

The Power of Second-Order Decision Tables^{*}

R. Hewett[†] and J. Leuchner[‡]

Abstract

The success of data mining techniques can be measured by the usefulness of models they produce. Often these models must be explainable as well as accurate. While decision tables are easy to interpret and explain to virtually all users, there has been little study of whether such simple models are powerful enough to use for data mining. This paper presents SORCER, a learning system that induces *second-order* decision tables from a given data set. Second-order tables are database relations in which rows have *sets* of atomic values as components. Using sets of values interpreted as disjunctions, second-order tables provide simple and compact representations that both enhance comprehensibility and facilitate efficient management. Unlike many other learning systems, to further promote comprehensibility, SORCER attempts to generate a minimum number of rows. SORCER's induction algorithm can be viewed as a table compression technique in which a traditional table, representing training data, is transformed into a second-order table with fewer rows by merging rows in ways that preserve consistency with the training data. We compare SORCER to three classification systems: C4.5, CBA and a Naive Bayesian classifier. Experimental results on 26 data sets show that the average error rate obtained from SORCER, using a simple compression method, is lower than those of the Naive Bayesian classifier and C4.5 and is competitive with CBA's average error rate. Using a slightly more sophisticated compression, SORCER's average error rate is the lowest of all.

Keywords: machine learning, induction, data mining, Boolean minimization technique

^{*} The research presented in this paper has been supported in part by ONR grant N00014-01-1-0926 and NASA grant NCC 2-1239.

[†] Research Scientist, The Institute for Human and Machine Cognition (IHMC), Pensacola, FL 32501, rhewett@ai.uwf.edu.

[‡] Research Associate, IHMC, Pensacola, FL 32501, jleuchner@ai.uwf.edu.

1 Introduction

Various data mining techniques have been studied in statistics, pattern recognition, machine learning, visualization and databases [Fayad et al., 1996; Glymour and Cooper, 1999; Mitchell, 1997; Ziarko, 1991] and significant advances have been made in various issues of model construction (e.g., scalability for larger databases, faster algorithms, high dimensionality, and mining from multiple databases) [Agrawal et al., 1996; Bayrdo and Agrawal, 1999; Fayad et al., 1996; Han et al., 1992; Liu et al., 1999]. Much research has concentrated on abstracting accurate models for prediction (or classification) from a given data set. However, a sophisticated technique may remain unused if the model it derives is not comprehensible. For a data mining technique to really be useful, the resulting models should be explainable as well as accurate. Many decisions (e.g., medical treatments, business decisions and financial investments) cannot be made based on black-box predictions alone. Easily interpretable models can give users confidence in the results obtained. The choice of a model affects not only accuracy but also users' understanding and confidence in the results.

Given a database of instances, each labeled by a corresponding class, supervised machine learning aims to find a *hypothesis* (or *classifier* or *predictive model*) that will correctly predict the class of future unlabeled instances. An induction algorithm can be viewed as a search for a hypothesis that best fits a given data set. Machine learning approaches have focused on models (e.g., neural nets, Bayesian nets, hyperplanes) that are unfamiliar to most non-analyst users. Although data mining models in the form of if-then rules [Apte and Hong, 1996; Holte, 1993; Michalski et al., 1986], decision trees [Quinlan, 1993] and association rules [Agrawal et al., 1996; Han and Fu, 1995; Liu et al., 1998] are considered to be easy to understand, problems arise when the size of trees or the number of rules become very large. Many successful applications that employ association rule mining algorithms tend to produce very large numbers of rules. Recent work on identifying "interesting" rules and on visualization techniques has been done to improve the comprehensibility of the models [Bayardo and Agrawal, 1999; Liu et al., 1999]. Decision trees are often viewed as one of the most comprehensible models. To their surprise, Kohavi and Sommerfield [Kohavi and Sommerfield, 1998] found that it took longer than expected to explain the meaning of decision tree models to their clients.

Decision tables, on the other hand, are easy to interpret and explain because of widespread familiarity with the tabular representations in spreadsheets and relational databases. However, relatively little work has been done on algorithms that use decision tables as representations of hypotheses induced from data sets, perhaps because complex models tend to be more accurate [Kohavi and Sommerfield, 1998]. Users however, would often be willing to sacrifice some accuracy for a more easily interpretable model, and it would be useful to know how well decision tables can perform in abstracting accurate yet comprehensible models from data.

In this paper, we describe **SORCER** (**S**econd-**O**rd**R**elation **C**ompression for **E**xtraction of **R**ules), a supervised learning system that induces a classifier in the form of a *second-order decision table*. Second-order decision tables are decision tables in which each row (or rule) assigns a *set* of atomic values to each attribute (or column header or problem variable). Using sets of values interpreted as disjunctions, second-order tables provide compact representations that both enhance comprehensibility and facilitate efficient management. Our goal is to evaluate the representational power of second-order tables for data mining, particularly the accuracy of induced tables. Unlike most other learning systems, SORCER attempts to produce the shortest decision table consistent

with the training data, a bias justified by Occam's razor [Blumer et al., 1987; Domingos, 1999]. Shorter tables are easier to understand and can be managed and applied more efficiently. The advantages of generating tables with a (near) minimal number of rows become more apparent when dealing with extremely large sets of examples. The size of a model can be taken as a simple measure of comprehensibility, and we believe the classifiers produced by SORCER have advantages of familiarity and simplicity, which are key supports of human comprehensibility [Chater, 1996; Wolff, 1993]. A proper evaluation of this hypothesis would require further experimentation and is beyond the scope of this paper.

The rest of the paper is organized as follows. Section 2 discusses other work in relation to our approach. Section 3 gives an overview of the second-order decision table framework and presents our table induction algorithm. Section 4 describes SORCER and the two compression algorithms used for our experiments. Section 5 describes our experiments with SORCER and compares the results to those of three other systems. Sections 6 and 7 discuss computational complexity and give our conclusions.

2 Related Work

Decision tables have been used to represent hypotheses in machine learning research that uses rough set theory to determine relevant features [Modrzejewski, 1993; Ziarko, 1991]. Han et al. [Han et al., 1992] present an attribute-oriented induction technique that exploits prior knowledge represented as conceptual hierarchies to extract generalized data from databases. This technique and its adaptation to multiple-level association rule mining [Han and Fu, 1995] shares with SORCER the use of nested relations [Thomas and Fischer, 1986] and set-oriented database operations that substantially reduce the computational complexity of the learning process. However, they differ from SORCER, and other classic learning algorithms such as candidate elimination [Mitchell, 1997], in that generalization is performed through hierarchies of attributes instead of coverage of attribute values. For example, in attribute-oriented induction, a data instance of a "freshman" college student with a GPA 3.9 may be generalized to an instance of an "undergraduate" student with an "excellent" GPA, given the facts that "freshman" is a type of "undergraduate" and GPAs of at least 3.8 are "excellent". Thus, the process of generalization loses the original low-level data values, which does not occur in SORCER. Furthermore, the number of rules generated in [Han and Fu, 1995] constrains the degree of generalization, whereas in SORCER, generalization proceeds, within cost constraints (e.g., time), until a table with as few rows as possible of rules is obtained.

Mani and Pazzani [Mani and Pazzani, 1998] describe an algorithm that induces decision tables. Unlike SORCER, they use Bayesian networks to represent hypotheses and then generate decision tables by viewing them as reverse Naïve Bayes structures. Langley [Langley, 1996] and Kohavi [Kohavi, 1995] present algorithms to induce decision tables from data by combining feature subset selection and computation of probabilities. Kohavi's IDTM algorithm [Kohavi, 1995] induces decision tables using a wrapper algorithm that selects features for a hypothesis with the highest future prediction accuracy. Kohavi and Sommerfield [Kohavi and Sommerfield, 1998] refine the approach in [Kohavi, 1995] and present the decision table learners, DTMaj and DTLoc. Empirical results show that decision table classifiers induced by IDTM and its variants perform surprisingly well but only on some data sets. These systems differ from SORCER in two ways. First, the IDTM family uses conventional decision tables (in which each row

assigns an atomic value to each attribute), whereas SORCER is based on a theoretical framework of *second-order relations (tables)* [Leuchner and Hewett, 1997]. Second, SORCER's induction aims to reduce the number of rows whereas IDTM's family aims to reduce the number of attributes, which is more closely related to feature selections than traditional induction.

Other induction techniques that generate models similar to those expressed by decision tables include decision tree learning, association rule mining and inductive logic programming techniques [Agrawal et al., 1996; Mitchell, 1997; Muggleton and Feng, 1990; Quinlan, 1993]. Decision tree learners such as C4.5 [Quinlan, 1993] have been applied successfully in various data mining applications. A decision tree model can be viewed as a decision table where each row (or rule) corresponds to a path from the root to the leaf of the tree. However, the rules generated by a decision tree learner are mutually exclusive, whereas the rules in a decision table may not be. Association rule mining [Agrawal et al., 1996; Bayardo and Agrawal, 1999; Liu et al., 1998, 1999] is one of the most influential data mining techniques. Each rule has corresponding parameters to signify "support" and "confidence" based on frequencies of occurrence of the events on the left and right sides of the rule. CBA [Liu et al., 1998] generates a classifier based on association rules. Unlike SORCER, CBA aims to obtain a complete set of predictive rules. Other approaches for rule mining include inductive logic programming (ILP) systems such as GOLEM [Muggleton and Feng, 1990]. These systems generate rules in disjunctive normal forms and apply logical inference to derive new rules, while SORCER uses set-based operations and heuristics for generalization.

R-MINI [Hong, 1994] is the system most like SORCER. Both aim at producing a "minimal" or near minimal set of rules that approximates a target function, and there is a parallel between SORCER's second-order relations and R-MINI's disjunctive normal forms. R-MINI is based on MINI [Hong et al., 1974], a heuristic technique for minimizing large Boolean functions, and has shown promising results for data mining applications [Apte and Hong, 1996]. However, SORCER is based on a more general representation and uses operations that are more general and easier to understand. R-MINI does not explicitly represent missing data and consequently ignores this information (which makes sense in logic synthesis). Since incomplete information can still be useful in identifying patterns in knowledge discovery, SORCER explicitly represents missing data by the empty set, which is interpreted using *null (unknown) values* [Codd, 1985]. Consequently, rules in SORCER express disjunctive normal forms with existential quantifiers (see details in Section 3.1). The inclusion of the empty set as a legitimate value representing unknown values adds considerable computational sophistications in defining operations for maintaining the tables (e.g., finding overlapping rules) and sets SORCER apart from learning approaches to generating disjunctive normal form rules including R-MINI, the Aq family [Michalski et al., 1986] and the approaches mentioned above. For more details, see our theoretical results in [Hewett and Leuchner, 1997a, 1997b].

3 The Approach

Our induction system, SORCER, uses second-order decision tables as representations of hypotheses. In this section we present an overview of the second-order decision table framework, including basic definitions and relevant operations, followed by a basic algorithm used in SORCER.

3.1 Representation of Induction Hypotheses and Preliminaries

Decision tables have two components: *scheme* and *body*. A scheme is a set of *attributes* (or *problem variables* or *features*) partitioned into two parts: *condition attributes* and *class* (or *action* or *classification* or *intervention*) *attributes*. We assume that there is only one class attribute. The *domain* of an attribute A , i.e., the set of values A can assume, is denoted by $dom(A)$. The body of a table consists of *rows* (or *tuples* or *instances* or *rules*), and a *row* is an assignment of values to each attribute of the table's scheme. A similar tabular notation is used for database *relations* (or *tables*), without the distinction between condition and classification attributes, and a decision table can be viewed as a special type of database relation. In conventional tables, the values that rows assign to an attribute A are taken from $dom(A)$.

Table 1

	<i>Hungry?</i>	<i>Type</i>	<i>Cost</i>	<i>#Patrons</i>	<i>Wait?</i>
1	yes	Chinese	med	few	yes
2	yes	Italian	med	few	yes
3	yes	American	med	many	no
4	yes	Chinese	low	few	yes
5	yes	Italian	low	few	yes
6	yes	American	low	many	no
7	no	Thai	high	-	no
8	no	Thai	low	-	no

Table 2

	<i>Hungry?</i>	<i>Type</i>	<i>Cost</i>	<i>#Patrons</i>	<i>Wait?</i>
1	{yes}	{Chinese, Italian}	{med, low}	{few}	{yes}
2	{yes}	{American}	{med, low}	{many}	{no}
3	{no}	{Thai}	{high, low}	\emptyset	{no}

Fig. 1 First-order vs. second-order decision tables.

Our hypotheses are represented as compact forms of decision tables called *second-order decision tables*. A *second-order decision table* is a decision table whose rows are *second-order tuples*. That is, each row r can be viewed as a function that assigns to each attribute A a finite *subset* of $dom(A)$. We refer to $r(A)$, i.e., the value of r at A , as the A *component* of r . A second-order table can be viewed as a type of one-level nested relation [Thomas and Fischer, 1986]. A second-order table T represents the conventional table consisting of rows that can be obtained by taking some row of T , replacing each of its nonempty entries by one of its elements and replacing the empty set by the null (unknown) value, denoted by "-". Figure 1 shows a conventional first-order decision table, Table 1, with eight rows, and a three row second-order decision table, Table 2 that represents it. For example, row 1 of Table 2 contains the same information that is in rows 1, 2, 4 and 5 of Table 1. The scheme of the tables contains four condition attributes and one action attribute (*Wait?*). Thus, row 3 of Table 2 can be expressed as:

$$\begin{aligned}
 & (Hungry? = \{no\}) \wedge (Type = \{Thai\}) \wedge (Cost = \{high, med\}) \wedge (\#Patrons = \emptyset) \\
 & \Rightarrow (Wait? = \{no\}).
 \end{aligned}$$

Letting D denote the domain of $\#Patrons$, this second-order rule can be interpreted as:

$$\begin{aligned} & [(Hungry? = no) \wedge (Type = Thai) \wedge (Cost = high) \wedge (\exists x \in D \text{ such that } \#Patrons = x)] \\ & \Rightarrow (Wait? = no) \\ \vee & [(Hungry? = no) \wedge (Type = Thai) \wedge (Cost = med) \wedge (\exists y \in D \text{ such that } \#Patrons = y)] \\ & \Rightarrow (Wait? = no). \end{aligned}$$

Thus, rules in second-order tables are more expressive than the simple disjunctive normal forms used in most machine learning systems. Using sets of values interpreted as disjunctions (choices), the second-order table framework uses a rich but comprehensible model to represent relatively complex logical rules compactly. With foundations in relational database theory, it has potential to provide easy integration with other relational database and decision support systems (e.g., SQL can be used to query second-order tables). From this point on, when the context is clear, the terms *table* and *row* (or *tuple* or *rule*) will refer to second-order structures.

We define the partial ordering *covers* on the set of all rows (over a fixed set of attributes) to be component-wise set inclusion. That is, row s is covered by row r if $s(A) \subseteq r(A)$ for each attribute A . A *flat* row has entries that are all either singletons or the empty set, and a *condition* is a row that associates the empty set to the class attribute. A decision table T associates a set of class values, denoted $T(c)$, to a flat condition c in the obvious way: $T(c)$ is the union of the class components of the rows in T that cover condition c . A table T *classifies* a condition c if $T(c)$ contains at least one value. Note that in applying a decision table to a condition to obtain a classification, an empty component in the condition is a “don’t care” value, since the corresponding component in the table’s rows are irrelevant. However, a “don’t care” value in the table is a complete component (i.e., containing all possible values for the attribute), since this makes the corresponding component of the condition irrelevant. If t and u are rules over the same scheme and t has more “don’t care” condition attributes (complete components) than u , we say that t is *shorter* than u . Shorter rules are preferable since they are easier to understand.

Two important concepts used in our induction are *equivalence* and *consistency*. The *flat extension* of table R is the set of all flat rows that are covered by at least one row in R . The *minimal flat extension* of a table is the set of maximal (with respect to covers) rows in its flat extension. A table S is said to *subsume* table R if the flat extension of R is a subset of the flat extension of S . Two tables are *equivalent* if each subsumes the other. Equivalent tables associate the same classification(s) to any given condition [Hewett and Leuchner, 1997a]¹. A condition having singleton components for all condition attributes is called a *simple* condition. A decision table R is *consistent* if it associates at most one class to any simple condition and is *consistent with* table S if $R(c)$ is a subset of $S(c)$ whenever c is simple and $S(c)$ is nonempty. For example, Table (2) in Figure 1 is consistent since each of the six simple conditions it covers has a unique classification. A decision table is *complete* if it classifies all simple conditions. Not all decision tables can be expected to be consistent. For example, a table describing antibiotics applicable to various diseases in patients with certain characteristics may associate more than one drug to a given condition.

¹ For a non-flat condition, a decision table assigns a value (classification) to the condition if the condition is subsumed by the set of rows in the table that assign that value to the class attribute.

3.2 The Table Transformations

Induction in SORCER is a transformation process in which an original table, representing a training data set, is transformed into a table with a minimal or near minimal number of rows that subsumes and is consistent with the original table. Thus, the approach can be viewed as a table compression technique.

SORCER applies two types of table transformations: *equivalence preserving* and *consistency preserving*. The transformation of a table R into table S is defined to be *equivalence preserving* if R is equivalent to S and *consistency preserving* if (1) S is consistent with R and (2) any simple condition which is covered by S but not by R is uniquely classified by S . In other words, a consistency preserving transformation does not introduce new inconsistency. Clearly, equivalence preserving operations preserve consistency.

We define three categories of equivalence preserving transformations: *rarefying*, which increase the size (i.e., number of rows) of a table, *condensing*, which decrease the size of the table, and *reshaping*, which change some component of a row. Let T be a table, and let r , s , and t be rules, and B be an attribute in the scheme of T . The *join* of r and s is defined to be a component-wise union of r and s , and the *A-join* (or *attribute join* over A) of r and s is defined to be the union of r and s on A but the intersection on all other attributes. In [Hewett and Leuchner, 1997a], we prove that the transformations below preserve equivalence.

Rarefying operations increase the length of a table. Rarefying operations include:

- *Add Directly Covered Rule:* t can be added to T if some rule in T covers t .
- *Add Redundant Rule:* t can be added to T if t is subsumed by T .
- *Split:* any r in T can be replaced by a pair of rules which agree with r on all attributes but one and whose join is r .
- *Add Attribute join:* for any attribute A , the A -join of any pair of rules in T can be added to T .

Condensing operations decrease the length of the table. Condensing operations include:

- *Delete Directly Covered Rule:* r can be removed from T if it is covered by some other rule in T .
- *Delete Redundant Rule:* r can be removed from T if it is subsumed by the other rules of T .
- *Merge local joinable rules:* a pair of rules in T which agree on all attributes except one (i.e., *local joinable* rules) can be replaced by their join.
- *Merge global joinable rules:* a pair of rules in T whose join is subsumed by T (i.e., *globally joinable* rules) can be replaced by their join.

Reshaping operations change some component of a row. Reshaping operations include:

- *Extend:* If r and s are in T and r covers s for all attributes except A , then s can be replaced by the A -join of r and s .
- *Reduce:* If r and s are in T and r covers s for all attributes except B , any value in $r(B)$ can be removed from $s(B)$.

Two types of consistency preserving transformation are defined below. Let r and s be rules, in relation R , that assign the same classification(s), and let A be a condition attribute.

Merge (or partially merge) consistently joinable rules (i.e., a pair of rules whose join does not introduce inconsistency.) There are three possible methods:

- *Cjoin*: Replace r and s by their join if this does not introduce inconsistency.
- *Njoin(n)*: If r covers s on n attributes, replace r and s by their join if this does not introduce inconsistency.
- *Pjoin(A)*: Let t be the join of r and s ; for each row u of R which classifies differently than t , if the conditions covered by u overlap those covered by t , remove from $t(A)$ the values in $u(A)$; t can then be added to R without introducing inconsistency.

Enlarge consistently expandable rules (i.e., rules whose attribute values can be added to one of its condition components without producing inconsistency)

- *Fatten (A)*: replace $r(A)$ by the largest possible set of A values that preserves consistency.

Adding attribute values may produce more "don't care" conditions in a table. These consistency preserving transformations "generalize" a relation to cover more conditions.

3.3 The Basic Algorithm

We can now describe the basic algorithm shown in Figure 2. Abstractly, the algorithm can be viewed as a search, through a complete hypothesis space of second-order tables, which converges to a suitable approximation of a target function. During the search, the algorithm uses an inductive bias that prefers the short and "dense" tables, where density is defined as the size of a table's minimal flat extension divided by its length. Rarefying and reshaping operations may enable further compression when backtracking is needed. Heuristics can be used to select an operation or pair of rows to merge. For example, attributes could be ranked by discriminatory power with respect to class. The system may look for pairs of rules that differ only on a particular attribute, starting from the least important attribute in terms of its influence on the classification variable. This may be beneficial since rules in a well-compressed table are likely to have larger attribute value sets for less important attributes. Cost (e.g., time) constraints may be necessary since some operations, e.g., removal of redundant rules, are likely to be prohibitively expensive in large tables. By applying equivalence preserving before consistency preserving transformations, the algorithm attempts to give priority to generalizing the relation according to the underlying structure of the knowledge partially formed from a training data set. (However, experience indicates that for some training data sets this does not produce significant differences in accuracy.)

Step (2) allows users to incorporate certain types of prior knowledge into the learning process, e.g., known monotone functional relationships between condition values and the class value in the target concept. For example, suppose that the domains of the class attribute C and some condition attribute A are both ordered and that a monotone non-decreasing functional relationship exists between A and C , i.e., if the value of A is increased, the value (class) of the target function cannot decrease. If a row has the greatest C value in its C component, then all A values greater than any value currently in its A component can be inferred and added.

Algorithm

Input: a (second-order) decision table T

Output: a (second-order) decision table R such that R subsumes and is consistent with T and the size of R is minimal or near minimal within cost constraints.

- (1) Apply *equivalence preserving transformations*, guided by heuristics, subject to cost constraints.
- (2) Infer additional rules or attribute values to components of individual rules.
- (3) Repeat Steps (1) and (2) until neither changes the relation.
- (4) Apply *consistency preserving transformations*, guided by heuristics, subject to cost constraints.
- (5) Go to Step (1). Stop when no further transformation has occurred within the cost constraints.

Fig. 2 Basic induction algorithm.

The classifier produced by the algorithm may not cover all possible conditions. For such cases, simple heuristics are applied to select a rule in the classifier to provide a classification. These heuristics include a preference for rules that (1) are closer to the test condition as measured by a “distance” function (e.g., the number of condition components not covered by the rule), (2) have a larger flat size, i.e., cover more conditions (and are therefore more general), or (3) give a classification that applies to a maximum number of covered conditions.

4 The SORCER System

Based on the theoretical framework of second-order relations [Hewett and Leuchner, 1997a, 1997b], we have implemented an experimental system, SORCER, written in C++ and using the C++ Standard Library (STL). Rules are implemented as STL bitsets so that basic operations on rows have an efficient underlying implementation based on bit vectors. The basic algorithm described in Section 3.3 can be instantiated at various levels of sophistication. SORCER facilitates interactive experimentation. A user can define induction algorithms (using several standard transformations) and test them by writing a “script” file of commands. For example, for the experiments in this paper, we apply two simple compression algorithms, C1 and C2, shown in Figure 3, omitting steps (1) and/or (2) from the basic algorithm of Figure 2. We use these very simple algorithms to ensure a strict comparison between our approach, based on second-order relations, and the approaches of other machine learning systems. Algorithms C1 and C2 use only the most basic techniques from our theory of second-order relations. We plan to use these results in future work to evaluate the contribution of using prior knowledge and statistical techniques within our system.

Since the order of training data may affect the result of compression, for fairness, we shuffle the table before compression in both algorithms C1 and C2. C1 simply merges pairs of consistently joinable rows until no more consistent joining is possible. Algorithm C1 is the simplest way to compress a (second-order) decision table while maintaining consistency.

C1: Shuffle Repeat Merge consistently joinable Until no change	C2: Shuffle Repeat Merge locally joinable Until no change If % partial rows > n then Merge preferred consistently joinable l else Merge all preferred consistently joinable l
---	---

Fig. 3 Two Compression Algorithms.

Algorithm C2 first merges locally joinable pairs of rows until no more such joins are possible and then preferentially merges consistently joinable pairs of rows based on a heuristic “preference” measure. In this way, C2 applies *generalizing* operations only after it has partially formed a classifier using only *equivalence* preserving transformations of the training data. The preference measure for merging a pair of rows is given by a function that assigns values between 0 and 1 to pairs of rows in such a way that smaller values indicate pairs to be preferred for merging. SORCER provides two methods to preferentially merge consistently joinable rows. In the first, given a preference limit l , pairs of rows whose preference value is no more than l are merged until the preference value of every pair of rows in the table exceeds l . This operation is invoked by the command *merge preferred consistently joinable l*. In the second, invoked by the command *merge all preferred consistently joinable l*, the preceding operation is applied repeatedly with preference limits ml for $m = 1, 2, 3, \dots$, until $ml \geq 1$. C2 uses the first method if the data set has missing values in $n\%$ or more of its rows and the second if it does not. The rationale for this is that rows with empty components contain less information, and are therefore less useful for generalization, and the first method results in fewer merges of rows involving such rows, at least with an appropriate preference function. Different preference functions can be specified in SORCER, and the particular preference measure used by C2 is defined by:

$$p(r, s) = \frac{1}{|\mathcal{C}|} \sum_{A \in \mathcal{C}} \frac{(|\text{dom}(A)| - |(r \sqcap s)(A)|) g(r, A) g(s, A)}{|\text{dom}(A)|^3},$$

where (1) r and s are rows, (2) \mathcal{C} is the set of condition attributes, (3) $\text{dom}(A)$ is the *domain* of A (i.e., the set of values for A), (4) $r \sqcap s$ is the component-wise intersection of r and s (the *meet* of r and s), and (5) $g(t, A) = \min\{|\text{dom}(A)|, |\text{dom}(A)| + 1 - |t(A)|\}$, for row t and attribute A . The expression within the summation sign is equivalent to the product of three factors. The first is the proportion of $\text{dom}(A)$ that is not in $r(A) \cap s(A)$, which gives preference to pairs of rows whose components overlap on large proportions of the condition domains. The other two factors are approximately the proportions of $\text{dom}(A)$ not in $r(A)$ and not in $s(A)$, which gives preference to merging rows with large components. The use of $g(t, A)$ rather than $|\text{dom}(A)| - |t(A)|$ in these factors prevents them from becoming either equal to zero or greater than one. Division by $|\mathcal{C}|$, the number of condition attributes, normalizes the preference measure to a value between zero and one. Thus, C2 favors merging pairs of rows that share many values for many attributes, especially if they share most of the values in attributes’ domains, but also favors merging “fat” rows, i.e., rows with large minimal flat extensions.

If the condition in a test example is not covered by a rule in the classifier, SORCER uses the first rule that covers the test condition on the greatest number of attributes. If no rule in the classifier covers the test condition on any attribute, the class appearing most often (in the classifier) is used.

5 Experiments and Results

We compare SORCER to three classification systems: C4.5, CBA and a Naive Bayesian classifier. These systems were selected for the following reasons. First, most published results report on accuracy but not the size of the model. Second, these published results were obtained from experiments on data sets that have continuous attribute values. In some classification systems including CBA and ours, a preprocessing step to discretize these data is required. To obtain a fair comparison, the experiments should use the same discrete data sets which, in our case, are made available through the published results of CBA in [Liu et al., 1998]. Finally, we would like to compare SORCER with systems that use representational models other than decision tables. C4.5 [Quinlan, 1993], a decision tree learner and CBA [Liu et al., 1998], an association rule-based classifier are among top performers of state of the art classification systems that have been applied successfully in various domains. The Naive Bayesian, a probability-based classifier, has been shown to be robust and can achieve accurate models even when there is a violation of the independence assumption [Domingos and Pazzani, 1996].

We used 26 data sets from the UCI Machine Learning Repository [Murphy and Aha, 1994]. These data sets have from two to ten classes and between five and 39 attributes. Twelve data sets contain inconsistent data and ten have missing values. The majority of the data sets contain attribute values requiring discretization. In order to compare SORCER with existing systems such as C4.5 and CBA, we used the same discretization boundaries (from the SGI MLC++ machine learning utilities [Kohavi et al., 1994]), as reported in [Liu et al., 1998] and removed inconsistency in training data before compression. SORCER provides an operation to resolve inconsistency in a training set of (flat) examples. If a condition is associated with two or more classes, the examples with the class that occurs less frequently for the condition are removed. If the classes occur with the same frequency, examples with the classes that occur less frequently in the entire table are removed. If the classes occur with the same frequency in the table, all but the first occurrence of a rule for the condition are eliminated.

To measure prediction accuracy, we applied 10-fold cross-validation [Kohavi et al., 1998]. A data set is randomly partitioned into 10 approximately equally sized subsets (or folds or tests). The induction algorithm is executed 10 times; each time it is trained on the data that is outside one of the subsets and the generated classifier is tested on that subset. The estimated accuracy for each cross-validation fold is a random variable that depends on the random partitioning of the data. So, for each data set, we repeated 10-fold cross-validation 10 times. The estimated accuracy is the average over the ten 10-fold cross-validations.

Figure 4 compares the results obtained by SORCER using algorithms C1 and C2 with the results produced by the Naive Bayesian classifier (as implemented in [Liu et al., 2000]), C4.5 (Release 8) and CBA (using 50% minimum confidence, 1% minimum support, and a limit of 80,000 candidate rules with pruning) as reported in [Liu et al., 1998; 2000]. Algorithm C2 was used with a preference limit $l = 0.2$ and a limit for the proportion of rows with missing values of $n = 80\%$. Results are given as percent error rates (i.e., the percent ratio of the number of misclassifications to the number of test

examples). For each data set, Figure 4 shows the average and standard deviation of SORCER's error rates over ten 10-fold cross-validations.

Our results show that SORCER can achieve remarkably high accuracy even using algorithm C1, the simplest consistency preserving compression method. (Ten data sets have low error rate of less than 10%, and in particular, results in the wine and hypo data sets are about 1%.) SORCER's error rates are lower than other systems for the data sets whose names are in bold type. The third to the last column shows the average training and testing time for one cross-validation test on C1 on a PC with a Pentium III 600 MHz processor. The run time of C1 for most of the data sets was under one second (and under 0.1 seconds for thirteen data sets). Five data sets (i.e., *german*, *hypo*, *sick*, *tic-tac-toe* and *vehicle* data sets) have running time a little over one second and all of them have relatively large domain sizes. The *waveform* data set has the largest compressed table and longest average running time. Similar relationships between data characteristics and accuracy or running time were observed on results obtained from SORCER with algorithm C2.

Data set	Naive Bayesian	C4.5 Tree	C4.5 Rules (disc.)	C4.5 Rules	CBA V1 (prun.)	CBA V2 (prun.)	SOR CER (C1)	C1's Stdev.	SOR CER (C2)	C2's Stdev.	C1's Time (secs)	C1's avg. #Rules	CBA V1 #Rules
anneal	2.7	7.5	6.5	5.2	3.6	2.1	6.0	0.18	3.7	0.31	0.294	5	34
australian	14.0	14.8	13.5	15.3	13.4	14.6	16.6	0.73	17.6	0.57	0.630	77	148
auto	32.1	17.6	29.2	19.9	27.2	19.9	21.6	1.61	21.5	2.22	0.037	27	54
breast	2.4	5.6	3.9	5.0	4.2	3.7	4.7	0.49	4.4	0.39	0.115	28	49
cleve	17.1	21.5	18.2	21.8	16.7	17.1	19.2	1.47	17.9	1.15	0.092	45	78
crx	14.6	15.0	15.9	15.1	14.1	14.6	17.8	0.98	17.0	0.91	0.663	84	142
diabetes	24.4	26.1	27.6	25.8	25.3	25.5	22.4	0.50	22.4	0.64	0.059	27	57
german	24.6	28.4	29.5	27.7	26.5	26.5	31.9	1.22	30.9	1.15	1.335	167	172
glass	29.4	30.4	27.5	31.3	27.4	26.1	23.2	0.62	22.1	0.62	0.015	18	27
heart	18.1	21.8	18.9	19.2	18.5	18.1	17.1	1.36	17.4	1.48	0.045	38	52
hepatitis	15.0	18.2	22.6	19.4	15.1	18.9	15.7	1.64	16.1	2.17	0.036	15	23
horse	20.6	14.7	16.3	17.4	18.7	17.6	31.0	2.22	18.8	0.73	0.181	20	97
hypo	1.5	0.7	1.2	0.8	1.7	1.0	1.1	0.11	1.1	0.07	1.029	18	35
ionosphere	11.9	10.5	8.0	10.0	8.2	7.7	7.1	0.70	7.3	0.47	0.156	17	45
iris	6.0	4.7	5.3	4.7	7.1	5.3	5.8	1.18	5.7	1.10	0.009	5	5
labor	14.0	22.3	21.0	20.7	17.0	13.7	8.2	2.20	6.8	1.00	0.010	2	12
led7	26.7	30.5	26.5	26.5	27.8	28.1	26.7	0.18	26.7	0.20	0.383	51	71
lymph	24.4	23.8	21.0	26.5	19.6	22.1	23.2	2.67	18.8	1.59	0.040	24	36
pima	24.5	25.8	27.5	24.5	27.6	27.1	22.7	0.96	22.7	0.85	0.060	28	45
sick	3.9	1.1	2.1	1.5	2.7	2.8	2.6	0.09	2.6	0.07	1.862	23	46
sonar	23.0	28.4	27.8	29.8	21.7	22.5	19.8	1.30	22.3	2.14	0.069	28	37
tic-tac-toe	30.1	13.8	0.6	0.6	0.0	0.4	8.4	0.49	8.2	0.52	1.497	60	8
vehicle	40.1	28.5	33.6	27.4	31.3	31.0	30.6	0.61	30.7	1.16	1.029	135	125
waveform	19.3	22.8	24.6	21.9	20.6	20.3	21.2	0.40	21.7	0.50	59.93	712	386
wine	9.5	7.3	7.9	7.3	8.4	5.0	0.8	0.55	0.4	0.35	0.045	5	10
zoo	13.7	7.8	7.8	7.8	5.4	3.2	5.1	0.91	6.1	0.91	0.023	9	7
Average	17.8	17.3	17.1	16.7	15.8	15.2	15.8	0.976	15.0	0.895	2.68	64.2	69.3

Fig. 4 Average error rates obtained by Naive Bayesian, C4.5, CBA and SORCER.

On the 26 data sets used for our experiments, SORCER performs competitively with Naive Bayesian classifier, C4.5 and CBA. On the average, the error rate produced by C1 is as good as that of CBA V1 which is lower than the error rate of C4.5 and Naive Bayesian classifier. The average error rate of CBA decreases from 15.8% to 15.2% when V2 is used instead of V1. However, SORCER's average error rate decreases from 15.8% to 15.0 % when C2 is used instead of C1. Most of this decrease is due to the improved performance of C2 on the *horse* data set. (Eliminating this one data set reduces the error rate of CBA V2.0 to about 15.1%, that of C1 to about 15.2% and the error rate of C2 to about 14.9%.) C2 produced classifiers with lower average error rates than those of C4.5 Rules and CBA V2 on 14 and 12 of the 26 data sets, respectively. Using a one-tailed *t*-test on the *mean pair difference* statistic [Lapin, 1973], C2's mean error rate is lower than those of Naive Bayesian classifier (with $t = -2.431$) and C4.5 Rules (with $t = -1.807$) but not CBA V2 (with $t = -0.249$) at a significance level 0.05 ($t_{0.05} = -1.706$).

These results, and the average number of rule sets sizes (in the last two columns of Figure 4), lead us to speculate that there are measurable properties of rule sets that distinguish between those on which SORCER performs better and those on which CBA has an advantage, e.g., the number of conditions in the classification rules versus the number of rules in the classifier. As observed by Holte [Holte, 1993], simple rules (e.g. 1-rules or rules that classify based on a single attribute) can yield surprisingly high performance. CBA tends to create a large number of classification rules containing a small number of condition attributes, whereas SORCER does the opposite.

6 Computational Complexity

Figure 5 gives examples of orders of complexity for computations involved in the table induction algorithm. It can be shown that determining whether a table does not subsume a row is NP-complete [Hewett and Leuchner, 1997a]. It follows that determining whether a row is subsumed by a table is in NP-Hard. The NP-complete "minimum disjunctive normal form" problem [Garey and Johnson, 1979] can be reduced to the problem of deciding whether a given table is equivalent to a table with K or fewer rows so that this problem too is NP-hard.

Type	Computation	Complexity
Row comparisons and operations	$r = s ?$, r covered by $s ?$	$O(k)$
	r joins s	$O(k)$
Condition checking	Is a pair of tuples locally joinable?	$O(k)$
	Is a pair of tuples globally joinable?	NP-Hard
Transformations	Merge locally joinable pairs to a fixed point.	$O(kn^2)$
	Delete all directly covered rules.	$O(kn^2)$
	Reshape all appropriate tuples found in one pass.	$O(kn^2)$
	Reshape all appropriate tuples to a fixed point.	$O(k^2n^3)$
	Merge consistently joinable pairs to a fixed point	$O(kn^3)$
$k =$ sum of domain size of each attribute in the scheme and $n =$ length of the table		

Fig. 5 Complexity of some computation.

Several techniques may increase the efficiency of table compression computations. Less expensive operations can be performed first to reduce the size of the table as much as possible before more expensive transformations are applied. For example, directly covered rules can be removed before attempting to remove redundant rules, and locally joinable pairs can be merged before globally joinable pairs. SORCER also uses the *intersection method*, described in [Hewett and Leuchner, 1997a], which can determine subsumption efficiently when a table has a small number of "fat" rules. SORCER also provides heuristics for selecting the rules and attributes to use in applying transformations. Rules can be ordered by the sizes of their minimal flat extensions in order to use either larger or smaller rules first. Condition attributes can be ranked with respect to their expected influence on the class attribute, using opinion of domain experts or information theory metrics such as entropy and gain [Mitchell, 1997].

7 Conclusion

We have presented an induction system that induces compact decision tables using a second-order relational framework. Although second-order relations have been, for some time, implicitly defined in database theory, our framework for second-order relation compression is unique and more recent. Experiments show that over all our approach using simple heuristics is competitive to (and on the average of the 26 data sets studied, is slightly better than) existing systems. In the future, we would like also to compare our approach with an induction algorithm based on standard decision tables. Other future work includes exploring strategies for handling inconsistent data and further analysis of the factors that impact our algorithm. The latter can be done by experimentation on synthetic data sets with various characteristics (e.g., many irrelevant variables and noisy data). For inconsistent training data sets, SORCER can be modified to generate inconsistent classifiers while avoiding the introduction of new inconsistency during generalization. Strategies for applying inconsistent classifiers may be as simple as using the first applicable rule or reporting all possible classes. With respect to comprehensibility, we currently only use the size of a table as an approximate measure for model comprehensibility. We would like to investigate other ways to assess comprehensibility of second-order tables. We plan to continue trying to improve SORCER's classification accuracy with better heuristics, adaptation to the individual characteristics of data sets, and by incorporating appropriate forms of prior domain knowledge, which can be verified by domain experts.

References

- Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A Fast discovery of association rules. In Fayad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.): *Advances in Knowledge Discovery and Data Mining*. AAAI Press/ The MIT Press, Menlo Park, CA, 307-328, 1996.
- Apte, C. and S. Hong, Predicting equity returns from securities data. In Fayad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.): *Advances in Knowledge Discovery and Data Mining*. AAAI Press/ The MIT Press, Menlo Park, CA, 307-328, 1996.
- Bayardo, R., Agrawal, R.: Mining the most interesting rules. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 145-154, 1999.

- Blumer, A., A. Ehrenfeucht, D. Haussler, and M. Warmuth, Occam's razor, *Information Processing Letters*, 24, 377-380, 1987.
- Chater, N., Reconciling simplicity and likelihood principles in perceptual organization, *Psychological Review* 103 (3): 566-581, 1996.
- Codd, E. F., 1985. "Does your DBMS run by the rules?", *Computer World*, October 21, 1985.
- Domingos, P., The role of Occam's Razor in Knowledge Discovery, *Data Mining and Knowledge Discovery*, 3: 409-425, 1999.
- Domingos, P. and M. Pazzani, Beyond independence: conditions for the optimality of the simple Bayesian classifier, in L. Saitta, ed., '*Machine Learning: Proceedings of the Thirteenth International Conference*', Morgan Kaufmann, pp. 105-112, 1996.
- Fayad, U., G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining*: AAAI Press / The MIT Press, Menlo Park, California, 1996.
- Garey, M. And D. Johnson, *Computers and Intractability*, W.H. Freeman and Co., 1979.
- Glymour, C. and G. Cooper, (eds.), *Computation, Causation & Discovery*, AAAI Press/The MIT Press, Cambridge, Massachusetts, 1999.
- Han, J. and Y. Fu, Discovery of Multiple-Level Association Rules from Large Databases, in *Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, 420-431, 1995.
- Han, J., Y. Cai and N. Cercone, Knowledge Discovery in Databases: An Attribute-Oriented Approach, in *Proceedings of the 18th International Conference on Very Large Databases*, Vancouver, Canada, 1992.
- Hewett, R. and J. Leuchner, Second-order relations and Decision Tables, Technical Report 97-27, CSE Department, Florida Atlantic University, Boca Raton, FL, 1997a.
- Hewett, R. and J. Leuchner, A local joinability condition and a computational technique for second order relations and decision tables, Technical Report 97-54, CSE Department, Florida Atlantic University, Boca Raton, FL, 1997b.
- Holte, R., Very simple classification rules perform well on most commonly used datasets, *Machine Learning*, 11, 63-91, 1993.
- Hong, S., R. Cain and D. Ostapko, MINI: A Heuristic Approach for Logic Minimization, *IBM Journal of Research and Development*, 443-458, 1974.
- Hong, S., R-MINI: A Heuristic Algorithm for Generating Minimal Rules from Examples, in *Proceedings of the Third Pacific Rim International Conference on Artificial Intelligence*, PRICAI'94, 331-337, 1994.
- Kohavi, R. and D. Sommerfield, Targeting Business Users with Decision Table Classifiers, in *Proceedings of Conference on Knowledge Discovery and Data Mining*, 1998.
- Kohavi, R., The power of decision tables, in *Proceeding of the European Conference on Machine Learning (ECML)*, 1995.
- Kohavi, R., G. John, R. Long, D. Manley and K. Pflieger. MLC++: a machine learning library in C++, *Tools with artificial intelligence*, 740-743, 1994.
- Langley, P., Induction of condensed determinations, in *Proceedings of the Second International Conference of Knowledge Discovery and Data Mining*, pp. 327-330, AAAI Press, 1996.
- Lapin, L., *Statistics for Modern Business Decisions*, Harcourt Brace Jovanovich, Inc., 1973.
- Leuchner, J. and R. Hewett, A Formal Framework for Large Decision Tables. In *Proceedings of International Conference of Knowledge Retrieval, Use, and Storage for Efficiency Symposium (KRUSE' 97)*, 165-179, 1997.

- Liu, B., Hsu, W., Ma, Y.: Pruning and summarizing the discovered associations. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 125-134, 1999.
- Liu, B., Y. Ma and C. Wong, Improving an exhaustive search based rule learner, in *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2000)*, Lyon , France, 2000.
- Liu, B., W. Hsu and Y. Ma, Integrating Classification and Association Rule Mining, in *Proceedings of Knowledge Discovery and Data Mining*, New York, 1998. (Also, <http://www.comp.nus.edu.sg/~dm2/result.html>)
- Mani, S. and Pazzani, M., Guideline Generation from Data by Induction of Decision Tables Using a Bayesian Network Framework, *JAMIA supplement*, pp. 518-522, 1998.
- Michalski, R., I. Mozetic, J. Hong and N. Lavrac, The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three medical Domains, in *Proceedings of the AAA-I 86*, 1041-1045, 1986.
- Mitchell, T., *Machine Learning*, McGraw-Hill Companies, New York, New York, 1997.
- Modrzejewski, M., Feature selection using rough sets theory, in *Proceedings of the European Conference on Machine Learning*, pp. 213-226, 1993.
- Muggleton, S. and C. Feng, Efficient Induction of Logic Programs, in *Proceedings of the First Conference on Algorithmic Learning Theory*, 368-381, Tokyo, 1990.
- Murphy, P. and D. Aha, UCI Repository of machine learning databases, Department of Information and Computer Science, University of California, Irvine, CA, 1994.
- Quinlan, J., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California, 1993.
- Thomas, S.J., P.C. Fischer, "Nested Relational Structures," *The theory of Databases, Advances in Computing Research III*, ed. By P.C. Kanellakis, JAI Press, 269 – 307, 1986.
- Wolff, J., Computing, Cognition and Information Compression, *AI Communication* 6(2): 107-127, 1993.
- Ziarko, W., The discovery, analysis and representation of data dependencies in databases, in G. Piatetsky-Shapiro & W. Frawley, eds, *Knowledge Discovery in Databases*, MIT Press., 1991.