Towards clausal discovery for stream mining

Anton Dries¹ and Luc De Raedt¹

Department of Computer Science,
Katholieke Universiteit Leuven
`{firstname.lastname}@cs.kuleuven.be`

Abstract. With the increasing popularity of data streams it has become time to adapt logical and relational learning techniques for dealing with streams. In this note, we present our preliminary results on upgrading the clausal discovery paradigm towards the mining of streams. In this setting, there is a stream of interpretations and the goal is to learn a clausal theory that is satisfied by these interpretations. Furthermore, in data streams the interpretations can be read (and processed) only once.

1 Introduction

The data base and data mining community has devoted a lot of attention recently to dealing with data streams. A data stream consists of a continuous, often high speed, flow of data points that is so large that each data point can be processed just once. Often these data streams are also susceptible to continuously evolving underlying patterns [1]. Special purpose querying and mining techniques have been developed to cope with such streams in the literature [2]. However, to the best of the authors' knowledge, such techniques have not yet been used in a logical and relational learning setting and there are several interesting questions that arise in this context: What is an appropriate formalization of stream mining in a logical and relational learning setting? and if we have such a setting, can we still develop efficient learning techniques? and for what purposes can this setting be employed?

In this note, we provide an initial answer to these questions. More specifically, we show that the learning from interpretations setting introduced by Valiant [3] in the propositional case and upgraded by De Raedt and Dzeroski [4] for the relational case, constitutes an appropriate setting for read-once stream mining. Furthermore, we also show how the algorithms used in these PAC-learning results can be adapted for use in an incremental read-once setting. The resulting algorithm computes a jk -clausal theory that is satisfied by all examples. While the learning from interpretations setting has been quite popular in inductive logic programming, cf. Claudien [5], Tertius [6], Logan-H [7], it is – to the best of the authors' knowledge – the first time that a read-once logical and relational learning algorithm is developed. The framework we present extends and generalizes that of Dries et al. for the propositional case of mining mining k -CNF theories from streams of item-sets [8]. Dries et al. have also shown that

the induced propositional theories can be used for a variety of purposes, such as classification, missing value imputation and detection of concept drift [9].

2 A logical and relational stream mining definition

We shall assume that examples are (Herbrand) interpretations, that is, sets of ground atoms. We shall induce clausal theories consisting of a set of (range-restricted) clauses, where each clause is a formula of the form $c : h_1 \vee \dots \vee h_n \leftarrow b_1 \wedge \dots \wedge b_m$ where the h_i and b_i are logical atoms. We call the disjunction of h_i s the head of the clause and the conjunction of b_i s the body of the clause¹. An example e *satisfies* or is *covered by* a clause $c : H \leftarrow B$, notation: $e \models c$, if and only if $\forall \theta : B\theta \subseteq e \vee H\theta \cap e \neq \emptyset$. When there exists a substitution θ such that $B\theta \subseteq e$ and $H\theta \cap e = \emptyset$ we say that the example *violates* the clause and call θ a *violating substitution*.

As a generality relation we shall employ the usual notion of θ -subsumption. Basically, a clause s θ -subsumes a clause g , notation $s \preceq g$, if and only if there exists a substitution θ such that $s\theta \subseteq g$. In the learning from interpretations setting, the direction of generality is reversed as compared to the learning from entailment setting, cf. [10]. Therefore, the θ -subsumption is also monotonic, that is, if a clause s is satisfied by an example e , and $s \preceq g$ then e will also satisfy g . This property will be important for the stream mining setting, because once an example is covered by a clause, it is guaranteed that all refinements under θ -subsumption (applying substitutions or adding literals to the head or the body) will also cover the example, and hence, the example can be forgotten.

Systems such as Claudien [5] and Tertius [6] now start from a given set of examples E , a language of clauses $\mathcal{L}$ and compute the theory

$$\mathcal{T} = \{c \in \mathcal{L} \mid E \models c \text{ and } \forall s \in \mathcal{L} : s \prec c \rightarrow E \not\models s\}.$$

This theory only computes the maximally specific clauses covering the examples, as the others ones are entailed by $\mathcal{T}$, and, hence, logically redundant. Claudien and Tertius compute – in a way – $\mathcal{T}$ and realize this by treating the examples in batch using top-down search. However, this does not satisfy the read-once requirement of stream mining. We therefore propose to compute $\mathcal{T}$ incrementally and requiring that each example is read just once. This leads to the following problem that has to be tackled once for each example:

Given:

- a language of clauses $\mathcal{L}$ with language bias
- a theory of clauses $\mathcal{T}_{n-1} \subseteq \mathcal{L}$
- an interpretation e

$$\begin{aligned} \textbf{Find } \mathcal{T}_n = & \{c \in \mathcal{T}_{n-1} \mid e \models c\} \cup \\ & \{g \in \mathcal{L} \mid \exists s \in \mathcal{T}_{n-1} : e \not\models s \text{ and } s \prec g \text{ and } e \models g \text{ and} \\ & \quad \neg \exists g' \in \mathcal{L} : s \prec g' \prec g : e \models g'\} \end{aligned}$$

¹ A clause is range-restricted if all variables appearing in the head of the clause also appear in its body.

It should be clear that the final theory that is computed in this way coincides with the theory that is computed directly, that is, by processing the examples in batch using a non-heuristic, complete algorithm, and, that the final theory is independent of the example order.

The problem setting can also easily be extended to take into account background knowledge in the usual way when learning from interpretations. It corresponds to employing a definite clause theory and computing for each example the least Herbrand model of the background theory and the set of facts describing the clauses, cf. [10, 5].

This setting is also natural for dealing with streams, because in many applications such as robotics and video analysis every state can be described in a natural way by an interpretation and over time streams of such interpretations would be generated. Although it remains an open question, it seems harder to adapt the usual notion of learning from entailment to mining streams because that setting assumes information about the examples resides in the background theory and has – mainly – been focussing on heuristic classification and not on description.

Finally, as shown for the propositional case by Dries et al. [8], the setting above can be used for classification and for missing information imputation. For example, when confronted with an unlabeled example a score is calculated for each possible class label. This score is based on the number of clauses that are violated by the example and the class label. The optimal class label is then chosen as the one with the best score, i.e., the one that violate the least amount of clauses. Because the learning algorithm does not make a distinction between a class attribute and any other attribute, the same method can be used to predict an arbitrary number of missing values. In the first-order case these missing value problems correspond to finding suitable completions for partial interpretations, or in other word, what information do we need to add to an interpretation to make it satisfy the current theory?

3 Algorithm

The algorithm that we introduce is based on the candidate-elimination algorithm of Mitchell [11], which computes border-sets of hypothesis consistent with the data. The set $\mathcal{T}$ that we compute can be considered the border of maximally specific clauses consistent with the examples. The resulting algorithm is shown below. The initial theory contains the maximally specific clause, that is, the empty one, which is not satisfied by any example.

For each example in the stream of examples, we then repeat the following three steps: find clauses in $\mathcal{T}_n$ that are not satisfied by the current example, refine these clauses, and possibly compress the resulting theory to remove those clauses that are not maximally specific, and, hence, logically redundant. Because the first step and third step are straight-forward, we focus on the second step.

The most interesting step is that of applying the refinement operator $\rho(c, e)$ to generate refinements of the clause c with respect to e , that is, $\rho(c, e) = \{c' | c \prec$

c' and $e \models c'$ and not $\exists d : c \prec d \prec c', e \models d$. This corresponds to computing the minimally general generalizations c' of c such that $e \models c'$.

As usual in logical and relational learning, the refinement operator is governed by a language bias that defines which literals can be added to the head and which literals can be added to the body of the clause, and that also specifies mode restrictions on the variables in the new literals. These mode restrictions specify whether a literal can use constants, existing variables, or whether it can introduce new variables. The refinement operator generates a complete set of refinements within the restrictions set out by the bias. This completeness guarantees that all possible (minimal) clauses are generated and that the final theory is independent of the order of the examples in the data stream.

Because there are typically many possible refinements of a clause, we want to minimize the number of generated refined clauses by eliminating redundant ones already during the refinement phase. Here, the read-once property comes in handy because the refined clause should only add those literals that are necessary for the current example. To this aim, we employ the set of substitutions Θ_F that violate the clause. We can distinguish between two types of refinements. On one hand we have literals that only use variables that are already part of the clause. Finding a minimal set of such literals is equivalent to solving a set covering problem where the chosen literals should form a minimal set that covers all violating substitutions. We do this by first listing the possible literals that can be added to the clause and the violating substitutions they resolve. This gives us a set of the following form $\{l_1 : [\theta_1, \theta_2, \theta_3], l_2 : [\theta_3, \theta_4], l_3 : [\theta_1, \theta_2, \theta_5], l_4 : [\theta_1, \theta_2, \theta_3, \theta_5]\}$, which indicates that, for example, l_1 resolves substitutions θ_1, θ_2 , and θ_3 . From this we find all minimal sets of literals that cover all substitutions. In this case $\{l_2, l_3\}$ and $\{l_2, l_4\}$ are the only such sets, and $\{l_1, l_2, l_3\}$ is not because l_1 is redundant.

On the other hand we have literals that add new variables to the clause. Avoiding redundancy for these literals is harder, because they cause dependencies between the new literals. For example, it would only be possible to add a literal that uses variable X to the head of a clause if a literal creating variable X is already added to the body of the clause. However, it is still possible to reduce the number of redundant refinements by taking into account the violating substitutions of the clause for the current example.

Finding an exact solution to this problem is hard because there are many cases to consider. Therefore, we use a heuristic approach that uses a scoring function on new literals based on the set Θ_F and the current example. It is

```

 $\mathcal{T}_0 = \{\square\}$ 
for  $e_n \in \mathcal{E}$  do
  for  $c \in \mathcal{T}_{n-1}$  do
    if  $e_n \models c$  then
      add  $c$  to  $\mathcal{T}_n$ 
    else
      add all  $c' \in \rho(c, e_n)$  to  $\mathcal{T}_n$ 
    end if
  end for
  compress( $\mathcal{T}_n$ )
end for

```

Fig. 1. incremental clausal discovery algorithm

important to note that this heuristic only reduces the number of non-minimal refinements, and has no impact on the final theory of the algorithm.

Our approach is based on a simple and effective way of avoiding the generation of permuted versions of the same clause. These permutations can be easily avoided by using a fixed order in which new literals are added. This approach is used in most refinement operators found in ILP. We modify this approach by using a dynamic order on the literals that allows us to greedily expand the best refinements first. This greedy breadth-first approach requires a scoring function that assigns a value to (partial) clauses based on how well they cover the current example. This scoring function is based on the fraction of all possible substitutions (for the current example) that violate the clause. This score can be easily calculated based on the set of violating substitutions Θ_F and the constants present in the example using the formula

$$Score(c, e) = \frac{|\Theta_F(c, e)|}{\prod_{Var \in c} |Values(e, Var)|}$$

In this formula $|Value(e, Var)|$ is the number of distinct values, present in example e , that can be assigned to Var (i.e. that have same type). This function has three interesting properties: (1) it can only decrease (or stay equal) when a literal is added to the clause, (2) it can deal with the introduction of new variables in the clause, and (3) it reaches zero if-and-only-if the clause is satisfied by the example and does not require further refinement. Furthermore, if a new literal does not decrease the value of this function, then it is redundant (unless it introduces a new variable).

4 Extended example

Consider the following simple language for describing scenes containing objects

$$\mathcal{L} = \{square(id), triangle(id), circle(id), in(id, id)\},$$

and its associated background knowledge

$$polygon(X) \leftarrow triangle(X). \quad polygon(X) \leftarrow square(X).$$

We use this language to represent the following two scenes.

$$\begin{aligned} e_1 &= \{triangle(a), circle(b), in(a, b)\} \quad \triangle \circ b \\ e_2 &= \{square(x), triangle(y), in(x, y)\} \quad \triangle \square \end{aligned}$$

We set our initial theory $\mathcal{T}_0$ to contain only the empty clause $\square$. After processing example e_1 the system learns 10 different clauses.

$$\begin{aligned} triangle(A) &\leftarrow in(A, B) & (1) & \quad \leftarrow polygon(A) \wedge circle(A) & (6) \\ circle(B) &\leftarrow in(A, B) & (2) & \quad \leftarrow polygon(A) \wedge in(B, A) & (7) \\ polygon(A) &\leftarrow in(A, B) & (3) & \quad \leftarrow square(A) & (8) \\ triangle(A) &\leftarrow polygon(A) & (4) & \quad \leftarrow circle(A) \wedge in(A, B) & (9) \\ &\leftarrow in(A, A) & (5) & \quad \leftarrow in(A, B) \wedge in(B, A) & (10) \end{aligned}$$

These are all the minimal clauses that can be learned from this limited example. There is still some redundancy in this set, for example clauses (1) and (6) combined with background knowledge imply clause (9). However, we cannot remove clause (9) because clauses (1) and (6) might not remain true for future examples. Indeed, when we update our theory with respect to example e_2 we see that clauses (3), (5), (6), (9) and (10) still apply, but the other, including clause (1), no longer do. If we had removed clause (9), it would no longer be part of the theory, and since it is not a refinement of any existing clause, it would not be reconsidered as part of the theory.

We can now repeat the process for example e_2 . Clause (7) $\leftarrow \text{polygon}(X) \wedge \text{in}(Y, X)$, for example, is violated by example e_2 because there is a substitution $\theta_F = \{X/b, Y/a\}$ which violates the clause. There are several refinements that resolve this substitution, for example $\text{triangle}(X)$, $\text{square}(Y)$, or $\neg \text{triangle}(Y)$. In total, theory $\mathcal{T}_2$ contains 45 clauses.

5 Experiments

Implementation We implemented a preliminary version of our algorithm in Sicstus Prolog. It uses a specialized trie-based data structure to store the theory of clauses, which enables fast discovery of violated clauses and clause subsumption. In its current version this implementation has two limitations: (1) it does not test for background subsumption, i.e., it does not remove clauses that are subsumed by the background knowledge, and (2) it enforces inequality constraints on variables, i.e., it forces different variables to take distinct values. Both these limitations lead to a final theory that contains more clauses than strictly necessary, but which is still minimal under these restrictions.

Poker dataset For our experiments we primarily used the Poker dataset to analyze the behavior and performance of our algorithm. This dataset was introduced by Blockeel et al. [12] and contains a descriptions of poker hands and their classification. We used a dataset containing 10.000 interpretations of the form,

$$\{\text{card}(R_1, S_1), \text{card}(R_2, S_2), \text{card}(R_3, S_3), \text{card}(R_4, S_4), \text{card}(R_5, S_5), \text{class}(C)\},$$

where R_i and S_i indicate rank (ace through king) and suit (spades, hearts, clubs or diamonds) respectively. This dataset contains seven classes: nothing, pair, double pair, full house, three of a kind, flush, and poker.

Rule learning behavior First we ran our algorithm directly on this dataset without any additional background knowledge or length restriction on the learned clauses. Processing the entire dataset took approximately 125 seconds and produced 154 rules. Within these clauses several subgroups can be discovered. For example, there are 60 different clauses to express that there can not be 6 cards in a hand, or there are 8 different rules to express the elements of class *pair*. Some of these groups could be simplified by using more flexible variable inequality constraints. Interestingly, only 32 examples contributed to the theory; the other 9.968 examples did not violate any clauses, and consequently did not update the theory.

Fig. 2. Evolution of runtime and theory size for the Bongard dataset.

For the second run, we defined two background predicates: *same_rank* and *same_suit* that count the number of cards of the same rank and suit respectively, for example, $\text{same_rank}(2, R) \leftarrow \text{card}(R, A) \wedge \text{card}(R, B) \wedge A \neq B$.

We modified our language bias to use these new predicates instead of the *card* atom. Running the algorithm with this setting took 25 seconds and produced 95 rules. In this case, only 12 examples were responsible for the entire theory. Interestingly, three of the classes (full house, flush and poker) were only represented by a single example in this reduced dataset.

We also repeated these experiments for a dataset containing Bongard problems [13] (of which a simplified example was used in the previous section). We restricted the theory to clauses of length 5. The results of these experiments are summarized in Figure 2. For this dataset the final theory was based on 94 out of 392 examples. Figure 2(b) shows clearly the effect of pruning due to the maximal length constraint.

Classification and completion of partial interpretations We also performed experiments to see whether the classification and imputation results from [8] still hold for the first-order case. For this experiment we randomly generated two sets of 100 poker hands, where we made sure that each class was represented in both datasets. We trained a single theory (without background knowledge) on the first set and evaluated this theory on the second set.

First we used the standard classification setting, where we removed the class label from the examples in the test set. We then assigned a score for each class label based on the number of clauses violated by the example extended with this class label. For most examples, assigning the correct class label did not violate any clauses, while assigning other class labels did. We repeated this experiment five times on different random datasets, producing an overall accuracy of 97.8%.

We also used our algorithm to complete a partial poker hand of four cards and a class label. We used the same theories learned in the previous setting and applied the same scoring mechanism. In this setting, multiple alternatives were predicted for each example. For example, when given the partial interpretation

$[\text{card}(2, \text{clubs}), \text{card}(9, \text{diamonds}), \text{card}(9, \text{spades}), \text{card}(2, \text{hearts}), \text{class}(\text{full_house})]$

the system would output

$[card(9, hearts), card(9, clubs), card(2, spades), card(2, diamonds)]$

as possible fifth cards. In all cases the original card was among those predicted.

6 Conclusions and future work

We introduced – for the first time – a framework for stream mining in the setting of logical and relational learning. Our approach upgrades the setting of clausal discovery in interpretations, as used by earlier systems such as Claudien [5], Tertius [6], and Logan-H [7], towards mining of streams. We proposed an algorithm based on the Candidate Elimination algorithm of Mitchell [11], and applied it to the problems of classification and completion of partial interpretations.

Future work in this area includes (1) a further optimization and implementation of our algorithm, (2) a thorough experimental verification of our preliminary results and a comparison with other ILP systems, and (3) application of the framework to other stream mining problems such as concept drift detection.

Acknowledgments

This work was partially supported by the GOA project 08/008 on Probabilistic Logic Learning and the European Commission FP7 project BISON.

References

1. Aggarwal, C.C.: Data streams: models and algorithms. Springer New York (2007)
2. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: a review. *SIGMOD Record* **34**(2) (2005) 18–26
3. Valiant, L.G.: A theory of the learnable. *Communications of the ACM* **27**(11) (1984) 1134–1142
4. De Raedt, L., Džeroski, S.: First order jk-clausal theories are PAC-learnable. *Artificial Intelligence* **70**(1-2) (October 1994) 375–392
5. De Raedt, L., Dehaspe, L.: Clausal discovery. *Machine Learning* **26**(2-3) (1997) 99–146
6. Flach, P.A., Lachiche, N.: Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning* **42**(1-2) (2001) 61–95
7. Arias, M., Khardon, R., Maloberti, J.: Learning horn expressions with LOGAN-H. *Journal of Machine Learning Research* **8** (2007) 549–587
8. Dries, A., Nijssen, S., De Raedt, L.: Mining predictive k-CNF expressions. *IEEE Transactions on Knowledge and Data Engineering* (**in preprint**) (2009)
9. Dries, A., Rückert, U.: Adaptive concept drift detection. In: *SIAM International Conference on Data Mining*, SIAM (May 2009) 233–244
10. De Raedt, L.: Logical and Relational Learning. Springer-Verlag (2008)
11. Mitchell, T.M.: Machine Learning. McGraw-Hill (1997)
12. Blockeel, H., De Raedt, L., Jacobs, N., Demoen, B.: Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery* **3** (2000) 59–83
13. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: *International Conference on Algorithmic Learning Theory*, London, UK, Springer-Verlag (1995) 80–94