CS 6604: Data Mining Fall 2007

Lecture 14 — October 3, 2007

Lecture: Naren Ramakrishnan *Scribe: Shubhangi Deshpande*

1 Overview

In the last few lectures, we introduced three software tools for relational data mining: FOIL, CIGOL, and PROGOL. In this lecture we discuss one of them, PROGOL, in more detail and go through the development of PROGOL input files to get a better understanding of its syntax.

2 Developing an input file for PROGOL

PROGOL is an ILP system which combines top-down and bottom-up approaches, specifically inverse entailment with general-to-specific search through a refinement graph. As with all ILP systems, PROGOL constructs logic programs from examples and background knowledge.

For instance, to learn a rule like

- All CS courses are easy.

We need to provide as input some cs courses which are easy and atleast one non cs course, to help it learn the determining factor for an easy course. Let us develop a PROGOL input file for this purpose:

```
% Mode declarations
:- modeh(*, easy(+course))?
:- modeb(*, cscourse(+course))?
:- modeb(*, easy(+course))?


% Types
course(cs6604).
course(cs2604).
course(bt101).


% Background knowledge
cscourse(cs2604).
cscourse(cs6604).
:- cscourse(bt101).


% Examples
easy(cs6604).
```

easy(cs2604).

Let us look at this file in more detail.

2.1 Types

These describe the categories of objects in the world under consideration. In this example we only need the type 'course', since all objects that we will be interested in are of this type.

course(cs6604).
course(cs2604).
course(bt101).
...

The lines above simply indicate the legal values for the 'course' type.

2.2 Modes

These describe the relations (predicates) which can be used either in the head (modeh declarations) or body (modeb declarations) of hypothesized clauses. For the head of the example above we have the head mode declaration as:

:- modeh(*, easy(+course))?

The declaration states that head atom has predicate symbol easy and has 1 argument of type course. The '+' sign indicates that an argument is an input variable. A '-' sign would indicate an output variable.

All modeh (and modeb) declarations contain a number called the recall. This is used to bound the number of alternative solutions for instantiating the atom. If in doubt use a large number or '*' as the recall. Here we have used '*' as we are not sure of the number of instantiations.

The modeb declarations look like:

:- modeb(*, cscourse(+course))?
:- modeb(*, easy(+course))?

The first of these declarations allows the use of the 'cscourse' predicate in forming a rule; the second allows the use of the 'easy' predicate in the rule.

2.3 The output

The output for the above PROGOL code looks like:

easy(A) :- cscourse(A).

Thus it has learned the rule that "All cs courses are easy" which was our aim.

3 More Examples

To learn a rule like ”All 6000 level cs courses are easy” we need to provide some 6000 level cs courses which are easy and atleast one cs course which is not 6000 level and is not easy.

cscourse(cs6604).
cscourse(cs6602).
cscourse(cs4232).
easy(cs6604).
easy(cs6602).
:- easy(cs4232).

The output looks like:

[C:-1,2,1,0 easy(A) :- cscourse(A).]
[2 explored search nodes]
f=0,p=2,n=1,h=0
[No compression]

[Generalising easy(cs6602).]
[Most specific clause is]

easy(A) :- cscourse(A).

[C:0,2,1,0 easy(A).]
[C:-1,2,1,0 easy(A) :- cscourse(A).]
[2 explored search nodes]
f=0,p=2,n=1,h=0
[No compression]

easy(cs6604).
easy(cs6602).

[Total number of clauses = 2]

[Time taken 0.00s]

Thus from the given inputs it has learned the rule:
easy(A) :- cscourse(A).

As you can see, this is not quite the rule we wanted to learn. This shows that we must have a better understanding of PROGOL’s algorithm and how it proceeds to hypothesize clauses. This is especially important when experimenting with completely unknown data.

To dig into more details, let us extend the above rule to say "A CS course is easy if its pre-requisite is easy". In predicate logic both the rules mentioned below are equivalent ways to encode the above:

easy(A) :- prereq(A,B),easy(B).
easy(A) :- easy(B), prereq(A,B).

But not so in PROGOL. For the first case, a modeb declaration like

:- modeb(1, prereq(+course,-course))?

will produce the desired result. Because 'B' being an output variable it can be produced with prereq(A,B) clause before its usage in easy(B) for which it is an input argument. But for the second case, we need to produce B somehow before its usage. Hence, easy might need to declare its (single) argument as both input and output.

Next class, we will investigate more details of PROGOL's algorithm and how to understand its functioning.

4 Applications of ILP

One classic application of ILP is mining chemical compounds, especially the predictive toxicology challenge where you are given the structure of a chemical compound and we must characterize its properties. Structures are highly recursive, hence traditional attribute-value techniques like decision trees might not be suited here.

Another application area for ILP is program synthesis from examples; this involves automatic generation of executable computer programs from high-level specifications of their behaviour. The idea originated in the 60s with the aim of using techniques from artificial intelligence to build an automatic programmer, exploiting deep connections between mathematics and the theory of programming. Today, although some researchers still work on formal approaches, more success has been obtained by combining pure deductive techniques with powerful heuristics, and limiting their application to specific domains.