Inductive Logic Programming in Haskell - Final Report

Marcus Peacock 0919274

February 12, 2013

1 Inductive Logic Programming

Inductive Logic Programming (ILP) is a research area which brings together Machine Learning and Logic Programming. The aim of ILP is to derive logical hypothesis from a set of facts. Logic Programming provides a uniform representation of relevant knowledge and is thus the structure of deduced hypothesis. An Inductive Logic System is a system which utilises ILP.

2 Knowledge Representation

Before a hypothesis can be constructed, a solid framework for the representation of knowledge must be established. A hypothesis is defined as a proposed explanation for a set of observations, therefore a language for describing these observations and dually a language for describing the hypothesis needs to be selected. A simple structure for the description of observations uses attribute-value objects, an observation can be described as discovering the value of a specific attribute related to an object. A set of attributes are defined along with an associating set of values for each attribute. A natural example involves coin currency, one attribute for a coin might be *Worth* which could use the set {1, 2, 10, 20, 50, 100, 200} and another *Shape* with the set {Circle, Heptagon}. This establishes a basis for describing a certain coin, it may be the case that a tuple system akin to databases is selected, for example a circular coin with a value of 5 may be represented as (5, Circle) with the system having an understanding that these values refer to the attributes Worth and Shape. Another possibility is the conjunction of observations a la [Worth = 5] $\wedge$ [Shape = Circle]. Logic programming uses the language of Horn Clauses, which represents observations as a set of ground facts, our coin could be represented as coin(5, Circle) where coin is now a predicate that takes two arguments of the form (Worth, Shape). This language allows collections of observations to be represented by conjunction and logic programming allows the introduction of variables. In the domain of hypothesis learning the concern is with discovering implications that

hold given a set of ground facts, for example:

$$coin(6, Shape_1) \leftarrow Shape_1 = Circle$$

This implication represents the idea that if a coin has a Worth of 5, then it will also be circular. It is implications similar to these that an Inductive Logic System is interested in learning from a set of ground facts.

In order to maintain a consistent framework for expressions the language of logic programs is often used, this language is detailed in [1] [3]. A logic program consists of a set of rules, which in turn are constructed from atoms of the form p(t_1,...,t_n) where ts are terms and p is a predicate symbol of arity n. Rules are of the form:

$$A_0 \leftarrow A_1, ..., A_m, not\ A_{m+1}, ..., not\ A_n \tag{1}$$

where A_is are atoms and not shares the functionality of logical not. The right hand side is labelled the premise and the left the conclusion. A rule without variables involved are called ground facts.

3 Learning From Examples

Given a language of representation for facts and hypotheses, a definition for what it means for a concept or hypothesis to hold is required. Continuing with the example above, given a ground fact of $coin(5, Circle)$ and the example hypothesis, we can say that this hypothesis covers this examples. For a given hypothesis, the set of positive examples is defined as the set of ground facts for which the hypothesis covers. Similar the negative examples are the examples which are not covered by the hypothesis. More formally, the problem of finding valid hypothesis can be described as follows.

Given a set E of examples including both positive and negative examples of a concept C, find a hypothesis H such that:

Every positive example e in E^+ is covered by H

No negative example e in E^- is covered by H

The overall goal of an Inductive Logic System is to find a hypothesis that is complete, in that it covers all the positive examples, and consistent, in that it covers none of the negative examples. It should be noted that this may not always be possible and different systems have different ways of dealing with these situations, each system has a concept of quality which can be used to evaluate a given hypothesis. For example, quality may be defined based on the number of positive examples it covers and how few negative examples it covers or on the length of the hypothesis or combinations of chosen attiributes.

Consider the following set of rules:

1. $flies(X) \leftarrow bird(X), not\ ab(X)$
2. $bird(X) \leftarrow penguin(X)$
3. $ab(X) \leftarrow penguin(X)$

Along with the following facts about particular birds:

f1. $bird(tweety) \leftarrow$
f2. $penguin(sam) \leftarrow$

$ab(X)$ represents the idea that the ability of X to fly is suspect. Rule 1 expresses that birds whose flying ability we are not suspect of can fly. Rule 2 expresses that penguins are birds and 3 that penguins might not be able to fly. This set of rules can be viewed as the understanding that a bird can fly until we observe otherwise. Fact f1 allows the simple deduction of the fact $flies(tweety)$, establishing this requires that we have $bird(tweety)$ but do not have $ab(tweety)$, which we can gather by inspecting the set of facts. Similarly, looking at fact f2, we can use rule 2 to deduce $bird(sam)$ and rule 3 presents $ab(sam)$. In contrast to $tweety$ we can say that $flies(sam)$ does not hold, since $flies(X)$ requires that no instance of $ab(X)$ exists in the fact set.

4 Language Bias

At this stage it should be recognised that the selection of the language used to represent the ground facts and hypothesis play a significant role in how the Inductive Logic System operates. Another form of bias is the way in which a system searches for a hypothesis. The selection of language is called the language bias and search method bias is called search bias. When designing a system, there is the option of using a less expressive hypothesis language which may reduce the search space considerably and therefore searching and learning become much quicker, however, care must be taken since a less expressive language may mean that a suitable hypothesis cannot be found whereas a system using a more expressive language could.

For example, in order to express the concept that two boolean attributes have the same value an attribute-value language would have to say $([A = false] \wedge [B = false]) \vee ([A = true] \wedge [B = true])$ whereas in a first-order language we can say $A = B$. This is important since one of the ways of measuring the quality of a hypothesis is the ease in which a human can understand it. There have been many different languages chosen for inductive learning systems. The most significant split between these systems is between the systems which focus on learning descriptions of attributes and relational learners, which attempt to learn about the relationships between objects. Two famous inductive learning algorithms, ID3 [4] and AQ [11] [7] use attribute-value based representation. In both these examples, objects are described entirely by a fixed set of attributes. ID3 differs from AQ in the representation of hypothesis, ID3 represents hypothesis using decision trees whereas AQ uses if-then rules. Attribute-value learners are known to be limited due to fact that they cannot take into account background knowledge that would allow for a deeper understanding of the information. The second set of learners are described as relational learners, these systems are designed to reason about the relationships between different objects. Most commonly the language used in these systems is some subset of first-order logic,

5 ILP Example

Inductive Logic Programming was first defined by Muggleton in 1991 [12] [8]. If the interest lies in relation learning, that is, using examples for the target relation and a set of facts to use as the background knowledge in order to find the most likely definition for the target relation in terms of the background knowledge, and the systems uses the language of logic progrems, then it is an inductive logic programming system. The following is a simple example that an ILP system could be presented.

A natural problem to use as an example is the task of learning family relations. Let the task be to find a definition of the relation *son*, any example of this relation will have 2 arguments, allowing the system to first learn it is more specifically trying to learn the definition of son(X,Y) where X and Y are variables representing some object in the background knowledge. Since *son* is a real world example with a known definition, it is possible to present the relation that would correspond to the real world in the form of Horn clauses:

$$son(X, Y) \leftarrow male(X), parent(Y, X)$$

This representation simply states that you are a son if you are male and have a parent. We know that this definition should be complete in the real world, everybody who is a son must have a parent and we also know it is consistent, anybody who is not a son is either not male or does not have a parent. These are two of the measures used to determine if the relation learnt is a valid one, or to come up with some sort of measure for how valid it is. It is also entirely possibly to have multiple complete and consistent relation definitions, for example:

$$son(X, Y) \leftarrow not\ female(X), parent(Y, X)$$

This is a second definition that we know holds since in the real world *male* and *female* are symmetric. This problem may be formulated for an ILP system as follows:

Examples	Background Knowledge
$son(tim, jon) \oplus$	$parent(jon, tim)$
$son(bill, sarah) \oplus$	$parent(tim, sarah)$
$son(sarah, jon) \ominus$	$parent(sarah, bill)$
$son(bill, jon) \ominus$	$parent(sarah, jill)$
	$male(tim)$
	$male(jon)$
	$male(bill)$

In this case, the goal would be to discover a relation which was valid for son(tim, jon) and son(bill, sarah) but specifically did not hold for son(sarah, jon) and son(bill, jon).

6 Noise and Lacking Background Knowledge

Ideally the information presented to an ILP system is error free and has no missing pieces of information, however this is not always possible. It may be the case that a system has to deal with imperfect data and often ILP systems are designed to be able to deal with some level of imperfection. There are 4 [6] main areas of imperfection that are commonly encountered in background knowledge; noise, missing values, too sparse training examples and inexactness of description language. A system which can deal with these problems will be able to solve more problems and be able to come up with more accurate solutions to some problems. When a system is dealing with problematic data the measure for how accurate a discovered definition must be needs to change, it may not even be possible to find a definition which is complete and consistent with some sets of knowledge. Many original ILP systems have been updated to include some methods of handling imperfect data [2] [10] [9] [5].

7 Basis of Logic Programming

This section will formally introduce the terminology of logic programming [3]. Firstly, the *alphabet* consists of variable symbols, predicate symbols and function symbols. *Variable* symbols are represented by a string consisting of an upper case letter followed by lower case letters and/or digits. *Predicate* symbols and *function* symbols are represented by a string consisting of a lower case letter followed by lower case letters and/or digits. Terms are either simply variables or they are functions, represented by a function symbol followed by a list of n terms where the function is said to be an n-ary function. A *constant* is a 0-ary function. Similarly, a predicate symbol followed by a list of n terms is called an *atom*. An atom and its negation are also called *literals*. A *clause* is a formula of the form:

$$\forall X_1 \forall X_2 ... X_i (L_1 \vee L_2 \vee ... L_j) \tag{2}$$

with X_n referring to the variables that occur in the list of L_i literals, variables in a clause may be negated. Negation can also be represened as implication:

$$(L_1 \vee L_2 \vee ... \vee \overline{L_a} \vee \overline{L_b} \vee ...) \tag{3}$$

Which can also be represented as:

$$L_1 \vee L_2 \vee ... \leftarrow L_a \wedge L_b ... \tag{4}$$

The following example illustrates this representation:

$$son(X, Y) \leftarrow male(X) \wedge father(Y, X) \tag{5}$$

This can also be written as:

$$\forall X \forall Y : son(X, Y) \vee \overline{male(X)} \vee \overline{father(Y, X)} \tag{6}$$

A logic program may represent this clause as a set:

$$\{son(X, Y), \overline{male(X)}, \overline{father(Y, X)}\} \tag{7}$$

References

[1] Baral C. and Gelfond M. Logic programming and knowledge representation. *The Journal of Logic Programming, pp.73-148*, 1994.

[2] Kononenko I. Cestnik B. and Bratko I. Asssistant 86: A knowledge elicitation tool for sophisticated users. *Progress in Machine Learning pp.31-45*, 1987.

[3] Lloyd J. Foundations of logic programming. 1987.

[4] Quinlan J. Induction of decision trees. *Machine Learning, pp.81-106*, 1986.

[5] Quinlan J. C4.5: Programs for machine learning. 1993.

[6] Cestnik B. Lazrac N. and Dzerski S. Use of heuristics in empirical inductive logic programming. *Proc. Second International Workshop on Inductive Logic Programming*, 1992.

[7] Hong J. Michalski R., Mozetic I. and Lavrawc N. The multi-purpose incremental learning system aq15 and its testing application on three medical domains. *Proc. Fifth National Conference on Artifical Intelligence, pp.1041-1045*, 1986.

[8] S. Muggleton. Inductive logic programming. *Academic Press, London*, 1992.

[9] Clark P. and Boswell R. Rule induction with cn2: Some recent improvements. *Proc. Fifth European Working Session on Learning pp.151-163*, 1991.

[10] Clark P. and Niblett T. Induction in noisy domains. *Progress in Machine Learning pp.11-30*, 1987.

[11] Michalski R. A theory and methodology of inductive learning. *Machine Learning: An Artificial Intelligence Approach, pp.83-134*, 1983.

[12] Muggleton S. Inductive logic programming. *New Generation Computing, pp.295-318*, 1991.