instance Tree Learning

Hendrik Blockeel HENDRIK.BLOCKEEL@CS.KULEUVEN.BE
Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium

David Page PAGE@BIOSTAT.WISC.EDU
Dept. of Biostatistics and Medical Informatics and Dept. of Computer Sciences,
University of Wisconsin, Madison, WI, USA

Ashwin Srinivasan ASHWIN.SRINIVASAN@IN.IBM.COM
IBM India Research Laboratory, Block 1, Indian Institute of Technology, Hauz Khas, New Delhi, 110 016, India.

Abstract

We introduce a novel algorithm for decision tree learning in the multi-instance setting as originally defined by Dietterich et al. It differs from existing multi-instance tree learners in a few crucial, well-motivated details. Experiments on synthetic and real-life datasets confirm the beneficial effect of these differences and show that the resulting system outperforms the existing multi-instance decision tree learners.

1. Introduction

In standard supervised concept learning, the training data set consists of a number of examples, each of which is classified as belonging to the concept ("positive") or not ("negative"). The so-called *multi-instance* learning setting differs from this, in that the data set consists of bags of instances, and it is indicated for each bag whether it contains at least one positive instance, or not. The classification of individual instances is unknown. Dietterich, Lathrop and Lozano-Pérez (1997) motivate the setting with several applications, including the so-called Musk problem: Molecules are classified into musk and non-musk molecules, and it is known that there exists some specific spatial structure that causes the musk property, but each molecule has multiple possible spatial configurations and it is not known which of these contains the structure that causes the musk property.

In this paper, we tackle the task of upgrading decision tree learning to the multi-instance setting. While several such attempts have been made before, they either work in a setting that is slightly different from the multi-instance setting originally proposed by Dietterich et al, or ignore certain difficulties that arise in this setting. We present an algorithm that works in the original setting and is very close to the standard tree learning algorithms. We motivate our algorithm with theoretical arguments and study its performance experimentally.

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

2. The multi-instance learning setting

As several researchers have studied relaxed versions of the multi-instance setting, it is useful to precisely state the learning problem as originally defined:

Given:

- a set of bags B_i, $i = 1, \ldots, N$, their classification $c(B_i) \in \{0, 1\}$, and the instances e_{ij} ($j = 1, \ldots, n_i$) belonging to each bag
- the existence of an unknown function f that classifies individual instances as 1 or 0, and for which it holds that $c(B_i) = 1$ if and only if $\exists e_{ij} \in B_i : f(e_{ij}) = 1$ (multi-instance constraint, MIC)

Find: the function f.

The task is to find the instance classifier f, not the bag classifier c. The latter follows, of course, from f; but if the learned hypothesis $\hat{f}$ only approximates f, it is not guaranteed that the best approximation $\hat{f}$, together with the definition $\hat{c}(B_i) = 1 \Leftrightarrow \exists e_{ij} \in B_i : \hat{f}(e_{ij}) = 1$, gives us the best approximation $\hat{c}$. For instance, if we also have information on the probability of a prediction $\hat{f}(e_{ij})$ being correct, using a *noisy or* will give us a better apprimation of $\hat{c}$ than using the MIC.

Not all research on multi-instance learning takes the viewpoint that f should be approximated. Xu (2003) presents an excellent overview of approaches to multi-instance learning and points out that many consider a relaxed version of multi-instance learning, where the MIC is not assumed to be true, and the goal is to approximate c instead of f. These methods usually do not yield any approximation of f, and hence cannot be used to reliably classify single instances or gain insight into f (for instance, in the Musk setting, to detect the substructure that gives a molecule the Musk property).

Before reviewing prior work on multi-instance tree learning, we briefly add further multi-instance terminology we need for this discussion. We call a bag positive or negative according to its given classification. As the data set does not contain the true classification of all individual instances, we call an instance "positive" (between quotes) if it is part of a positive bag. We call a "positive" example a true positive, if its true class is positive; and a false positive, if its true class is negative. All instances in negative bags are called negative.

We next turn to the specific case of decision tree learning. We assume the reader is familiar with standard greedy tree learning algorithms, which recursively partition a data set by choosing split variables and, for continuous variables, split values. In particular, we assume the reader is familiar with the notion of an impure node and standard measures of the impurity of a node, such as entropy or Gini index. We further assume the reader is familiar with the gain functions based on these measures, such as information gain (Quinlan, 1986) or Gini gain (Breiman et al., 1984), respectively, and their use to choose splits.

3. Existing approaches to decision tree learning in a multi-instance setting

Dietterich et al. showed experimentally that the "strawman" approach of ignoring the MIC when training (assigning to each instance its bag's class and applying a standard tree learner) does not work well. This is not surprising: many negative instances are assigned a positive class in this way, so it introduces a lot of (one-sided) class noise.

Blockeel and De Raedt (1998) pointed out that the multi-instance learning is a special case of inductive logic programming (ILP). ILP systems typically use a declarative language bias that allows the user to (syntactically or otherwise) constrain the kind of hypotheses it can return. Blockeel and De Raedt showed how the inductive logic programming system TILDE, a decision tree learner, can be given a bias that makes it learn multi-instance trees. But the bias they propose does not include all multi-instance concepts, nor only those. A detailed analysis requires strong familiarity with TILDE, and is beyond the scope of this paper; for illustration, Figure 1 shows a tree that can be generated with Blockeel and De Raedt's bias and does not represent a valid multi-instance concept. Using other biases, it is possible to define a strict superset of multi-instance concepts, underconstraining the learning problem, or to define the correct set by using TILDE's lookahead feature and forcing it to build linear trees, which essentially reduces it to a rule learning system. Neither is desirable from the point of view of studying multi-instance tree learning.

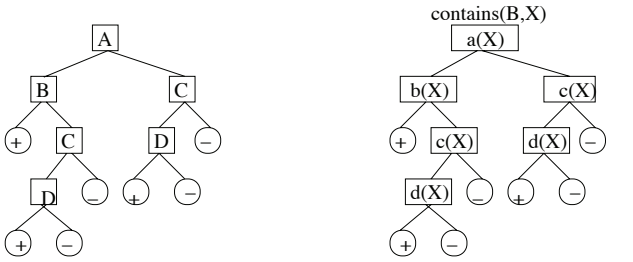

Figure 1. A propositional and first order tree. The propositional tree represents $A \wedge B \vee C \wedge D$. The first order tree does not represent $\exists x \in B : (a(x) \wedge b(x) \vee c(x) \wedge d(x))$, although that is a valid multi-instance concept. Instead, it expresses $(\exists x \in B : a(x) \wedge b(x)) \vee (\exists x : a(x) \wedge c(x) \wedge d(x)) \vee (\nexists x \in B : a(x) \wedge \exists x \in B : c(x) \wedge d(x))$, a concept outside the space of multi-instance concepts.

Ruffo's (2000) RELIC works in the relaxed setting, learning concepts such as "contains at least three instances with property P".

Chevaleyre and Zucker (2001) present a general approach to upgrading learners to the multi-instance setting, apply it to RIPPER (Cohen, 1995) with good results, and briefly discuss how decision tree learners can be upgraded in a similar way. Their main point is that changing the heuristic for choosing the best split in a node is sufficient. Instead of using a heuristic based on the class distribution of instances (which is highly noisy), they propose to use the class distribution of the bags that have at least one instance in the set under consideration. In addition, in the case of decision tree induction, they state that once an instance is predicted positive by the tree, all its co-instances (instances in the same bag) should be removed from the data set. They have implemented this method in the ID3-MI system, but do not present any empirical evaluation.

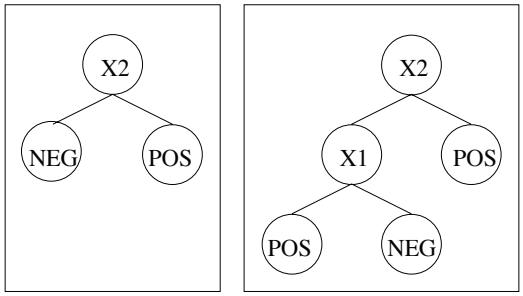

Figure 2. Depth-first versus best-first node expansion

4. MIC in Decision Tree Learning

Decision tree learning can be adapted to the multiple instance case in two different ways. The first kind of trees handles bags directly. Such a tree takes a bag as input, the tests in its nodes are defined on bags (typically of the form "does the bag contain an instance for which C holds", with C some condition), and a whole bag is thus eventually assigned to a leaf and classified as positive or negative. The second kind are just standard trees, which take a single instance as input, contain the standard attribute value tests, and eventually classify the instance as positive or negative. The multi-instance aspect is only visible in the use of the tree to make predictions: each instance of a bag gets a separate classification by the tree, and the bag is classified as positive if at least one if its instances is.

The first approach is the one that has been taken by Tilde, and lead to the problems mentioned above. The second approach avoids these problems and has the advantage that a standard decision tree is obtained. However, this approach causes new difficulties for the learning algorithm.

A problem with permitting bags to be divided at training time is that such division removes the ability to score a split *locally*. In ordinary decision tree learning, it can be shown that splitting an impure node on a variable leads to gain (a decrease in the entropy, Gini index, or other common measure of impurity) at that node if and only if it leads to gain for the entire tree. In fact, a stronger result can be shown: for any given node, a larger *local* gain (gain at that node) yields a larger *global gain* (gain for the tree). Hence choosing the best split at a node requires computing only the local gain for each split, as done by existing greedy tree learners, rather than computing the global gain for each split. This property of locality is lost when we move to the multiple instance setting.

As an illustration, consider the decision tree in Figure 2a. In ordinary tree learning, if the positive node in Figure 2a is pure, and the negative node is maximally impure (consists of equal numbers of positive and negative instances), then splitting the negative node on another variable, say x_1, where the children of x_1 are nearly pure, will have high local gain and high global gain. The tree resulting from this split is shown in Figure 2b. In the multiple instance setting, when we say the positive node in Figure 2a is pure, we mean it contains instances from positive bags only. When we say the negative node is maximally impure, we mean it has equal numbers of instances from positive and negative bags. Then splitting the negative node on x_1, where the children of x_1 are nearly pure, again has high local gain but it may have zero or negative global gain. In other words, the resulting tree in Figure 2b may have higher entropy than the original tree in Figure 2a. To see this, suppose that in Figure 2a the positive instances in the negative node all belong to bags that also have instances in the positive node. Then the split on x_1 has not decreased the entropy or Gini index of the tree; indeed, if the positive child of x_1 contains a negative instance, then the split has increased entropy and Gini, and in fact decreased accuracy. More generally, for any node, the split with the best local gain is not the split with the best global gain.

We have seen that the loss of locality means we may need to score splits differently. One option for a multiple instance tree learner might be to simply replace the local scoring function with a global one, such as the difference in the entropies of the old tree and the new tree. But the loss of locality has another implication. Locality implies that splits at different nodes are independent. Hence the order in which nodes are expanded in ordinary decision tree learning is unimportant; we may consider them in any order. In the multiple instance setting, the value of a split at one node can be changed by first performing a split at another node, if the two nodes have instances from common bags. The order in which nodes are expanded is critical in this setting. Therefore, simply replacing the local scoring function of a greedy tree learner with a global one is likely to be insufficient. We would like to somehow generate as soon as possible the positive leaf nodes about which we can be most confident, to set the stage for scoring other splits throughout the tree. More generally, we argue in the following that dealing with the loss of locality motivates several design principles for a multiple instance tree learner:

1. the tree learner should focus on finding pure *positive* subsets, not on negative ones

2. it should try to find pure positive subsets that are *as large as possible*
3. it should try to find the *largest pure positive subsets first*
4. once it has discovered a positive leaf, it should disregard instances in other parts of the tree that belong to a bag that contains an instance covered by the positive leaf
5. it makes sense to give higher weights to instances from smaller bags

For ease of argumentation, we assume the data to be noise-free.

4.1. Focus on the positives.

A first important observation is that when building the tree, it makes sense to try to form pure positive leaves, but it does not make sense to try to form pure negative leaves. Indeed, the MIC implies that *a truly positive instance can only occur in positive bags.* Because a leaf expresses a sufficient condition for an instance to be positive, the bag-classes of all the instances in that leaf must be positive.

Negative leaves are assumed to cover only negative instances, which may have both positive and negative bag-classes. Hence, a set with both "positive" and negative instances may make a perfectly valid negative leaf (all the "positives" are then false positives). Trying to obtain pure negative leaves implies that the tree tries to separate false positives from negatives, which we know cannot be done using the information in the instance description. (The reason for an instance being a false positive lies in some other instance in the same bag.)

The commonly used heuristics for choosing the best test in a node (information gain (ratio), Gini index) are symmetric with respect to the classes, and therefore try equally hard to obtain pure negative subsets as pure positive ones. In the multi-instance setting, we want an asymmetric heuristic that just aims at creating pure positive subsets.

4.2. Find large pure-positive leaves

While covering only positive instances is a necessary condition for a positive leaf to be correct (i.e., cover only truly positives), it is not a sufficient one: some false positives may be covered as well. But we have no way of telling this. Any reason to believe that a node is a correct positive leaf, necessarily stems from the fact that it contains only "positive" training instances. However, the more "positives" are covered by the pure positive leaf, the more reason we have to believe that it is correct. Indeed, if a node is not a correct positive leaf, then the more instances it covers, the higher the probability that a negative instance slips in, rebutting the hypothesis that it is a correct positive leaf.

We conclude that our heuristic for choosing a test in a node should reward the creation of subsets that have a high chance of leading not just to a pure-positive subset (as stated previously), but to a *large* pure-positive subset.

4.3. Identifying a sufficient condition allows us to remove some class noise

Now assume that we have identified a positive leaf. This leaf not only explains the class of the instances that it covers, according to the MIC it also explains away the bag-class of all the other instances in the bags to which the covered instances belong.

Therefore, a multi-instance tree learner, once it has identified a positive leaf, should from that point onwards disregard other instances in the bags that contain an instance in the positive leaf. In our algorithmic description, we will say that these other instances are *deactivated.* Note that this deactivation effectively removes false positives from the dataset, which will make it more feasible to identify correct sufficient conditions for the remainder of the data.

By introducing this deactivation procedure, the order in which nodes are split becomes important. When the possible splits of a node are evaluated, the results may be different depending on whether this evaluation is before or after deactivation of certain instances, in other words, whether it happens before or after creation of a certain leaf.

Deactivation can have a detrimental effect if an incorrect pure positive leaf has been found. It is therefore important that the leaves that are most likely to identify a correct sufficient condition (i.e., the largest pure-positive leaves) are found first. They are most likely to lead to deactivation of many false positives and increase the chances of correctly constructing the remainder of the tree.

We already know that the heuristic should reward splits that bring the tree close to the identification of a large pure-positive subset. We can now add that when several nodes still need to be split, we should first split the node that is most likely to lead to the largest pure-positive leaf. In other words, our heuristic should not only be used to choose the best split of a given node, but also to choose which node to split next.

```
Q := {root node};
while Q is not empty
    remove first node N from Q
    if N is a pure positive node
    then
        make N a positive leaf
        deactivate all instances of all bags that are
            represented in N
    else if N contains no positives
    then make N a negative leaf
    else
        find the best test t for N
        split N according to t
        add the children of N to Q
    sort Q
```

Figure 3. A multi-instance tree learning algorithm

Clearly, the commonly used depth-first order in which nodes are split is no longer sufficient. In the multi-instance case, nodes should be split in a best-first order.

4.4. Some positives are more likely to be noisy than others

We know that a positive bag of n instances contains at least one true positive. Hence, the probability of a single, randomly chosen, "positive" instance being truly positive is at least $1/n$, with n the size of its bag. It makes sense, then, to give instances from small bags a higher weight than instances from large bags.

4.5. An Algorithm

Figure 3 presents an algorithm, devised according to the above reasoning. The algorithm is rather rudimentary: it has no non-trivial stopping criterion and no pruning mechanism. For our current purposes, however, it turns out to be sufficient: the conclusions we will draw further in the paper would only be reinforced by a more sophisticated implementation.

The algorithm can be seen as a variant of ID3, with as most important change that it expands the nodes of the tree in a best-first, instead of depth-first, order. It keeps an ordered queue Q of nodes that may still need to be split, and proceeds by repeatedly doing the following: if the first node in the tree contains only "positives", make it a positive leaf, and deactivate all instances that belong to a bag of which an instance is covered by this leaf; otherwise, find the best split for this node, perform the split, creating children for this node; add those children to Q, keeping Q sorted. Note that after creation of a leaf node, the evaluation of any nodes may have changed due to deactivation of instances; hence Q has to be reordered.

It should be kept in mind that at each point, only instances are considered that have not been deactivated yet. E.g., the test "N contains no positives" used to decide whether we create a negative leaf, returns true also if N originally contained "positive" instances but all of them have been deactivated by now.

The behaviour of the algorithm is influenced by several parameters, which we need to discuss in some detail.

Sorting-heuristic determines the order in which nodes will be expanded. Nodes more likely to lead to large pure-positive leaves should be expanded first; the question is then how we trade off the absolute and relative number of positive instances in a node to be expanded. We consider three heuristics: *unbiased proportion, laplace estimate, tozero estimate*. Let p be the number of "positives" and t the total number of instances covered by the node. Unbiased proportion is simply p/t, which represents no trade-off: a small set with proportion 1 is always preferred over a larger set with proportion 0.9. The Laplace estimate is often used to counter this effect: it is defined as $(p+1)/(t+2)$. It is an interpolation between p/t and an a priori estimate of 0.5, getting closer to p/t as t grows. However, given the asymmetry in our problem, we may prefer an interpolation between p/t and 0. The *tozero* estimate does exactly that: it is defined as $p/(t+k)$, with k a parameter that influences how strongly the estimate is pulled towards 0. Like the Laplace estimate, it is a special case of the m-estimate (Cestnik, 1990). We call all these estimates *biased estimates for the proportion of positives* (*bepp*'s).

The queue can in principle also be ordered in such a way that the standard depth-first order is simulated: it is sufficient to always add children to the front of the queue. This is the effect of setting the **node-expansion** parameter to *depthfirst*, instead of its default *bestfirst*. The sorting heuristics only have effect in combination with *bestfirst*. The *depthfirst* order will serve as a reference point.

Splitting-heuristic determines the heuristic that is used to determine the best test for splitting a node. As mentioned, this heuristic should be asymmetric.

We consider five heuristics: *max-bepp, ss-bepp, gini, acc-bags, bag-entropy*. *Max-bepp* defines the quality of the split as the maximal *bepp* among its children; *ss-bepp* computes the sum of squared *bepps* and as such tries to obtain many children with high *bepp*; *gini* is the standard Gini index as used by CART (Breiman

et al., 1984); *acc-bags* defines the quality as the predictive accuracy of the tree created by adding this split, when predicting bag classes according to the MIC; *bag-entropy* is the class entropy of the bags having representatives in the data set, counting each bag only once (the heuristic proposed by Chevaleyre and Zucker).

Note that *ss-bepp* can be seen as a one-sided Gini index; it rewards the creation of close-to-pure-positive subsets in exactly the same way as Gini rewards the creation of close-to-pure subsets.

Pos-threshold is the percentage of positives that a leaf should contain before it makes a positive prediction. This is only relevant if it turns out to be impossible to obtain pure leaves (i.e., if the node being split is non-pure but no acceptable best test t can be found). In single-instance tree learning, the expected predictive accuracy is maximized if this threshold is 0.5 (i.e., the majority class is predicted). In the multi-instance case, one can argue that any leaf that is not 100% positive cannot possibly be truly positive (in a noise-free setting), but can easily be negative (if all "positive" instances are false positives). It would therefore make sense to increase this threshold. It is not clear, however, how to compute the threshold's optimal value.

Instance-weights determines whether instances are weighted according to their bag size. As explained, instances from small bags can be assumed to carry more information than instances from large bags, hence we may want to give them a higher weight. We consider two options: not using any weights, and *inverse bag size* (IBS) weighting, where instances get a weight $1/n$ with n the size of the bag they belong to.

The algorithmic description in Figure 3 together with the settings for these parameters uniquely determines the algorithm's behaviour, which should render our experiments reproducible.

4.6. Algorithm complexity

Decision tree learners generally run in time $O(ANd)$ with A the number of attributes, N the number of instances, and d the depth of the tree (Quinlan, 1986).

The multi-instance tree learner has slightly higher computational complexity. The new heuristics are not more complex to compute than the usual ones. Deactivation of instances can be done in time linear in the number of instances covered by the leaf, so this affects the complexity only with a constant factor. After each deactivation, nodes in the queue must be re-evaluated before sorting, which involves recounting the (positive) instances they cover (now disregarding newly deactivated ones). This is at most linear in N. Sorting the queue takes time $q \log q$ with q the length of the queue.

As re-evaluation and re-sorting need to be done for each positive leaf, a term $O(L(N + q \log q))$, with L the number of positive leaves, is introduced in the complexity formula. This gives a total complexity of $O(ANd + L(N + q \log q))$. The second term may dominate the first one when the number of nodes grows linearly in N, behaviour not uncommon for decision trees when no simple target concept exists. In that case, both L and q may become linear in N, and we get a complexity of $O(N^2 \log N)$. But when the tree size remains constrained, behaviour linear in the number of examples can be expected, so it performs as efficiently as standard tree learners, up to a constant factor.

5. Experiments

We have implemented the algorithm as described before (with parameters as mentioned) in a system called MITI (multi-instance tree inducer). We show in this section that MITI outperforms TILDE on multiple instance datasets.

We have conducted experiments on synthetic datasets as well as the Musk and Mutagenesis (Srinivasan et al., 1996) benchmarks. The synthetic datasets allow us to control certain properties of the datasets, to gain insight into their effect on the learner's behaviour. The experiments with real-life datasets allow us to compare the performance of the system with that of other systems.

Our synthetic dataset generator uses the following parameters: A, the number of (non-class) attributes; V, a set of values each attribute can take; B, the bag size; N, the number of bags; T, the target hypothesis. The A non-class attribute values in a single instance are generated independently and randomly from a uniform distribution. B such instances are independently generated to form a single bag. The class of the bag is then determined as positive if at least one of its instances is positive according to T. N such bags are generated independently. The experimental procedure is always as follows: a training set is generated with parameters A, V, B, T, N; a tree is learned from this training set; the tree is evaluated on a test set of 1000 cases that is independently generated from the training set but with the same parameters.

5.1. Effect of Design Choices

We experimentally compare the behaviour of the system when using dept-first node expansion, as ID3 does, and best-first expansion, expanding the node

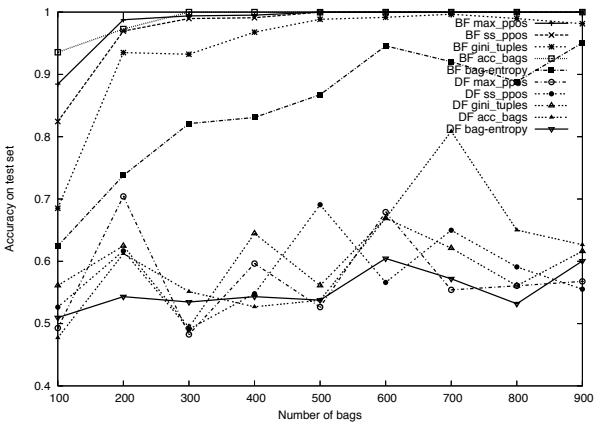

Figure 4. Depth-first versus best-first node expansion

that seems closest to a large pure positive leaf first.

Figure 4 shows learning curves for one particular type of dataset, with $A = 10$, $B = 10$, $|V| = 3$, $T =$ "$A_1 = a \wedge (A_2 = a \wedge A_3 = a \vee A_4 = a \wedge A_5 = a)$". Test accuracy is shown as a function of N, which varies from 100 to 900. The depth-first and best-first strategies are compared for five splitting heuristics (max-bepp, ss-bepp, gini, acc-bags, bag-entropy). The pos-threshold parameter is 0.5. Test accuracies reported are always the average of five runs with different datasets generated randomly with the parameters just mentioned. The *bepp* used for these results is `tozero(5)`, but different *bepp*'s yield essentially the same picture.

The curves clearly show that the best-first strategy is crucial to the success of the method: its accuracy converges to 1 quite fast, whereas the depth-first strategy consistently performs much worse and does not improve much with a growing number of bags. The difference between the different splitting heuristics is negligible compared to the effect of the node expansion strategy, with the possible exception of the bag-entropy heuristic, which performs worse.

It is also interesting to study the behaviour of MITI on the Musk benchmark datasets. First, the number of instances per bag varies greatly, which makes it an interesting set to study the effect of IBS weighting. Second, our trees can represent disjunctive concepts, whereas the target concept is expected to be conjunctive. As the Musk data sets contain few examples (92, 102), and, especially in the case of Musk 2, many instances per example, there is a concern that the weaker bias of our method yields worse results. However, there is a simple way to add a bias towards conjunctive concepts to the tree learner. If the target concept is conjunctive, all truly positive instances should be in the same leaf. If there are P positive examples, there must be at least P truly positive instances, hence any subset with fewer than P "positive" instances is probably not a good one, even if it is 100% pure. The heuristic that trades off the relative and absolute number of "positives" should give more weight to the absolute number. This can be achieved by using the `tozero(`p`)` heuristic with a large p.

p	5	20	100	1000	APR
Musk 1	35	30	22	16	7-18
Musk 1 IBS	20	20	16	11	7-18
Musk 2	37	36	33	24	11-34
Musk 2 IBS	27	25	19	9	11-34

Table 1. Results of MITI, with and without IBS weighting, and for increasing p, on the Musk datasets. The numbers indicate the number of incorrect predictions in a tenfold cross-validation. APR gives the range of errors obtained by Dietterich et al.'s APR approaches.

Table 1 illustrates the effects of (a) instance weighting and (b) the p parameter in the `tozero(`p`)` heuristic. The table clearly shows that increasing p gives a sufficiently strong bias to the decision tree learner so that it performs comparably to Dietterich et al.'s APR approaches, which are also biased towards a conjunctive concept. Manual inspection of the learned trees confirmed that the hypothesis found tends to have several small disjuncts for small p, and represents a single conjunctive concept for sufficiently large p. The table further confirms that at least in these benchmark data sets, with bags of highly variable size, IBS instance weighting has a positive effect on performance.

5.2. Comparison to Other Tree Learners

We compare MITI to Tilde and to ID3-MI', a version of MITI tuned to resemble Chevaleyre and Zucker's ID3-MI (by making it use depthfirst node expansion and ID3-MI's bag-entropy heuristic).

Comparison on synthetic datasets. For six different concepts, Tilde, MITI and ID3-MI' were run on the same sequence of training sets of increasing size (100 to 3000; for the last concept 100 to 20000 because of a strongly skewed class distribution). Table 2 reports the lowest error obtained on a test set and the training set size for which it was obtained. The table shows that MITI typically finds much better theories, using smaller training sets. While the table only shows test set errors, also training set errors for Tilde did not reach 0 except in the first two cases.

Comparison on real-life datasets. Table 3 shows the accuracies obtained with a tenfold crossvalidation

Concept	Tilde	MITI	ID3-MI'
$a \vee b$	0 / 100	0 / 2k	0 / 500
abc	0 / 200	0 / 100	0 / 200
$ab \vee cd$	128 / 2k	0 / 300	173 / 2k
$abc \vee ade$	22 / 3k	0 / 200	198 / 200
$ab \vee cd \vee ef$	152 / 2k	3 / 3k	153 / 2k
$a \vee bc \vee bde \vee fg$	15 / 10k	3 / 10k	29 / 100

Table 2. Lowest test error / number of training examples for which it was obtained, for Tilde, MITI and ID3-MI' learning a variety of target concepts.

Data set	Tilde	MITI	ID3-MI'
Musk 1	0.815	0.837	0.793
Musk 2	0.775	0.882	0.735
Muta 188	0.771	0.787	0.750
Muta 42	0.81	0.81	0.83

Table 3. Test set accuracy obtained on benchmark datasets by Tilde, MITI and ID3-MI'.

(using the same folds for all systems) on the Mutagenesis and Musk benchmark datasets, by Tilde, MITI and ID3-MI', all run with default parameter settings (except $p = 1000$ for MITI on Musk). There are theoretical arguments for assuming Musk is MIC-compliant, though Xu (2003) presents counterarguments. To our knowledge, there are no theoretical arguments for Mutagenesis being MIC-compliant (though it is a popular benchmark problem for multi-instance learning).

6. Conclusions

We have introduced a novel decision tree learning algorithm for the multi-instance setting as originally proposed by Dietterich, Lathrop and Lozano-Pérez (1997). The algorithm is similar to standard tree learning algorithm, with as major changes the adoption of a best-first node expansion strategy during tree learning, and heuristics that are better tuned to the multi-instance setting by focusing on the creation of pure positive leaves and by giving higher weights to examples from smaller bags. An experimental study shows that especially the best-first node expansion strategy is crucial in obtaining good performance. In an experimental comparison with the existing decision tree learners proposed for the original multi-instance setting, our approach outperforms them on synthetic datasets (constructed to fulfill the multi-instance assumptions) and performs equally good or better on real-life datasets.

The experimental study shows the value of our approach. However, further optimization is possible: issues such as the stopping criterion, the pruning strategy, and the threshold proportion of positives required for a leaf to be positive, remain to be investigated.

7. Acknowledgements

H.B. is a post-doctoral fellow of the Fund for Scientific Research of Flanders, Belgium (FWO-Vlaanderen).

References

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont: Wadsworth.

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. *Proceedings of the 9th European Conference on Artificial Intelligence* (pp. 147–149). London: Pitman.

Chevaleyre, Y., & Zucker, J.-D. (2001). Solving multiple instance and multiple part learning problems with decision trees and rule sets. application to the mutagenesis problem. *14th Canadian Conference on Artificial Intelligence* (pp. 204–214).

Cohen, W. (1995). Fast effective rule induction. *Proceedings of the twelfth International Conference on Machine Learning* (pp. 115–123). Morgan Kaufmann.

Dietterich, T. G., Lathrop, R. H., & Lozano-Pérez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, *89*, 31–71.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Ruffo, G. (2000). *Learning single and multiple instance decision trees for computer security applications*. Doctoral dissertation, Department of Computer Science, University of Torino.

Srinivasan, A., Muggleton, S., Sternberg, M., & King, R. (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, *85*, 277–299.

Xu, X. (2003). Statistical learning in multiple instance problems. Master's thesis, University of Waikato.