Probabilistic Logic-based Process Mining

Elena Bellodi, Fabrizio Riguzzi, and Evelina Lamma

ENDIF – Università di Ferrara – Via Saragat, 1 – 44122 Ferrara, Italy.
{elena.bellodi,evelina.lamma,fabrizio.riguzzi}@unife.it

Abstract. The management of business processes has recently received much attention, since it can support significant efficiency improvements in organizations. One of the most interesting problems is the description of a process model in a language, also equipped with an operational support, that allows checking the compliance of a process execution (trace) to the model. Another problem of interest is the induction of these models from data. In this paper, we present a logic-based approach for the induction of process models that are expressed by means of a probabilistic logic. The approach first uses the DPML algorithm to extract a set of integrity constraints from a collection of traces. Then, the learned constraints are translated into Markov Logic formulas and the weights for each formula are tuned using the Alchemy system. The resulting theory allows to perform probabilistic classification of traces. We tested the proposed approach on a real database of university students' careers. The experiments show that the combination of DPML and Alchemy achieves better results than DPML alone.

Keywords: Business Process Management, Process Mining, Declarative Process Models, Statistical Relational Learning

1 Introduction

Organizations usually rely on a number of processes to achieve their mission. These processes are typically complex and involve a large number of people. The performance of the organization critically depends on the quality and accuracy of its processes. Formal ways of representing business processes have been studied in the area of Business Processes Management (see e.g. [13]).

Recently, the problem of automatically mining a structured description of a business process directly from real data has been studied by many authors (see e.g. [5,1,14]). The data in this case consist of execution traces (or histories) of the process and their collection is performed by information systems which log the activities performed by the users. This problem has been called Process Mining or Workflow Mining.

Most work in the field of Process Mining has been devoted to inducing models in the form of graphs or Petri nets [4]. Recently, however, new modeling languages have started to appear that are *declarative*, in the sense that they express only constraints on process execution rather than encoding them as paths in a graph.

DecSerFlow [3], ConDec [2] and $\mathcal{S}$CIFF [7,6] are examples of such languages. In particular, $\mathcal{S}$CIFF adopts first-order logic in order to represent the constraints. The works [16,15,8] presented approaches for learning models in these languages.

Starting from them, in this paper we investigate the adoption of a logic-based language for representing a process model that is able to encode probabilistic information. In fact, the complexity and uncertainty of real world domains require both the use of first-order logic and the use of probability. Recently, various languages have been proposed in the field of Statistical Relational Learning that combine the two. One of these is Markov Logic [19,12], that extends first-order logic by attaching weights to formulas.

We propose to represent process models by means of Markov Logic. Moreover, we present a logic-based approach for inducing these descriptions that involves first learning a logical theory with DPML [16] and then attaching weights to the formulas by means of the Alchemy system [19].

The effectiveness of the approach is illustrated by considering **as a process** the careers of real students at the University of Ferrara. The experiment showed that the combined use of DPML and Alchemy for Process Mining outperforms the use of DPML only.

The paper is organized as follows: we first discuss how we represent execution traces and process models with logic programming. Then we present the learning technique we have adopted for performing Process Mining. After having evaluated the proposed approach on a real world dataset, we discuss related works and conclude.

2 Process Mining

A trace t is a sequence of events. Each event is described by a number of attributes. The only requirement is that one of the attributes describes the event type. Other attributes may be the executor of the event or event specific information.

An example of a trace is

$\langle a, b, c \rangle$

where a, b and c are events.

We define ** a *process model* PM a formula in a language, for which an interpreter exists that, when applied to a model PM and a trace t, returns answer yes if the trace is compliant with the description **(namely the formula is true in the trace) and false otherwise.

A bag of process traces L is called a *log*. The aim of Process Mining is to infer a process model from a log. Usually, in Process Mining, only compliant traces are used as input to the learning algorithm, see e.g. [5,1,14]. We consider instead the case where we are given both compliant and non compliant traces, since both are relevant for the case under study.

2.1 Representing Process Traces and Models with Logic

A process trace can be represented as a logical interpretation (set of ground atoms): each event is modeled with an atom whose predicate is the event type and whose arguments store the attributes of the event. Moreover, the atom contains an extra argument indicating the position in the sequence. For example, the trace:

$\langle a, b, c \rangle$

can be represented with the interpretation

$\{a(1), b(2), c(3)\}$.

Besides the trace, we may have some general knowledge that is valid for all traces. This information will be called *background knowledge* and we assume that it can be represented as a normal logic program B[1]. **By using a background knowledge we are able to encode each trace parsimoniously, by storing only once the rules that are not specific to a single trace but are true for every trace. For example, background knowledge contains clauses which define precedence and succession relations involving the argument 'position' of the atom representing an event.**

Rather than simply t, we **therefore** consider $M(B \cup t)$, the model of the program $B \cup t$ according to Clark's completion [10].

The process language we consider is a subset of the $\mathcal{S}$CIFF language, originally defined in [6,7], for specifying and verifying interaction in open agent societies.

A process model in our language is a set of Integrity Constraints (ICs). An IC, C, is a logical formula of the form

$$\begin{aligned} Body \rightarrow \exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \\ \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m) \end{aligned} \tag{1}$$

where $Body$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ are conjunctions of literals built over event atoms or over predicates defined in the background knowledge.

We will use $Body(C)$ to indicate $Body$ and $Head(C)$ to indicate the formula $\exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m)$ and call them respectively the *body* and the *head* of C. We will use $HeadSet(C)$ to indicate the set $\{ConjP_1, \ldots, ConjP_n, ConjN_1, \ldots, ConjN_m\}$. **(spostato avanti)The quantifiers in the head apply to all the variables not appearing in the body. The variables of the body are implicitly universally quantified with scope the entire formula.**

$Body(C)$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ will be sometimes interpreted as sets of literals, the intended meaning will be clear from the context. All the formulas $ConjP_j$ in $Head(C)$ will be called P *disjuncts*; all the formulas $ConjN_j$ in $Head(C)$ will be called N *disjuncts*.

[1] A normal logic program is a program containing clauses of the form $H \leftarrow B_1, \ldots, B_n$ where H is an atom and the B_is are literals, i.e., atoms or negations of atoms

An example of an IC is

$$\begin{aligned} & order(bob, camera, T), T < 10 \\ \rightarrow & \exists T1(ship(alice, camera, T1)), \\ & bill(alice, bob, 100, T1), T < T1 \\ & \vee \\ & \forall T1, V \neg bill(alice, bob, V, T1), T < T1 \end{aligned} \tag{2}$$

The meaning of the IC (2) is the following: if *bob* has ordered a camera at a time $T < 10$, then *alice* must *ship* it and *bill bob* 100$ at a time $T1$ later than T or *alice* must *not bill bob* any expense at a time $T1$ later than T.

An IC C is true in an interpretation $M(B \cup t)$, written $M(B \cup t) \models C$, if, for every substitution θ for which $Body(C)$ is true in $M(B \cup t)$, there exists a disjunct $\exists(ConjP_i)$ or $\forall\neg(ConjN_j)$ in $Head(C)$ that is true in $M(B \cup t)$. If $M(B \cup t) \models C$ we say that the trace t is *compliant* with C. A process model H is true in an interpretation $M(B \cup t)$ if every IC of H is true in it and we write $M(B \cup t) \models H$. We also say that trace t is *compliant* with H.

Similarly to what has been observed in [18] for disjunctive clauses, the truth of an IC in an interpretation $M(B \cup t)$ can be tested by running the query:

$? - Body, not(ConjP_1), \ldots not(ConjP_n),$
$ConjN_1, \ldots, ConjN_m$

against a Prolog database containing the clauses of B and the atoms of t as facts. Here we assume that B is *range-restricted*, i.e., that all the variables that appear in the head of clauses also appear **in positive literals of **the body. If this holds, every answer to a query Q against $B \cup t$ completely instantiate Q, i.e., it produces an element of $M(B \cup t)$.

If the N disjuncts in the head share some variables, then the following query must be issued

$? - Body, not(ConjP_1), \ldots not(ConjP_n),$
$not(not(ConjN_1)), \ldots, not(not(ConjN_m))$

that ensures that the N disjuncts are tested separately without instantiating the variables.

If the query finitely fails, the IC is true in the interpretation. If the query succeeds, the IC is false in the interpretation. Otherwise nothing can be said.

2.2 Learning ICs Theories

In this section, we briefly describe the Declarative Process Model Learner (DPML) algorithm that was proposed in [16].

DPML finds an IC theory solving the learning problem by searching the space of ICs. The space is structured using a generality relation based on the following definition of subsumption.

Definition 1 (Subsumption). *An IC D subsumes an IC C, written $D \geq C$, iff it exists a substitution θ for the variables in the body of D or in the N disjuncts of D such that*

- $Body(D)\theta \subseteq Body(C)$ *and*
- $\forall ConjP(D) \in HeadSet(D),\ \exists ConjP(C) \in HeadSet(C)\ :\ ConjP(C) \subseteq ConjP(D)\theta$ *and*
- $\forall ConjN(D) \in HeadSet(D),\ \exists ConjN(C) \in HeadSet(C)\ :\ ConjN(D)\theta \subseteq ConjN(C)$

If D subsumes C, then D is more general than C. For example, let us consider the following clauses:

$$C = accept(X) \vee refusal(X) \leftarrow invitation(X)$$
$$D = accept(X) \vee refusal(X) \leftarrow true$$
$$E = accept(X) \leftarrow invitation(X)$$

Then C is more general than D and E, while D and E are not comparable.

** The aim of DPML is to discover a set of clauses built through a refinement operator, on the base of the generality relation. In order to define a refinement operator, we must first define the *language bias*.

Language bias consists of a set of IC templates which define the literals that can be added to clauses. In particular, each template specifies:

- a set of literals BS allowed in the body,
- a set of disjuncts HS allowed in the head. For each disjunct, the template specifies:
  - whether it is a P or an N disjunct,
  - the set of literals allowed in the disjunct.

As a consequence language bias prescribes which refinements can be realized, so that a finite number of those have to be considered, performing the search in the space of ICs from specific to general. Given an IC D, the finite set of refinements $\rho(D)$ of D is a set of ICs that are more general than D. The set of refinements $\rho(D)$ of D is obtained by performing one of the following operations:

- adding a literal from the IC template for D to the body;
- adding a disjunct from the IC template for D to the head;
- adding a literal to an N disjunct in the head;
- removing a literal from a P disjunct in the head.

Given a language bias which prescribes that the body literals must be chosen among $\{invitation(X), test(X)\}$ and that the head disjuncts must be chosen among $\{accept(X), refusal(X)\}$, an example of refinements sequence is:

$$false \leftarrow true$$
$$accept(X) \leftarrow true$$
$$accept(X) \leftarrow invitation(X)$$
$$accept(X) \vee refusal(X) \leftarrow invitation(X)$$

**

The DPML algorithm solves the following learning problem:
Given

```
function DPML(I+, I−, B)
initialize H := ∅
do
    C := FindBestIC(I+, I−, B)
    if C ≠ ∅ then
        add C to H
        remove from I− all interpretations
            that are false for C
while C ≠ ∅ and I− is not empty
return H

function FindBestIC(I+, I−, B)
initialize Beam := {false ← true}
initialize BestIC := ∅
while Beam is not empty do
    initialize NewBeam := ∅
    for each IC C in Beam do
        for each refinement Ref of C do
            if Ref is better than
                BestIC then BestIC := Ref
            if Ref is not to be pruned then
                add Ref to NewBeam
                if size of NewBeam > MaxBS
                    then remove worst clause
                        from NewBeam
    Beam := NewBeam
return BestIC
```

Fig. 1. DPML learning algorithm

- a space of possible process models $\mathcal{H}$
- a set I^+ of positive traces;
- a set I^- of negative traces;
- a definite clause background theory B.

Find: a process model $H \in \mathcal{H}$ such that

- for all $i^+ \in I^+$, $M(B \cup i^+) \models H$;
- for all $i^- \in I^-$, $M(B \cup i^-) \not\models H$;

If $M(B \cup i) \models C$ we say that IC C *covers* the trace i and if $M(B \cup i) \not\models C$ we say that C *rules out* the trace i.

Every IC in the learned theory is seen as a clause that must be true in all the positive traces (compliant traces) and false in some negative traces (non compliant traces). The theory composed of all the ICs must be such that all the ICs are true when considering a compliant trace and at least one IC is false when considering a non compliant one.

The DPML algorithm is an adaptation of ICL [11] and consists of two nested loops: a covering loop (function DPML in Figure 1) and a generalization loop (function FindBestIC in Figure 1). In the covering loop negative traces are progressively ruled out and removed from the set I^-. At each iteration of the loop a new IC C is added to the theory. Each IC rules out some negative interpretations. The loop ends when I^- is empty or when no IC is found.

The IC to be added in every iteration of the covering loop is returned by function FindBestIC. It looks for an IC by using beam search with $p(\ominus|\overline{C})$ as a heuristic function. The search starts from the IC $false \leftarrow true$ that is the most specific and rules out all the negative traces but also all the positive traces. ICs in the beam are gradually generalized by using the refinement operator. $MaxBeamSize$ is a user-defined constant storing the maximum size of the beam.

At the end of the refinement cycle, the best IC found so far is returned.

2.3 Probabilistic Integrity Constraints

Markov Logic (ML) [19] is a language that extends first-order logic by attaching weights to formulas. Semantically, weighted formulas are viewed as templates for constructing Markov networks. In the infinite-weight limit, ML reduces to standard first-order logic.

Definition 2 (Markov logic network). *A Markov logic network (MLN) L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_m\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each atom appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*
2. *$M_{L,C}$ contains one feature (real-valued function) for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature associated to F_i is w_i.*

For example, an MLN containing the formula $\forall x Smokes(x) \rightarrow Cancer(x)$ (smoking causes cancer) applied to the set of constants $C = \{Anna, Bob\}$ yields the features Smokes(Anna) →Cancer(Anna) and Smokes(Bob) →Cancer(Bob), and a ground Markov network with 4 nodes (Smokes(Anna), Cancer(Anna), Smokes(Bob), Cancer(Bob)).

A possible world $\mathbf{x}$ is an assignment of truth values to every ground atom. The probability distribution specified by the ground Markov network $M_{L,C}$ over possible worlds $\mathbf{x}$ is given by

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(\mathbf{x})\right) \tag{3}$$

where F is the number of formulas in the MLN, $n_i(\mathbf{x})$ is the number of true groundings of F_i in $\mathbf{x}$, Z is a partition function given by $\sum_{\mathbf{x}} \exp\left(\sum_{i=1}^{F} w_i n_i(\mathbf{x})\right)$ that ensures that $P(\mathbf{x})$ sums to one.

A set of ICs can be seen as a "hard" first-order theory that constrains the set of possible worlds: if a world violates even one formula, it is considered impossible. The basic idea in Markov Logic is to soften these constraints: when a world violates one of them it is just less probable, but not impossible. The weight associated to each formula reflects how strong the constraint is: the higher the weight, the greater the difference in probability between a world that satisfies the formula and one that does not, other things being equal.

Once an IC theory has been learned from data, integrity constraints are transformed into ML formulas and weights are learned for them using the discriminative weight learning algorithm of [12] that is implemented in the Alchemy system[2].

Each IC of the form (1) is translated into the following ML formula:

$$\begin{aligned} &Body \wedge \neg(ConjP_1) \wedge \ldots \wedge \neg(ConjP_n) \\ &\wedge(ConjN_1) \wedge \ldots \wedge (ConjN_m) \rightarrow neg \end{aligned} \tag{4}$$

where *neg* means that the trace is negative. In absence of disjuncts in the head, the IC $Body \rightarrow false$ reduces to $Body \rightarrow neg$. The head of all the formulas always contains only the atom *neg*, while all disjuncts in the head are moved to the body.

An example of IC referred to the analyzed domain is:

$$\begin{aligned} &true \\ \rightarrow &\forall A \neg registration(A, year(2005)) \\ &\vee \\ &\forall B, C \neg enrollment2(B, C, repeating(O))). \end{aligned}$$

This IC states that the students who graduated (positive traces) do not present registration in the year 2005 or an enrollment in the second year as an out-of-course student.

The translation into a formula in Markov logic is:

$$\begin{aligned} &registration(A, 2005) \wedge \\ &enrollment2(B, C, oc) \rightarrow neg \end{aligned}$$

The resulting MLN can then be used to infer the probability of *neg* given a database consisting of atoms representing the trace.

3 Review of ROC curves

Research in machine learning has shifted away from simply presenting accuracy results when performing an empirical validation of new algorithms. This is especially true when evaluating algorithms that output probabilities of class values.

[2] http://alchemy.cs.washington.edu/

Provost et al. (1998) have argued that simply using accuracy results can be misleading. They recommended when evaluating binary decision problems to use Receiver Operator Characteristic (ROC) curves, which show how the number of correctly classified positive examples varies with the number of incorrectly classified negative examples. In a binary decision problem, a classifier labels examples as either positive or negative. The decision made by the classifier can be represented in a structure known as a confusion matrix, that has four categories: True positives (TP) are examples correctly labeled as positives; False positives (FP) refer to negative examples incorrectly labeled as positive; True negatives (TN) correspond to negatives correctly labeled as negative; finally, false negatives (FN) refer to positive examples incorrectly labeled as negative. The confusion matrix can be used to construct a point in ROC space: one plots the False Positive Rate (FPR) on the x-axis and the True Positive Rate (TPR) on the y-axis. The FPR measures the fraction of negative examples that are misclassified as positive (FPR=FP/(FP+TN)). The TPR measures the fraction of positive examples that are correctly labeled (TPR=TP/(TP+FN)). Informally, one point in ROC space is better than another if it is to the northwest (tp rate is higher, fp rate is lower, or both) of the first. The point (0; 1) represents perfect classification.

A discrete classifier, which gives only a class decision, i.e., a Yes or No on each instance, produces a single point in ROC space. Some classifiers instead yield an instance *probability* or *score*, a numeric value that represents the degree to which an instance is a member of a class. An important point about ROC graphs is that they measure the ability of a classifier to produce good relative instance scores. A classifier needs not produce accurate, calibrated probability estimates; it needs only produce relative accurate scores that serve to discriminate positive and negative instances.

Starting from the set of test examples, the probabilistic classifier's estimate that each example is positive, the number of positive and negative examples, an efficient generation algorithm of ROC curves has been proposed which, for each positive instance, increments TP, and for every negative instance increments FP. It maintains a stack R of ROC points, pushing a new point onto R after each instance is processed. Finally it graphs the *set* R which contains the points on the curve (and not a single point like a discrete classifier).

ROC curves are typically generated to evaluate the performance of a machine learning algorithm on a given dataset, since a dataset contains a fixed number of positive and negative examples. To evaluate and also to compare different classifiers one must reduce ROC curve to a single scalar value representing expected performance. A common method is to calculate the area under the ROC curve, abbreviated AUC: it may be computed easily using a small modification of the previous algorithm. Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1. Given 2 classifiers and their AUC, the one with the greater area has better average performance.

4 Experiments

Our goal is to demonstrate that the combined use of DPML, for learning an IC theory, and Alchemy, for learning weights for formulas, produces better results than the sharp classification realized by the IC theory alone.

The experiments have been performed over a real dataset regarding university students, where the careers of students that graduated are positive traces and the careers of students who did not finish their studies are negative ones. We want to predict whether a student graduates on the basis of her career. To perform our experiments, we collected 813 careers of students enrolled at the Faculty of Engineering of the University of Ferrara from 2004 to 2009. The traces have been labeled as compliant or non compliant with respect to the classification specified above. There are 327 positive and 486 negative traces.

We first induce an IC theory from these data. Every trace was therefore adapted to the format required by the DPML algorithm, transforming it into an interpretation. We considered the main activities performed by a student together with parameters describing the activities. An example of an interpretation for a student is the following:

$$
\begin{aligned}
&\{registration(par_1, \ldots, par_n, 1),\\
&exam(par_1, \ldots, par_m, 2),\\
&exam(par_1, \ldots, par_m, 3),\\
&\ldots\\
&career\_end(par_1, n)\}
\end{aligned}
$$

where par_i means the i-th parameter for a certain activity. **Each activity has a fixed number of parameters which reflect the corresponding attributes stored in the database used. The complete list of logical predicates used, corresponding to activities, is the following:

- *registration*, which stores some personal and school information about a student, with parameters type, mark and year of high school diploma, town and country of residence, year of registration at university, student's ID;

- *enrollmentN*, with N=1..9, which stores the enrollments to years following the first (of registration), with parameters enrollment year, course year (1,2,3), student's status (out-of-course or not), student's ID;

- *exam*, with parameters course id, mark (number), honours (yes/no) and mark category (low, medium, high);

- *career_end* which stores the career conclusion, with values degree (positive traces) or abandon, not-renewed enrollment, transfer to another faculty, transfer to another University (negative traces).

A ten-fold cross-validation was used, i.e., the dataset was divided into ten sets (containing roughly the same proportion of positive and negative traces as the whole dataset) and ten experiments were performed, where nine sets were used for training and the remaining one for testing, i.e., for evaluating the accuracy of the learned theory. In particular, test sets contain either 33 positive and 49 negative traces or 32 positive and 48 negative traces.

The same language bias was used in all ten experiments,** including two IC templates. The first template prescribed as body literal the *exam* predicate (with only the parameter *honours* specified) and as head disjunct the *registration* predicate (without specifying any parameter) and the *enrollment*1 predicate (without specifying any parameter); the second template prescribed as body literals the *registration* predicate (specifying the year) and the *enrollmentN* predicate with N=1..9 (repeating it with different parameters specified: first no parameters, then enrollment year, course year and student's status (out-of-course or not) separately, then course year and student's status together), and as head disjunct the *registration* predicate (distinguishing town and country of residence separately).**

The accuracy is defined as the number of compliant traces that are correctly classified as compliant by the learned model plus the number of non compliant traces that are correctly classified as not compliant divided by the total number of traces.

Ten different IC theories were learned, composed of a number of rules between 25 and 31. The accuracy of the theories on the test sets ranges from 54% to 86%, with an average of 67.5%. **Examples of ICs that were obtained, in $\mathcal{S}$CIFF language, are:

$$\begin{array}{l} \forall A, B, C, D, E \quad registration(A, town(rimini), B, C, D, E) \\ \rightarrow \\ false. \end{array}$$

which states that students from the town of Rimini didn't graduate,

$$\begin{array}{l} true \\ \rightarrow \\ \forall A, B \neg enrollment6(year(2009), A, B). \end{array}$$

which states that students who enrolled 6 times (year 2009 being the sixth after registration) didn't graduate,

$$\begin{array}{l} \forall F, G, H, I, J \quad registration(F, town(bologna), G, H, I, J) \\ \rightarrow \\ \forall A \neg enrollment1(A, year(2005)). \end{array}$$

which states that students living in Bologna and enrolled the first time in 2005 didn't graduate.**

The second step was the assignment of weights to the ICs, by creating ten MLN containing the theories translated into ML. Each of the ten MLNs were given as input to Alchemy for discriminative weight learning.

Ten MLNs were also generated from the learned IC theories by assigning the pseudo-infinite weight 10^{10} to all the clauses, in order to approximate a purely logical theory.

The corresponding MLNs to the three ICs above are respectively:

$$registration(A, rimini, B, C, D, E) \rightarrow neg$$

$$enrollment6(2009, A, B) \rightarrow neg$$

$$registration(F, bologna, G, H, I, J) \wedge enrollment1(A, 2005) \rightarrow neg$$

The Alchemy system performs also structure learning, able to learn a complete MLN composed of both ML formulas and weigths, but in our experiments it gave problems of memory lack so it could not be completed.

In the third step, we computed the probability of each test trace of being negative. This was performed by running the belief propagation inference algorithm of [21] (implemented in Alchemy) both on the MLNs with learned weights and on the MLNs with pseudo-infinite weights. In practice, we computed the marginal probabilities of the atoms of the form $neg(i)$, with i representing the identifier of a student in the test dataset.

Finally, we compared the sharp MLN with the weighted MLN using the the average area under the ROC curve (AUC) [17] that has been identified as a better measure for evaluating the classification performances of algorithms with respect to accuracy, because it also takes into account the different distribution of positive and negative examples in the datasets. The sharp MLN achieved an average AUC of 0.7107528, while the weighted MLN achieved and average AUC of 0.7227286. We also applied a one-tailed paired t test: the null hypothesis that the two algorithms are equivalent can be rejected with a probability of 90.58%.

5 Related Works

Most works on process mining deal with process models in the form of graphs or Petri nets, that represent the allowed sequences of events as paths in the diagram. [5] proposed an approach for inducing a process representation in the form of a directed graph encoding the precedence relationships.

[4] proposed the α-algorithm that induces Petri nets. The approach discovers binary relations in the log, such as the "follows" relation. The α-algorithm is guaranteed to work for a restricted class of models.

In [14] the result of induction is a process model in the form of a disjunction of special graphs called *workflow schemes*.

Recently, a new approach for the representation of process models has appeared, in which the models are seen as sets of constraints over the executions of the process. These models are called declarative because they state the conditions that process executions must satisfy rather than encoding them as paths in graphs.

Examples of declarative languages for representing process models are DecSerFlow [3], ConDec [2] and $\mathcal{S}$CIFF [7,6]. [9] describes the relationships between these languages and shows that ConDec/DecSerFlow can be translated into $\mathcal{S}$CIFF and a subset of $\mathcal{S}$CIFF can be translated into ConDec/DecSerFlow.

[16] proposed the DPML algorithm that learns process models expressed in a subset of $\mathcal{S}$CIFF. [15,8] presented the DecMiner system that is able to infer ConDec/DecSerFlow models by first inducing a $\mathcal{S}$CIFF theory and then translating it into ConDec/DecSerFlow.

This paper extends the works [16,15,8] by including a probabilistic component in the process models. This allows to better model domains where the relationships among events are uncertain.

Recently, [20] discussed mining of process models in the form of AND/OR workflow graphs that are able to represent probabilistic information: each event is considered as a binary random variable that indicates whether the event happened or not and techniques from the field of Bayesian networks are used to model a probability distribution over events. The paper presents a learning algorithm that induces a model by identifying the probabilistic relationships among the events from data. Thus the approach of [20] provides a probabilistic extension to traditional graph-based models, while we extend declarative modeling languages by relying on a first-order probabilistic language.

6 Conclusions

We propose a methodology, based both on Logic and Statistical Relational Learning, for analyzing a log containing several traces of a process, labeled as compliant or non-compliant. From them we learn a set of declarative constraints expressed as ICs. Then we represent ICs in Markov Logic, a language extending first-order logic, to obtain a probabilistic classification of traces, by using the Alchemy system. Finally we evaluate the performances of the two models concluding that probabilistic ICs are more accurate than the pure logical ones. The experiments have been performed on process traces belonging to a real dataset of university students' careers.

Supplementary material, including the code of the systems and an example dataset, can be found at http://sites.google.com/a/unife.it/ml/pdpm/

7 Acknowledgements

This work was possible thanks to the Audit Office of the University of Ferrara, in particular Alberto Domenicali and Susanna Nanetti, that supplied the university dataset for experiments.

References

1. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
2. van der Aalst, W.M.P., Pesic, M.: A declarative approach for flexible business processes management. In: Business Process Management Workshops, BPM 2006 International Workshops, Vienna, Austria, September 4-7, 2006. LNCS, vol. 4103, pp. 169–180. Springer (2006)
3. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) Proceedings of the Third International Workshop on Web Services and Formal Methods (WS-FM 2006). LNCS, vol. 4184. Springer (2006)
4. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
5. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Proceedings of the 6th International Conference on Extending Database Technology, EDBT'98. LNCS, vol. 1377, pp. 469–483. Springer (1998)
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., P.Torroni: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. 9(4) (2008)
7. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An abductive interpretation for open societies. In: Cappelli, A., Turini, F. (eds.) Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence (AI*IA 2003). LNAI, vol. 2829. Springer Verlag (2003)
8. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. LNCS Transactions on Petri Nets and Other Models of Concurrency, ToPNoC II 5460, 278–295 (2009), `http://www.springerlink.com/content/c4j2k38675588759/`
9. Chesani, F., Mello, P., Montali, M., Storari, S.: Towards a decserflow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, DEIS, Bologna, Italy (2007)
10. Clark, K.L.: Negation as failure. In: Logic and Databases. Plenum Press (1978)
11. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: Proceedings of the 6th Conference on Algorithmic Learning Theory. LNAI, vol. 997. Springer Verlag (1995)
12. Domingos, P., Kok, S., Lowd, D., Poon, H., Richardson, M., Singla, P.: Markov logic. In: Probabilistic Inductive Logic Programming. Lecture Notes in Computer Science, vol. 4911, pp. 92–117. Springer (2008)
13. Georgakopoulos, D., Hornick, M.F., Sheth, A.P.: An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases 3(2), 119–153 (1995)

14. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. 18(8), 1010–1027 (2006)
15. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Proceedings of the 5th International Conference on Business Process Management, BPM 2007. pp. 344–359. No. 4714 in Lecture Notes in Computer Science, Springer, Heidelberg, Germany (2007)
16. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: Proceedings of the 17th International Conference on Inductive Logic Programming, ILP 2007. pp. 132–146. No. 4894 in Lecture Notes in Artificial Intelligence, Springer, Heidelberg, Germany (2008), http://dx.doi.org/10.1007/978-3-540-78469-2_16
17. Provost, F.J., Fawcett, T.: Robust classification for imprecise environments. Machine Learning 42(3), 203–231 (2001)
18. Raedt, L.D., Dehaspe, L.: Clausal discovery. Machine Learning 26(2-3), 99–146 (1997)
19. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
20. Silva, R., Zhang, J., Shanahan, J.G.: Probabilistic workflow mining. In: Grossman, R., Bayardo, R.J., Bennett, K.P. (eds.) Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 275–284. ACM (2005)
21. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008. pp. 1094–1099. AAAI Press (2008)