Predictive and Descriptive Approaches to Learning Game Rules from Vision Data

Abstract. Systems able to learn from visual observations have a great deal of potential for autonomous robotics, scientific discovery, and many other fields. We describe an application to learning rules of a dice game using data from a vision system observing the game being played. While there have been many approaches to learning from such visual data, using neural networks for instance, we are interested in using learned rules for higher level spatial and temporal reasoning. For this reason, we chose to employ an ILP system for the learning tasks. We experimented with two broad approaches: (i) a predictive learning approach with the Progol system, where explicit concept learning problems are posed and solved, and (ii) a descriptive learning approach with the HR system, where a general theory is formed with no specific problem solving task in mind and rules are extracted from the theory. In the former case, difficulties arise in specifying the problems to solve, and sometimes over-generalisation can result in key rules being omitted from the output. In the latter case, difficulties arise in determining exactly what elements a theory should contain, and theory formation may involve searching a prohibitively large space. Drawing on the results from these experiments we suggest ways for combined predictive/descriptive approaches to learning from vision data.

1 Introduction

We are interested in combining machine learning, automated reasoning and visual information processing, so that the integrated systems can make discoveries from their observations, and the robotic systems within which they may be situated can adapt accordingly. In applications where there is interaction with lay users (such as in the nurse robot project, [14] or the interactive room project [20]) the representation and reasoning about common sense knowledge – in particular about everyday objects, events and time – is a desirable feature in a system. Therefore, it is our goal to develop an autonomous system that is capable of reasoning about qualitative space (accounting for everyday visual objects) and about actions and change (accounting for temporal reasoning). There has been much work towards automatically learning from visual data, such as that described in [10]. However, given that our systems will require further processing of the rules learned, in particular within spatial and temporal reasoning systems, black box approaches to learning from the vision data are not entirely appropriate. Moreover, we also want the learned theories to be understandable so that we (as developers) can determine how well the system is learning by looking at how precise and complex are concepts learned are. For these reasons, we have investigated the use of Inductive Logic Programming systems to learn rules from vision data.

The usual mode within which ILP is applied is known as *predictive* learning. In these approaches, a particular categorisation problem, or a set of such problems, are specifi ed, and the learning system derives a theory which performs well when the derived rules in the theory are used for predicting the category of unseen examples. ILP

systems which perform well at these tasks include the Progol [17] and FOIL [21] systems. An alternative mode within which ILP can be applied is known as *descriptive* learning. In such approaches, there is no particular problem to solve, other than that of discovering interesting concepts relating to the data provided and interesting facts which relate these concepts. Descriptive learning programs such as the HR system, [4] and the CLAUDIEN [8] and WARMR [9] systems, produce theories containing examples, concepts which categorise the examples, conjectures which relate the concepts and explanations which illustrate the truth or falsity of the conjectures.

We have used the Progol and HR systems to perform predictive and descriptive learning respectively, given data from a vision system which observed a dice game being played. The overall purpose of the application is to automatically learn protocol behaviour, including how players move their hands, how they move pieces, how their facial expressions change, etc. One of the tasks towards this goal is to learn the rules of the game, given the visual information expressed using a minimal number of background concepts. In the case of Progol, this involved specifying a particular machine learning problem and using Progol to learn a series of rules to solve the learning problem. The rules which solve the learning problem are taken as the game rules identified by the system. In the case of HR, the application involved specifying the theory formation parameters, forming a theory for a given number of steps, then extracting rules from the theory. Both of these approaches have limitations. In particular, the predictive approach may generalise when more specific rules are required, and the descriptive approach can search too large a space to effectively produce useful rules.

The long term goal of this research is to close the loop, so that the learned rules can inform the action of an autonomous agent within which the vision system is situated. For full autonomy, agents will require higher level abilities to reason about low level data collected from the environment, so that they can adapt to the environment and make discoveries which are necessary for the completion of tasks. Similarly, scientific discovery systems such as the "Robot Scientist" [11] would be greatly enhanced if they could learn from visual observations of experiments. The purpose of the experimentation and the reporting of it here is primarily to compare and contrast descriptive and predictive approaches to learning rules from vision data. In addition, HR has had many applications in mathematics, where the amount of noise in the data is very low (often zero), and it's application to noisy vision data has provided the first real test of it's ability to handle uncertain data. Moreover, while qualitative comparisons of HR and Progol have been given in [2] and [5], this is the first quantitative comparison of these systems.

The paper is organised as follows. In section 2, we describe the vision system which generated the data from observing the dice game. We also briefly describe the Progol and HR systems. In particular, we take the opportunity to describe the mechanisms HR has for handling noisy data, which have not been described in detail elsewhere. In section 3, we describe the experimental setup, including the rules of the game and how HR and Progol were employed for this application. In section 4, we present the results from our experiments with the vision data. To conclude, we describe the lessons we have learned from this application, with a comparison of predictive and descriptive approaches in general. We also speculate some future approaches to using ILP for vision data which may use a combination of descriptive and predictive approaches.

2 System Descriptions

2.1 The Vision System

The vision system used in this work is composed of a video camera observing a table top where a dice game is taking place. Figure 1 shows a picture of this setup, and also depicts a schema of the system modules.

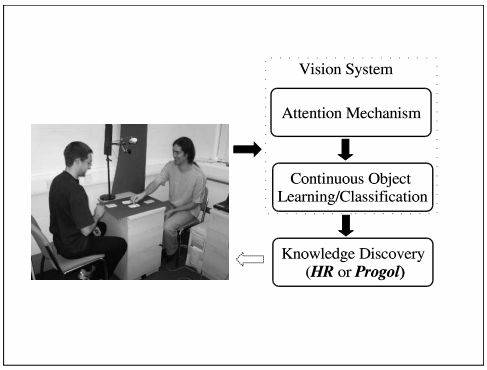


Fig. 1. A scheme of the experimental setup.

Visual information obtained by the vision system is turned into a symbolic description of states of the objects observed on the table top. This is used in turn as input data for the knowledge discovery module (i.e., HR and Progol in our experiments). Outputs from the knowledge discovery module are further used to drive a virtual agent to play the observed game. However, discussion of how the learned rules are automatically interpreted and turned into actions which the agents perform is beyond the scope of this paper. To turn video streams into symbolic information, the vision system uses a spatio-temporal attention mechanism and an object classifier, both of which are described below.

- Spatio-temporal attention

Video streams of dynamic scenes contain large quantities of data, much of which is irrelevant to scene learning and interpretation. An attention mechanism is therefore required to identify 'interesting' parts of the stream, in terms of spatial location ('where') and temporal location ('when'). For autonomous learning, models or heuristics are required to determine what is of interest, and what is not. Such models could be based on motion, novelty, high (or low) degree of spatial variation, or a number of other factors. In our framework, it is merely important that an attention mechanism exists to identify interesting areas of space and time. For this reason, we have chosen to use motion in our implementation, as this is straightforward to work with. We make no claim that attention based on motion only is suitable in all scenarios, however it is appropriate in our chosen domain.

The spatial aspect of our attention mechanism is based around a generic blob tracker [13] that works on the principle of multi-modal (Gaussian mixture) background modeling, and foreground pixel grouping. This identifies the centroid location, bounding box and pixel segmentation of any separable moving objects in the scene in each frame of the video sequence. If multiple objects are non-separable from the point of view of the camera, they are tracked as a single object, until such time as they are separable.

The temporal aspect of our attention mechanism identifies key frames where there is qualitatively zero motion for a number of frames (typically 3), which are preceded by a number of frames (typically 3) containing significant motion. Motion is defined as a change in any objects' centroid or bounding box above a threshold value (typically 5 pixels, determined from observed tracker positional noise). This method for temporal attention is based on the assumption that all objects remain motionless following a change in state (and that the process of change is not in itself important). This is valid for the example scenarios we present within this paper, however we are actively researching more complex temporal attention mechanisms that do not make these assumptions.

- Continuous object learning and classification

Each object in the frames selected by the attention mechanism is classified in two steps. Firstly, features are extracted using banks of wavelets. Secondly, a set of example feature vectors is partitioned into classes using a graph partitioning method [23, 12]. The resulting partitions are used as supervision for training a statistical multi-class classification algorithm (we use a nearest neighbour classifier based on a vector quantisation of the set of training examples, but other methods such as Radial Basis functions or Multi-layer perceptrons could be used). The results from this algorithm are used to construct the symbolic information. In effect, for each object identified by the attention mechanism, a symbol is associated according to the specific model in which this object is classified. For example, the statement: $state([a, b], t_{20})$ indicates that there are two distinct objects in the scene, at time t_{20}, represented by the feature classes a and b respectively. A predicate $successor/2$ representing temporal succession is also provided for each subsequent pair of time points that appear in the argument of statements $state/2$. Atoms $successor(t_i, t_j)$ state that the time point t_i is a *successor* of the time point t_j. So, for example, the vision system may output the following symbolic representation of what it has observed:

state($[a]$, t_{307}).
successor(t_{309}, t_{307}).
state($[a, b]$, t_{309}).

This indicates that the system has observed a single object of class a on the table at time 307, and the next recorded event was at time 309, which was when there were two objects on the table, of class a and class b respectively.

2.2 The Progol System

For the experiments described here, we used CProgol4.5 [19], which is an implementation of the *definite modes language* [18]. This system allows the generalisation of a set

of positive examples without the need of negative examples to guide the search. This characteristic suits well our aim to explain passive visual observation, since, in this case, negative examples cannot be easily input without supervision. In brief, Progol works as follows. For each positive example, it generates a most specific Horn clause constructed according to user defined mode declarations. Mode declarations in Progol account for restrictions in the possible form of the proposed generalisations and hence enable the imposing of language biases on the search space. The initial most specific clause is further contrasted with the remaining examples in the search for a more general formula capable of subsuming most of the data set.

2.3 The HR System

The HR system performs descriptive induction to form theories about given data sets using background information which includes a set of concepts and possibly a set of axioms which relate the concepts. It has mostly been applied to mathematical domains such as number theory, where it has been used to make some interesting discoveries [1]. How HR operates[1] has been described numerous times, so we only provide an overview of its operation here.

At the heart of HR's functionality is an ability to form new concepts from old ones, with the original concepts being supplied as background knowledge. HR uses 15 production rules to perform concept formation, each of which is either binary – taking two old concepts as input – or unary – taking one old concept as input, and each production rule generates a clausal, range restricted definition of a new concept. Each production rule is parameterised so that the input set can lead to the production of multiple concepts. Five important production rules are:

- The *compose* production rule. This takes the definitions of two concepts and combines them by conjoining the two sets of literals in their definitions.
- The *match* production rule. This takes a single old definition and unifies variables to produce a new one.
- The *exists* production rule. This introduces existential quantification by changing ground values to variables in a definition.
- The *split* production rule. This performs instantiation in definitions.
- The *negate* production rule. This negates literals within definitions.

For more information about the production rules, see [4] or [5]. Each concept is assessed using a variety of measures of interestingness [6], and this drives a heuristic search through the space of definitions.

In addition to generating the definitions of new concepts, HR also calculates the success sets of the definitions and uses this data to induce hypotheses about the concepts empirically. In particular, if the success set is empty, rather than adding the new definition to the set of concepts in the theory, rather HR adds a non-existence conjecture to the set of hypotheses. Similarly, if a success set is exactly the same as the success

[1] In particular, a detailed description of HR as an Inductive Logic Programming system has been provided in [4].

set of an existing concept, rather than adding the new concept to the theory, HR adds an equivalence conjecture to the set of hypotheses. In certain domains, in particular algebraic domains such as group theory and ring theory, HR employs a third party automated theorem prover (usually Otter [15], but other have been employed) to attempt to prove that some of the hypotheses follow from a set of axioms supplied by the user, e.g., the axioms of group theory. It also employs a third party model generator (usually Mace [16]) to attempt to show that certain hypotheses contradict the axioms, by generating a counterexample. In other domains, HR may use the theorem proving functionality to show that a hypothesis can be proved from first principles given only some simple axioms. Such hypotheses are unlikely to be interesting to the user, hence HR uses theorem proving as a filter for dull conjectures, for instance in number theory [7].

New functionalities, allowing it to handle noisy data have recently been implemented in HR. These processes were used for this application to learning from vision data, and as they have not been described elsewhere, we provide an overview here. Whenever HR adds a new concept to the theory, it attempts to make certain types of conjecture involving the new concept. In particular, it attempts to make implication conjectures by finding concepts where the success set is a proper subset or superset of the success set for the new concept, and formulating the hypotheses accordingly. Previously, this was restricted to cases where the subset/superset relation was *exact* . This was because, in mathematical domains, a conjecture which is 99% true empirically is normally 100% false, as a counterexample is evident in the data. However, in non-mathematical domains, such conjectures are usually interesting, and the mis-matches may be due to noise or omissions in the data, rather than indicating the falsity of the hypothesis. Hence, we have relaxed the 100% constraint, and HR is now able to make *near-implications*. That is, that, given a user-specified percentage P, HR will add an implication to the theory if it is supported by at least $P\%$ of the data relating to the implication. Note that HR records the proportion of the data that does actually support the conjecture. This is called the *plausibility* of the near-implication.

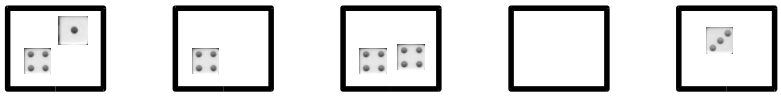
Similarly, HR is able to look for *near-equivalences* by finding old concepts where the success set is the same as the success set for the new concept, with a fraction of the tuples in the success sets breaking this pattern (again, the maximum allowed percentage of such tuples is specified by the user). With near-equivalences, HR can be told to calculate the proportion of supporting examples by looking at only the positives for the equivalence. For instance, given concepts C_1 and C_2 which had very similar success sets, HR would make the near equivalence conjecture that $C_1 \leftrightarrow C_2$. If told to use positive data only, when calculating the proportion of supporting examples, it would only look at tuples for which either C_1 was true or C_2 was true, or both. Without this constraint, HR can produce large numbers of fairly specialised near-equivalences which are empirically true because the majority of examples are true for neither the left hand nor right hand side of the equivalence. Such conjectures are usually uninteresting. Given near-equivalences and near-implications, HR can extract implicates from these: if a conjunction of literals implies another conjunction, then HR can extract the set of clausal implications where a conjoined set of literals in the body implies a single literal in the head. From these implicates, HR can also extract prime implicates, where no proper subset of the body literals implies the goal literals, as described in [3].

3 Experimental Setup

Eight distinct sessions where a human played the game described in section 3.1 below were observed by the vision system. This produced eight separate datasets, which were used as inputs to the Progol and HR systems. Details of how Progol and HR were set up to perform the learning task are given in sections 3.2 and 3.3 respectively.

3.1 Rules of the Dice Game

The dice game scenario assumed in this paper is described as follows. Two dice are thrown on an initially empty board. The game consists of keeping on the table the die with the highest face value while the other die is replayed. Both dice are withdrawn from the table when their faces show the same figure. Some example rounds of this game are shown in Figure 2. Typical outputs of the vision system are represented in the subfigure captions, where a, b and d are constants for dice faces, [ ] represents an empty state (no dice on the table), and t_i ($i \in [1, \ldots, 5]$) are time points.



(a) state([a,d],t1). (b) state([d],t2). (c) state([d,d],t3). (d) state([ ],t4). (e) state([c],t5).

Fig. 2. Five states of the dice game and their typical representation output from the vision system.

Given no knowledge about the numerical value of the die faces, the rules of the game can be expressed most succinctly using three general rules:

- An empty board is followed by a board with 1 die on it;
- A board with one die, D, on is followed by a board with D and another one on;
- A board with two equal dice face showing is followed by an empty board;

and fifteen specific rules relating to the ordering of dice faces:

- A board with dice faces 1 and 2 showing is followed by a board with a single die with face 2 showing;
- A board with dice faces 1 and 3 showing is followed by a board with a single die with face 3 showing;
- etc.

The rules could be expressed in different ways, (in particular, the third general rule above could be expanded into a set of six rules specific to each pair of equal dice faces showing). However, we felt that these 18 game rules provided a balance between the kinds of general and specific rules we would expect a discovery system to find.

3.2 Setup for Progol

In the first set of experiments with Progol, subsequent states in each data set were re-written into a predicate representing the transition between pairs of state, $trans/4$. For instance, the sequence of states represented in Figure 2 would be re-written as:

$$trans([a,d],[d],t1,t2), trans([d],[d,d],t2,t3),$$
$$trans([d,d],[\,],t3,t4), trans([\,],[c],t4,t5).$$

Given that all 18 game rules we are interested in can be expressed in terms of a transition from one board state to another, this translation enabled us to use Progol in a single run to learn a set of rules for predicting such transitions, from which we could hopefully extract rules of the game. After some experimentation with Progol's mode declarations, we found a suitable mode declaration which enabled Progol to solve the problem. An example theory and an analysis of the theories it produced to solve the problem are given in section 4. Note that relevant facts about any tuples of state transitions could be generalised from the data if appropriate representations are assumed, but in this paper we only discuss transitions between pairs of states.

Progol's answer is always going to be the rule (or set of rules) that best generalises a given example set. However, if only (potentially noisy) positive examples constitute this set, the rules that best generalise the data set may not provide the best rules for representing interesting concepts about a particular application domain. This is due to the fact that the output rules may predict some of the noisy data. In effect, in Progol it is possible to manually set the amount of noise that can be predicted by the generalised rules. This, however, is not an option if Progol is to be incorporated in an autonomous agent to generalise vision data, as the percentage of noise present in the data depends on the fluid conditions of the environment the agent is immersed in.

Hence, as another approach to using Progol, in order to filter out rules that predict noise, we have also used a macro method as described in [22]. This consists of applying Progol to multiple example sets, as well as using various distinct mode declarations for each set. The result of this procedure is, therefore, a collection of generalisations about a particular concept. Rules in this collection are ranked according to a voting procedure, which takes into account the number of occurrences of each formula in the resultant answer sets. We have experimented in this fashion in [22], but we do not report on this approach here.

3.3 Setup for HR

As with Progol, some manipulation of the symbolic information provided by the vision system was required. HR cannot yet handle list representations, so the state predicates such as *state([a,c],t1)* were split into three sets of background predicates for HR. These were: (i) *0state/1* predicates, which indicate time points when the board was empty, (ii) *1state/2* predicates, which indicate the single object on a board at a particular time, and (iii) *2state/2* predicates, which indicate the pair of objects on a board at a time point. Note that HR is able to write definitions of concepts differently to how they are used internally, so we were able to disguise the three different predicates and add some syntactic sugar which made HR's output appear as if it were handling lists. In

addition to the three predicates indicating states of boards, HR was also supplied with the *successor/2* predicate from the vision system.

We performed some initial experimentation using different sets of production rules taken from the five described in section 2.3 above. We decided not to use the negate rule, but did use the match, exists, compose and split rules. HR was told to form equivalences, implications and near-implications, but not near-equivalences as we found this led to a duplication of effort. HR was also told to extract implicates from the conjectures formed, and to only report the implicates to the user. In addition to the background concepts, we also supplied HR with some axioms pertaining to the working of the vision system. In the format of the Otter theorem prover, these were:

```
all a b c ((1state(a,b) & 1state(a,c)) -> (b=c)).
all a b c d e ((2state(a,b,c) & 2state(a,d,e)) -> (b=d & c=e)).
all a (0state(a) -> (-(exists b (1state(a,b))))).
all a (0state(a) -> (-(exists c d (2state(a,c,d))))).
all a ((exists b (1state(a,b))) -> -(0state(a))).
all a ((exists b (1state(a,b))) -> -(exists c d (2state(a,c,d)))).
all a ((exists b c (2state(a,b,c))) -> -(0state(a))).
all a ((exists b c (2state(a,b,c))) -> -(exists d (1state(a,d)))).
all a (exists b (successor(a)=b)).
all a (exists b (successor(b)=a)).
all a b c ((successor(a)=b & successor(c)=b) -> (a=c)).
all a b c ((successor(a)=b & successor(a)=c) -> (b=c)).
```

These do not express anything about the actual dice game, but do state some obvious aspects of the vision system, e.g., that only one state can occur at the same time and that every time point has a successor. Any hypothesis that HR makes which can be proved to follow from these axioms will not relate to the game being played, hence can be discarded. Hence, HR was told to use Otter with these axioms to attempt to prove every implicate it produces. Any proved implicate was hidden from the output.

There are a large number of settings that can be altered before HR is asked to form a theory. We experimented with altering the following:

- The number of theory formation steps allowed before HR stopped. A theory formation step is an attempt to build a new concept from old ones (which may result in a new concept, an equivalence conjecture or a non-existence conjecture).
- The complexity limit for the concept formation. This is a limit on the number of theory formation steps required to produce a concept. For instance, the default is set to 6, so if a concept has been produced as the result of six theory formation steps, no further concepts can be built from it. One effect of this constraint is that it imposes a limit on the number of literals in the clausal definitions HR produces.
- The plausibility measure for near-implications as described in section 2.3 above. We tried values ranging from 50% to 100%.
- The heuristic search for concepts. We experimented with (i) a simple breadth first search (ii) a *split-first* search, where the split production rule was used greedily before any other and (iii) a *compose-last* search, where all production rules except compose

Dataset	Number of states	Number of faces	Mis-categorisations (%)	Coverage
4	55	72	25	0.47
6	76	99	4	0.73
7	69	96	25	0.67
8	108	147	8	0.73
9	58	84	12	0.47
10	72	96	8	0.47
11	61	27	7	0.47
12	61	78	11	0.53
all	560	699	12	0.93

Table 1. Details of observations made by the vision system

are used greedily, with the compose rule only being employed when no other theory formation steps are available.

4 Results

The observations from the vision system over eight distinct sessions are summarised in table 1. In particular, we recorded the number of faces that were shown and the proportion of times a die face was mis-categorised by the system. We also recorded the proportion of all possible transitions from two faces to one face that occurred during each session and were correctly recorded (for reasons given later). For instance, if in a particular session, a board with dice faces a and d showing was followed by one with just die face d showing, this is counted in the *coverage* of 2-to-1 transitions observed during the session. Finally, details of a dataset with *all* the observations is given, as this was also used in learning sessions.

We were primarily concerned with the sensitivity of the learning approaches we experimented with. In particular, we measured the proportion of the 18 game rules from section 3.1 which were found in the theories produced by HR and Progol, with details provided below. Of secondary importance were the selectivity and the effi ciency of the systems. In both accounts, Progol performed better than HR. For instance, the number of rules produced by Progol for data set 4 was 17, whereas HR produced nearly four hundred, and in general the size of HR's theories were an order of magnitude larger than Progol's. Similarly, while Progol took 4 seconds to complete the task for data set 4, HR took around 200 seconds, and in general, HR was more than an order of magnitude slower than Progol. It is worth noting, however, that, unlike Progol, HR's search space doesn't change a great deal if the amount of data it is provided with increases. This is highlighted by the fact that with the largest data set, namely the *all* data, Progol took 260 seconds to process this, while HR took around 400 seconds. Hence it does appear that HR scales better with respect to data size.

4.1 An Example Progol Theory

In the session described here, Progol was supplied with the vision data from data set 6, represented as described in section 3.2 above, and the appropriate mode declarations. After a single run, it found rules (1) to (19) – given below in the order produced by Progol – to solve the learning problem. Note that the symbols A and B are variables over dice faces; C and D are variables over time points, and a, b, c, d, e and f are constants encoding the emergent classes representing the dice faces.

$$trans([\,], [A], C, D) :- successor(D, C). \quad (1)$$
$$trans([A], [A, B], C, D) :- successor(D, C). \quad (2)$$
$$trans([A], [B, A], C, D) :- successor(D, C). \quad (3)$$
$$trans([b, a], [b], C, D). \quad (4)$$
$$trans([A, B], [\,], C, D) :- successor(D, C). \quad (5)$$
$$trans([f, d], [f], C, D). \quad (6)$$
$$trans([c, f], [f], C, D). \quad (7)$$
$$trans([a, f], [f], C, D) :- successor(D, C). \quad (8)$$
$$trans([f, c], [f], C, D) :- successor(D, C). \quad (9)$$
$$trans([c, e], [e], C, D). \quad (10)$$
$$trans([e, a], [e], C, D). \quad (11)$$
$$trans([e, f], [f], C, D) :- successor(D, C). \quad (12)$$
$$trans([d, f], [f], C, D) :- successor(D, C). \quad (13)$$
$$trans([c, a], [c], C, D). \quad (14)$$
$$trans([c, d], [d], C, D). \quad (15)$$
$$trans([f, a], [f], C, D) :- successor(D, C). \quad (16)$$
$$trans([f, b], [f], C, D). \quad (17)$$
$$trans([f, e], [f], C, D) :- successor(D, C). \quad (18)$$
$$trans([e, b], [e], C, D). \quad (19)$$

Rules (1) to (19) include most of the 18 rules of the dice game we identified in section 3.1. Rules (1),(2) and (3) represent two of the three general game rules we were looking for. In particular, rule (1) represents the fact that the subsequent state of an empty state is a state with only one object (one die face). Rules (2) and (3) represent the game rule that from a state with one die on the table, a state with two dice always follows. Progol did not find the general rule that a board state with two equal dice faces showing is followed by an empty board. Rule (5) is close to this rule, but is an over generalisation. Rules (4) and (6) to (19) represent the ordering between the faces of the dice. In total, Progol reproduced 13 out of the 18 game rules we were looking for, hence scored 13/18 = 0.72 for sensitivity.

The same method applied to data set 7, however, only produced a sensitivity value of 0.61. Part of the problem was due to the over generalisation of rules. For example,

rule 20 below was produced:

$$trans([A],[B,C],D,E) :- successor(E,D). \quad (20)$$

This is a generalisation of rules (2) and (3) above. While this was useful for solving the learning problem, the generalisation did not represent a relevant fact about the observed scenario. The reason rule (20) was learned rather than the two more specific ones was because the vision system made errors in the classification of the objects on the board. In particular, data set 7 had an error rate of 25%. These led Progol to produce similarly incorrect rules, such as these below:

$$trans([a,a],[a],A,B) :- successor(B,A). \quad (21)$$
$$trans([b,a],[e],A,B). \quad (22)$$
$$trans([a,c],[b],A,B). \quad (23)$$
$$trans([a,a],[b],A,B). \quad (24)$$
$$trans([b,c],[f],A,B). \quad (25)$$
$$trans([c,a],[e],A,B). \quad (26)$$

It is worth noting that Progol constructed some of the rules above using the predicate $successor/2$, whereas it ignored it in some others, where probably there were fewer examples to support them. It can be argued that the relation representing the succession of time points could be dropped in this data set as it is implicit in the representation of transitions as $trans/4$ relations. $successor/2$ statements were kept in the representation mainly in order to differentiate rules about transitions between pairs of states from rules about the transition between longer sequences of states.

4.2 An Example HR Theory

We will look at the theory produced by HR using a complexity limit of 5, 4000 theory formation steps, a near-implication plausibility of 50% and a breadth first search. This contained 544 implicates, of which 151 were proven to follow from the axioms supplied to Otter (as described in section 3.3). For instance, the following theorem was proved: `2state(A,B,B) :- 2state(A,C,B),2state(A,B,D)`. This states that if a board state has object B in position 2 with something else in position 1, and it has object B in position 1 with something else in position 2, then it must have object B in both positions, which is an uninteresting side effect of the vision process. Such proved theorems were not shown to the user, and hence only 393 were presented.

The following rules in the theory were identified as relating to the 18 game rules:

```
state(A,[C]) :- successor(A,B), state(B,[nil]).
state(A,[nil]) :- successor(A,B), state(B,[C,C]).
state(A,[b]) :- successor(A,B), state(B,[b,a]).
state(A,[d]) :- successor(A,B), state(B,[d,a]).
state(A,[e]) :- successor(A,B), state(B,[e,a]).
```

```
state(A,[f]) :- successor(A,B), state(B,[f,a]).
state(A,[d]) :- successor(A,B), state(B,[b,d]).
state(A,[e]) :- successor(A,B), state(B,[b,e]).
state(A,[f]) :- successor(A,B), state(B,[b,f]).
state(A,[e]) :- successor(A,B), state(B,[e,d]).
state(A,[f]) :- successor(A,B), state(B,[f,d]).
state(A,[f]) :- successor(A,B), state(B,[e,f]).
```

As these cover 12 of the 18 game rules, HR scored 12/18 = 0.67 for sensitivity. It is interesting to note that, as in the large majority of sessions, HR found the general result that a board state with two equal faces showing is followed by an empty board state, whereas in the majority of sessions, Progol did not find this. Conversely, while Progol often found the general rule that a board state with one face showing is followed by a board state with that face showing, and another one, HR never found this rule. Theoretically, HR should have found this rule, and we are currently investigating its failure.

Note also that in this session, HR did not find the rule that a board state with face b and face f showing is followed by a board state with just face f showing, whereas Progol identified this as rule (17) above. This was due to an interesting anomaly in the data: whenever face b was showing with something else, it was always showing with face f. Thus, when HR tried to invent the concept of board states with both b and f showing, it was conjectured to be equivalent to the concept of board states with b and *anything* else showing, hence the new concept was not allowed into the theory. This had the eventual effect that the rule we expected did not appear in the theory produced by HR. There are ways to stop HR making equivalences which may help in this situation, but may also greatly enlarge the search space. We are currently exploring these possibilities.

In other sessions, where the vision mis-categorisation rate was higher, like Progol, HR also produced false results such as:

```
state(A,[a]) :- successor(A,B), state(B,[a,b])
```

These were produced because they were above the plausibility threshold of 50%, as there were no correct situations recorded where state [a,b] was followed by state [b]. In the larger datasets, however, such incorrect rules were less common. Also like Progol, HR produced some over- generalised rules such as this:

```
state(A,[D,E]) :- successor(A,B), state(B,[C,D])
```

which is the conjecture that a 1-object board is followed by a 2-object board, but doesn't refer to the retention of the object on the board. HR also produced many incorrect conjectures about the predecessors of board states, for example:

```
state(B,[b]) :- successor(A,B), state(A,[b,f])
```

which is the conjecture that board states with faces b and f showing were produced after a board state with just b on it. While these do not reflect correct facts about the board game, they wouldn't affect the way in which an agent would play the game, so to some extent, they are harmless.

4.3 Sensitivity

To gain an impression of the sensitivity of our approaches, we ran Progol as described in section 3.2 for each of the eight vision observation datasets, and a ninth data set consisting of all the data taken over the 8 session. We ran HR for 4000 theory formation steps using a breadth first search, a complexity limit of 5 and a near-implication minimum plausibility of 50%. We then recorded the proportion of the 18 rules identified in section 3.1 which were found in the output from the systems. We then repeated the session, but using a heuristic search where the split production rule is used greedily as described in section 3.3 above. We ran a third session with HR using a compose-last search, also described in section 3.3. Note that, for example, we did not require the rule that board states [a,f] are followed by board state [f] to be in the output, if the rule that board states [f,a] are followed by board state [f] was in the output. In table 2, we have recorded the sensitivity results from these sessions, along with the coverage of 2-state to 1-state board sequences as described above.

Dataset	Coverage	Progol	HR (breadth first)	HR (split first)	HR (compose last)
4	0.47	0.44	0.44	0.44	0.44
6	0.73	0.72	0.67	0.67	0.67
7	0.67	0.61	0.61	0.56	0.61
8	0.73	0.67	0.67	0.67	0.67
9	0.47	0.5	0.33	0.44	0.33
10	0.47	0.44	0.44	0.44	0.44
11	0.47	0.44	0.39	0.44	0.39
12	0.53	0.5	0.5	0.5	0.5
all	0.93	0.83	0.83	0.83	0.83

Table 2. Sensitivity analysis for HR and Progol on 9 datasets

4.4 Further Experiments with HR

As this is one of the first applications of HR to data with noise, we undertook various additional experiments to see how the sensitivity of the system was affected by changes in three parameters (set before the theory formation session begins). We first altered the number of theory formation steps HR was allowed to perform, ranging from 1000 to 6000. The runs were performed using a breadth first search, a near-implication plausibility constraint of 50%, and a complexity limit of 5. The results are presented in table 3 of the appendix. Next, we altered the complexity limit for the sessions, with HR performing runs with complexity limit 4, 5 and 6. The number of steps allowed was 4000, the search strategy was breadth first, and the plausibility of near-implications was constrained to 50%. The results are presented in table 4 of the appendix. In the final set of sessions, we experimented with the plausibility measure, and ran sessions with the measure at 50%, 70% and 100%, again with a complexity limit of 4, a breadth first search and 4000 theory formation step limit. The results are presented in table 5 of the appendix.

4.5 An Analysis of the Results

Looking at table 2, we can see that HR and Progol perform as well as could be expected on the particular learning task we set. Given that very little background information was supplied, it is difficult to imagine a learning program hypothesizing that a particular board state is followed by another particular board state if that sequence of events was never observed. Hence, as 15 of the 18 game rules were about particular sequences of boards, we could not expect Progol or HR to score much more than the coverage measure provided in tables 1 and 2. Hence, measured against this, we see that the sensitivity results were always in the same range as the coverage, so the systems were doing as well as could be expected. For this particular task, Progol outperformed all of HR's searches for data sets 6 and 9, and for at least one of HR's searches for data set 7 and 11. However, the margin of difference isn't great enough to come to any conclusion about whether either approach was more suitable. The theories produced by HR and Progol are used by the autonomous agent to select the next move in a game. To do this, it orders the rules in terms of the support they have from the data, then chooses the rule nearest to the top of this list which applies to the particular game playing situation. Hence, the fact that HR produced many more rules than Progol isn't debilitating for the agent. It is more worrying that both Progol and HR found incorrect game rules, as these could be chosen for playing the game. However, it is likely that such rules would be lower down the agent's list than the correct rules.

The results in the appendix show that altering parameters affects the sensitivity of HR in predictable ways. In particular, from table 3, we see that increasing the number of theory formation steps increases the sensitivity accordingly. However, for the dice problem, the increase reaches a limit after HR is allowed 4000 steps. This highlights the fact that experimentation with HR's step limit is required and perhaps an iterated deepening search (IDS) rather than a depth-limited breadth first search may be more appropriate in cases where experimentation is not possible. With the increase in complexity, as portrayed in table 4, the difference is more dramatic. Clearly, the concepts required to derive the rules of the game are of complexity 5, as the sensitivity at complexity 4 is never more than 0.06. This shows that it is also important to experiment with the complexity limit, and again, an IDS search strategy might be appropriate. Table 4 also shows that, when using a higher complexity limit, additional theory formation steps may be required. This is most notable with the *all* data set: the search at complexity 5 found rules with 0.83 selectivity for both 4000 and 5000 steps, but the search at complexity 6 found rules scoring only 0.5 and 0.78 for sensitivity respectively.

We see from table 5, that, as expected, decreasing the near-implication plausibility threshold improves the yield of correct game rules returned. This flexibility helps overcome the errors in the data from the vision system. For example, in data set 7, the rule that board states with two equal faces is followed by an empty board was empirically true only two thirds of the time. Hence, with a 100% or a 70% minimum requirement on plausibility, this rule was not found by HR, but with the setting lowered to 50%, it was found. We also note from table 2 that the difference in sensitivity between the different heuristic searches is not particularly marked, with the split-first approach occasionally outperforming the breadth-first approach and vice-versa (the compose-last approach always produced the same sensitivity as the breadth first approach).

5 Conclusions

We have presented and motivated the problem of learning rules of a dice game from vision data, in the larger context of the requirement for autonomous agents to learn understandable, logical, rules about protocol behaviours. We have presented a predictive ILP approach to this task using the Progol system and a descriptive ILP approach using the HR system, and we have compared and contrasted the systems in terms of their sensitivity when learning a particular required rule set. Moreover, we have run some additional tests on HR in order to understand it's behaviour in the presence of noisy and incomplete data. We conclude that: (i) Progol and HR perform roughly equally in terms of sensitivity (ii) both perform the task as well as could be expected given the data, and (iii) altering parameters for HR's search alters its sensitivity in predictable ways.

In many ways, Progol is the wrong tool for learning tasks similar to that presented here. This is firstly because, in the general case, it will not be possible to specify a single categorisation problem from which all the relevant rules of the game can be learned. The dice game presented here was a special case, where all rules could be output as the answer to a single prediction problem, but in the general case, it will require a human to correctly specify a set of such problems if predictive approaches are used. An alternative that has been explored in [22] is to follow a macro approach whereby a meta-program automatically produces a set of problems to solve, and this approach has some promise. Secondly, predictive systems are designed to generate rule sets which perform as well as possible in terms of predictive accuracy for unseen test sets. This does not necessarily mean that they will find the most succinct, pertinent or otherwise interesting rules of protocol behaviour. As we saw with Progol, given noisy data, prediction systems may produce faulty rules in order to increase the predictive accuracy of the answer.

In theory at least, HR is the right tool for learning tasks similar to that presented here. This is because it is not constrained to solving a particular problem and hence can find rules relating to many different aspects of the background information. Also, as it doesn't have to worry about predictive accuracy, it will only supply faulty rules if they are sufficiently expressed in the given data. In practice, unfortunately, such an all-encompassing approach means searching a huge space of concepts and conjectures, and this may mean that rules of interest are not found efficiently enough.[2] Even with the few background predicates for the dice problem, HR often took up to five minutes to find the rules, and usually produced 50 rules to every one which was required.

Our experiments and analysis suggest that a combination of predictive and descriptive approaches may be more suitable than one alone. For instance, a descriptive system could be used to form an initial theory which may contain some interesting rules. If sufficiently many rules were found, then these could guide the use of a predictive learning program, e.g., by suggesting background concepts and mode declarations for Progol. Certain rules to solve the problem would be learned quickly, and may be as useful as the ones found by the descriptive system. This compromise may provide the best possible approach to learning in situations where autonomous agents have to rely on observations to understand and adapt to their environment.

[2] HR has hundreds of possibilities, however, for heuristic searches which may solve this problem (see chapters 8 and 9 of [2]). We plan further experimentation to address this possibility.

References

1. S Colton. Refactorable numbers: a machine invention. *Journal of Integer Sequences*, 2, 1999.
2. S Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.
3. S Colton. The HR program for theorem generation. In *Proceedings of the Eighteenth Conference on Automated Deduction*, 2002.
4. S Colton and S Muggleton.. ILP for mathematical discovery. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, 2003.
5. S Colton, A Bundy, and T Walsh. Automatic identification of mathematical concepts. In *Machine Learning: Proceedings of the 17th International Conference*, 2000.
6. S Colton, A Bundy, and T Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.
7. S Colton and S Huczynska. The Homer system. In *Proceedings of the 19th International Conference on Automated Deduction (LNAI 2741)*, pages 289–294, 2003.
8. L De Raedt and L Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
9. L Dehaspe and H Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
10. A. Fern, R. Givan, and J. Siskind. Specific-to-general learning for temporal events with application to learning event definitions from video. *Journal of Artifi cial Intelligence Research*, 17:379–449, 2002.
11. R King, K Whelan, F Jones, P Reiser, C Bryant, S Muggleton, D Kell, and S Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.
12. D. Magee, D.C. Hogg, and A.G. Cohn. Autonomous object learning using multiple feature clustering in dynamic scenarios. Technical Report 2003.15, School of Computing Research Report, University of Leeds, 2003.
13. D. R. Magee. Tracking multiple vehicles using foreground, background and motion models. *Image and Vision Computing*, 20(8):581–594, 2004.
14. J Matthews, S Engberg, M Montemerlo, J Pineau, N Roy, J Rogers, S Thrun, S Handler, T Starrett, D Ting, and R Travis. The nursebot project: Results of prelimiminary field studies during development of a personal robotic assistant for older adults. In *Proceedings of the Greater Pittsburgh 14th Annual Nursing Research Conference*, 2002.
15. W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.
16. W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical Report ANL/MCS-TM-194, Argonne National Laboratories, 1994.
17. S Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13, 1995.
18. S Muggleton. Learning from positive data. In *International Conference on Inductive Logic Programming*, 1996.
19. S Muggleton. CProgol4.4: a tutorial introduction. In *Relational Data Mining*, pages 160–188. Springer, 2001.
20. C Pinhanez and A Bobick. It/i: a theater play featuring an autonomous computer character. *Presence: Teleoperators and Virtual Environments*, 11(5):536–548.
21. J Quinlan and R Cameron-Jones. FOIL: A midterm report. In *Proceedings of the 6th European Conference on Machine Learning (LNAI 667)*. Springer, 1993.
22. P Santos, D Magee, and A Cohn. Combining multiple answers for learning mathematical structures from visual observation. In *Proceedings of the European Conference on Artifi cial Intelligence*, 2004 (forthcoming).
23. A. Stehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

Appendix

Dataset	1000 steps	2000 steps	3000 steps	4000 steps	5000 steps	6000 steps
4	0.06	0.17	0.44	0.44	0.44	0.44
6	0.06	0.33	0.67	0.67	0.67	0.67
7	0.06	0.22	0.56	0.61	0.61	0.61
8	0.06	0.33	0.67	0.67	0.67	0.67
9	0.06	0.28	0.33	0.33	0.33	0.33
10	0.06	0.22	0.44	0.44	0.44	0.44
11	0.06	0.28	0.39	0.39	0.39	0.39
12	0.06	0.22	0.5	0.5	0.5	0.5
all	0.06	0.28	0.72	0.83	0.83	0.83

Table 3. Sensitivity variation with number of steps

Dataset/steps	Complexity 4	Complexity 5	Complexity 6
4	0.06/0.06	0.44/0.44	0.44/0.44
6	0.06/0.06	0.67/0.67	0.56/0.67
7	0.06/0.06	0.61/0.61	0.39/0.56
8	0.06/0.06	0.67/0.67	0.67/0.67
9	0.06/0.06	0.33/0.33	0.33/0.33
10	0.06/0.06	0.44/0.44	0.44/0.44
11	0.06/0.06	0.39/0.39	0.39/0.39
12	0.06/0.06	0.5/0.5	0.5/0.5
all	0.06/0.06	0.83/0.83	0.5/0.78

Table 4. Sensitivity variation with complexity and for 4000 & 5000 steps

Dataset	100% requirement	70% requirement	50% requirement
4	0.38	0.44	0.44
6	0.61	0.61	0.67
7	0.56	0.56	0.61
8	0.61	0.61	0.67
9	0.28	0.28	0.33
10	0.28	0.33	0.44
11	0.39	0.39	0.39
12	0.33	0.33	0.5
all	0.61	0.72	0.83

Table 5. Sensitivity variation with conjecture plausibility