Knowledge Discovery in Databases – An Inductive Logic Programming Approach

Katharina Morik

Univ. Dortmund, Computer Science Department, LS VIII

Abstract. The need for learning from databases has increased along with their number and size. The new field of Knowledge Discovery in Databases (KDD) develops methods that discover relevant knowledge in very large databases. Machine learning, statistics, and database methodology contribute to this exciting field. In this paper, the discovery of knowledge in the form of Horn clauses is described. A case study of directly coupling an inductive logic programming (ILP) algorithm with a database system is presented.

1 Introduction

Databases are used in almost all branches of industry and commerce. The aim of KDD is to discover rules hidden in these collected data.

The task of KDD is challenging, for the following reasons:

- The amount of database tuples from which an algorithm learns exceeds the number of examples that are usually handled by a learning algorithm or a statistical procedure. Real-world KDD applications have to cope with a number of tuples on the order of several hundred thousand.
- The amount of database attributes (on the order of several hundred) exceeds the number of features that are usually handled by a learning algorithm.
- The task of finding all valid rules hidden in a database is more difficult than other learning tasks (see Section 2).

Several such learning algorithms for KDD exist: Bayesian and Neural Networks, induction of decision trees, minimal description length algorithms, learning of association rules – to name but the most common ones [1]. In order to reduce the number of attributes, these algorithms reduce the database in one or both of the following ways:

- Analysis is only done of one database attribute or of the equivalent set of attributes with binary values.
- Analysis is only done on one database table comprised of the most interesting attributes. Frequently, this table is created in a pre-processing step. [2]

[1] The KDD books give a comprehensive overview of the field [10] [4]. For fast neural learning see also [11]

[2] In principle, learning from just one database table is not a restriction – it could be the universal relation and cover what was originally stored in many tables. In practice, however, the universal relation of real-world databases is so large that it cannot be efficiently managed, nor can the learning algorithms handle it.

In addition to restricting the database, most algorithms restrict the discovered knowledge to an attribute-value representation (propositional logic), possibly with the extension of probabilistic annotations or certainty factors. This is a severe restriction, since it excludes relational or structural knowledge. Saso Dzeroski has given a nice example to illustrate this [3]: Figure 1 shows data in two tables of a database. If restricted to propositional knowledge, an algorithm could discover the following rules in the data:

$income(Person) \geq 100000 \rightarrow customer(Person) = yes$

$sex(Person) = f \& age(Person) \geq 32 \rightarrow customer(Person) = yes$

Rules like the following cannot be expressed (and therefore not be learned) by these algorithms:

(i) $married(Person, Spouse) \& customer\_yes(Person)$
$\rightarrow customer\_yes(Spouse)$

(ii) $married(Person, Spouse) \& income(Person, \geq 100000) \rightarrow customer\_yes(Spouse)$

potential customer					married	
person	age	sex	income	customer	husband	wife
Ann Smith	32	f	10 000	yes	Jack Brown	Jane Brown
Joan Gray	53	f	1 000 000	yes	Bob Smith	Ann Smith
Mary Blythe	27	f	20 000	no		
Jane Brown	55	f	20 000	yes		
Bob Smith	30	m	100 000	yes		
Jack Brown	50	m	200 000	yes		

Fig. 1. Relational database with two tables

Other relations that cannot be expressed or characterized in propositional logic include spatial relations, time relations, or the connectivity of nodes in a graph. For such knowledge, a (restricted) first-order logic is necessary. Inductive Logic Programming (ILP) is the field within machine learning which investigates Learning hypotheses in a restricted first-order logic. In Section 3 we show how to overcome the common restrictions (learning propositional rules from one attribute or one database table) by an ILP algorithm without becoming inefficient.

2 The Learning Task in a Logical Setting

The task of knowledge discovery in a broad sense includes interaction with the knowledge users, the database managers and data analysts. Learning rules from data is then one important part of the overall task of KDD. The rule learning task has been stated formally by Nicolas Helft [5] using the logical notion of minimal models of a theory $\mathcal{M}^+(Th) \subseteq \mathcal{M}(Th)$.

Minimal model. An interpretation I is a model of a theory Th, $\mathcal{M}(Th)$, if it is true for every sentence in Th. An interpretation I is a *minimal model* of Th, written $\mathcal{M}^+(Th)$, if I is a model of Th and there does not exist an interpretation I' that is a model of Th and $I' \subset I$.

Rule learning
Given observations $\mathcal{E}$ in a representation language $\mathcal{L}_{\mathcal{E}}$ and background knowledge $\mathcal{B}$ in a representation language $\mathcal{L}_{\mathcal{B}}$,
find the set of hypotheses $\mathcal{H}$ in $\mathcal{L}_{\mathcal{H}}$, which is a (restricted) first-order logic, such that

(1) $\mathcal{M}^+(\mathcal{B} \cup \mathcal{E}) \subseteq \mathcal{M}(\mathcal{H})$ (validity of $\mathcal{H}$)
(2) for each $h \in \mathcal{H}$ there exists $e \in \mathcal{E}$ such that $\mathcal{B}, \mathcal{E} - \{e\} \not\models e$ and $\mathcal{B}, \mathcal{E} - \{e\}, h \models e$ (necessity of h)
(3) for each $h \in \mathcal{L}_{\mathcal{H}}$ satisfying (1) and (2), it is true that $\mathcal{H} \models h$ (completeness of $\mathcal{H}$)
(4) There is no proper subset $\mathcal{G}$ of $\mathcal{H}$ which is valid and complete (minimality of $\mathcal{H}$).

Since the hypotheses $\mathcal{H}$, i.e. sets of rules, are valid in all minimal models of background knowledge and examples, the first condition asks for a deductive step. It corresponds to the application of the closed-world assumption, i.e. all but the stated assertions are false. The second condition restricts the learning result to those rules that are related to given observations. There might be additional valid rules, for instance tautologies, but we are not interested in them. The most important condition is the third one which demands that *all* valid and necessary rules must be found. Condition (4) formalizes the non-redundancy requirement. This learning task has been taken up by several ILP researchers, e.g., [7], [2]. It is more difficult than the classical concept learning task described below:

Concept learning
Given positive and negative examples $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ in a representation language $\mathcal{L}_{\mathcal{E}}$ and background knowledge $\mathcal{B}$ in a representation language $\mathcal{L}_{\mathcal{B}}$,
find a hypothesis $\mathcal{H}$ in a representation language $\mathcal{L}_{\mathcal{H}}$, which is a (restricted) first-order logic, such that

(1) $\mathcal{B}, \mathcal{H}, \mathcal{E}^+ \not\models \Box$ (consistency)
(2) $\mathcal{B}, \mathcal{H} \models \mathcal{E}^+$ (completeness of $\mathcal{H}$)
(3) $\forall e^- \in \mathcal{E}^- : \mathcal{B}, \mathcal{H} \not\models e^-$ (accuracy of $\mathcal{H}$)

The difference between the two learning tasks can be illustrated by our small example database (Figure 1). For the sake of clarity, let us reduce the table *potential customer* to the attributes *customer* and *income* and replace the numerical values of *income* by *low, high* in the obvious way. We consider the observation $\mathcal{E}$ to be the attribute *customer* and the background knowledge $\mathcal{B}$ to be *income* and *married_to*. In addition to rules (i) and (ii) – where ≥ 100000 is replaced by *high* – rule learning will find

(iii) $income(Person, high) \rightarrow customer\_yes(Person)$

Let us now inspect two variants of our example in order to clarify the differences. Case 1: It is unknown whether *jane* is a customer. Case 2: Both, the income of *jack* and whether *jane* is a customer are unknown. In the first case, condition (1) of rule learning leads to the rejection of rules (i) and (ii), since *jane* is not a customer in all minimal models of $\mathcal{E}$ and $\mathcal{B}$. Only rule (iii) is valid regardless of whether or not *jane* is a customer. If we regard the attribute value *yes* of *customer* as the classification of positive examples and the value *no* as the classification of negative examples, then we can apply concept learning on *customer_yes*. Not knowing whether *jane* is a customer does not prevent concept learning from finding rules (i) and (ii). *customer_yes*(*jane*) will be predicted. Concept learning will deliver (i) and (iii), or (ii) and (iii), depending on its preference. The task of deriving the positive examples is accomplished with either of the two sets of rules. If negative examples are not explicitly stated but are derived using the closed-world assumption, however, concept learning rejects rules (i) and (ii) (accuracy condition of concept learning). Rule (iii) alone cannot derive that *ann* is a customer. Because of its completeness condition, concept learning fails when using the closed-world assumption.

In the second case, the rules (i) – (iii) cannot derive *customer_yes*(*jack*). The completeness condition of concept learning leads to the rejection of rules (i) – (iii). Even without the closed-world assumption, concept learning does not deliver a result at all. Rule learning still finds rule (iii). In summary, rule learning finds rules in all cases, whereas concept learning fails to find any rule in case 2 and under the closed-world assumption in case 1. The rules found by rule learning are closer to the data (i.e. less predictive) than the rules found by concept learning. The rule learning task is more difficult than the concept learning task because there are learning problems that can be solved by rule learning, but not by concept learning (case 2), and there are no learning problems that can be solved by concept learning but not by rule learning [3]. The completeness condition of finding *all* valid and possibly interesting rules is the formalization of the learning task in KDD.

3 Discovery of Horn Clauses

The previous section states formally what it means to find *all* valid, necessary, and non-redundant rules. In addition, the user indicates the kind of hypotheses in which he or she is interested. The system then restricts $\mathcal{L}_{\mathcal{H}}$ to those hypotheses that fit the user-given declarative bias. This method offers two advantages: it allows users to tailor the learning procedure to their goals and, by restricting the space of all hypotheses, it makes rule learning feasible. Of course, there is a price to pay: the tailored learning procedure misses interesting rules that do not fit into the user-given hypothesis space. Our learning procedure, the Rule Discovery Tool

[3] Stephen Muggleton and Luc De Raedt have shown for languages $\mathcal{L}_{\mathcal{B}}$ and $\mathcal{L}_{\mathcal{H}}$ being restricted to definite clauses that rule learning covers concept learning [9]. Jörg-Uwe Kietz has generalized their finding and gives proofs of the difficulty of learning for various restrictions of first-order logic [6].

(RDT) and its adaptation in order to learn directly from databases (RDT/DB) has been described elsewhere [7], [8], [1]. Here, we shortly summarize its main algorithm and give an example of its declarative bias.

The user-given bias is a set of rule schemata. A rule schema has predicate variables that can be instantiated by predicates of the domain. An instantiated rule schema is a rule. Rule schemata are partially ordered according to their generality. For our small database, a user might specify the following rule schemata $m1$, $m2$, and $m3$ which restrict hypotheses to one or two conditions for the conclusion. An additional predicate in the premise makes a rule more specific.

m1(P, Q): $P(X) \rightarrow Q(X)$
m2(P1, P2): $P1(X) \& P2(X) \rightarrow Q(X)$
m3(P1, P2): $P1(X, Y) \& P2(Y) \rightarrow Q(X)$

RDT's learning procedure consists of two steps: hypothesis generation and testing. In a top-down, breadth-first manner, all possible instantiations of the rule schemata are generated and tested via SQL-queries to the database. The following rules instantiate $m1$ in our example:

$age\_young(X) \rightarrow customer\_yes(X)$
$age\_middle(X) \rightarrow customer\_yes(X)$
$age\_old(X) \rightarrow customer\_yes(X)$
$sex\_f(X) \rightarrow customer\_yes(X)$
$sex\_m(X) \rightarrow customer\_yes(X)$
$income\_high(X) \rightarrow customer\_yes(X)$
$income\_low(X) \rightarrow customer\_yes(X)$

Rules and rule schemata depend on a mapping from the database representation to predicates. Here, we have chosen a representation in propositional logic for the *customer* table. The discretized attribute values are used as predicates. The name of the person (i.e. the key of the table) is the only argument of these predicates. An alternative mapping is implicitly given by rule (iii) above. There, the attribute of the database becomes the predicate; the key and the attribute value become its arguments. Another alternative is to map the table name onto the predicate and have all attributes as arguments. This is the natural mapping for the *married* table as is implicit in rules (i) and (ii). The user can specify a mapping from the database to the rule representation.

Hypothesis testing uses the specified mapping when creating SQL queries. A user-given acceptance criterion is used to decide whether the result of the queries for supporting and non supporting tuples is sufficient for accepting the hypothesis. If a rule has enough support but too many non supporting tuples, it is considered too general. Later on, it becomes a partial instantiation of a more specific rule schema if this exists. If a rule does not have enough support, it is considered too specific. In this case, the rule need not be specialized further, since this cannot increase the number of supporting tuples. RDT safely prunes the search in this case. RDT learns all valid rules that fit the declarative bias. The important points of this learning procedure its declarative bias and its top-down, breadth-first refinement strategy which allows the pruning of large parts of the hypothesis space. This makes rule learning from very large databases tractable.

4 Experiments

We have run RDT/DB on various large data sets. An illustrative example is vehicle data from Daimler Benz AG. Here, we have solved two learning problems. First of all, we learned characteristics of warranty cases. This learning task is close to concept learning, since the learned rules characterize cars of one class or concept, namely the warranty cases. However, the task was not to find a set of rules that completely derives all warranty cases (the completeness condition of concept learning). Instead, the task was to find all relevant rules about warranty cases (the completeness condition of rule learning). That is, we wanted to find out what the data tell us about warranty cases.

RDT/DB accomplishes this learning task by instantiating the predicate variable of the conclusion in all rule schemata by the predicate $warranty(X)$. Following the advice of the Mercedes technicians, we separated the data into three subsets: gasoline engines and manual gearshift, gasoline engines and automatic gearshift, and diesel engine and automatic gearshift. In each of these subsets, the number of database tables is 23, and the largest number of tuples in a database table was 111,995. We selected up to 3 relevant attributes of each table so that one observation consists of 26 attributes which must be collected using 23 database tables. Using 3 rule schemata, RDT/DB learned rules like the following:

(iv) $rel\_niveaureg(X) \rightarrow warranty(X)$ stating that vehicles with the special equipment of *Niveauregulierung* are more likely to be warranty cases than all other cars, taking into account the distribution of cars with this equipment.
(v) $motor\_e\_type(X, Type) \& mobr\_cyl(Type, 6.0) \rightarrow warranty(X)$ stating that vehicles of an engine type that has 6 cylinders are more likely to be warranty cases than one would predict on the basis of the overall warranty rate and the proportion of cars with this characteristic.

The second learning task was to analyze warranty cases in which parts of a particular functional or spatial group (e.g., having to do with fuel injection) were claimed to be defective. We investigated 9 groups of car parts. Here, RDT/DB inserted – one after the other – 9 predicates as the conclusion of all rule schemata. We used 13 rule schemata. We specified an acceptance criterion inspired by Bayes:

$$\frac{pos}{pos+neg} \geq \frac{concl}{concl+negconcl}$$

where *pos* is the number of tuples for which the premise and the conclusion are true, *neg* is the number of tuples for which the premise is true but the conclusion is not true, *concl* is the number of tuples for which the conclusion is true, regardless of any premise, and *negconcl* is the number of tuples for which the conclusion is not true. The first expression corresponds to the conditional probability of the conclusion given the conditions described by the premise. The second expression corresponds to the a priori probability of the conclusion [4].

[4] Note that the data set is complete, i.e. all vehicles of a certain engine and gearshift are known to the system. This means that prior probabilities can be reduced to

For 5 groups of car parts, RDT/DB found no rules that fulfill the acceptance criterion. In contrast, 4 groups of parts were described by rules like the following:

(vi) $rel\_warranty(X1, X2, RB, X4, X5, X6, X7, Config, Part) \&$
$rel\_warranty\_gasman(X1, X2, RB, X4, X5, X6, X7, VID) \&$
$rel\_motor(VID, Type, Variant) \& engine\_perform(Type, Variant, 236)$
$\rightarrow class\_15(Config, Part)$

(vii) $rel\_warranty(X1, X2, RB, X4, X5, X6, X7, Config, Part) \&$
$rel\_warranty\_gasman(X1, X2, RB, X4, X5, X6, X7, VID) \&$
$rel\_motor\_type(VID, 206) \& regions\_italy(RB) \rightarrow class\_419(Config, Part)$

Rule (vi) combines 3 database tables (*rel_warranty*, *rel_warranty_gasman*, and *rel_motor*). It states that engines with a particular performance have more trouble with parts of the group *class_15* than we would expect, given their overall frequency. Rule (vii) states that Italian engines of type 206 often have fuel injections (*class_419*) that have been claimed as defective. The regions of the workshops where a car has been repaired are background knowledge that has been added to the database.

Figure 4 summarizes the system's performance, running on a Sparc 20 computer. The variation in CPU run-times for the first learning task is due to pruning. The run-time for the second learning task is justified by the difficulty of the learning task: 9 instead of 1 conclusion predicates, 13 instead of 3 rule schemata. Moreover, for the second learning task a join using a table with 700,000 tuples was necessary. Note, that operations which other learning procedures perform in a preprocessing step, such as join operations, selections, groupings in the database, are here performed within the hypothesis testing step, since we are working directly on the database.

Data set	tables	tuples	rules learned	run-time
gas. manual	23	111995	17	3 min. 15 secs.
gas. autom.	23	111995	79	3 h. 1 min. 46 secs.
diesel autom.	23	111995	15	7 min. 29 secs.
car parts	16	700000	92	49 h. 32 min. 44 secs.

Fig. 2. RDT/DB learning from databases

5 Conclusion

We have formalized the learning step within KDD as a rule learning task and have contrasted it with the concept learning task. Since some of the interesting rules that are hidden in databases cannot be expressed in propositional logic,

frequencies. The criterion is applicable, because for every particular rule to be tested, we have two classes: the group of parts in the conclusion and all other groups of parts.

we have adopted the approach of ILP, which can express rules in restricted FOL. We have shown that RDT/DB, an ILP algorithm which directly accesses ORACLE databases, is capable of solving the difficult task of rule learning. Using a declarative bias which narrows the hypothesis space allows RDT/DB to learn first-order logic rules from diverse tables, even when the number of tuples is large. However, directly using the database makes hypothesis testing inefficient. Each hypothesis is tested using two SQL queries, which count supporting and non supporting tuples. These tuples are not stored in the main memory under the control of the learning system. If a hypothesis is specialized later on, a subset of the already computed set of tuples must be selected. In our approach, however, the result of previous hypothesis testing is lost and the new SQL query computes the large set of tuples anew and then selects within it. Now that we have restricted hypothesis generation successfully, improvements to hypothesis testing are necessary.

References

1. Peter Brockhausen and Katharina Morik. A multistrategy approach to relational knowledge discovery in databases. *Machine Learning Journal*, to appear 1997.
2. L. DeRaedt and M. Bruynooghe. An overview of the interactive concept–learner and theory revisor CLINT. In Stephen Muggleton, editor, *Inductive Logic Programming.*, number 38 in The A.P.I.C. Series, chapter 8, pages 163–192. Academic Press, London [u.a.], 1992.
3. Saso Dzeroski. Inductive logic programming and knowledge discovery in databases. In Usama M. Fayyad et al., editors, see 4., pages 117–152.
4. Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press Series in Computer Science. A Bradford Book, The MIT Press, Cambridge Massachusetts, London England, 1996.
5. Nicolas Helft. Inductive generalisation: A logical framework. In *Procs. of the 2nd European Working Session on Learning*, 1987.
6. Jörg Uwe Kietz. *Induktive Analyse relationaler Daten*. PhD thesis, Technische Universität Berlin, 1996.
7. Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task–oriented models. In Stephen Muggleton, editor, *Inductive Logic Programming*, chapter 16, pages 335–360. Academic Press, London, 1992.
8. Guido Lindner and Katharina Morik. Coupling a relational learning algorithm with a database system. In Kodratoff, Nakhaeizadek, and Taylor, editors, *Statistics, Machine Learning, and Knowledge Discovery in Databases*, MLnet Familiarization Workshops, pages 163 – 168. MLnet, April 1995.
9. S. Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
10. Gregory Piatetsky-Shapiro and William J. Frawley, editors. *Knowledge Discovery in Databases*. The AAAI Press, Menlo Park, 1991.
11. J. Schmidhuber and D. Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625 – 635, 1993.