Towards clausal discovery for stream mining

Anton Dries[1] and Luc De Raedt[1]

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

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

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 $c : h_1 \vee \ldots \vee h_n \leftarrow b_1 \wedge \ldots \wedge b_m$ where the h_i and b_i are logical atoms. We call the disjunction of h_is the head of the clause and the conjunction of b_is the body of the clause[1]. An example e *satisfies* or is *covered by* a clause $c : H \leftarrow B$, notation: $e \models c$, if and only if $\forall \theta : B\theta \not\subseteq e \vee H\theta \cap e \neq \emptyset$. 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

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

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

Find $\mathcal{T}_n = \{c \in \mathcal{T}_{n-1} | e \models c\} \cup$
$\{g \in \mathcal{L} | \exists s \in \mathcal{T}_{n-1} : e \not\models s \text{ and } s \prec g \text{ and } e \models g \text{ and}$
$\neg \exists g' \in \mathcal{L} : s \prec g' \prec g : e \models g'\}$

[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.

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 Herband 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 $e \models c'$.

$\mathcal{T}_0 = \{\square\}$
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
  end
  `compress(`$\mathcal{T}_n$`)`
end

Algorithm 1: incremental clausal discovery algorithm

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.

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.

Refinements come in two flavors. 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 is still possible to reduce the number of redundant refinements by taking into account the violating substitutions of the clause for the current example. For example, we want to avoid adding a literal to the clause that covers the same substitutions as another new literal already added to the clause. By keeping track of which new literals cover

which violating substitutions we can avoid this type of redundancy. This method is similar to the set covering approach but with the added complexity of keeping track of the newly introduced variables.

4 Experiments

We implemented a preliminary version of our algorithm in SWI-Prolog and ran it on the poker dataset to examine its behavior and performance. This dataset was introduced by Blockeel et al. [12] and contains a collection of descriptions of poker hands and their classification. We used a dataset containing 10.000 interpretations of the form, where R_i and S_i indicate rank and suit respectively

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

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 40 minutes and produced 736 rules. Interestingly, only 25 examples contributed to the theory; the other 9.975 examples did no violated any clauses. Running the algorithm on the reduced dataset containing only these 25 examples took 4 minutes and produced the same final theory.

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.

$$same\_rank(2, R) \leftarrow card(R, A) \wedge card(R, B) \wedge A \neq B$$

We modified our language bias to use these new predicates instead of the *card* atom. Running the algorithm with this setting took 80 seconds and produced 53 rules. In this case, only 11 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.

Classification and completion of partial interpretations We also ran some experiments to see whether the classification and imputation results from [8] also 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 where predicted for each example. For example, when given the partial interpretation

$$[card(2, clubs), card(9, diamonds), card(9, spades), card(2, hearts), class(full\_house)]$$

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.

5 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.

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) (November 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 k-CNF expressions. (submitted)
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. Technical report, Katholieke Universiteit Leuven (2000)