Rule induction from noisy examples

Laura Firoiu
Computer Science Department
University of Massachusetts Amherst
lfiroiu@cs.umass.edu

Motivation

This work is a part of an effort aimed at creating an intelligent agent embodied in a robot (Pioneer), which learns by interacting with its environment. The specific problem addressed here is rule induction from a relational representation of robot experiences. The rules are expected to capture the definitions of types of experiences. Perceptual relations describing the robot's interaction with the environment are computed by hand-coded functions from the stream of values returned by the robot's sensors. In this work it is assumed that experiences have been correctly [1] labeled with relations denoting their types. The goal is to organize the experiences' perceptual relations into rules that define their relational labels, *i.e.* to learn their intensional definition. Rules are desirable because they represent in a compact way the robot's interaction with the environment and can be further used for planning.

Rule learning is the subject of inductive logic programming (ILP) and in this work the application of the basic ILP technique of least general generalization under subsumption(*lggs*) is investigated. Given a set of positive examples, *lggs* creates a rule that logically entails each example, by selecting only what these examples have in common. The problem is that in our domain the examples represent robot experiences and as such are subject to noise generated by both sensor and perceptual limitations. Specifically, while the classification of examples is correct, there may be either missing or extraneous relations in the description of examples. This is exactly the opposite of the usual ILP task, where examples may be missclassified, but their description is assumed correct. ILP algorithms usually deal with noise by finding a subset of the examples for which a rule can be induced. But the robot's experiences may yield too few or no correct examples and a classical ILP technique may be unable to generalize. The solution presented here is to replace the strong requirement that the induced rule entail every example in the subset with the soft requirement that the rule literals be supported by enough evidence in the data. The result is a new induction technique applicable in very noisy domains.

[1] For now, the experience labels are assigned by the person running the experiment.

Perceptual Relations and Example Clauses

The percepts are atomic propositions computed by specialized hand-coded functions from the stream of sensor values. At each time step a proposition is either true or false. The atomic propositions encode:

- features of external objects; for example *is_red_A* means that an object is perceived on visual channel A which detects red objects
- robot features: *moving_forward_R* is true when the robot's *translational velocity* sensor returns a value greater than a positive threshold
- relations involving the robot and external objects; examples are:
  - *approach_R_A* holds when the visual distance sensor associated with channel A returns decreasing values.
  - *left_of_A_C* holds when the object in the visual field of channel A is positioned to the left of the object observed on channel C (blue colors); it is computed from the *(x, y)* coordinates of the two objects in the visual fields of channels A and C.

Relations are easily derived from these atomic propositions. For example, if proposition *approach_R_A* is true at all times between t_i and t_f during experience e, then it becomes relation $approach(e, t_i, t_f, r, a)$. There are no function terms in the resulting first order language.

Figure 1 shows the computed atomic propositions for a *pass_right* experience – the robot r moves forward past a red object a on its right. The perceptual relations that hold during an experience are grouped into an example clause. The resulting clause for the experience in figure 1 is a positive example of a *pass_right* concept:

$$stop(e_0, T_0, T_1, r),\ moving\_forward(e_0, T_2, T_5, r)$$
$$approach(e_0, T_3, T_4, a, r),$$
$$move\_to\_the\_left(e_0, T_3, T_4, r, a),$$
$$is\_red(a)$$
$$T_0 < T_1 < T_2 < T_3 < T_4 < T_5$$
$$\rightarrow\ pass\_right(e_0, T_0, T_5, r, a).$$

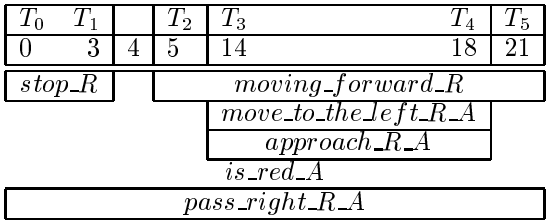

Figure 1: The atomic propositions and their temporal extentions for a *pass_right* experience.

In the example clauses the time constants [2] are replaced by time variables, because we do not want event occurrence at the same time steps during different experiences to be considered meaningful for induction. As a result, a learned rule will not state that the robot must start moving at the $5th$ time step, but at some unspecified time T_k.

ILP, Subsumption and Merging

The general problem inductive logic programming tries to solve is that of finding a hypothesis that together with background knowledge explains a set of positive examples and does not contradict a set of negative examples. Usually, the background knowledge, the examples and the hypothesis are sets of first order clauses.

In this work we deal with a restricted setting of the induction problem: the language is function free, neither background knowledge, nor negative examples are given and the concepts to be learned are assumed non-recursive. Also, the positive examples are definite clauses, meaning that no constraints are imposed on the learned rules. In this simplified setting, the hypothesis can be found among the clauses that subsume the set of positive examples. Specifically, the hypothesis is taken to be the least general generalization under subsumption ($lggs$) of the set of examples. Subsumption and $lggs$ are basic ILP concepts due to Plotkin and will be presented only informally here. Formal definitions and properties regarding subsumption and the subsumption ordering of clauses can be found in ILP books such as (Nienhuys-Cheng & de Wolf 1997).

A clause C subsumes a clause D if there exists a substitution θ such that $C\theta \subseteq D$. $C \preceq D$ denotes "C subsumes D". If $C \preceq D$ then clause C is considered more general than D. If both $C \preceq D$ and $D \preceq C$ then C and D are equivalent under subsumption, denoted $C \sim D$. It is immediate that if $C \preceq D$ then $C \models D$. So if we are looking for a clause that entails D, we may find it among the clauses that subsume D.

Muggleton((Muggleton 1992)) showed that if $C \models D$, D is not a tautology and C is not recursive, then $C \preceq D$. This means that if we do not want to learn recursive concepts, as in our case, then looking for rules in the set of clauses that subsume the examples is an appropriate method, since the set covers the entire hypotheses space.

A generalization under subsumption of a set of clauses $\{C_1, \ldots C_n\}$ is a clause D that subsumes every clause C_i in the set. Finding a generalization under subsumption of two clauses C_1 and C_2 means finding two substitutions θ_1 and θ_2 and a clause D such that $D\theta_1 \subseteq C_1$ and $D\theta_2 \subseteq C_2$. It can be noticed that $D \subseteq C_1\theta_1^{-1} \cap C_2\theta_2^{-1}$.

The least general generalization under subsumption ($lggs$) of a set of clauses $\mathcal{C} = \{C_1, \ldots C_n\}$ is a clause D that is a generalization of this set and is subsumed by any other generalization of $\mathcal{C}$.

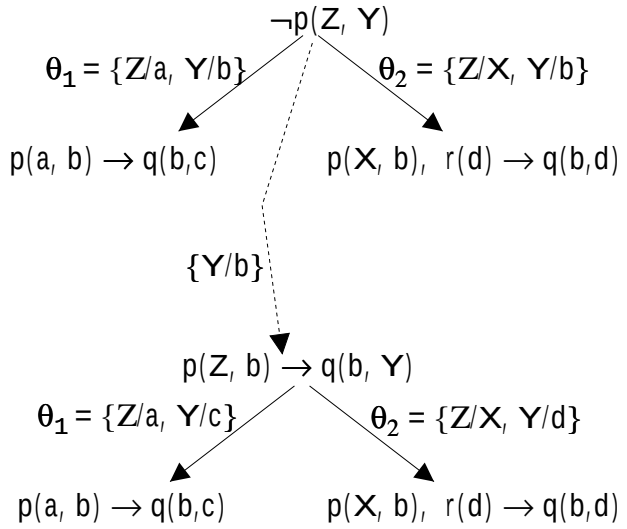

Figure 2: $\neg p(Z, Y)$ is a generalization under subsumption of clauses $p(a, b) \rightarrow q(b, c)$ and $p(X, b), r(d) \rightarrow q(b, d)$. $p(Z, b) \rightarrow q(b, Y)$ is the $lggs$ of the two clauses.

Both generalization and least general generalization under subsumption are illustrated in figure 2. Because the $lggs$ of a set of example clauses is the least general clause that entails all the examples, it makes a good hypothesys.

In order to understand why the $lggs$ of a set of clauses does not always yield an adequate rule in our domain, let us look at three "push" experiences:

$$stop(e_{27}, T_0, T_1, r),\ moving\_forward(e_{27}, T_2, T_5, r),\ approach(e_{27}, T_3, T_4, r, c),\ \mathbf{contact}(e_{27}, \mathbf{T_5}, \mathbf{T_5}, r),\ is\_blue(c) \rightarrow push(e_{27}, T_0, T_5, r, c)$$

$$stop(e_{28}, T_0, T_1, r),\ moving\_forward(e_{28}, T_2, T_6, r),\ approach(e_{28}, T_3, T_4, r, c),\ \mathbf{contact}(e_{27}, \mathbf{T_5}, \mathbf{T_6}, r)\ is\_blue(c) \rightarrow push(e_{28}, T_0, T_6, r, c)$$

[2] The variables are capitalized and the constants are written in lowercase.

$$stop(e_{29}, T_0, T_1, r),\ moving\_forward(e_{29}, T_2, T_5, r),$$
$$approach(e_{29}, T_3, T_4, r, c),$$
$$is\_blue(c)$$
$$\rightarrow\ push(e_{29}, T_0, T_5, r, c)$$

The *lggs* of the three clauses is:

$$stop(E, T_0, T_1, r),\ moving\_forward(E, T_2, T_5, r),$$
$$approach(E, T_3, T_4, r, c),$$
$$is\_blue(c)$$
$$\rightarrow\ push(E, T_0, T_5, r, c)$$

It can be noticed that the literal $contact(\cdot)$ does not occur in the third example and therefore it does not occur in the induced [3] rule. But we know that *contact* is a defining predicate for a "push" experience. Its absence from the third clause is due to sensor limitations not recovered by the perceptual system. If negative examples or constraints would be present, a classical ILP method might discard the third example as noisy and create a rule based only on the first two examples. The problem is that in this domain the noise level is high and there may be only a few or no correct examples.

The solution adopted here is to create a *merged* clause that keeps all literals present in the examples, but remembers for each literal the number of times it was encountered in examples. The algorithm that creates this clause is called the *merging* algorithm and is described in the next section. The *merged* clause of the three examples above is:

$$stop(E, T_0, T_1, r),\ moving\_forward(E, T_2, T_6, r),$$
$$approach(E, T_3, T_4, r, c),$$
$$contact(E, T_5, T_6, r),\ contact(E, T_6, T_6, r),$$
$$is\_blue(c)$$
$$\rightarrow\ push(E, T_0, T_6, r, c)$$

A *merged* clause for a kind of experiences is not the defining rule for that kind, but an accumulation of evidence. After a number of examples, the rule can be extracted from the merged clause by selecting only the literals whose strength is above a certain threshold. For example, notice that in the above *merged* clause there are two *contact* literals, with distinct time structures. Future examples will decide which one will appear in the rule.

The merging algorithm

The algorithm is incremental. For each concept it finds a generalization under subsumption of two clauses: the new example and the *merged* clause of the previous examples.

Algorithm 1 *merging*($\{E\}$)
input: a set of example clauses $\{E\}$ for a concept
output: the *merged* clause for the concept

- $M \leftarrow \emptyset$
- for each example clause E:
  1. $size(p) \leftarrow 1$, $strength(l) \leftarrow 1$ for each literal $l \in E$
     $size(E) \leftarrow$ #literals $l \in E$ + #terms $t \in E$
  2. if $M = \emptyset$ then $M \leftarrow E$
     else $M \leftarrow match\_clauses(M, E)$

Algorithm *match_clauses*, invoked at step 2 of algorithm *merging*, finds a generalization under subsumption of two clauses, M and E such that the size of the resulting merged clause is minimized heuristically – the size of a clause is the sum of the sizes of its distinct literals and terms. For the merged clause the size of a literal is initialized with its strength, so algorithm *match_clauses* will first match M's strongest literals. The algorithm does not backtrack, so the search for a generalization is mostly influenced by these literals.

Algorithm 2 *match_clauses(M, E)*
input: a merged clause M and a clause E
output: a merged clause M'

1. $size(l) \leftarrow strength(l)$ for each literal $l \in M$, $M' \leftarrow \emptyset$
2. while there exists a pair of compatible literals $\langle l, p\rangle$, $l \in M$, $p \in E$
   (a) find the pair of compatible [4] literals $\langle l, p\rangle$ whose matching with substitutions α and β yields the clause $M\alpha^{-1} \cup E\beta^{-1}$ with the smallest size
   (b) $M \leftarrow M\alpha^{-1}$, $E \leftarrow E\beta^{-1}$
   (c) for every $l \in M$ identical with a $p \in E$:
   - $M \leftarrow M - \{l\}$, $E \leftarrow E - \{p\}$
   - $strength(l) \leftarrow strength(l) + strength(p)$
   - $M' \leftarrow M' \cup \{l\}$

   (d) mark the new terms in α and β as non-replaceable during future literal matchings
3. $M' \leftarrow M' \cup M \cup E$

Algorithm 2 computes a generalization of clauses M and E subject to two restrictions: (1) each term in M or E is replaced (i.e. matched with another term) at most once and (2) each literal in M or E is matched at most once. Due to the first restriction the non-matched literals in the example clause are added to the merged clause with their terms properly replaced. The second restriction avoids the eventual match of a literal representing an event in the experience clause, with distinct events of the same type – literals with the same predicate symbol – in the *merged* clause.

Preliminary results

Our data set has forty two experiences grouped in nine types. The experiences are very simple and involve either one object – approaching, passing, or pushing the object, or two objects – passing both objects or first passing one object and then pushing the other one. These types of experiences represent the concepts whose definitions must be learned. Because very few examples

[3] No substitution can make a rule with $contact(\cdot)$ in its body be included in the third clause.

[4] Two literals l and p are compatible if they have the same predicate symbol and sign and there exist two substitutions α and β that match l with p: $l\alpha^{-1} = p\beta^{-1}$.

are present for each concept, the threshold for including literals was set very low, at 51 percent of the maximum strength of the literals in the merged clause.

The learned definitions of the 9 concepts are shown in table 1. Time relations were removed for clarity, but the indices of the time variables observe the temporal order.

$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_3,r),$ $approach(E,t_3,t_4,r,V)$ $\rightarrow\ approach\_exp(E,t_0,t_4,r,V)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_6,r),$ $approach(E,t_3,t_4,r,V),$ $contact(E,t_5,t_6,r)$ $\rightarrow\ push(E,t_0,t_6,r,V)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_3,r)$ $\rightarrow\ pass\_right(E,t_0,t_3,r,V)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_3,r)$ $\rightarrow\ pass\_left(E,t_0,t_3,r,V)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_6,r),$ $move\_to\_the\_right(E,t_3,t_4,r,a),$ $is\_red(a),\ is\_blue(c),$ $contact(E,t_5,t_6,r)$ $\rightarrow\ pass\_left\_then\_push(E,t_0,t_6,r,a,c)$
$stop(E,t_0,t_2,r),\ moving\_forward(E,t_4,t_9,r),$ $approach(E,t_5,t_7,r,a),$ $move\_to\_the\_left(E,t_6,t_7,r,a),$ $right\_of(E,t_1,t_3,a,c),\ front\_of(E,t_1,t_3,a,c),$ $is\_red(a),\ is\_blue(c),$ $contact(E,t_8,t_9,r)$ $\rightarrow\ pass\_right\_then\_push(E,t_0,t_9,r,a,c)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_6,r),$ $approach(E,t_4,t_5,r,a),$ $move\_to\_the\_right(E,t_3,t_5,r,a),$ $is\_red(a),\ is\_blue(c),$ $left\_of(E,t_5,t_5,a,c),\ front\_of(E,t_5,t_5,a,c)$ $\rightarrow\ pass\_left\_then\_pass\_left(E,t_0,t_6,r,a,c)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_3,r)$ $\rightarrow\ pass\_left\_then\_pass\_right(E,t_0,t_3,r,V_0,V_1)$
$stop(E,t_0,t_1,r),\ moving\_forward(E,t_2,t_8,r),$ $approach(E,t_3,t_5,r,c),$ $move\_to\_the\_left(E,t_4,t_5,r,c),$ $approach(E,t_3,t_6,r,a),$ $move\_to\_the\_left(E,t_5,t_7,r,a),$ $is\_red(a),\ is\_blue(c),$ $behind(E,t_0,t_5,a,c),\ left\_of(E,t_5,t_5,a,c)$ $\rightarrow\ pass\_right\_then\_pass\_right(E,t_0,t_8,r,c,a)$

Table 1: Rules learned by *merging* for the nine types of experiences in the data set.

It can be noticed that in general, the rules capture the meaningful events for each experience type, with the exception of "pass_right", "pass_left" and "pass_left_then_pass_right" which have the same rule. This might eventually be avoided by lowering the threshold for extracting rules from their *merged* clauses until distinguishing literals are promoted in the rule.

For comparison, table 2 lists some of the rules learned by CProgol4.4 (Muggleton 1995) from the same data set, with the constraint that rules for different concepts must be distinct. Because Progol tries to find short

$approach(A,E,C,r,D) \rightarrow approach\_exp(A,B,C,r,D).$
$push(e29,0,19,r,c).$ $contact(A,E,C,r) \rightarrow push(A,B,C,r,D).$
$moving\_forward(A,E,C,r) \rightarrow pass\_right(A,B,C,r,D).$
$move\_to\_the\_right(A,E,F,r,D) \rightarrow pass\_left(A,B,C,r,D).$
$move\_to\_the\_right(A,F,G,r,D), is\_red(D)$ $\rightarrow pass\_left\_then\_push(A,B,C,r,D,E).$
$left\_of(A,F,G,D,E)$ $\rightarrow pass\_left\_then\_pass\_left(A,B,C,r,D,E).$

Table 2: Rules learned by CProgol4.4 for the nine types of experiences in the data set.

hypotheses, the rules it induces have few literals, and important features may be ignored in some cases. For example, it can be noticed in table 2 that *contact* is missing from the rule for *pass_left_then_push*.

Discussion

In order to deal with noise in the example descriptions, the method investigated in this work gives up the requirement that the hypothesis be consistent with every example. The resulting rule may contain literals that do not occur in a particular example, but their presence is supported by other examples. Preliminary results show that in a very noisy domain, where a method following crisp logic may lead to rules based on irrelevant predicates, this approach leads to rules that intuitively make sense to a human observer. This technique can be extented to deal with missclassified or unclassified examples: a new example can be assigned to one of the previous clusters based on how tightly it fits the merged clause of that cluster. More experiments, with larger data sets and more objective criteria for evaluating the learned rules are needed to validate the method, but the preliminary results presented here warrant future efforts.

Acknowledgements

This research is supported by DARPA/AFOSRF and DARPA under contracts No. F49620-97-1-0485, No. N66001-96-C-8504 and No. DASG60-99-C-0074. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the or the U.S. Government.

References

Muggleton, S. 1992. Inverting implication. In *Proceedings of the Second Inductive Logic Programming Workshop*, 19–39.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming* 13(3-4):245–286.

Nienhuys-Cheng, S.-H., and de Wolf, R. 1997. *Foundations of Inductive Logic Programming.* Springer.