CS 6604: Data Mining Fall 2007

Lecture 12 — September 26, 2007

Lecture: Naren Ramakrishnan *Scribe: Sheng Guo*

1 Overview

In the last lecture we began discussion of relational data mining, and described two major methods of RDM: taking logic and adding some induction (inductive logic programming - ILP), and taking induction and adding some logic. In this lecture, we will focus on the Inverse Resolution method.

2 Inverse Resolution Method

Resolution is a rule of inference typically used in a refutation-mode of theorem-proving, for sentences in propositional logic and in predicate logic. In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a given formula is unsatisfiable. By negating the goal and verifying that it is unsatisfiable, we can prove that the goal logically follows from the premises.

We will first study resolution before formally characterizing inverse resolution. Further, we will first study propositional logic before turning our attention to predicate logic.

2.1 Resolution and Inverse Resolution for Propositional Logic

Here we give an example to show how it works.

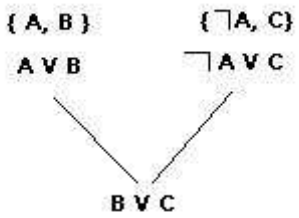

Figure 1: Resolution in propositional logic.

We think of a clause (disjunction) as a set of (negated or non-negated) propositional variables. We define formally resolution and inverse resolution procedures.

2.1.1 Resolution

Given clauses C_1 & C_2,

1. find a literal L such that L appears in C_1 and $\neg L$ appears in C_2.
2. Then the resolvent is given by $(C_1 - \{L\}) \cup (C_2 - \{\neg L\})$.

2.1.2 Inverse Resolution

Given C_1 which is of the form $A \vee B$, and resolvent which is of the form $B \vee C$, the aim is to find C_2.

1. Find a literal L that appears in C_1 but not in the resolvent.
2. Then C_2 is given by either

$$(Resolvent - (Resolvent \cap C_1)) \cup \{\neg L\}$$

or by

$$(Resolvent - (C_1 - \{L\})) \cup \{\neg L\}$$

2.2 Resolution and Inverse Resolution for Predicate Logic

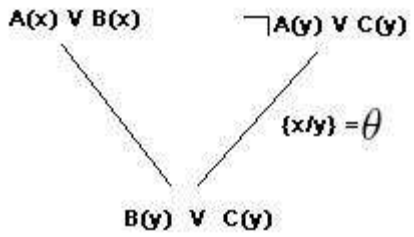

Figure 2: Resolution in predicate logic. There is a typo; the expression on the right must have a negation for A(y).

2.2.1 Resolution

Given C_1, C_2,

1. find a literal L_1 in C_1 and a literal L_2 in C_2 that $L_1\,\theta = \neg\, L_2\,\theta$. (For instance, L_1 is A(x), L_2 is $\neg\, A(Y)$, and θ is {x/y}).
2. Resolvent is

$$(C_1 - \{L_1\})\,\theta \cup (C_2 - \{L_2\})\,\theta$$

2.2.2 Inverse Resolution

Before we present inverse resolution, observe that to resolve two expressions such as:

- $A(x) \vee B(x)$
- $A(y) \vee \neg B(y)$

there could exist multiple substitutions as shown in Fig. 3. If we assume that substitutions can be factored into two parts θ_1 and θ_2 that apply to each of the two clauses being resolved (on separate literals L_1 and L_2), we get:

$$\begin{array}{ll} L_1\theta & = \neg L_2\theta \\ L_1\theta_1 & = \neg L_2\theta_2 \\ L_1\theta_1\theta_2^{-1} & = \neg L_2 \\ L_2 & = \neg\, L_1\theta_1\theta_2^{-1} \end{array}$$

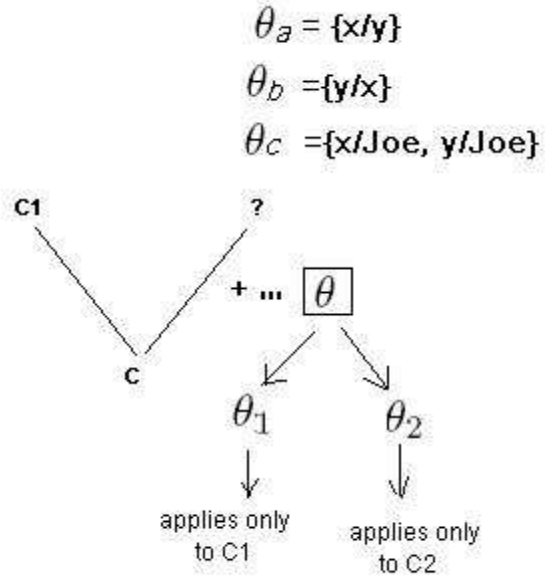

Figure 3: Inverse resolution in predicate logic.

Thus the inverse resolution procedure can be cast as:

1. Find two subsitutions θ_1, θ_2 as suggested by the above figure.
2. Then C_2 is given by

$$(Resolvent - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \;\cup\; \{\neg\, L_1\theta_1\theta_2^{-1}\}$$

This ideas is used in the CIGOL ILP system.

3 θ-subsumption

Consider the example below, which depicts multiple relations and assume our goal is to learn the two clauses:

- ancestor(x,z) ← parent(x,z).
- ancestor(x,z) ← ancestor (x,m), parent(m,z).

+	mary	ann
+	eve	tom
-	tom	eve
-	eve	ann

Table 1: daughter

ann	mary
ann	tom
tom	eve
tom	ian

Table 2: parent

ann
mary
eve

Table 3: female

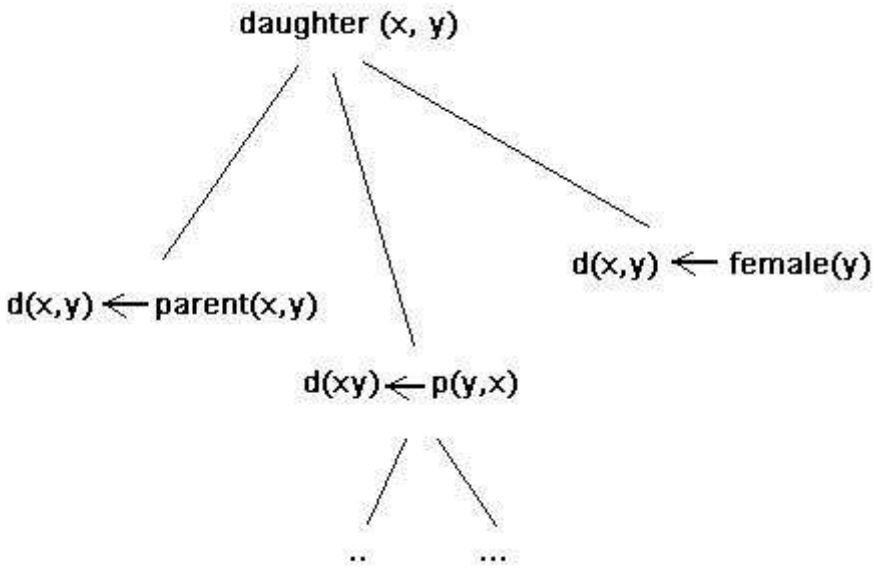

Figure 4: Searching for relational rules with same consequent.

To search across the specialization/generalization relationship, we introduce the idea of θ-subsumption.

C_1 θ-subsumes C_2 if there exists a substitution θ s.t. $C_1\theta \subseteq C_2$. If C_1 θ-subsumes C_2, then C_1 is at least as general as C_2. C_1 is more general than C_2, if C_1 θ-subsumes C_2, but C_2 does not θ-subsume C_1. Observe that θ-subsumption is purely a syntactic notion, not a semantic one.

In semantic terminology, we say $C_1 \models C_2$ iff mdoels of C_1 are also models of C_2. Note that if C_1 θ-subsumes C_2, then $C_1 \models C_2$, but not the other way round. Moral of the story: θ-subsumption is a poor cousin to entailment.

For instance, suppose we have two clauses:

- C_1: elephant(x) → elephant(father(x)).
- C_2: elephant(x) → elephant(father(father(x))).

It is easy to see that $C_1 \models C_2$, i.e., if we treat C_1 as a given rule, we can conclude C_2, since knowing that an elephant's father is an elephant, we can say that the elephant's father's father is also an elephant. But observe that C_1 does not θ-subsume C_2.