A Framework for Learning Rules from Multiple Instance Data

Yann Chevaleyre and Jean-Daniel Zucker

LIP6-CNRS, University Paris VI,
4, place Jussieu,
F-75252 Paris Cedex 05, France
{Yann.Chevaleyre,Jean-Daniel.Zucker}@lip6.fr

Abstract. This paper proposes a generic extension to propositional rule learners to handle multiple-instance data. In a multiple-instance representation, each learning example is represented by a \bag" of fixed-length \feature vectors". Such a representation, lying somewhere between propositional and first-order representation, offers a tradeoff between the two. NAIVE-RIPPERMI is one implementation of this extension on the rule learning algorithm RIPPER. Several pitfalls encountered by this naive extension during induction are explained. A new multiple-instance search bias based on decision tree techniques is then used to avoid these pitfalls. Experimental results show the benefits of this approach for solving propositionalized relational problems in terms of speed and accuracy.

1   Introduction

In most ML applications, the choice of knowledge representation for a learning example is between a fixed-length "feature vector" and a first-order representation. The motivation for using first-order representation is that it is the natural extension to propositional representation. However, a known drawback of using first-order logic is that its expressivity is so high that in order to learn efficiently, strong biases such as *determinacy*, are often required on the hypothesis space. Giordana *et al.* have recently shown that there is a phase transition in relational learning [10] linked to the exponential complexity of matching. They argued that relational learners could hardly search in practice for target concepts having more than four non-determinate variables. The difficulty of learning relations has stimulated attempts towards extending Attribute/Value representation rather than directly using first-order logic based representation. Multiple-instance representation, where each example is represented by a "bag" of fixed-length "feature vectors" [8], is an extension that offers a good tradeoff between the expressivity of relational learning and the low complexity of propositional learning. Data represented as bags of vectors may either be found naturally in chemical domains [8], in images classification tasks [12], or be produced after multiple-instance propositionalization of first-order data [17, 1].

Much work has been done on multiple-instance learning. Unfortunately, available learners are not able to efficiently generate easily interpretable rule sets or decision trees. Also, the generated models cannot be reformulated into first-order theories; these

learners can therefor not be used to solve relational learning problems with multiple-instance propositionalized data. Because propositionalization based relational learners (such as STILL [14]) often outperform classical relational learners, relational learning based on multiple-instance propositionalization (which is much more adapted to non-determinate domains than standard propositionalization [17]) is a promising field for which efficient multiple-instance rule learners will be needed.

This paper proposes a framework for extending propositional rule learners to handle multiple-instance data. A first extension is presented and implemented in the RIPPER rule learning algorithm. The resulting algorithm, called NAIVE-RIPPERMI, is evaluated and analysed on artificial datasets. Several pitfalls encountered by this naive extension are then characterized before showing that a modification of the refinement procedure implemented in RIPPERMI avoid these pitfalls. Experiments on artificial datasets are used to validate these improvements. The last section presents experiments on relational data and shows the benefits of a multiple-instance learner for relational learning. As our algorithms generate rule sets, it is possible to use them on relational learning problems reformulated into multiple-instance learning tasks, to generate first-order rules. NAIVE-RIPPERMI and RIPPERMI are compared against three popular relational learners on the mutagenesis prediction problem.

2   The Multiple Instance Learning Problem

2.1   Definition and Notation

In the traditional setting of machine learning, an object is represented by a feature vector x, to which is associated a label $f(x)$. Let $\mathcal{X}$ be a feature vector space, and $\mathcal{Y}$ the finite set of labels or classes. For the sake of simplicity, we will restrict ourselves to the two-class case, i.e. $\mathcal{Y} = \{\oplus, \ominus\}$. The goal then, typically, is to find a classifier $h : \mathcal{X} \to \mathcal{Y}$ which minimizes the probability that $f(x) \neq h(x)$ on a newly observed example $(x, f(x))$.

Within the multiple instance framework, objects are represented by *bags of vectors* of variable size. Vectors are also called *instances*. As in the traditional setting, they can contain numeric as well as symbolic features. The size of a bag b is noted $\sigma(b)$. Its instances are noted $b_1 \ldots b_{\sigma(b)}$. The multiple instance induction task consists of finding a classifier $H : 2^{\mathcal{X}} \to \mathcal{Y}$, which accurately predicts the label $F(b)$[1].

The multiple instance learning problem has been associated to a bias introduced by [8], which will here be refered to as the *single-tuple bias*, as opposed to the multi-tuple bias proposed by [13]. It can be formally defined as follows:

Definition 1 *The single-tuple bias is a restriction on the set of functions H : $2^{\mathcal{X}} \to \{\oplus, \ominus\}$ to those for which there exists a function $h : \mathcal{X} \to \{\oplus, \ominus\}$ such that $H(b) \equiv \exists i, h(b_i)$.*

The underlying idea is that for certain learning tasks, if a bag is labeled positively, then at least one of its instances must be responsible for this.

---

[1] Note that functions on the instance space (resp. bag space) will be noted lower case (resp. upper case).

This idea can be particularly well illustrated in an example on which this bias has been extensively used: the task of predicting whether molecules smell musky or not [8]. Dietterich chooses to represent molecules as bags of vectors, each vector describing a steric configuration of the molecule. It is well known by chemists that a molecule is musky iff at least one of its configurations has given properties, which make the entire molecule smell musky. Thus, there exists a function h representing these properties, such that the function $H(b)$ - which is derived from h as shown earlier - is an accurate classification function for this learning task.

2.2   Related Work

Previous work on learning from multiple-instance examples has focused on the problem of learning axis-parallel rectangles (APR) under the single-tuple bias. In particular, Dietterich *et al.* [8] have designed APR algorithms to solve the task of predicting whether a molecule is musky or not. Other APR algorithms such as MULTIINST [3] have been tested on this learning task, and many interesting learnability results have been obtained [5, 2]. More recently, Maron *et al.* proposed a new multiple-instance algorithm called *Diverse Density* [12], which they applied to image classification. Finally, the lazy learning approach to multiple-instance learning has been investigated by Jun *et al.* [16].

The algorithms mentioned here do not generate interpretable hypotheses such as rule sets, which is our purpose. In the following, a method for inducing multiple-instance rules with a modified traditional rule learner will be presented. Note that Blockeel and De Raedt [4] already presented a method for extending propositional learners to handle relational data. The extension of a propositional learner to the multiple-instance case is less complex, and yields specific multiple-instance issues, as will be shown in the following.

3   Extending a Propositional Learner

3.1   Motivation

This section presents a method for the extension of a propositional learner to handle multiple-instance data using the single-tuple bias. Our choice to adapt a propositional learner instead of designing new multiple-instance algorithms is justified by the three following points. First, the two learning problems are very similar. In fact, multiple-instance data can easily be represented by a single set of vectors, and under the single-tuple bias, the multiple-instance and the single-instance search spaces are identical. Thus, the extension may be simple. Secondly, the existing multiple-instance learners [8,3,12] do not generate interpretable rules or decision trees. Note that an MI learner able to generate rule sets can be used to solve relational learning problem with an appropriate reformulation algorithm such as REPART [17]. This will be detailed in the final section. Finally, propositional learning is a mature field, and many of the available algorithms can efficiently handle large databases, while achieving low error rates. The extension of such a learner to the multiple-instance case will thus benefit from this maturity.

Extending a decision tree induction algorithm to the multiple-instance case raises several algorithmic problems, due to the *divide-and-conquer* aspect of the learner. These

issues are beyond the scope of this paper and will be addressed elsewhere. Fortunately, these problems are not encountered in the rule learning coverage algorithms. We have therefore chosen to propose an extension of propositional rule learners using a coverage algorithm.

3.2   A Single-Tuple Naive Extension of Ripper

Let us now study the modifications needed by a single-instance (i.e. traditional) rule learning algorithm in order to incorporate the single-tuple bias. Let us consider a generic single-instance rule learner using a coverage algorithm. It can be seen as an algorithm iteratively searching for a function h under a given bias, such that this function will minimize a given criterion related to the prediction error of h on the training dataset D. This criterion varies from one algorithm to another. Ripper [7] and C4.5 both use a criterion based on the information gain. To compute the value of this criterion, the learners first evaluate $count(h, D, \oplus)$ and $count(h, D, \ominus)$, which denote the number of positive (resp. negative) examples from D covered by h.

In order to adapt a single-instance learner to the multiple instance case, we first need to transcribe a multiple instance bag set into a simple set of vectors, without any loss of information. This can be done by adding to each instance two attributes, the first one named $bagid$ identifying the bag it belongs to and the second one named $bagclass$ encoding the class of its bag.

After having done this, we must now modify the evaluation criterion, such that $H(b) = \exists i, h(b_i)$ is evaluated instead of h. To do so, we will replace the function $count(h, D, c)$ by $count_{single-tuple}(h, D, c)$ evaluating the number of bags of class c encoded in D covered by H. Note that because of the single-tuple bias, if h covers a single vector x, then the bag identified by $bagid(x)$ will be considered as covered by H. Thus, we have:

$$count_{single-tuple}(h, D, c) = \\ |\{bagidx(x); \; x \in D \land h(x) \land bagclass(x) = c\}|$$

We have chosen to implement these modifications in Ripper, a fast efficient rule learner designed by Cohen [7] which has been shown to be as accurate as C4.5 on classical datasets. In addition, the rule sets induced by Ripper are usually very short, thus being easily interpretable. Ripper includes several functionalities such as pruning and rule optimization, which also had to be adapted to handle single-tuple hypotheses. The rule refinement strategy of Ripper consists in greedily adding the best literal without any backtracking. The optimization phases are thus important to improve the accuracy of the rules induced. The resulting algorithm, which we call Naive-RipperMI, inherits most of Ripper's qualities, such as the ability to efficiently handle large datasets in nearly linear time with the number of examples and the number of features.

3.3   Evaluating Naive-RipperMI

In order to compare Naive-RipperMI to the other multiple-instance learners, we chose to run experiments on the musk datasets, already presented in section 2.2. For a detailed

description of these datasets, see [8]. Table 1 presents the results of Naive-RipperMI measured using a tenfold cross-validation on the data.

On the musk1 dataset, the hypotheses generated by Naive-RipperMI contain an average of seven literals. This is primarily due to efficient pre-pruning and post-pruning techniques implemented in Ripper. In addition, the average induction time is less than sixty seconds on a Sun SparcStation 4 computer. The two algorithms which are here more accurate than Naive-RipperMI on musk1 generate models which are not directly interpretable. Both Iterated-discrim-APR and All-pos-APR have been specifically designed for this learning task [8]. In contrast, the ILP algorithms such as Tilde which give comprehensible theories are slower than our learner on this specific task.

On the musk2 dataset, Naive-RipperMI obtains an accuracy of 77%, which is far from the results on the musk1 dataset. In the former, the average number of instances per bag is much bigger than in the latter. More precisely, during cross-validations, Naive-RipperMI generates from the musk2 dataset concise hypotheses achieving low error rates on the training data, but whose error rates on the test data are significantly higher. This may be due to the large number instances and attributes, which causes some non predictive hypotheses to be consistent with the training data. Instance selection algorithms based on prototype selection techniques are currently under investigation by the authors to overcome this problem. Finally, note that because the musk datasets only contain numerical attributes, we do not expect our algorithms to compete with fully numerical methods such as APR learners.

In the following, the relation between consistency on training data and predictive power will not be addressed. Hence, our goal will not be to improve the accuracy of our learner on the musk datasets. Instead, we will focus on the ability of naive-RipperMI to find consistent hypotheses. Considering that naive-RipperMI is a simple MI extension of an optimized single-instance algorithm, it is likely to be sub-optimal. For example the greedy search procedure of Ripper may not be adapted to finding consistent MI hypotheses on datasets containing many instances. In the following section, the behavior of our algorithm will be analyzed carefully on artificial datasets, in order to design improvements.

Table 1. Compared accuracy of MI learners on both musk datasets.

Learner	Musk1	Musk2	Model
Iterated-discrim-APR [8]	0.92	0.89	Axis-parallel Rectangle
Citation-kNN [16]	0.92	0.86	k-nearest neighbour
Diverse Density [12]	0.89	0.82	Points in $\mathcal{X}$
RipperMI	0.88	0.77	rule set
Naive-RipperMI	0.88	0.77	rule set
Tilde [4]	0.87	0.79	horn clauses
All-pos-APR [8]	0.80	0.73	APR
MultiInst [3]	0.77	0.84	APR

4   Analysis of RIPPERMI Algorithms

The purpose of this section is to analyze and to understand the behavior of the algorithm presented earlier as NAIVE-RIPPERMI. This analysis will enable us to discover potential drawbacks, which we will try to solve. The following questions will guide our research. When the number of instances is equal to one, NAIVE-RIPPERMI is equivalent to RIPPER; how, therefore, does the algorithm react when the number of instances increases? Is the search procedure of NAIVE-RIPPERMI adapted to large numbers of instances? When does the algorithm fail to induce a theory? Considering that a multiple-instance learner can be viewed as a biased ILP learner [13], how well does an ILP algorithm compare to ours?

To answer these question, we need datasets on which all is known, in order to run several experiments. We have therefore decided to design a simple artificial dataset generator. The following subsection presents the generation of these datasets and their use.

4.1   Validation Protocol Using Artificial Datasets

In order to test and validate the multiple instance abilities of NAIVE-RIPPERMI, we constructed an artificial dataset generator which builds MI datasets according to parameters provided by the user[2]. As stated above, we were primarily interested in understanding the behavior of our algorithm as the number of instances per bag increases. For this reason, we measured the accuracy of our algorithms on several randomly generated datasets having a given number of instances per bag.

Each artificial dataset contains 200 bags, a given number of instances, and 12 boolean attributes. The target concept is a boolean conjunction of three literals combining 3 attributes out of 12. The distribution of the values of each attribute is chosen randomly by the artificial dataset generator. The bags are then built by drawing a given number of instances independently from this distribution, and labeled according to the target concept chosen by the generator. The decision to use a conjunction of boolean attributes, and a static number of instances per bag was intended to focus only on the multiple instance aspect of NAIVE-RIPPERMI, without taking into account its capability of handling numerical attributes or bags of variable size. Note that in the single-instance case this class of target concepts is PAC-learnable, whereas in the multiple-instances case, it is not. In the latter case, if viewed in the ILP setting, these concepts are *12-nondeterminate linked horn clauses*, which were proven not to be PAC-learnable [11]. Thus, the complexity shift from one to more than one instances is very large.

NAIVE-RIPPERMI will finally be tested on the mutagenesis dataset containing both numerical attributes and bags of variable size in the last section.

The different MI extensions of RIPPER described in this paper were run on these datasets with the default parameters, which consist of two optimization passes each followed by a pruning phase. Hundreds of datasets containing a given number k of instances per bag were generated; then, the accuracy of each algorithm was measured

---

[2] the source code and further experimentation details can be found on
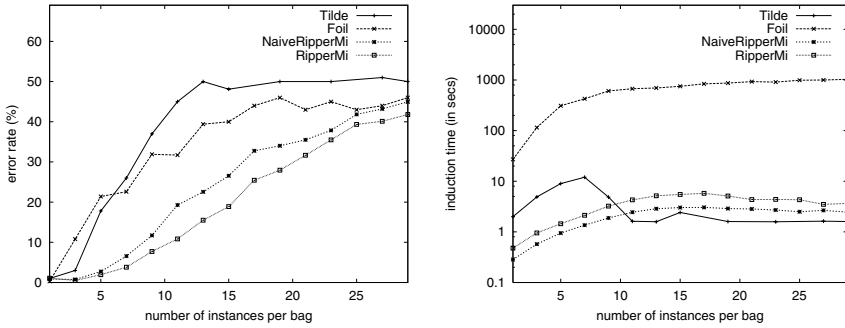`http://www-poleia.lip6.fr/~chevaley/ART_DAT_GEN/`

Fig. 1. Classification error rate (left figure) and induction time (right figure) of FOIL, TILDE, NAIVE-RIPPERMI, RIPPERMI (see section 4.3), on artificial datasets with various numbers of instances per bag

by averaging two-fold cross-validations over these datasets. The average classification error is ploted on figure 1, as well as the corresponding learning time using a log-scale. For example, on datasets containing 15 instances per bag, NAIVE-RIPPERMI obtains an average classification error rate of 26.5%, and the induction phase lasts less than a three seconds on a Sun SparcStation 4 computer.

The ILP learners FOIL and TILDE [4] were also run on these datasets in order to evaluate the ability of ILP tools on multiple-instance data. The top curve on the left part of figure 1 shows their accuracy with various numbers of instances per bag. On this particular task, they are outperformed by NAIVE-RIPPERMI in terms of accuracy. However, TILDE's induction time is very low, due to the "learning from interpretation" framework it implements.

4.2   Pitfalls during Induction

In this section, NAIVE-RIPPERMI will be analyzed, its pitfalls will be described, and the next section will propose algorithmic modifications to overcome them.

Let $x_j(b_i)$ denote the jth attribute of the instance b_i. For the sake of simplicity, multiple-instance rules $H(b)$ of the form $\exists i, x_1(b_i) = 0 \wedge x_2(b_i) = 1 \wedge \ldots \wedge x_j(b_i) = 0$ will be noted   as $(x1 = 0) \wedge (x2 = 1) \wedge \ldots \wedge (x_j = 0)$. A careful examination of the theories induced on the artificial datasets revealed three pitfalls of NAIVE-RIPPERMI. To illustrate these pitfalls, let us consider the four bags shown in table 2. The target concept is $(x1 = 1) \wedge (x2 = 0) \wedge (x3 = 1)$. NAIVE-RIPPERMI's strategy to refine a rule will be to examine each possible literal, and to add the one which brings the highest gain. Here, starting with an empty rule, the candidate rules $(x1 = 1)$, $(x2 = 0)$, and $(x3 = 1)$ each cover all four bags, $(x1 = 0)$ and $(x3 = 0)$ both cover one positive and two negative bags, and $(x2 = 1)$ covers two positive and one negative. Thus the best literal to start with, in terms of information gain, is $(x2 = 1)$. This literal is *misleading* w.r.t. target concept. Given a target concept $F(b) \equiv \exists i, f(b_i)$, a literal ℓ will be said *misleading* iff $\ell \Rightarrow \neg f$. We can easily show that with the artificial data sets used here,

bag	class	x_1	x_2	x_3
$bag1$	$\oplus$	**1**	**0**	**1**
		1	1	0
$bag2$	$\oplus$	0	1	1
		1	**0**	**1**

bag	class	x_1	x_2	x_3
$bag3$	$\ominus$	0	0	0
		1	1	1
$bag4$	$\ominus$	0	0	1
		1	0	0

Fig. 2. Two positive bags and two negative bag, with two instances each. Target: $(x_1 = 1) \land (x_2 = 0) \land (x_3 = 1)$

rules containing misleading literals have the following property: whatever their empirical error rate[3] is, their true error rate is higher than that of the default rule. In addition, when the number of instances per bag increases, the probability of having misleading literals correlated with the target concept on the dataset also increases, so does the probability that the induction algorithm chooses a misleading literal. Note that misleading literals is a typical multiple-instance phenomenon which cannot appear in the single-instance case. In the latter case, any rule containing a misleading literal would have an empirical error rate of 100%. Thus, empty rules would always be preferred to rules with misleading literals. In the following section, an algorithmic modification will be proposed to cope with this pitfall. The second pitfall can again be observed on the examples of table 2. From the six candidate rules proposed by NAIVE-RIPPERMI three rules cover the four bags. These three rules are thus *indistinguishable* for the learner. To avoid this pitfall, a new coverage measure has been developed. Due to space limitations, this measure which is based on counting the number of instances per bag covered by a rule will be described in a forthcoming paper. The last pitfall described in this paper consists in *irrelevant literals* added to rules. Irrelevant literals are literals which do not belong to the target concept, but which are not misleading. In single-instance rule learning, irrelevant literals are generally added at the end of rules because of overfitting. In multiple-instance learning, irrelevant literals may appear anywhere in a rule because candidate literals are often *indistinguishable*, as explained earlier. Although this phenomenon appears very often with multiple-instances, it can also appear with single-instance data.

4.3   Avoiding Pitfalls

Algorithmic modifications of NAIVE-RIPPERMI's search procedure to avoid misleading and irrelevant literals are now described. Suppose we are refining a rule R which is known not to contain any misleading literal yet. Let ℓ be the best literal to add to R, according to NAIVE-RIPPERMI's greedy strategy. Of course, we cannot be sure that ℓ is not a misleading literal. Yet, it is clear that at least one of the two literals ℓ and $\neg\ell$ is not misleading. Hence, by considering both $R \cup \ell$ and $R \cup \neg\ell$, at least one of the two rules will not contain any misleading literal. The induction process thus undoubtedly avoids this pitfall. Note that the process of examining two rules at each refinement step can be seen as building a binary decision tree from which a single rule is extracted. In such a tree, each node corresponds to a literal and paths from the root node to the leaves corresponds to the candidate rules.

---

[3] the empirical error rate of a rule is generally defined as $\frac{fp}{fp+tp}$ with tp and fp being the number of covered examples which label is (resp. is not) that predicted by the rule
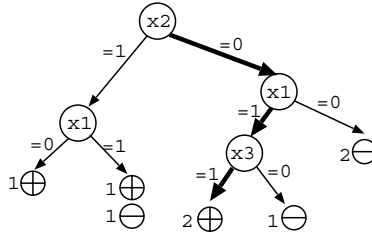
Fig. 3. Decision tree induced from bags in table 2

Our new refinement procedure builds such a decision tree, starting with a root node. Let $\{s_i\}$ be the set of leaves of the current tree, and r_i, the rule associated with the path from the root node to the leaf s_i. Let $gain(r, r*)$, the gain function used by NAIVE-RIPPERMI to evaluate the benefit of replacing the rule r by $r*$. Let ℓ_i be the literal which maximizes the gain $g_i = gain(r_i, r_i \cup \ell_i)$ for each rule r_i. The leaf s_j which has the highest gain g_j is chosen for expansion: the leaves corresponding to literals ℓ_j and $\neg \ell_j$ are added to s_j, which is now an internal node. This refinement process stops when all gains g_i are null. At last, the rule r_i which brings the highest value of $gain(\{\}, r_i)$ is extracted from the tree.

Considering the worst case, the storage requirement of this algorithm is linear with the total number of instances in the training set. In practice, small trees are generated by this algorithm, as all gains g_i become null rather quickly. This is due to the fact that often in multiple-instance learning tasks, large parts of the instance space are of no use to separate positive bags from negative ones. Note that the complexity of building this tree is similar to building a single-instance decision tree, which is $O(ma \log m)$ where m is the number of examples in the single-instance case and a the number of attributes. In the multiple-instance setting, m represents the total number of instances. Assuming as in [7,9] that the number of generated rules is approximately constant, the complexity of RIPPERMI is thus $O(ma \log m)$. Further experiments conducted on the artificial datasets confirmed that the algorithm's runtime was approximately linear with the number of bags. Due to lack of space, these experiments will be detailed elsewhere.

When running this algorithm on the small dataset described in table 2, it explores the decision tree as shown in figure 3. The leaves of the tree indicate how many positive and negative bags are covered by the corresponding rule. Here, the rule $(x2 = 0) \wedge (x1 = 1) \wedge (x3 = 1)$ covers two positive bags ($2\oplus$) and no negative one. This rule will thus be extracted from the tree, and the pitfall will be avoided. Much work has been done recently on the use of decision trees as a temporary representation for single-instance rule induction. Nevertheless, as stated by Frank and Witten [9], in the single-instance case, decision trees are used as a substitute to a global optimization on rule sets. Thus they do not provide a qualitative algorithmic improvement, unlike in the multiple instance case for which they enable pitfalls to be avoided.

In addition to misleading literals, induced theories may contain irrelevant literals anywhere in the rules. In the single-instance case, irrelevant literals usually appear at the end of the rule, because of overfitting. To avoid this, RIPPER implements a *reduced*

error pruning technique which tests and removes literals at the end of rules. We therefore added after this pruning step another step consisting in a modified reduced error pruning algorithm examining literals in the current rule in any order. Using the same validation procedure as earlier, the graphs of figure 1 respectively show the average classification error rate and induction time of RIPPERMI, the new algorithm implementing both improvements. With the dataset containing 15 instances per bag, for example, the classification error decreases from 26.5% to 18.9%. Note that these algorithmic improvements, aimed at inducing consistant hypotheses, have no impact on the musk learning task, as NAIVE-RIPPERMI was already consistant on these data.

5   Experiments on Relational Data

It has been shown that under various biases, the problem of learning from first-order data can be converted to a lower-order learning problem, in particular to attribute-value learning tasks. This process, called *propositionalization*, has already been investigated within the multiple-instance framework in [17]. In this section NAIVE-RIPPERMI and RIPPERMI will be used in association with REPART [17] to solve a traditional ILP problem : the mutagenesis prediction task [15].

5.1   Solving the Mutagenesis Problem with a Multiple-Instance Learner

The mutagenesis prediction problem [15] consists in inducing a theory which can be used to predict whether a molecule is mutagenic or not. To achieve this, a dataset describing 188 molecules with prolog facts is used. Several relational descriptions of the domain are available. We will use the description termed $\mathcal{B}_2$ [15] where atoms and bonds are described, and $\mathcal{B}_3$ which includes $\mathcal{B}_2$ as well as two global molecular properties.

The algorithm REPART [17] has been used to generate several propositionalizations. After each propositionalizations, the MI learner is launched on the reformulated data, and it outputs an hypothesis and its accuracy on the training set. The process stops if this accuracy is sufficiently high. If not, another more complex reformulation is chosen, and so forth. Using the description $\mathcal{B}_2$, REPART first represents molecules as bags of atoms. Thus, each instance contains the three attributes describing a single atom. As expected, this reformulation did not yield good results. During the second step, REPART represented molecules as bags of pairs of bonded atoms. The following subsection describes the results using this reformulation. With the $\mathcal{B}_3$ description level, the first reformulation

Table 2. Compared accuracy of RipperMi with ILP learners on the mutagenesis dataset.

	$\mathcal{B}_2$	$\mathcal{B}_3$
RIPPERMI	0.82	0.91
NAIVERIPPERMI	0.78	0.91
TILDE	0.77	0.86
PROGOL	0.76	0.86
FOIL	0.61	0.83

chosen by REPART has shown to be sufficient. This reformulation consisted in representing each molecule as a bag of atoms each to which was added global molecular properties.

5.2 Experiments and Results

The results of NAIVE-RIPPERMI and RIPPERMI are compared to those of state of the art ILP learners able to generate comprehensible hypotheses PROGOL [15], TILDE [4], and FOIL. Table 2 shows the accuracy of these learners measured with a tenfold cross-validation . Both NAIVE-RIPPERMI and RIPPERMI perform equally well on the $\mathcal{B}_3$ description, which is not surprising, because most literals added to the induced theories are global literals. Therefore, their multiple-instance ability is not challenged here. On the other hand, the reformulation using the $\mathcal{B}_2$ description level does not contain any global attributes. This explains the higher accuracy obtained by RIPPERMI compared to that of NAIVERIPPERMI. The following is an example of rule generated by our learner: `active ← (type1 = 1) ∧ (ch1 < 0.288) ∧ (ch2 < -0.404)` It indicates that if a molecule has a pair of bonded atoms such that the first one is of type 1 and has a partial charge lower than 0.288 and that the second one has a partial charge lower than -0.404, then the molecule is mutagenic. Both MI learners are faster than ILP algorithms. For example, on the $\mathcal{B}_2$ description level, NAIVE-RIPPERMI induces an hypothesis less than 150 seconds on a Sun SparcStation 4. In comparison, PROGOL requires 117039 seconds, TILDE requires 539 seconds, and FOIL requires 4950 seconds.

6 Conclusion

The problem of supervised multiple-instance learning is a recent learning problem which has raised interest in the machine learning community. This problem is encountered in contexts where an object may have several alternative vectors to describe its different possible configurations. Solving multiple-instance problems using propositional algorithms raises subtle issues that are related to the notion of bags of instances whose coverage is by essence different from that of mono-instance problems. We have proposed an method to extend a propositional rule learning algorithm to the multiple-instance case. Some drawbacks of this method have been detected and a better search procedure was developed. Each refinement has been validated on artificial datasets.

With the help of the REPART [17] algorithm, which reformulates first-order examples into bags of instances, our algorithm has been tested on the well known mutagenesis relational dataset. RIPPERMI yielded good results compared to those of FOIL TILDE and PROGOL on this problem. It also showed to be significantly faster. We therefore argue that the multiple instance paradigm may be very useful for solving a wide range of relational problems. Relational data mining tasks may also be addressed by multiple-instance learners, in particular when it is possible to create bags of instances making sense by joining tables together [6]. Finally, a future application of our learner will be to embed it in a mobile robot to recognize real-world objects from segmented images.

Many questions remain opened. The pitfalls described here appear more often when instances are independantly drawn from a distribution $\mathcal{D}$. How often do they appear if

this does not hold any more ? In fact, most theoretical studies were made under this statistical assumption which was shown to be reasonable in many cases. An interesting research issue would be to develop weaker assumptions which would be more realistic.

References

1. Erick Alphonse and Celine Rouveirol. Lazy propositionalization for relational learning. In *ECAI*, 2000.
2. P. Auer, P. Long, and Ashwin Srinivasan. Approximating hyper-rectangles: Learning and pseudo-random sets. In *Annual ACM Symposium on Theory of Computing*, 1997.
3. Peter Auer. On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proc. 14th International Conference on Machine Learning*, 1997.
4. Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.
5. Avrim Blum and Adam Kalai. A note on learning from multiple-instance examples. *Machine Learning*, 30, 1998.
6. Yann Chevaleyre and J.D. Zucker. Noise tolerant rule induction for multiple-instance data and potential data mining application. Tech. Rep. University of Paris 6, available at `http://www-poleia.lip6.fr/~chevaley/michurning.ps`, 2001.
7. William W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.
8. Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2), 1997.
9. Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proc. 15th ICML*, 1998.
10. Attilio Giordana, Lorenza Saitta, Michele Sebag, and Marco Botta. Analyzing relational learning in the phase transition framework. In *Proc. 17th ICML*, 2000.
11. J.U. Kietz. Some lower bounds for the computational complexity of inductive logic programming. In *ECML*, 1993.
12. Oded Maron and Aparna Lakshmi Ratan. Multiple-instance learning for natural scene classification. In *Proc. 15th ICML*, pages 341–349, 1998.
13. Luc De Raedt. Attribute-value learning versus inductive logic programming: The missing links. In *Proc. 8th International Conference on ILP*, 1998.
14. Michele Sebag and Celine Rouveirol. Tractable induction and classification in first order logic. In *IJCAI*, Nagoya, Japan, 1997.
15. A. Srinivasan and S. Muggleton. Comparing the use of background knowledge by two ilp systems. In L. de Raedt, editor, *ILP-95.*, Katholieke Universiteit Leuven, 1995.
16. Jun Wang and Jean-Daniel Zucker. Solving multiple-instance problem: a lazy learning approach. In *Proc. 17th ICML*, 2000.
17. Jean-Daniel Zucker and Jean-Gabriel Ganascia. Learning structurally indeterminate clauses. In *Proc. 8th International Conference on ILP*. Springer-Verlag, 1998.