CS 6604: Data Mining Fall 2007

Lecture 11 — September 24, 2007

Lecture: Naren Ramakrishnan *Scribe: Ying Jin*

1 Overview

In the last lecture we discussed the BOAT algorithm, deriving rules from trees, both simultaneous and sequential covering, classifier validation methods, and other ways of classification. In addition, we briefly introduced relational data mining.

In this lecture we discuss the new topic of relational data mining in more detail, and describe two major methods of RDM: (i) taking logic and adding some induction (Inductive Logic Programming (ILP)); and (ii) taking induction and adding some logic.

2 Approaches to RDM

Relational data mining is a data mining technique for learning patterns from multiple relations (tables in relational databases). Recall that relations in a RDBMS correspond to predicates (in logic programming terminology).

2.1 Propositional Logic

Given some sentences as below, what can we infer from them?

1. All CS courses are easy.
2. CS6604 is a CS course.
3. CS6604 is easy.

Here are example inferences that are all consistent with the above facts:

1. 6000 level CS course are easy.
2. All courses are easy.
3. All courses are CS courses.
4. ...

Hence data mining (like always) requires some bias, in order to express preferences on hypotheses.

Let's begin by refreshing our memory about some concepts in propositional logic and predicate Logic.

Modus ponens is a common rule of inference that concludes q given $p \rightarrow q$ and q. Recall that $p \rightarrow q$ can be written as $\neg p \vee q$.

Entailment ($\models$): $p \models q$ means that models of p are also models of q.

For example,

- $p \models p \vee q$,
- $p \wedge q \models p$,
- $p, p \rightarrow q \models q$,
- $\neg q, p \rightarrow q \models \neg p$.

Resolution rule: $(\neg p \vee q, \neg q \vee r) \models (\neg p \vee r)$.

Usually resolution is used in refutation mode of operation, i.e., negate the goal and try to derive a contradiction.

As a practice problem, let us prove $p \wedge q \models p \Leftrightarrow q$.

We are hence given two clauses: p (clause 1) and q (clause 2). The goal to be proved: $p \Leftrightarrow q$ can be equivalently written as $(\neg p \vee q) \wedge (p \vee \neg q)$. We negate this and obtain:

$$\begin{aligned} &\neg((\neg p \vee q) \wedge (p \vee \neg q)) \\ &\neg(\neg p \vee q) \vee \neg(p \vee \neg q) \\ &(p \wedge \neg q) \vee (\neg p \wedge q) \\ &(p \vee q) \wedge (\neg q \vee \neg p) \end{aligned}$$

This thus gives two clauses: $p \vee q$ (clause 3) and $\neg q \vee \neg p$ (clause 4).

Applying resolution rule on clauses 2 and 4 gives $\neg p$, which in turn resolving with clause 1 gives a contradiction.

2.1.1 Predicate Logic

Predicates contain arguments as opposed to propsitions. For example:

- $P(x)$: x is an animal.
- $Z(x)$: x is a zoo.
- $l(a, m)$: a lives in m.

Modus ponens can be generalized as $p(x), p(x) \rightarrow q(y) \models q(y)$, under the substitution $\{x/y\}$.

Here is another example.

1. Whoever can read is literate.
2. Dolphin are not literate.
3. Some dolphins are intelligent.
4. Some who are not intelligent can not read.

Given 1, 2, and 3, we want to prove 4. First define the vocabulary:

- $D(x)$: x is dolphin.
- $L(x)$: x is literate.
- $I(x)$: x is intelligenet.
- $R(x)$: x can read

Above sentences can be rewritten as:

1. $\forall x R(x) \Rightarrow L(x)$.
2. $\forall x D(x) \Rightarrow \neg L(x)$.
3. $\exists x D(x) \wedge I(x)$.
4. $\exists x I(x) \wedge \neg R(x)$.

Converting every sentence to a clause, we get:

1. $\neg R(x) \vee L(x)$.
2. $\neg D(x) \vee \neg L(x)$.
3. we need to assign specific value to variable x.
   a. $D(flipper)$.
   b. $I(flipper)$.
4. $\neg I(x) \vee R(x)$ (because the goal is negated).

The proof is shown in Fig. 1.

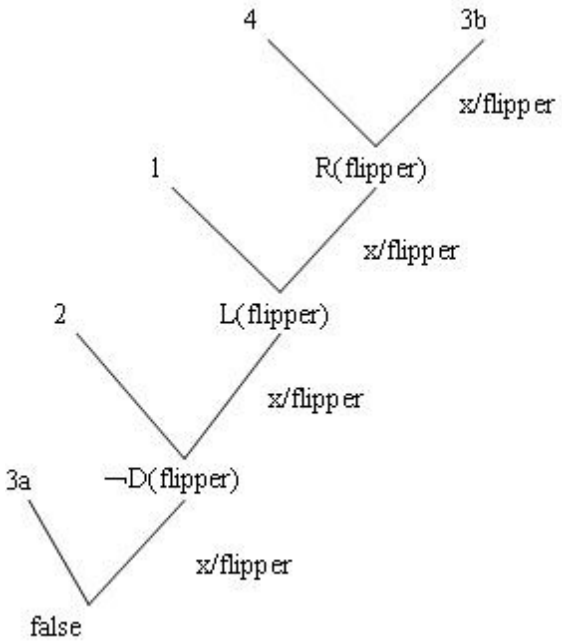

Figure 1: Proof for the Dolphins example.

Let us take another example: the following tables give data from multiple relations.

+	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

Suppose there is a rule for the "daughter" relationship, $daughter(x, y) \Leftarrow parent(y, x) \wedge female(x)$. We can examine if the daughter relationship is satisfied for any two people in the dataset by applying resolution rules. For example, given

1. $female(mary)$.
2. $parent(ann, mary)$.
3. $\neg parent(y, x) \vee \neg female(x) \vee daughter(x, y)$.

Fig. 2 gives an example of applying resolution to conclude a specific daughter relationship from the above data.

PROLOG is a well known logic programming language. What is its shorcomming? It uses depth first search, so can wander down a lost path in search of a proof. Also, most PROLOG implementations have deliberately missed out the "occurs check", i.e., unifing a variable against a term containing that variable.

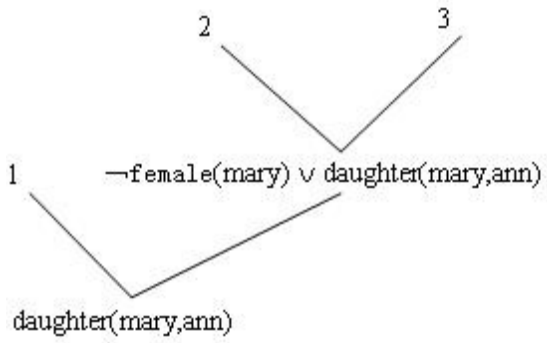

Figure 2: Proving that daughter(mary,ann) is true.

2.2 LP + Induction

Supposer we want to infer the general rule $\neg parent(y, x) \vee \neg female(x) \vee daughter(x, y)$, we just run the proof from Fig. 2 "in reverse." This is called inverse resolution.

2.3 Induction + LP

We have discussed decision tree induction in previous classes. Let's think about a tree whose conditions are based on predicate argument, as shown in Fig. 3.

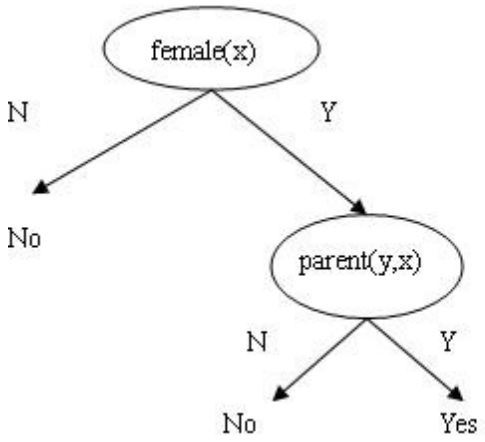

Figure 3: Example of relational decision tree.

Just as we grow decision trees inductively, we can determine the best question to ask and incrementally grow the tree based on predicate arguments.

Another approach, to directly learn rules, is to do a levelwise search for rules. Initially start with a rule that has no predicates in the antecdent, i.e., that always predicts that X is the daughter of Y. Then consider rules that have one condition, then rules that have two conditions, and so on. The notion of θ -subsumption is relevant here: clause 1 θ -subsumes clause 2 if there exists a substitution θ, s.t. $C_1\theta \subseteq C_2$. We will discuss this in more detail in the next class.

2.4 Software for relational data mining

- FOIL: adopts the 'add LP to an inductive algorithm' approach by a top-down search for specialiations.
- CIGOL: adops the 'add induction to LP' approach, specifically implements bottom-up inverse resolution.
- PROGOL (a play on PROLOG): Combines top-down and bottom-up approaches, specifically inverse entailment with general-to-specific search through a refinement graph.