

Learning Rules from Multiple Instance Data : Issues and Algorithms

Yann Chevaleyre

LIP6-CNRS - Univ. Paris 6
4, place Jussieu
F-75252 Paris - France
yann.chevaleyre@lip6.fr

Nicolas Bredeche

LIMSI-CNRS - Univ. Paris 11 and
LIP6-CNRS - Univ. Paris 6
4, place Jussieu
F-75252 Paris - France
nicolas.bredeche@lip6.fr

Jean-Daniel Zucker

LIP6-CNRS - Univ. Paris 6
4, place Jussieu
F-75252 Paris - France
jean-daniel.zucker@lip6.fr

Abstract

This paper proposes a generic extension to propositional rule learners to handle multiple-instance data. Multiple-instance representation lies between propositional and first-order representation and offers a tradeoff between the two. This naive extension, which is implemented on the rule learner RIPPER, encounters several pitfalls which are analyzed. Then, solutions are described to avoid these pitfalls.

Keywords: Multiple-instance learning, rule learning.

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.

Multiple-instance representation, where each example is represented by a "bag" of fixed-length "feature vectors" [4], is an extension of Attribute/Value representation 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

[4], in images classification tasks [6], or be produced after multiple-instance propositionalization of first-order data [8].

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 [3] 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 as well as a new multiple-instance probabilistic coverage measure implemented in RIPPERMI-REFINED-COV 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

The multiple instance (MI) learning problem was first introduced by Dietterich *et al.* [4]. In this section, it will be formally defined as well as a language bias traditionally used in MI learners [4, 6]. Finally, related work will be briefly reviewed.

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 $h : \mathcal{X} \rightarrow \mathcal{Y}$ 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 $b_1 \dots b_{\sigma(b)}$. The multiple instance induction task consists of finding a classifier $H : 2^{\mathcal{X}} \rightarrow \mathcal{Y}$, which accurately predicts the label $F(b)$ ¹.

The multiple instance learning problem has been associated to a bias introduced by [4], which will here be referred to as the *single-tuple bias*. It can be formally defined as follows:

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

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 [4]. 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 (which can be seen as learning from ambiguity). Thus, there exists a function h representing these properties, such that the function $H(b)$ - which is derived from h as shown ear-

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

lier - 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 [4]. More recently, Maron *et al.* proposed a new multiple-instance algorithm called *Diverse Density* [6], which they applied to image classification. Finally, the lazy learning approach to multiple-instance learning has been investigated by various authors. In the following, a method for inducing multiple-instance rules with a modified traditional rule learner will be presented.

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, as the multiple-instance and the mono-instance search spaces are structurally identical. Secondly, the existing multiple-instance learners [4, 6] 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 [8]. 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.

3.2 A single-tuple naive extension of RIPPER

Let us now study the modifications needed by a mono-instance (i.e. traditional) rule learning algorithm in order to incorporate the single-tuple bias. Let us consider a generic mono-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 [3] and C4.5 both use a criterion based on the information gain. To compute the value of this criterion, the learners first evaluate $\text{count}(h, D, \oplus)$ and $\text{count}(h, D, \ominus)$, which denote the number of positive (resp. negative) examples from D covered by h .

In order to adapt a mono-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 $\text{count}(h, D, c)$ by $\text{count}_{\text{single-tuple}}(h, D, c)$ 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 $\text{bagid}(x)$ will be considered as covered by H . Thus, we have: $\text{count}_{\text{single-tuple}}(h, D, c) = |\{\text{bagid}x(x); x \in D \wedge h(x) \wedge \text{bagclass}(x) = c\}|$

We have chosen to implement these modifications in RIPPER, a fast efficient rule learner designed by Cohen [3] 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.

Experiments on the traditional musk1 dataset

provided good results using a 10-fold cross-validation for NAIVE-RIPPERMI (with an accuracy of 0.88), compared to ITERATED-DISCRIM-APR [4] (0.92), DIVERSE DENSITY [6] (0.89), TILDE [1] (0.87), ALL-POS-APR [4] (0.80) and MULTIINST (0.77). For a more detailed description of these experiments, see [2].

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, 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 user². 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.

²the source code and further details can be found on our web site www-poleia.lip6.fr/~chevaley/ART_DAT_GEN/

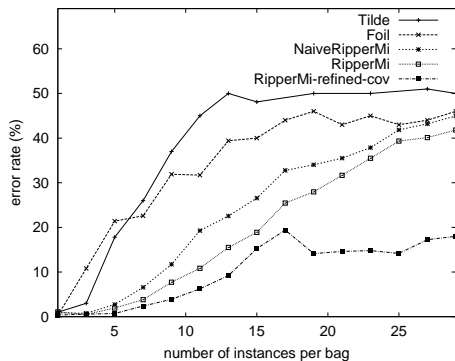


Figure 1: Classification error rate of FOIL, TILDE, NAIVE-RIPPERMI, RIPPERMI (see section 4.3) and RIPPERMI-REFINED-COV (see section 4.4), on artificial datasets

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 labelled 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. Nevertheless, NAIVE-RIPPERMI will 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 by averaging two-fold cross-validations over these datasets. The average classification error is plotted on figure 1. 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 com-

puter.

The ILP learners FOIL [?] and TILDE [1] were also run on these datasets in order to evaluate the ability of ILP tools on multiple-instance data. The top curves of figure 1 show their accuracy with various numbers of instances per bag. On this particular task, they are outperformed by NAIVE-RIPPERMI in terms of accuracy.

4.2 Pitfalls during induction

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

Let $x_j(b_i)$ denote the j^{th} attribute of the instance b_i . For the sake of simplicity, multiple-instance rules $H(b)$ of the form $\exists i, x_1(b_i) = 0 \wedge x_2(b_i) = 1 \wedge \dots \wedge x_j(b_i) = 0$ will be noted as $(x_1 = 0) \wedge (x_2 = 1) \wedge \dots \wedge (x_j = 0)$. 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 1. The target concept is $(x_1 = 1) \wedge (x_2 = 0) \wedge (x_3 = 1)$. 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 $(x_1 = 1)$, $(x_2 = 0)$, and $(x_3 = 1)$ each cover all four bags, $(x_1 = 0)$ and $(x_3 = 0)$ both cover one positive and two negative bags, and $(x_2 = 1)$ covers two positive and one negative. Thus the best literal to start with, in terms of information gain, is $(x_2 = 1)$. This literal is clearly *incompatible with the target concept*. It will thus be called a *misleading literal*. 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, rules containing misleading literals have the following property: whatever their empirical error rate³ 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

³the 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

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 mono-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.

bag	class	x_1	x_2	x_3
<i>bag1</i>	\oplus	1	0	1
		1	1	0
<i>bag2</i>	\oplus	0	1	1
		1	0	1
<i>bag3</i>	\ominus	0	0	0
		1	1	1
<i>bag4</i>	\ominus	0	0	1
		1	0	0

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

The second 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 mono-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 appear with mono-instance data.

The last pitfall can again be observed on the examples of table 1. From the six candidate rules proposed by NAIVE-RIPPERMI three rules cover the four bags. These three rules are thus *indistinguishable* for the learner. In our datasets, instances of positive and negative bags are drawn independantly from the same distribution. Thus, for datasets containing many instances per bag, most of the candidate rules cover the entire dataset, which makes them indistinguishable. To avoid this pitfall, we describe in section 4.4 a new coverage measure that has been theoretic-

cally developed and implemented as RIPPERMI-REFINED-COV.

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 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 recursively building a binary decision tree from which a single rule is extracted. Figure 2 describes a simplified recursive implementation of this refinement algorithm.

refine(R, E)

% return the best refinement of R, using the dataset E

```

if stopping_criterion( $R, E$ ) then return  $R$ 
 $\ell^* = \operatorname{argmax}_{\ell} \operatorname{gain}(R, R \cup \ell, E)$ 
 $R_1 = \operatorname{refine}(R \cup \ell^*, E)$ 
 $R_2 = \operatorname{refine}(R \cup \neg\ell^*, E)$ 
if  $\operatorname{gain}(R, R_1, E) > \operatorname{gain}(R, R_2, E)$  and
 $\operatorname{gain}(R, R_1, E) > 0$  then return  $R_1$  else
if  $\operatorname{gain}(R, R_2, E) > 0$  then return  $R_2$  else
return  $R$ 

```

Figure 2: Refinement procedure modified to avoid misleading literals. The $\operatorname{gain}(R, R', E)$ function based on RIPPER’s entropy and on $\operatorname{count}_{\text{single-tuple}}$ is similar to that used by the refinement procedure of NAIVE-RIPPERMI. It computes the information gain brought by selecting the rule R' instead of R to classify E

When running this algorithm on the small dataset described in table 1, 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 $(x_2 = 0) \wedge (x_1 = 1) \wedge (x_3 = 1)$ covers two positive bags ($2\oplus$) and no negative one. This rule will thus be extracted from the tree, and the pitfall will

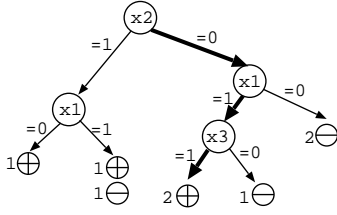


Figure 3: Decision tree induced from table 1

be avoided.

Much work has been done recently on the use of decision trees as a temporary representation for mono-instance rule induction. Some ideas such as partial tree induction [5] can be transposed to our case in order to avoid exponential growth of the tree. Note that as stated by Frank and Witten [5], in the mono-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 mono-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 extended the pruning algorithm to a *global pruning*, examining literals in rules 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%.

4.4 A more refined coverage measure

Up to now, the coverage measure that has been used in our experiments is the function $cover_{signe_tuple}(h, D, c)$ introduced in section 3. This function returns an integer indicating the number of bags in D of class c covered by H . If a bag b^1 has one of its instances covered by h , and a bag b^2 has 15 of its instances covered by h , both bags will count exactly the same way. This

has severe drawbacks; in our experiment, when the number of instances becomes high, given any rule made of a single literal, the probability that each bag has at least one instance covered by this rule is very high. Thus the learner does not have enough information to choose which literal to add first. Note that even though each bag is covered by the candidate rule, some bags may have more covered instances than others. This section will present a new single-tuple coverage measure exploiting this information.

We will now suppose that there exists function f , such that $F(b) \equiv \exists i, f(b_i)$, with $F(b)$ denoting the label of any bag b . Suppose we succeed in inducing an hypothesis $H(b) \equiv \exists i, h(b_i)$ complete and consistent with the training data. If f and h often disagree on instances of the training data, then it is likely that H often disagrees with F on test data. H will thus have a low predictive power. Thus, inducing complete and consistent hypotheses is not sufficient. The associated functions h must cover at least one instance covered by f per positive bag. Because we do not have access to f , the previous requirement cannot deterministically be guaranteed. Hence, we will evaluate its probability. Let b be a positive bag containing σ instances. Let k be the number its instances covered by h . Let $R = \{b_{i1} \dots b_{ik}\}$ be those instances. The probability that at least one instance of b is covered by h and f is: $P(f(b_{i1}) \vee \dots \vee f(b_{ik}))$.

The new coverage measure on positive bags is thus the sum of the above probability on each positive bag. Of course, the coverage measure on negative bags remains the same as before.

From now on, we will need to assume that every positive bags containing σ instances were build by independently drawing $\sigma - 1$ instances from a distribution \mathcal{D} , and by drawing a single instance from a distribution \mathcal{D}_f defined as: $\mathcal{D}_f\{x \mid \neg f(x)\} = 0$ and $\mathcal{D}_f\{x \mid f(x)\} = \mathcal{D}\{x \mid f(x)\}$.

Let q be the index of an instance of b chosen randomly from all instances of b covered by f . The above probability becomes: $P(b_q \in R) \times P(f(b_{i1}) \vee \dots \vee f(b_{ik}) \mid b_q \in R) + P(b_q \notin R) \times P(f(b_{i1}) \vee \dots \vee f(b_{ik}) \mid b_q \notin R)$.

Clearly, the first term is equal to $\frac{k}{\sigma}$. The first probability of the second term is thus $1 - \frac{k}{\sigma}$. Note that in a positive bag at least one instance fires

f . Thus, the distribution of instances in positive bags differs from \mathcal{D} . Suppose that an instance firing f is removed from each positive bag; then the distribution of instances in these bags becomes \mathcal{D} . Hence, when $b_q \notin R$, we have $P(f(b_{ij})) = P_{\mathcal{D}}(f)$. Thus, the equation becomes: $\frac{k}{\sigma} + (1 - \frac{k}{\sigma}) \times (1 - P_{\mathcal{D}}(\neg f)^k)$ and then, after simplification: $1 - (1 - \frac{k}{\sigma}) \times P_{\mathcal{D}}(\neg f)^k$.

This refined probabilistic coverage function has the following desired properties: if $k = 0$, the probability is null, and if $k = \sigma$, it is equal to one. Note that $P_{\mathcal{D}}(f)$ is still unknown. To evaluate it, our algorithm generates a first hypothesis h , and computes the total number of instances covered by h from positive bags. Given $nbag$ the number of positive bags covered by h , and $mpos$ the number of instances from these positive bags. We approximate $P_{\mathcal{D}}(f)$ with $\frac{mpos - nbag}{mpos - nbag}$.

This measure was implemented in RIPPERMI (thereby referred to as RIPPERMI-REFINE-COV). The results can be seen in figure 1. This refined measure prevents the accuracy from decreasing suddenly (when the number of instances increases), as it did with the earlier algorithms. For reasons mentioned earlier results on the musk datasets were not significantly improved by these new algorithms.

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 [8]. In this section NAIVE-RIPPERMI and RIPPERMI will be used in association with REPART [8] to solve a traditional ILP problem: the mutagenesis prediction task [7].

5.1 Solving the Mutagenesis Problem with a Multiple-Instance Learner

The mutagenesis prediction problem [7] 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 de-

scriptions of the domain are available. We will use the description termed \mathcal{B}_2 [7] 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 [8] has been used to generate several propositionalizations. After each propositionalization, 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 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-REFINED-COV are compared to those of state of the art ILP learners able to generate comprehensible hypotheses PROGOL [7], TILDE [1], and FOIL.

Both NAIVE-RIPPERMI and RIPPERMI-REFINED-COV 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

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

Table 2: Accuracy (10-fold validation) of RipperMi with ILP learners on the mutagenesis dataset.

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-REFINED-COV compared to that of NAIVERIPPERMI. 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 [8] 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. Finally, we are currently embedding our learner in a mobile robot to recognize real-world objects from segmented images.

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] 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.
- [2] Yann Chevaleyre and J.D. Zucker. A framework for learning rules from multiple instance data. 2001.
- [3] William W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995.
- [4] 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.
- [5] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proc. 15th ICML*, 1998.
- [6] Oded Maron and Aparna Lakshmi Ratan. Multiple-instance learning for natural scene classification. In *Proc. 15th ICML*, pages 341–349, 1998.
- [7] 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.
- [8] Jean-Daniel Zucker and Jean-Gabriel Ganascia. Learning structurally indeterminate clauses. In *Proc. 8th International Conference on ILP*. Springer-Verlag, 1998.