Learning First-order Relations from Noisy Databases using Genetic Algorithms

Man Leung Wong
Department of Computing & Decision Sciences
Lingnan University, Tuen Mun
Hong Kong
mlwong@ln.edu.hk

Kwong Sak Leung
Department of Computer Science
The Chinese University of Hong Kong
Hong Kong
ksleung@cse.cuhk.edu.hk

Abstract

In knowledge discovery from databases, we emphasize the need for learning from huge, incomplete and imperfect data sets (Piatetsky-Shapiro and Frawley, 1991). To handle noise in the problem domain, existing learning systems avoid overfitting the imperfect training examples by excluding insignificant patterns. The problem is that these systems use a limiting attribute-value language for representing the training examples and induced knowledge. Moreover, some important patterns are ignored because they are statistically insignificant. This paper describes a system called **GLPS** that combines Genetic Algorithms and a variation of **FOIL** (Quinlan, 1990) to learn first-order concepts from noisy training examples. The performance of **GLPS** is evaluated on the chess endgame domain. A detail comparison to **FOIL** is accomplished and the performance of **GLPS** is significantly better than that of **FOIL**. This result indicates that the Darwinian principle of natural selection is a plausible noise handling method which can avoid overfitting and identify important patterns at the same time.

1. Introduction

In knowledge discovery from databases, we emphasize the need for learning from huge, incomplete and imperfect data sets (Piatetsky-Shapiro and Frawley, 1991). Existing inductive learning systems employ noise-handling mechanisms to cope with different kinds of data imperfections such as noise, insufficiently covered example space, inappropriate description language and missing values in the training examples (Dzeroski and Lavrac, 1993).

However, these learning systems use attribute-value language for representing the training examples and induced knowledge and allow a finite number of objects in the universe of discourse. This representation limits them to learn only propositional descriptions in which concepts are described in terms of values of a fixed number of attributes. Recently, there has been increasing interests in systems that induce first-order logical expressions. In this formalism, domain knowledge represented in the forms of relations can be used in the induced relational descriptions of concepts. For example, **FOIL** (Quinlan, 1990) efficiently learns function free Horn clauses, a useful subset of first-order predicate logic. It uses a top-down, divide and conquer approach guided by information-based heuristics to produce a concept description that covers all positive examples and excludes all negative examples.

Nevertheless, only a few relation learning systems such as **FOIL** and **LINUS** (Dzeroski and Lavrac, 1993) address the issue of learning from imperfect data. This paper describes a system called **GLPS** that combines a variation of **FOIL** and Genetic Algorithms to learn first-order concept from noisy examples. It gives an empirical comparison of **GLPS** and **FOIL** in the domain of learning illegal chess endgame positions from noisy examples. Section 2 of this paper presents a high level description of **GLPS**, followed by the description of the mechanism used to generate the initial population of concepts. One of the genetic operators, crossover, is detailed in section 4. Section 5 presents the noise handling methods of **FOIL** and **GLPS**. The experiment results are presented in the section 6. The last section is the conclusion.

2. Genetic Logic Programming System (GLPS)

The task of inducing first-order concept can be formulated as a search problem (Mitchell, 1982) in a hypotheses space of first-order concepts. Various approaches differ mainly in the search strategy used and the heuristics used to guide the search. Since the search space is extreme large, strong heuristics are required in order to manage the problem. Most first-order concept learning systems are based on a greedy search strategy. The systems generate a sequence of first-order concepts from general to specific ones (or from specific to general) until a consistent target concept is found. Each concept in the sequence is obtained by specializing or generalizing the previous one. For example, **FOIL** applies the hill climbing search strategy guided by an information-gain heuristics to search concepts from general to specific ones. However, these strategy and heuristics are not always applicable because they may trap the systems in a local maxima.

An alternate search strategy is Genetic Algorithm (**GA**) which performs parallel searches implicitly (Holland, 1975; Goldberg, 1989). Genetic Algorithm perform both exploitation of the most promising solutions and exploration of the search space. It is featured to tackle hard search problems and thus it may be applicable to first-order concept induction.

In **GLPS**, populations of first-order concepts are genetically bred using the Darwinian principle of survival and reproduction of the fittest along with genetic crossover operation appropriate for mating first-order concepts. **GLPS** starts with an initial population of first-order concepts generated randomly, induced by other learning systems, or provided by the user. In this paper, the initial concepts are learned by a variation of **FOIL**. For concept learning, each individual concept in the population is measured in terms of how well it covers positive examples and excludes negative examples.

The initial concepts in generation 0 are normally incorrect and have poor performance. However, some individuals in the population will be fitter than the others. The Darwinian principle of reproduction and survival of the fittest and the genetic operation of sexual crossover are used to create new offspring population of concepts from the current population. The reproduction operation involves selecting a concept from the current population and allowing it to survive by copying it into the new population. The selection is based on either fitness (fitness proportionate selection) or tournament (tournament selection).

The genetic process of crossover is used to create two offspring concepts from the parental concepts selected by either fitness proportionate or tournament selection. The parental concepts are usually of different sizes and structures and the offspring concepts are composed of clauses, literals from their parents. These offspring concepts are typically of different sizes and structures from their parents. The new generation replaces the old generation after the reproduction and crossover operations are performed on the old generation. Fitness of each concept in the new generation is estimated and the above process is iterated over many generations until the termination criterion is satisfied.

This algorithm will produce populations of concepts which tend to exhibit increasing average fitness in producing correct answers for the training examples. **GLPS** returns the best concept found in any generation of a run as the result.

3. Generate the initial population

The fundamental difficulty in **GLPS** is to represent first-order concepts appropriately so that initial population can be generated easily and the genetic operator such as crossover and reproduction can be performed effectively. A first-order concept can be represented as a forest of AND-OR trees. The leaves of an AND-OR tree are positive or negative literals generated using the predicate symbols and terms of the problem domain. For example, the AND-OR tree in figure 1 represents the following first-order concept description:

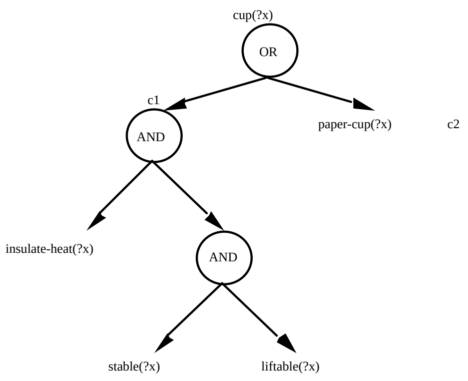


Figure 1: An AND-OR Tree

C1: cup(?x) :- insulate_heat(?x), stable(?x), liftable(?x)
C2: cup(?x) :- paper_cup(?x)

In the experiment presented in this paper, an initial population of first-order concepts are induced by a variation of **FOIL** using a portion of the training examples. Then a forest of AND-OR trees can be generated randomly for each concept learned.

4. Crossover of first-order concepts

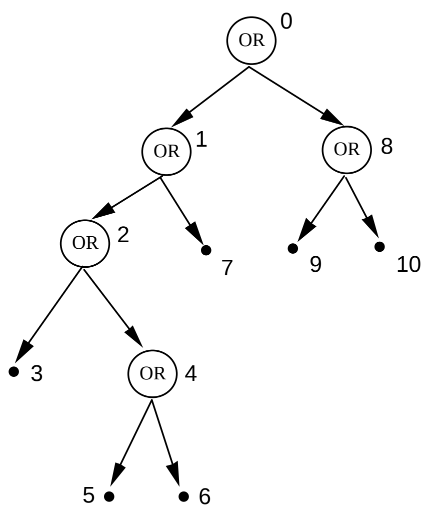


Figure 2: An OR Tree

Components of a first-order concept that are subjected to crossover are the whole concept, rules, clauses and antecedent literals. In **GLPS**, terms of literals cannot be exchanged. Thus crossover components are referred to by a list of numbers. The list can have at most three elements:

1. {} refers to the whole concept.
2. {m} refers to the m^{th} rule in the concept. A rule has one or more clauses.
3. {m, n'} refers to a clause or a number of clauses of the m^{th} rule in the concept where n' is a node number of the corresponding sub-tree. For instance, let the m^{th} rule has N_m clauses which are arranged in an OR-tree (Figure 2). Each leaf in the tree represents a clause. In the example, the tree has six clauses, i.e. $N_m = 6$. There are 11 nodes in the tree, and the number of nodes is denoted by N'_m. n' in the list {m, n'} is between 0 and N'_{m-1}. Thus, { m, n'} represents a clause if n' corresponds to a leaf node. It refers to a set of clauses if n' corresponds to an internal node in the tree.

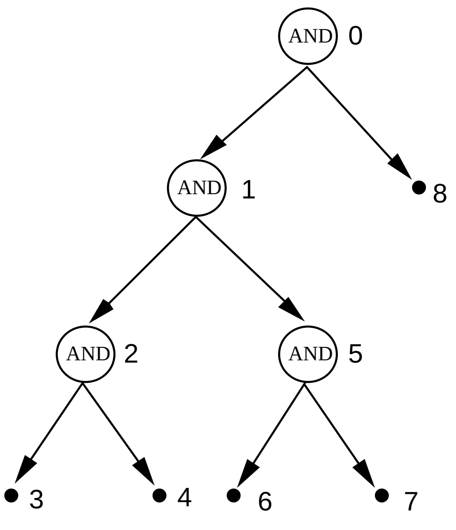


Figure 3: An AND Tree

4. {m, n, l'} refers to a literal or a set of literals of the n^{th} clause of the m^{th} rule where l' is also a node number of the corresponding sub-tree. For example, let the clause has $L_{m,n}$ antecedent literals. These literals are arranged in an AND-tree (Figure 3). Each leaf in the tree represents an antecedent literal and there are 5 antecedent literals, i.e. $L_{m,\ n} = 5$. Let the number of nodes in an AND tree be $L'_{m,n}$ which is 9 for the above tree. The third number in {m, n, l'} can have value between 0 and $L'_{m,\ n-1}$. {m, n, l'} represents a literal if l' refers to a leaf node. It is a set of literals if l' refers to an internal node.

There are four kinds of crossover points represented by the above lists of numbers. Two crossover points are compatible if their representations (i.e. lists) have the same number of elements. In **GLPS**, crossover between two parental concepts can only occur at compatible crossover points. Consider the first-order concept P_a represented in Horn clauses:

C1:	cup(?x) :- insulate_heat(?x), stable(?x), small(?x)
C2:	stable(?x) :- bottom(?x, ?b), flat(?b)
C3:	stable(?x) :- bottom(?x, ?b), concave(?b)

and the concept P_b

C1':	cup(?x) :- insulate_heat(?x), stable(?x)
C2':	stable(?x) :- bottom(?x, ?b), flat(?b), concave(?b), has_support(?x)

The AND-OR trees of P_a and P_b are depicted in figures 4 and 5 respectively. If the crossover points are empty lists {}, the offsprings are identical to their parent and the crossover operation degenerates to reproduction. For this reason, **GLPS** has not reproduction operation which can be emulated by crossover. There is a parameter P_0 which controls the probability of reproduction. The parameter P_1 controls the probability that a list of one element is generated. If the crossover points are {1} and {1} respectively, the offsprings are:

C1: cup(?x) :- insulate_heat(?x), stable(?x), small(?x)
C2': stable(?x) :- bottom(?x, ?b), flat(?b), concave(?b), has_support(?x)

and

C1': cup(?x) :- insulate_heat(?x), stable(?x)
C2: stable(?x) :- bottom(?x, ?b), flat(?b)
C3: stable(?x) :- bottom(?x, ?b), concave(?b)

The parameter P_2 determines the probability that a list of two elements is generated. If the crossover points are {1, 1} for P_a and {1, 0} for P_b, the offsprings are

C1: cup(?x) :- insulate_heat(?x), stable(?x), small(?x)
C2': stable(?x) :- bottom(?x, ?b), flat(?b), concave(?b), has_support(?x)
C3: stable(?x) :- bottom(?x, ?b), concave(?b)

and

C1': cup(?x) :- insulate_heat(?x), stable(?x)
C2: stable(?x) :- bottom(?x, ?b), flat(?b)

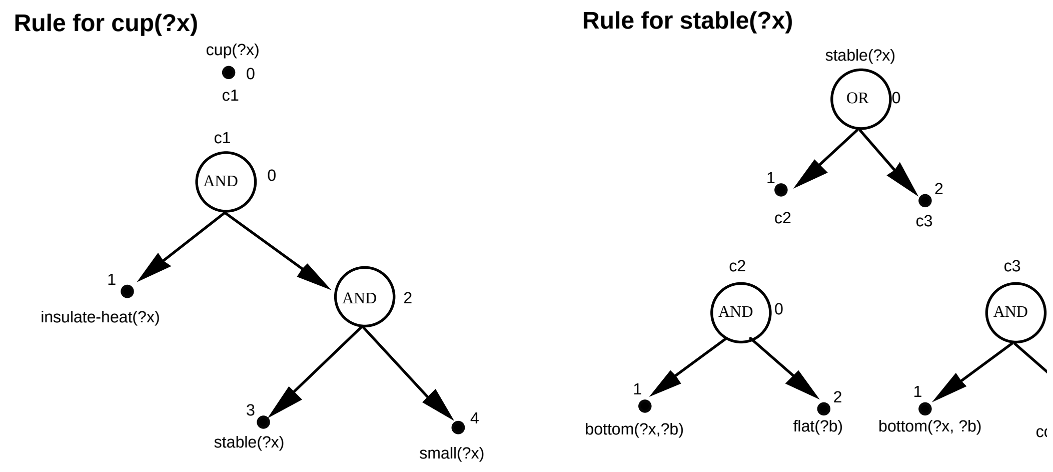


Figure 4: And-Or tree of concept P_a

The parameter P_3 determines the probability that a list of three elements is created. If the crossover points are {1, 2, 0} for P_a and {1, 0, 1} for P_b, the offsprings are

C1: cup(?x) :- insulate_heat(?x), stable(?x), liftable(?x)
C2: stable(?x) :- bottom(?x, ?b), flat(?b)
C3: stable(?x) :- **bottom(?x, ?b), flat(?b)**

and

C1': cup(?x) :- insulate_heat(?x), stable(?x)
C2': stable(?x) :- **bottom(?x, ?b), concave(?b)**, concave(?b), has_support(?x)

5. Noise handling in FOIL and GLPS

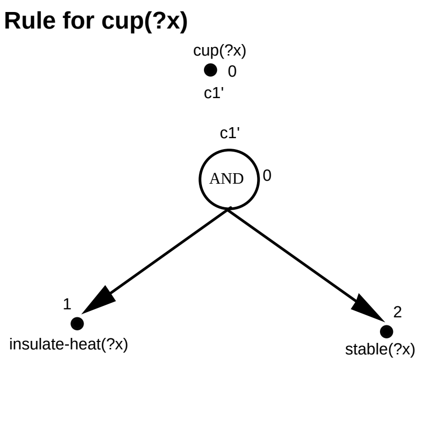
In **FOIL**, the noise handling mechanism is the encoding length restriction. The idea is that the number of bits required to encode the clause should never exceed the total number of bits needed to indicate explicitly the positive training examples covered by the clause. Thus, if a clause covers r positive examples out of n examples in the training set. The number of bits available to encode the clause is $\log_2(n) + \log_2(\binom{n}{r})$. If there are no bits available for adding another literal, but the clause has more than 85% accuracy, it is retained in the induced set of clauses, otherwise it is deleted. This heuristics avoids overfitting the training examples because insignificant literals are excluded from clauses of the inducing concept. The obtained concept description is smaller, simpler, more accurate and more comprehensible. Since **GLPS** employs a variation of **FOIL** to find the initial population of concepts. It uses the same noise handling mechanism of **FOIL**. Currently, no other noise handling method has been implemented.

6. Experiments

A preliminary implementation of **GLPS** is developed in CLOS (Common Lisp Object System). It has been tested on various CLOS implementations and hardware platforms, including CMU Common Lisp on SparcStation, LUCID Common Lisp on DecStation and MCL on Macintosh. This section discusses the performance of **GLPS** on learning concepts from imperfect data. The chess endgame domain is taken from Quinlan (Quinlan, 1990) and the results are compared to the results obtained by **FOIL**. The latest version of **FOIL** (**FOIL6**) is used in this comparison.

In this domain, the target concept illegal(?WKf, ?WKr, ?WRf, ?WRr, ?BKf, ?BKr) states whether the position where the white king at (?WKf, ?WKr), the white rook at (?WRf, ?WRf) and the black king at (?BKf, ?BKr) is not a legal white-to-move position. In **FOIL6** and **GLPS**, the background knowledge is represented by two predicates, adjacent(?X, ?Y) and less_than(?W, ?Z), indicating that rank/file ?X is adjacent to rank/file ?Y and rank/file ?W is less than rank/file ?Z respectively. The training set contains 1000 examples (336 positive and 664 negative examples). The testing set has 10000 examples (3240 positive and 6760 negative examples). Since the current implementation of **GLPS** does not accept declarations of argument type in predicates. The arguments of the background and target predicates are not typed, in order to ease the comparison between **GLPS** and **FOIL**.

Different amounts of noise are introduced into the training examples in order to study the performances of both systems in learning concepts in noisy environment. To introduce n% of noise into an argument ?X of the examples, the value of the argument ?X is replaced by a random value of the same type from a uniform distribution, independent to noise in other arguments. For the class variable, n% positive examples are labeled as negative ones while n% negatives examples are labeled as positive ones. Noise in an argument is not necessarily incorrect because it is chosen randomly, it is possible that the correct argument value is selected. In



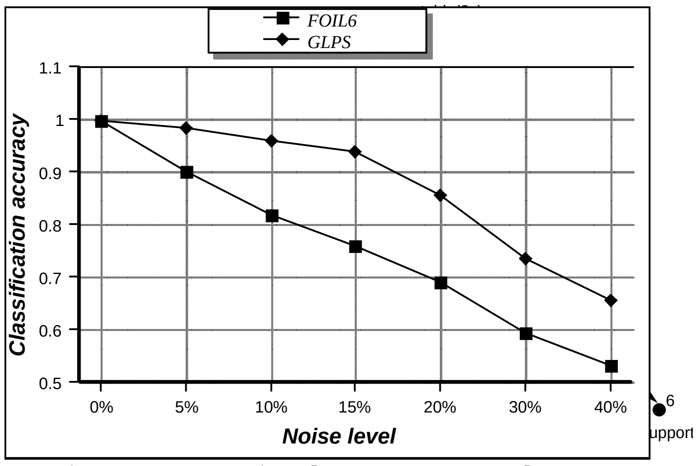


Figure 5: And-Or tree of concept P_b

Figure 6: Comparison between GLPS and FOIL

contrast, noise in classification implies that this example is incorrect. Thus, the probability for an example to be incorrect is $1-\{[(1-n\%)+n\%*\frac{1}{8}]^6*(1-n\%)\}$. In this experiment, the percentages of introduced noise are 5%, 10%, 15%, 20%, 30% and 40%. Thus, the probabilities for an example to be incorrect are respectively 27.36%, 48.04%, 63.46%, 74.78%, 88.74% and 95.47%. Background knowledge and testing examples are not corrupted with noise.

A chosen level of noise is first introduced in the training set. First-order concept descriptions are then induced from the training set using **GLPS** and **FOIL6**. Finally, the classification accuracy of the learned concepts is estimated on the testing set. For **GLPS**, the initial population of concepts are induced by a variation of **FOIL** using a portion of the training examples. The parameters P_0, P_1, P_2 and P_3 are 0.0, 0.1, 0.3 and 0.6 respectively. The population size is 10 and the maximum number of generations for each experiment is 50. Since **GLPS** is a non-deterministic learning system, the process is repeated for five times on the same training set and the average of the five results is reported as the classification accuracy of **GLPS**.

Ten runs of the above experiments are performed on different training examples. The results of ten runs are summarized in figure 6. From this experiment, the classification accuracy of both systems degrades seriously as the noise level increases. Nevertheless, the classification accuracy of **GLPS** is better than that of **FOIL** by at least 5% at the 99.995% confidence interval at all noise levels (except the noise level of 0%.) The largest difference reaches 24% at the 20% noise level. This result is surprising because both systems use the same noise handling mechanism. One possible explanation of the better performance of **GLPS** is that the Darwinian principle of survival and reproduction of the fittest is a good noise handling method. It avoids overfitting noisy examples, but at the same time, it can finds interesting and useful patterns from these noisy examples.

7. Conclusion

In this paper, we describe how to combine Genetic Algorithms and **FOIL** in learning first-order concepts. The performance of **GLPS** in a noisy domain is evaluated by using the chess endgame problem. A detailed comparison to **FOIL** has been conducted. The experiment demonstrates that **GLPS** is a promising alternative to other famous inductive logic programming systems and sometimes is superior for handling noisy data.

Reference

Dzeroski, S. and Lavrac, N. (1993). Inductive Learning in Deductive Databases. *IEEE Transactions on Knowledge and Data Engineering*; **5,** 939-949.
Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.
Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.
Mitchell, T. M. (1982). Generalization as Search. *Artificial Intelligence*; **18**, 203-226.
Piatetsky-Shapiro, G. and Frawley, W. J. (1991). *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI Press.
Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, **5**, 239-266.