Machine Learning manuscript No.
(will be inserted by the editor)

ALLPAD: Approximate Learning of Logic Programs with Annotated Disjunctions

Fabrizio Riguzzi

Dipartimento di Ingegneria, Università di Ferrara, Via Saragat 1
44100 Ferrara, Italy, e-mail: fabrizio.riguzzi@unife.it

Abstract Logic Programs with Annotated Disjunctions (LPADs) provide a simple and elegant framework for representing probabilistic knowledge in logic programming. In this paper we consider the problem of learning ground LPADs starting from a set of interpretations annotated with their probability. We present the system ALLPAD for solving this problem. ALLPAD modifies the previous system LLPAD in order to tackle real world learning problems more effectively. This is achieved by looking for an approximate solution rather than a perfect one. A number of experiments have been performed on real and artificial data for evaluating ALLPAD, showing the feasibility of the approach.

1 Introduction

Logic Programs with Annotated Disjunctions (LPADs) (Vennekens, Verbaeten, & Bruynooghe, 2004; Vennekens & Verbaeten, 2003) are a relatively new formalism for representing probabilistic information in logic programming. They have been recognized as one of the simplest and most expressive (Blockeel, 2004) languages that combine logic and probability.

In (Riguzzi, 2004) the definition of a learning problem for LPADs has been proposed together with an algorithm for solving it called LLPAD. However, LLPAD does not work well on non-toy problems because it relies on the exact solution of a large constraint satisfaction problem. On real world problems such a solution may not exist or may be too expensive to find. Therefore in this paper we propose the system ALLPAD (Approximate Learning of Logic Programs with Annotated Disjunctions) that modifies LLPAD in order to be able to learn in real world scenarios by looking for a solution that "approximately" satisfies the learning problem. ALLPAD solves the constraint satisfaction problem in an approximate way by

transforming it into an optimization problem. Moreover, the clause search phase is also modified in order to return only the most significant clauses so that the complexity of the optimization problem is kept inside acceptable bounds.

To show that ALLPAD is able to work in practice, we have applied it to two real and one artificial problems. The real problems are the classification of proteins into SCOP classes and the classification of Unix traces into user classes. The accuracy obtained is compared to that of a naïve Bayes approach and with the results of previous works. The artificial problem consists of learning back an LPAD from data generated from it by random sampling. The log likelihood obtained is compared with that of naïve Bayes and of the original LPAD.

The paper is organized as follows. Section 2 provides some preliminary notions regarding LPADs together with their semantics as given in (Vennekens et al., 2004). In section 3 we introduce the learning problem and we describe LLPAD and ALLPAD. Section 4 presents the experiments. In section 5 we discuss related works and in section 6 we conclude and present directions for future work.

2 Preliminaries

A Logic Program with Annotated Disjunctions P consists of a finite set of formulas of the form

$$(h_1 : p_1) \vee (h_2 : p_2) \vee \ldots \vee (h_n : p_n) \leftarrow b_1, b_2, \ldots b_m$$

called *annotated disjunctive clauses*. In such a clause the h_i are logical atoms, the b_i are logical literals and the p_i are real numbers in the interval $[0, 1]$ such that $\sum_{i=1}^{n} p_i = 1$. If $n = 1$, then the annotation can be omitted. For a clause C of the form above, we define $head(C)$ as the set $\{(h_i : p_i)|1 \leq i \leq n\}$ and $body(C)$ as the set $\{b_i|1 \leq i \leq m\}$. $\mathcal{P}_\mathcal{G}$ is the set of all finite ground LPADs. Let $H_B(P)$ be the Herbrand base of P and let $\mathcal{I}_P$ be the set of all the possible Herbrand interpretations of P (i.e., subsets of $H_B(P)$). If P contains function symbols, then $H_B(P)$ is infinite, otherwise it is finite. If P contains functions symbols and variables then its grounding is infinite, otherwise it is finite.

Example 1 Let us see an example of an LPAD taken from (Blockeel, 2004).

$$\begin{aligned}
&mother(m, c). \quad father(f, c).\\
&cg(m, 1, w). \quad cg(m, 2, w). \quad cg(f, 1, p). \quad cg(f, 2, w).\\
&(cg(X, 1, A) : 0.5) \vee (cg(X, 1, B) : 0.5) \leftarrow\\
&\quad mother(Y, X), cg(Y, 1, A), cg(Y, 2, B).\\
&(cg(X, 2, A) : 0.5) \vee (cg(X, 2, B) : 0.5) \leftarrow\\
&\quad father(Y, X), cg(Y, 1, A), cg(Y, 2, B).
\end{aligned}$$

$$color(X, purple) \leftarrow cg(X, \_N, p).$$
$$color(X, white) \leftarrow cg(X, 1, w), cg(X, 2, w).$$

This program encodes the Mendelian rules of inheritance of the color of pea plants. The color of a pea plant is determined by a gene that exists in two forms (alleles), p and w, that stand for purple and white. Each plant has two alleles for the color gene that reside on a couple of chromosomes. *cg(X,N,A)* indicates that plant X has allele A on chromosome N. The facts of the program express that c is the offspring of f and m and that the alleles of m are ww and of f are pw. The disjunctive rules encode the fact that an offspring inherits the allele on chromosome 1 from the mother and the allele on chromosome 2 from the father. In particular, each allele of the parent has a probability of 50% of being transmitted. The definite clauses for *color* express the fact that the color of a plant is purple if at least one of the alleles is p, i.e., that the p allele is dominant.

2.1 Semantics

The semantics of an LPAD was given in (Vennekens et al., 2004) for finite ground programs, i.e., programs in $\mathcal{P}_\mathcal{G}$. A non-ground program can be assigned a semantics only if its grounding is finite, i.e., if it does not contain function symbols. The semantics is given in this case in terms of its grounding.

Each ground annotated disjunctive clause represents a probabilistic choice between a number of ground non-disjunctive clauses. By choosing a head atom for each ground clause of an LPAD we get a normal logic program called an *instance* of the LPAD. A probability distribution is defined over the space of instances by assuming independence between the choices made for each clause.

An instance is identified by means of a selection function. Let P be a program in $\mathcal{P}_\mathcal{G}$: a *selection* (Vennekens et al., 2004) σ is a function which selects one pair $(h : p)$ from each rule of P, i.e. $\sigma : P \rightarrow (H_B(P) \times [0, 1])$ such that, for each R in P, $\sigma(R) \in head(R)$. For each rule R, we denote the selected atom h by $\sigma_{atom}(R)$ and the selected probability p by $\sigma_{prob}(R)$. Furthermore, we denote the set of all selections σ by $\mathcal{S}_P$.

Let σ be a selection in $\mathcal{S}_P$: the *instance* P_σ *chosen by* σ (Vennekens et al., 2004) is obtained by keeping only the atom selected for R in the head of each rule $R \in P$, i.e. $P_\sigma = \{\text{“}\sigma_{atom}(R) \leftarrow body(R)\text{”} | R \in P\}$.

We now assign a probability to a selection function σ and therefore also to the associated program P_σ. The *probability of a selection* (Vennekens et al., 2004) σ in $\mathcal{S}_P$ is the product of the probabilities of the individual choices made by that selection, i.e. $C_\sigma = \prod_{R \in P} \sigma_{prob}(R)$.

The semantics of the instances of an LPAD can be given by any of the semantics defined for normal logic programs (e.g. Clark's completion,

Fitting semantics, stable models, well founded semantics (Van Gelder, Ross, & Schlipf, 1991)). In (Vennekens et al., 2004) the authors have considered only the well founded semantics, the most skeptical one. Since in LPADs the uncertainty is modeled by means of the annotated disjunctions, the instances of an LPAD should contain no uncertainty, i.e. they should have a single two-valued model. Therefore, given an instance P_σ, we require that its well founded model $WFM(P_\sigma)$ is two-valued.

$P \in \mathcal{P}_\mathcal{G}$ is called *sound* (Vennekens et al., 2004) iff, for each selection σ in $\mathcal{S}_P$, the well founded model $WFM(P_\sigma)$ of the program P_σ chosen by σ is two-valued.

An LPAD P is *locally stratified* iff all of the atoms in $H_B(P)$ can be assigned a countable ordinal *rank* such that, for every rule $(h_1 : p_1) \vee \ldots \vee (h_n : p_n) \leftarrow B$ in the grounding P' of P, for every h_i: (1) the rank of h_i is greater than the rank of every atom that appears negatively in B, and (2) the rank of h_i is greater or equal to the rank of every atom that appears positively in B.

If $P \in \mathcal{P}_\mathcal{G}$ is locally stratified then P is sound (Van Gelder et al., 1991).

We now define the probability of interpretations. Let P be a sound program. For each of its interpretations I in $\mathcal{I}_P$, the *probability* $\pi^*_P(I)$ *assigned by* P *to* I (Vennekens et al., 2004) is the sum of the probabilities of all selections which lead to I, i.e. with $S(I)$ being the set of all the selections σ for which $WFM(P_\sigma) = I$: $\pi^*_P(I) = \sum_{\sigma \in S(I)} C_\sigma$.

Let us now introduce the definition of the probability of a formula. For each formula ϕ, the *probability* $\pi^*_P(\phi)$ *of* ϕ *according to* P (Vennekens et al., 2004) is the sum of the probabilities of all the interpretations in which ϕ holds, i.e. $\pi^*_P(\phi) = \sum_{I \in \mathcal{I}^\phi_P} \pi^*_P(I)$ with $\mathcal{I}^\phi_P = \{I \in \mathcal{I}_P \; I \models \phi\}$.

We can also define the conditional probability of a formula given another one. Given two formulas ϕ and ψ, the *conditional probability of* ϕ *given* ψ *according to* P is indicated by $\pi^*_P(\phi|\psi)$ and is given by $\frac{\pi^*_P(\phi \wedge \psi)}{\pi^*_P(\psi)}$.

2.2 Properties

In (Riguzzi, 2004) a definition and two theorems were given that will be useful in the following. We will report them here, together with a correction: in (Riguzzi, 2004) it was not specified that the theorems hold only for ground clauses and for locally stratified programs.

Definition 1 (Mutually exclusive bodies) *Ground clauses* $H_1 \leftarrow B_1$ *and* $H_2 \leftarrow B_2$ *have* mutually exclusive bodies over a set of interpretations $\mathcal{J}$ *if,* $\forall I \in \mathcal{J}$, B_1 *and* B_2 *are not both true in* I.

Theorem 1 *Consider a locally stratified LPAD* $P \in \mathcal{P}_\mathcal{G}$ *and a clause* $C \in P$ *of the form*

$$C = (h_1 : p_1) \vee (h_2 : p_2) \vee \ldots (h_m : p_m) \leftarrow B.$$

Suppose you are given the function π_P^* over $\mathcal{I}_P$ and suppose that all the clauses of P that share an atom in the head with C have mutually exclusive bodies with C over the set of interpretations $\mathcal{J} = \{I|\pi_P^*(I) > 0\}$. The probabilities p_i are given by the conditional probability of the head atoms given the body:

$$p_i = \pi_P^*(h_i|B)$$

This theorem states that, under certain conditions, the probabilities of the head disjuncts of a ground rule can be computed from the probabilities of the interpretations. In particular, the probability of a disjunct h_i is given by the conditional probability of h_i given the body, i.e. by the sum of the probabilities of interpretations where the body of the clause and h_i are true divided by the sum of the probabilities of interpretations where the body is true.

Theorem 2 *Consider an interpretation I and a locally stratified LPAD $P \in \mathcal{P}_\mathcal{G}$ such that all the couples of clauses of P that share an atom in the head have mutually exclusive bodies with respect to the set of interpretations $\{I\}$. If $S(I) \neq \emptyset$ then all the selections $\sigma \in S(I)$ agree on the clauses of P with body true in I and*

$$\pi_P^*(I) = \prod_{R \in P, I \models body(R)} \sigma_{prob}(R) \tag{1}$$

where σ is any element of $S(I)$. If $S(I) = \emptyset$ then $\pi_P^*(I) = 0$.

This theorem states that, given an interpretation I, under certain conditions, all the selections σ in the set $S(I)$ agree on all the rules in the program with the body true in I and that the probability of I can be computed by multiplying the probabilities of the head disjuncts selected by a $\sigma \in S(I)$ for all the clauses with the body true in I.

Proofs of these theorems can be found in (Riguzzi, 2006).

3 Learning LPADs

We consider a learning problem of the following form (Riguzzi, 2004):
Given:

- a set E of examples that are couples $(I, \pi(I))$ where I is an interpretation, $\pi(I)$ is its associated probability and $\sum_{(I,\pi(I))\in E} \pi(I) = 1$;
- a space of possible LPAD $\mathcal{S}$ (described by a language bias LB)

Find: an LPAD $P \in \mathcal{S}$ such that $\forall (I, \pi(I)) \in E \;\; \pi_P^*(I) = \pi(I)$

Instead of a set of couples $(I, \pi(I))$, the input of the learning problem can be a multiset E' of interpretations. From this case we can obtain a learning problem of the form above by computing a probability for each interpretation in E' using relative frequency.

We now describe LLPAD (Riguzzi, 2004) and then ALLPAD.

3.1 LLPAD

LLPAD learns ground LPADs in four phases: the first consists in finding all the definite clauses that satisfy certain constraints, the second consists in finding all the disjunctive clauses that satisfy certain constraints, the third consists in annotating the head atoms of disjunctive clauses with probabilities and the fourth consists in solving a constraint satisfaction problem.

The first and second phases can be cast in the framework proposed by (Stolle, Karwath, & Raedt, 2005) in which the problem of descriptive ILP is seen as the problem of finding all the clauses in the language bias that satisfy a number of constraints. Exploiting the properties of constraints, the search in the space of clauses can be usefully pruned.

A constraint is *monotonic* if it is the case that when a clause does not satisfy it, none of its generalizations (in the θ-subsumption generalization order) satisfies it. A constraint is *anti-monotonic* if it is the case that when a clause does not satisfy it, none of its specializations satisfies it.

The first phase of LLPAD can be formulated in this way: find all the definite clauses that satisfy the following constraints:

D1 they have their body true in at least one interpretation
D2 they are satisfied in all the interpretations
D3 they are maximally general (there does not exist another clause that is strictly more general and that satisfies D1 and D2)

LLPAD searches the space of definite clauses by performing a complete depth-first and top-down search in the space of bodies for each ground atom a allowed by the language bias in the head of clauses. The search starts with the clause with an empty body ($a \leftarrow$) and exploits constraints D3 and D1: as soon as a body is found such that the clause is satisfied in all interpretations, the clause is returned and the search along that branch is stopped because any specialization will not satisfy constraint D3; as soon as a body that is true in zero interpretations is found, the search along that branch is stopped, because constraint D1 is anti-monotonic.

The second phase can be formulated in this way: find all the disjunctive clauses that satisfy the following constraints:

C1 they have their body true in at least one interpretation
C2 they are satisfied in all the interpretations
C3 their atoms in the head are mutually exclusive over the set of interpretations where the body is true (i.e. no two head atoms are both true in an interpretation where the body is true)
C4 they have no redundant head atom, i.e. no head atom that is false in all the interpretations where the body is true

LLPAD searches the space of disjunctive clauses by first searching depth-first and top-down for bodies true in at least one interpretation and then, for each such body, searching for a head satisfying the remaining constraints. When a body is found that is true in zero interpretations, the search along that branch is stopped (constraint C1 is anti-monotonic).

The system searches the clause space in a complete way and employs bottom-up search in the space of heads, exploiting the monotonic constraint C2 that requires the clause to be true in all the interpretations for pruning the search.

Definite clauses are searched separately because if we allow the search in the space of heads in the second phase to reach single atom heads we would return also non maximally general definite clauses. In fact, given a body B, an atom a and a literal b, if $a \leftarrow B$ satisfies constraints C1-4 and B, b is true in at least one interpretation, then both the clauses $a \leftarrow B$ and $a \leftarrow B, b$ will be returned, thus violating constraint D3.

The third phase is performed by using theorem 1: given a ground clause generated by the second phase, the probabilities of head atoms are given by the conditional probability of the head atoms given the body according to the distribution π.

In the fourth phase, LLPAD partitions the set $\mathcal{D}$ of all the disjunctive clauses that have been found in subsets that are solutions of the learning problem. This is done by assigning to each clause $C_i \in \mathcal{D}$ a variable x_i that is 0 if the clause is absent from a solution P and is 1 if the clause is present. The system must ensure that the couples of clauses that share an atom in the head have mutually exclusive bodies over the set of interpretations E. This is achieved by testing, for each couple of clauses, if they share an atom in the head and, if so, if the intersection of the two sets of interpretations where their body is true is non-empty. LLPAD asserts the constraint $x_i + x_j \leq 1$ for all such couples of clauses (C_i, C_j). Finally the solution must assign the correct probability to each interpretation. For each I such that $(I, \pi(I)) \in E$, the system asserts the constraint:

$$\prod_{C_i \in \mathcal{D}, I \models body(C_i)} p_i^{x_i} = \pi(I) \tag{2}$$

where p_i is the probability of the single head of C_i that is true in I.

In fact, if an assignment of the x_is is found that satisfies the mutual exclusion constraints and the interpretation constraints (2), then the left hand side of (2) for an interpretation I becomes $\prod_{C_i \in \mathcal{D}, x_i=1, I \models body(C_i)} p_i$ and thus theorem 2 tells us that the probability $\pi(I)$ of every interpretation I will be equal to the one assigned to it by the set of selected clauses. This result was proved to hold (Riguzzi, 2006) in the case in which the language bias allows only acyclic LPADs.

Definite clauses are not considered in the constraints because they would contribute only with a factor 1^{x_j} that has no effect on the constraint for any value of x_j. Therefore, for each assignment of the other variables, x_j can be either 0 or 1. For this reason, all the found definite clauses are included in every solution and, therefore, we need only the most general definite clauses.

If we take the logarithm of both members we get the following linear constraint:

$$\sum_{C_i \in \mathcal{D}, I \models body(D)} x_i \log p_i = \log \pi(I) \tag{3}$$

We can thus find the solutions of the learning problem by solving the above linear constraint satisfaction problem.

3.2 ALLPAD

ALLPAD learns ground LPADs in five phases. The first and the third are the same as those of LLPAD. The second and the fourth modify those of LLPAD and the fifth one is new.

Let us first consider the fourth phase: in it ALLPAD solves an optimization problem rather than a constraint satisfaction problem. In fact, with real world problems, a perfect solution of the learning problem may not exist in the solution space, because the language bias is too restricted or because the data available is noisy. In these cases the constraint phase returns a failure. In order to solve this problem, we may enlarge the language bias, but this can lead to unacceptable run times, or look for an approximate solution. Thus the constraint problem is transformed into an optimization problem where the system tries to satisfy the interpretation constraints as much as possible. i.e., it tries to minimize the absolute value of the difference between the left and right members of the interpretation constraints (3).

However the absolute value function is not linear, therefore we have to transform the cost function so that we can use linear programming techniques. To this purpose, we introduce a "slack variable" named max and two "slack variables" s_I^+ and s_I^- for each interpretation I. These variables are real. Then each interpretation constraint in the form of equation (3) is replaced with two constraints of the form

$$\frac{\sum_{C_i \in \mathcal{D}, I \models body(C_i)} x_i \log p_i}{\log \pi(I)} - 1 \leq s_I^+$$

$$1 - \frac{\sum_{C_i \in \mathcal{D}, I \models body(C_i)} x_i \log p_i}{\log \pi(I)} \leq s_I^-$$

Moreover the slack variables must satisfy the following constraints for every I

$$s_I^+ \geq 0 \qquad s_I^+ \leq max \qquad s_I^- \geq 0 \qquad s_I^- \leq max$$

The cost function to be minimized can now be expresses as

$$\alpha \times max + (1 - \alpha) \times \frac{\sum_{(I,\pi(I)) \in E} (s_I^+ + s_I^-)}{|E|}$$

where α is a parameter between 0 and 1. This function provides a trade-off between the objective of minimizing the average error and the objective of minimizing the maximum error. The user can modify the α parameter depending on her needs: if she is going to use the learned model to evaluate the probability of a small number of interpretations, she may prefer to have

a high α, so that single large errors are avoided; if she is going to use the model to evaluate the probability of a large number of interpretations, she may prefer to have a low α, because single large errors have a lower influence on the overall performance. In the experiments of Section 4 α was chosen to be 0.5 in order to take a neutral position between these two competing requirements.

Since now both the constraints and the cost function are linear, we can use mixed-integer programming (MIP) techniques. If no perfect solution exist, a non zero optimum will be found.

However, the optimization problem, as the constraint problem, is NP-hard and thus solvable only for small instances. To overcome this problem, we exploit the possibility of setting a time limit offered by many MIP packages: at the deadline, the best admissible solution found up to that point is returned. An admissible solution in this case is one that respects all the mutual exclusion constraints. If no admissible solution has been found, the package returns an error. In this way, ALLPAD looks for the best solution given the available time.

To make sure that an admissible solution will be found within the time limits, one can reduce the dimension of the problem by reducing the number of clauses found in the second phase. Thus one may prefer the clauses that apply to examples with a high probability, because they will give a large contribution to the probability of interpretations. For this reason the complete search in the space of bodies performed by LLPAD is given up for an incomplete search strategy, beam search. The heuristic to be used for ranking bodies is the sum of the probabilities of the interpretations where the body is true. This heuristic ensures that the clauses that apply only to a small number of improbable interpretations are discarded and the dimension of the optimization problem is reduced.

Moreover, in ALLPAD it is possible to set a limit on the number of nodes explored in the search for bodies, in order to further limit the number of generated clauses.

As regards the search in the space of heads, the system can employ either bottom-up search or top-down search at the user choice. When it searches bottom-up it exploits the monotonic constraint C2 that requires the clause to be true in all the interpretations for pruning the search. When it searches top-down it exploits the anti-monotonic constraint C3 that requires head atoms to be mutually exclusive. This second possibility was proposed in (Stolle et al., 2005) where the authors present the system *Classic'cl*.

ALLPAD can also employ a third modality for finding heads that does not require search. If the head atoms in a clause template in the language bias are mutually exclusive by construction, then a clause can be found by starting with a head containing all the possible atoms. Then, if the clause is satisfied in all interpretations, ALLPAD removes the head atoms that are not true in at least one interpretation where the body is true and it returns the clause. This modality has been employed in all the experiments presented in section 4.

Fig. 1 Function SecondPhase

```
function SecondPhase(
    inputs : E : set of couples (I, π(I)),
        LB : a language bias expressed as a set of clause templates,
        d : dimension of the beam,
        n : maximum number of explored nodes,
    returns : C : a set of disjunctive clauses)

C := ∅
for each clause template T in LB
    let ρ_T be the downward refinement operator for
        bodies relative to clause template T
    Beam := [(true, 1)]
        /* Beam is a list of couples ordered on the second argument */
    i := 0
    while Beam ≠ ∅ and i < n
        i := i + 1
        remove the first couple (Body, P) from Beam
        if Body is true in at least one interpretation (i.e. P > 0) then
            C := C∪SearchHeads(E, Body, T)
            Ref:= ρ_T(Body)
            for each body R ∈Ref
                let ER be the set of couples (I, Pr(I)) of E such that
                    R is true in I
                PR := Σ_{(I,π(I))∈ER} π(I)
                insert (R, PR) in Beam in descending order of PR
            discard the elements of Beam after the d-th
return C
```

The algorithm for the second phase in pseudo-code is given in Figure 1. In it the function SearchHeads performs the search in the space of heads and returns the set of clauses satisfying the constraints.

When searching the space of bodies, the refinement operator is downward while when searching the space of heads the refinement operator is either upward or downward depending on user choice. In both cases the refinement operator can be optimal or non-optimal: the user can choose an optimal operator to further reduce the number of returned clauses.

Since the learned clauses are ground, the refinement operator used by ALLPAD simply adds (in the case of a downward operator) or removes (in the case of an upward operator) literals from the clauses.

ALLPAD can employ two types of language bias: either DLAB (Dehaspe & Raedt, 1996) or a simpler language bias that consists of a set of couples (H, B) where H is the set of ground atoms allowed in the head of clauses and B is the set of ground literals allowed in the body of clauses with heads in H. More details on ALLPAD language biases and refinement operators can be found in (Riguzzi, 2006).

In the fifth phase, the definite clauses not mutually exclusive with the selected disjunctive clauses are removed. To this purpose, for each definite clause, ALLPAD looks for the disjunctive clauses that share an atom in the head with the definite clause and checks if the bodies are both true in one of the input interpretations. If this is true, the definite clause is removed. In this way, in the output program, all clauses sharing an atom in the head have mutually exclusive bodies.

ALLPAD was shown to be correct (Theorem 6 in (Riguzzi, 2006)) when a minimum of 0 is reached in the optimization phase and when the language bias allows only acyclic programs (Riguzzi, 2006). ALLPAD is not complete because it performs beam search in the space of bodies and because it may not find the optimum of the optimization problem if a time limit is set.

4 Experiments

In this section we report on the experiments performed for evaluating ALLPAD.

All the experiments have been performed on a PC with an Athlon XP 2600+ processor at 2138 MHz, 1GB of RAM and Windows 2000. For solving the MIP problem we used Xpress-Optimizer by Dash Optimization. This tool allows the user to set a time limit to the optimization.

4.1 Protein Tertiary Structure

ALLPAD was applied to the problem of predicting the tertiary structure of proteins by classifying them into one of the SCOP classes (Turcotte, Muggleton, & Sternberg, 2001). Each protein is described by a sequence of secondary structure elements. The sequence is represented in first order logic as an interpretation where each atom is either of the form *he(Type, Length, Position)* or of the form *st(Orientation, Length, Position)*. The last argument is an ordinal number indicating the position in the sequence. The first atom form indicates that the element is an helix and specifies its type and length. The possible types are *h(left, alpha)*, *h(right, alpha)*, *h(left, gamma)*, *h(right, gamma)*, *h(left, omega)*, *h(right, omega)*, *h(right, pi)*, *h(right, f3to10)*, *27ribbon* and *polyproline*. The second atom form indicates that the element is a strand and specifies its orientation and length. The possible orientations are *null* (the beginning of a strand), *plus* (a parallel strand of a sheet) or *minus* (an anti-parallel strand of a sheet). The length of helices and strands is defined as the number of amino acids they are composed of and was discretized by diving the range into three intervals of equal length.

The dataset available (Kersting, Raiko, Kramer, & Raedt, 2003) (kindly provided by Kristian Kersting) regards the prediction of domains at the second level of the SCOP hierarchy, namely the level of folds. In particular, the data regards the alpha beta protein class (a/b) and, in this class, the

five most populated subclasses (i.e. folds) are considered: TIM beta/alpha-barrel, NAD(P)-binding Rossmann-fold domains, Ribosomal protein L4, glucosamine 5-phosphate deaminase/isomerase and leucine aminopeptidas. The folds will be respectively indicated in the following with the names fold1, fold2, fold23, fold37 and fold55.

The dataset available has 721 examples for class fold1, 360 for fold2, 274 for fold 23, 441 for fold37 and 290 for fold55. In the dataset, only two helix types are actually present, namely *h(right, alpha)* and *h(right, f3to10)*.

ALLPAD can be used for classification by learning a model for each class using only the examples for that class and by assigning a test case to the class whose model gives it the highest probability.

In order to learn an LPAD that describes a class, each interpretation given as input to the system is annotated with the same probability, given by one over the total number of interpretations in the training set, since no interpretation appears more than once.

Proteins are modeled with LPADs as stochastic processes: we want to predict the structure at position p on the basis of the structures in the k previous positions. To this purpose, ALLPAD learns programs containing rules having all the possible structures with position equal to p in the head and a conjunction of structures in the body with positions belonging to the set $S(p,k) = \{p-1, p-2, \ldots, p-k\}$ for a given k. Obviously we cannot have two different structures for the same position $t \in S(p,k)$ in the body since in that case the body would be false in every interpretation. An example of such a rule is:

```
helix(h(right,alpha),long,7):0.571 ;
helix(h(right,f3to10),short,7):0.429 :-
  helix(h(right,alpha),long,5),
  strand(null,medium,4).
```

Therefore we give ALLPAD a language bias expressed using DLAB that contains a rule template for each position p from 1 to the maximum length of the protein sequences in the training set. Each rule template allows each possible structure in the head at position p and each possible structure in the body for positions $p-1$, $p-2$ up to position $p-k$, if they are larger or equal to 1.

Since the optimization phase finds only an approximate solution, we must use an approximate procedure for testing the theories learned. When the test case is a model of an instance of the program, its probability can be computed in the following way: all the disjunctive rules whose bodies are true in the interpretation are identified and, for each such rule, the single head that is true in the interpretation is identified. The probability of the case is obtained by multiplying the probabilities associated to all the head atoms identified in this way. For each position, there is a single rule whose body is true in the interpretation because the rules have mutually exclusive bodies.

Table 1 Results of the SCOP experiments.

Experiment	Av. acc.	St. dev.	Significance	Av. def. rules
Naive Bayes	82.79%	0.03087	-	20.09
First	85.14%	0.01990	98.3%	16.83
Second	85.67%	0.02394	98.6%	16.64

When the test case is not a model of an instance of the program, the probability assigned to the case is 0. This happens for example when for one or more positions there is no rule with the body true in the interpretation. Since rules may be missing because the solution is approximate, in the testing procedure we adopt the following approach: if for a position no rule is applicable, the probability for the structure in that position is given by the conditional probability of the structure given the position and the class. This corresponds to using a rule with an empty body (i.e. a default rule).

The accuracy of the learned LPADs is compared with the accuracy of a naïve Bayes classifier obtained by applying the approximate testing procedure so that the conditional probability given the position and the class is used for all positions.

Two experiments were performed using 10-fold cross validation. The time limit for the MIP optimization has been set to 1 hour for each class in the first experiment and to 100 minutes for each class in the second experiment.

The other important parameters are: the value of k (the number of previous positions to consider), set to 4; the size of the beam, set to 100, and the maximum number of bodies to be refined for each clause template, set to 100 in the first experiment and to 125 in the second experiment for all classes and cross-validation folds apart from two folds for class fold1 and one fold for class fold37 where it was set to 115 because in 100 minutes no admissible solution was found for the MIP problem.

The results obtained are summarized in Table 1. LLPAD has been tested as well on the dataset but the constraint satisfaction problem led to an insufficient memory error in all cases.

A cross-validated paired two-tailed t test has been performed for comparing the accuracy of ALLPAD with that of naïve Bayes. The null hypothesis of equivalence can be rejected with the probability indicated in the Significance column in Table 1. The last column of the table shows the average number of default rules used in testing. In the case of naive Bayes, this is the average length of the sequences. This means that, on average, 3.26 learned rules were used in the testing phase of the first experiment and 3.45 learned rules were used in the testing phase of the second experiment. The accuracy improvement between the first and second experiment, even if it is significant only with probability 65.9%, shows that the results can be improved by employing more computation time.

As regards the execution times, they were dominated by the optimization problem: the other phases only took a few minutes per fold.

The first work applying ILP to the problem was (Turcotte et al., 2001) where the authors obtain an average accuracy of 78.28% using Progol. However, their dataseet is very different because they consider other classes besides the alpha beta class. (Kersting et al., 2003; Kersting & Gärtner, 2004; Kersting, Raedt, & Raiko, 2006; Gutmann & Kersting, 2006) use a dataset similar to the one used here. (Kersting et al., 2003) and (Kersting et al., 2006) use logical hidden Markov models and achieve respectively accuracies of 74% and 76%. (Kersting & Gärtner, 2004) uses Fisher kernels for logical sequences and achieves an accuracy of 83.6%. Finally (Gutmann & Kersting, 2006) uses conditional random fields for logical sequences and achieves an accuracy of 92.96%. The results of ALLPAD compare favourably with all the previous results apart from the last one, even if the system is not specifically tailored to learning sequences, as all previous systems are apart from (Turcotte et al., 2001).

4.2 Unix Traces

ALLPAD has been applied to the problem of identifying the class of a user from a trace of Unix commands. The data set considered was collected by Greenberg (Greenberg, 1988) and contains the traces of 168 users divided into four classes: novice programmer, experienced programmer, computer scientist and non programmer. For each user the data records the commands entered at the *csh* shell on a number of different sessions: overall 303,628 commands were collected divided into 110 sessions on average per user. Each session constitutes a trace.

We used the same settings used in (Kersting & Raiko, 2005): we choose one user for class novice (user novice-1, NV from now on) and one user for class non programmer (user non-programmer-4, NP from now on). 62 and 159 traces were collected respectively for users NV and NP. From these we removed the traces containing non printable characters giving respectively 53 and 133 traces.

For each command the dataset records the line entered by the user, the current working directory, the alias expansion of the command (if any), whether the line had a history expansion in it and the eventual error code returned by the shell.

Of these data we used only the line, the current directory and the alias. We represent each command with two facts: *command(Command, Position*) and *directory(Directory, Position). Command* is the Unix command effectively executed by the user: if there is an alias, it is obtained from the alias expansion by removing the arguments, otherwise it is obtained from the line by removing the arguments. *Directory* is the name of the last folder in the directory path. *Position* is the position of the command in the trace. Class NV has 117 different commands and 15 different folders, while class NP has 124 different commands and 72 different folders.

The approach for applying ALLPAD to this problem is similar to the one used for protein tertiary structure: each trace is stored in a different interpretation with probability given by relative frequency in the dataset and traces are modeled with LPADs as stochastic processes, where the command and the directory at the current position depend on the commands and directories at previous positions. In particular, the directory at position p depends on the commands at positions $\{p-1,\ldots,p-k\}$ and on the directories at positions $\{p-1,\ldots,p-k\}$. The command at position p depends on the directories at positions $\{p,\ldots,p-k+1\}$ and on the commands at positions $\{p-1,\ldots,p-k\}$. In the experiments k was set to 2.

Thus the language bias admits two kinds of templates, one for the predicate *command* and one for the predicate *directory*. The templates for *directory* allow, in the head, all possible directories for position p and, in the body, all possible commands with positions $\{p-1,p-2\}$ and all possible directories with positions $\{p-1,p-2\}$. The templates for *command* allow, in the head, all possible commands for position p and, in the body, all possible commands with positions $\{p-1,p-2\}$ and all possible directories with positions $\{p,p-1\}$.

Five runs were executed. For each run 35 traces were randomly selected for each class and used for training while the rest was used for testing. The theories learned were tested using the approximate procedure described in section 4.1. The probabilities computed for each trace were multiplied by the class prior as in (Kersting & Raiko, 2005) in order to take into account the skew present in the data. Besides ALLPAD, also a naïve Bayes approach was applied, as in section 4.1.

The time limit of the MIP optimization was set to 1 hour for each class, the size of the beam was set to 20 and the maximum number of bodies to be refined for each clause template was set to 100.

The average accuracy, precision and recall of ALLPAD and naïve Bayes are reported in Table 2 together with the results obtained with SAGEM (Kersting & Raiko, 2005) and with k-NN and J48 (Jacobs & Blockeel, 2003). SAGEM (for structural generalized EM) is an Expectation Maximization algorithm for learning the structure and parameters of a logical hidden Markov model. (Jacobs & Blockeel, 2003) describes a k-NN approach where the edit distance is used as the measure of distance between the traces. By comparison, the authors also report the results of applying J4.8 to the dataset. The results shown were obtained on a dataset containing 50 examples.

The comparison of ALLPAD with naïve Bayes shows that ALLPAD achieves a higher accuracy that is significant at 90.8 % (according to a paired two-tailed t test).

ALLPAD outperforms J4.8, equates k-NN and is inferior to SAGEM in terms of accuracy. In terms of precision and recall, all figures are higher than those of SAGEM except precision for class NV. Again, ALLPAD shows good performances given the generality of the approach, even if it does not outperform more specific systems.

Table 2 Results of the Unix traces experiments.

Algorithm	ALLPAD	Naïve Bayes	SAGEM	k-NN	J4.8
Accuracy	91%	86%	94%	91%	87%
St. Dev.	3%	3%	6%	-	-
Precision NV	73%	55%	94%	-	-
St. Dev.	11%	8%	6%	-	-
Precision NP	95%	97%	91%	-	-
St. Dev.	1%	1%	2%	-	-
Recall NV	74%	88%	67%	-	-
St. Dev.	6%	5%	3%	-	-
Recall NP	94%	85%	89%	-	-
St. Dev.	4%	5%	5%	-	-

4.3 Artificial Data

ALLPAD was also applied to artificial data. An LPAD was chosen and datasets were generated from it by random sampling. The LPAD chosen is the one shown in example 1 with the following differences: besides the two parents and the child, the program considers as well the four grandparents of the child; the alleles of the grandparents are randomly chosen between p and w with uniform probability, and the clause that specifies the dependence of the alleles on chromosome 2 from those of the father is replaced with

$$(cg(X,2,A):0.6) \vee (cg(X,2,B):0.4) \leftarrow$$
$$father(Y,X), cg(Y,1,A), cg(Y,2,B).$$

This is done because otherwise, with the choices made for grandparent alleles, ALLPAD would have no means of identifying that the alleles on chromosome 1 depend on those of the mother and the alleles on chromosome 2 depend on those of the father.

The performances of ALLPAD are evaluated by computing the log likelihood assigned to a test dataset containing 20,000 cases by the programs learned. The approximate testing procedure was used for computing the probability of the cases. The results are compared with those of naïve Bayes and with those of the original LPAD.

The language bias given as input to ALLPAD contains one template for each color gene of each member of the family. The head allows the two possible alleles p and w and the body allows the color gene atoms for all the ancestors of the plant.

Twenty datasets with 5,000 cases have been generated and twenty with 10,000 cases. ALLPAD and naïve Bayes models were learned from each dataset and the log likelihoods were averaged.

The time limit for the MIP optimization was set to 1 hour, the size of the beam was set to 50, and the maximum number of bodies to be refined for each clause template was set to 200.

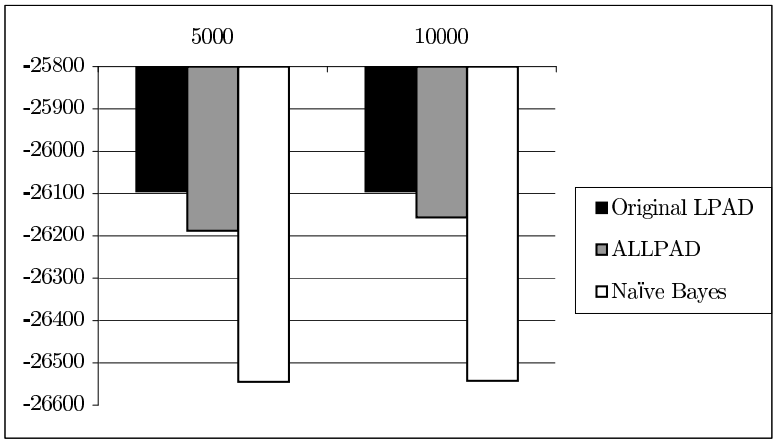

Fig. 2 Average log likelihood of the original LPAD, ALLPAD and naïve Bayes

The results are shown in Figure 2. The difference between ALLPAD and naïve Bayes is significant at 100.0 % (according to a paired two-tailed t test).

5 Related Work

There has recently been a growing interest in the field of probabilistic logic programming: a number of works have appeared that combine logic programming or relational representations with probabilistic reasoning. Among these works, we cite: Bayesian Logic Programs (BLPs) (Kersting & Raedt, 2000, 2001a), Probabilistic Relational Models (PRMs) (Getoor, Friedman, Koller, & Pfeffer, 2001), Independent Choice Logic (ICL) (Poole, 1997), Stochastic Logic Programs (Muggleton, 2000; Cussens, 2000), the Meta Interpreter Approach (MIA) (Blockeel, 2003), Logical Bayesian Networks (LBNs) (Fierens, Blockeel, Ramon, & Bruynooghe, 2004) and CLP(BN) (Santos Costa, Page, Qazi, & Cussens, 2003).

In (Blockeel, 2004) the author compares MIA, CLP(BN), BLPs, LBNs and LPADs on the problem of representing Mendel's law of inheritance of pea color. The author concludes that LPADs are the only ones that are both among the most readable and among those that are able to express the most stringent constraints on the joint distribution by using only qualitative information.

In (Vennekens & Verbaeten, 2003) the authors compare LPAD with Bayesian Logic Programs (Kersting & Raedt, 2000, 2001a). They show that every BLP can be expressed as an LPAD with a semantics that matches that of the BLP. Moreover, they also show that a large subset of LPADs

can be translated into BLPs in a way that preserves the semantics of the LPADs. Such a subset is the set of all finite ground LPADs that are acyclic. Therefore, the techniques developed in (Kersting & Raedt, 2001b) for learning BLPs can be used for learning this class of LPADs as well and, on the other side, ALLPAD can be used for learning BLPs, thus representing an alternative approach to those in (Kersting & Raedt, 2001b).

Stochastic Logic Programs (SLPs) (Cussens, 2000; Muggleton, 2000) are another formalism integrating logic and probability. Each clause C in a SLP has a label λ_C that specifies, as in probabilistic context-free grammars, the probability that C is used in a SLD refutation when an atom with the same predicate as the head of C has to be proved. In (Vennekens & Verbaeten, 2003) the authors have shown that a SLP can be translated into an LPAD, while whether the opposite is possible is not known yet. Therefore, it is not clear at the moment whether the techniques used for learning SLPs can be used for learning LPADs.

PRISM (Sato, 1995) is a logic language in which a program is composed of a set of facts and a set of rules such that no atom in the set of facts appears in the head of a rule. In a PRISM program, each atom is seen as a random variable taking value true or false. A probability distribution for the atoms appearing in the head of rules is inferred from the probability distribution given for the set of facts. PRISM differs from LPADs because PRISM assigns a probability distribution to ground facts while LPADs assign a probability distribution to the atoms in the head of rules. PRISM programs resemble programs of ICL, in the sense that PRISM facts can be seen as ICL abducibles. In (Sato & Kameya, 2001) the authors also propose an algorithm for learning the parameters of the probability distribution of facts from a given probability distribution for the observable atoms (atoms in the head of rules). However, no algorithm for learning the rules of a PRISM program has been defined. Inferring the parameters of the distribution is performed in ALLPAD analytically by means of theorem 1 rather than by means of the EM algorithm as in PRISM.

Considering the problem of learning LPADs, a system related to ALLPAD is *Classic'cl* (Stolle et al., 2005). ALLPAD differs from *Classic'cl* in the following respects: it is able to solve the constraint problem in an approximate way, it learns definite clauses separately, it can search the space of heads bottom-up and it adopts beam search in the space of bodies.

6 Conclusion and Future Works

The learning algorithm ALLPAD has been presented. It improves LLPAD by solving the constraint satisfaction problem in an approximate way so that real world problems can be solved.

ALLPAD was tested on the problem of classifying proteins into SCOP classes and on the problem of classifying Unix traces into user classes. While not overcoming the best systems in the domains, ALLPAD achieved good

accuracy, thus showing the viability of the approach. Moreover, ALLPAD was also tested on artificial data and showed a significantly higher log likelihood with respect to naïve Bayes.

In the future, work will be devoted to the definition of a generality relation among LPAD clauses and of the relative generalization operators, so that the ground clauses that are returned by ALLPAD can be generalized in a sixth phase.

Moreover, given that Bayesian networks can be translated into LPADs (Vennekens et al., 2004), we will explore an alternative approach for learning LPADs: learning a Bayesian network from data and converting it into an LPAD. In this way a ground LPAD will be obtained so generalization techniques will have to be employed as well.

Finally, the problem of learning a more general class of LPADs, namely LPADs where clauses do not have mutually exclusive bodies, will be investigated.

7 Acknowledgements

This work was partially funded by the Ministero dell'Istruzione, della Ricerca e dell'Università under the PRIN 2005 project "Specification and verification of agent interaction protocols". The author would like to thank Evelina Lamma, Marco Gavanelli and Maddalena Nonato.

References

Blockeel, H. (2003). Prolog for first-order bayesian networks: A meta-intepreter approach. In *The second workshop on multi-relational data mining.*

Blockeel, H. (2004). Probabilistic logical models for Mendel's experiments: An exercise. In *Proceedings of the fourteenth international conference on inductive logic programming, work in progress track.*

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *The sixteenth conference on uncertainty in artificial intelligence* (p. 115-122). San Francisco, CA: Morgan Kaufmann.

Dehaspe, L., & Raedt, L. D. (1996). DLAB: A declarative language bias formalism. In Z. W. Ras & M. Michalewicz (Eds.), *Proceedings of the ninth international symposium on foundations of intelligent systems* (p. 613-622). Berlin, Germany: Springer.

Fierens, D., Blockeel, H., Ramon, J., & Bruynooghe, M. (2004). Logical bayesian networks. In *Third workshop on multi-relational data mining* (p. 19-30).

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. In S. Dzeroski & N. Lavrac (Eds.), *Relational data mining.* Berlin, Germany: Springer-Verlag.

Greenberg, S. (1988). *Using unix: Collected traces of 168 users* (Tech. Rep. No. Research Report 88/333/45). Department of Computer Science, University of Calgary.

Gutmann, B., & Kersting, K. (2006). TildeCRF: Conditional random fields for logical sequences. In *Seventeenth european conference on machine learning.* Berlin, Germany: Springer.

Jacobs, N., & Blockeel, H. (2003). User modeling with sequential data. In *Proceedings of the tenth international conference on human-computer interaction* (p. 557-561). Mahwah, NJ: Lawrence Erlbaum Associates.

Kersting, K., & Gärtner, T. (2004). Fisher kernels for logical sequences. In *Fifteenth european conference on machine learning* (p. 205-216). Berlin, Germany: Springer.

Kersting, K., & Raedt, L. D. (2000). Bayesian logic programs. In *The tenth international conference on inductive logic programming, work in progress track.* Available from `http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-35/`

Kersting, K., & Raedt, L. D. (2001a). *Bayesian logic programs* (Tech. Rep. No. 151). Freiburg, Germany: Institute for Computer Science, University of Freiburg.

Kersting, K., & Raedt, L. D. (2001b). Towards combining inductive logic programming and bayesian networks. In C. Rouveirol & M. Sebag (Eds.), *The eleventh international conference on inductive logic programming.* Berlin, Germany: Springer-Verlag.

Kersting, K., Raedt, L. D., & Raiko, T. (2006). Logical hidden markov models. *Journal of Artificial Intelligence Research*, *25*, 425-456.

Kersting, K., & Raiko, T. (2005). 'Say EM' for selecting probabilistic models for logical sequences. In *Proceedings of the twenty first conference on uncertainty in artificial intelligence.* Arlington, VA: AUAI Press.

Kersting, K., Raiko, T., Kramer, S., & Raedt, L. D. (2003). Towards discovering structural signatures of protein folds based on logical hidden markov models. In *Pacific symposium on biocomputing* (p. 192-203). Singapore: World Scientific Press.

Muggleton, S. H. (2000). Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, *4*(041). Available from `http://www.ida.liu.se/ext/epa/cis/2000/041/tcover.html`

Poole, D. (1997). The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, *94*(1-2), 7-56.

Riguzzi, F. (2004). Learning logic programs with annotated disjunctions. In *Fourteenth international conference on inductive logic programming* (p. 270-287). Springer.

Riguzzi, F. (2006). *ALLPAD: Approximate learning of logic programs with annotated disjunctions* (Tech. Rep. No. CS-2006-01). University of Ferrara. Available from `http://www.ing.unife.it/aree_ricerca/informazione/cs/technical_reports/CS-2006-01.pdf`

Santos Costa, V., Page, D., Qazi, M., & Cussens, J. (2003). CLP($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge. In *The nineteenth conference on uncertainty in artificial intelligence.* San Francisco, CA: Morgan Kaufmann.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *The twelfth international conference on logic programming* (p. 715-729). Cambridge, MA: MIT Press.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, *15*, 391-454.

Stolle, C., Karwath, A., & Raedt, L. D. (2005). *Cassic'cl*: An integrated ILP system. In *The eigth international conference on discovery science.* Berlin, Germany: Springer.

Turcotte, M., Muggleton, S., & Sternberg, M. J. E. (2001). The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, *43*(1/2), 81-95.

Van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, *38*(3), 620-650.

Vennekens, J., & Verbaeten, S. (2003). *Logic programs with annotated disjunctions* (Tech. Rep. No. CW386). K. U. Leuven. Available from `http://www.cs.kuleuven.ac.be/~joost/techrep.ps`

Vennekens, J., Verbaeten, S., & Bruynooghe, M. (2004). Logic programs with annotated disjunctions. In *The twentieth international conference on logic programming.* Berlin, Germany: Springer.