Mining Classification Rules from Datasets with Large Number of Many-Valued Attributes

Giovanni Giuffrida[1], Wesley W. Chu[1], and Dominique M. Hanssens[2]

[1] Dept. of Computer Science, Univ. of California, Los Angeles
giovanni@cs.ucla.edu, wwc@cs.ucla.edu
[2] Anderson Grad. School of Manag., Univ. of California, Los Angeles
dominique.hanssens@anderson.ucla.edu

Abstract. Decision tree induction algorithms scale well to large datasets for their univariate and divide-and-conquer approach. However, they may fail in discovering effective knowledge when the input dataset consists of a large number of uncorrelated many-valued attributes. In this paper we present an algorithm, Noah, that tackles this problem by applying a multivariate search. Performing a multivariate search leads to a much larger consumption of computation time and memory, this may be prohibitive for large datasets. We remedy this problem by exploiting effective pruning strategies and efficient data structures. We applied our algorithm to a real marketing application of *cross-selling*. Experimental results revealed that the application database was too complex for C4.5 as it failed to discover any useful knowledge. The application database was also too large for various well known rule discovery algorithms which were not able to complete their task. The pruning techniques used in Noah are general in nature and can be used in other mining systems.

1 Introduction

Decision tree induction algorithms, such as C4.5 [15], are characterized by the following two properties:

(i) *Univariate splitting.* The partitioning criteria is based on a single variable at a time. Therefore, for n independent variables, n partitions are compared with each other. The variable that generates the partition with the best statistical significance is chosen as the next *test* [15].

(ii) *Divide–and–Conquer approach.* After a univariate split, each child node covers only a subset of the initial dataset. Thus, subsequent splits are based on the remaining portions of the training set.

Although these two properties are the backbone of the efficient implementation of decision tree induction algorithms, they limit their learning ability in certain situations. In this paper, we first discuss this shortcoming in more details. Then, we present a rule discovery algorithm, Noah, that performs a more exhaustive search than C4.5 because it uses a *multivariate* approach. We introduce effective pruning strategies to control the combinatorial explosion of the multivariate search. Then, we discuss the application of Noah to a real marketing cross-selling application. The application dataset was too large to be processed by CN2 [4], Ripper [5], CBA [11], and Apriori [1] and too complex for C4.5 to induce useful knowledge.

2 Preliminaries

The goal of a classifier is to *learn* a body of knowledge $\mathcal{M}$ from an input dataset I. The derived knowledge $\mathcal{M}$ can then be used to *predict* (i.e., classify) new tuples. Let us consider a set of n *independent* variables $X_1, \ldots, X_n$ such that each X_k takes on values from a domain D_{X_k}. In addition, we have another variable X_c, called the *class variable*, whose domain is $D_C = \{c_1, \ldots, c_m\}$, with m being the number of classes. The task of a classifier is:

(i) Given a training dataset I consisting of a set of *(n+1)*-tuples: $(t_1, \ldots, t_n, c)$, where $t_k \in D_{X_k}$ $(k = 1, \ldots, n)$ and $c \in D_C$;
(ii) Construct a mapping $\mathcal{M} : (D_{X_1}, \ldots, D_{X_n}) \rightarrow D_C$.

$\mathcal{M}$ can now be used to predict the class of new tuples. Thus, given a n-tuple $t' = (t'_1, t'_2, \ldots, t'_n)$, such that $t'_i \in D_{X_i}, i = 1, \ldots, n$, the predicted class c' will be: $c' = \mathcal{M}(t')$.

Let "$X = v$" be a *term* where X is an independent variable and v one of its values, $v \in D_X$. A term "$X = v$" *covers* a tuple t when the attribute X in t has value v.[1] Let a *pattern* be a *conjunction* of *terms*. A pattern ρ *covers* a tuple t when *all* the terms in ρ cover t. A rule r is a statement of the form: "if ρ then Φ" where ρ is a pattern and Φ is a class distribution. r covers a tuple t when ρ covers t. The *support* of r, *supp*(r), is the number of tuples in I covered by ρ. Φ is the class distribution over the tuples covered by r. Φ is represented as a vector of m counters, i.e., one counter for each class, of the form: $[n_1, n_2, \ldots, n_m]$, where each n_i is the number of tuples in the training set that are (1) covered by r and (2) whose class attribute is c_i. Thus, $supp(r) = \sum_i n_i$. The *confidence* of a rule "if ρ then Φ" is a measure of *goodness* of the rule, it is function of Φ and is often based on the *entropy* concept [15]. (More details on the "confidence" in Noah are discussed later.)

3 Shortcomings of tree induction algorithms

Let us now discuss the shortcomings of the univariate and divide-and-conquer approaches used in C4.5.

Shortcomings of the univariate splitting. Consider the database of Figure 1(a) where "PLAY?" is the class attribute. We can synthesize it as: *"We do not play tennis when it is hot and highly humid at the same time; we play in all other*

[1] We assume only discrete variables.

Temp	Humid.	Play?
hot	normal	yes
hot	medium	yes
hot	dry	yes
cool	high	yes
mild	high	yes
v_hot	high	yes
hot	high	no
hot	high	no
hot	high	no

(a) The training set

(b) The tree produced by C4.5

Fig. 1: The univariate approach of C4.5 fails to discover the evidence that `(hot,high)` leads to `no`

circumstances." We ran C4.5[2] over this dataset; it induced the *one-node* tree shown in Figure 1(b). Basically, C4.5 does not find any good predictive variable, among Temp and Humidity, to split the database upon. Therefore, it is unable to learn the strong rule: "if Temp=hot and Humid=high then No." Thus, C4.5 overlooks this piece of knowledge which is described by the *interaction* of the two independent variables. Consequently, the tree in Figure 1(b) misclassifies the tuple `<hot,high>` which should be classified as "No."

Shortcomings of the divide-and-conquer approach. Let us now consider the training set in Figure 2(a). By running C4.5 over it, we get the tree shown in Figure 2(b). This is a perfect tree because each leaf covers only tuples of the same class. However, C4.5 did not discover the rule r_1:"if Humid=high then No." This is because r_1 is *subsumed* by the more general rule r_2:"if Temp=hot then No" that was discovered by choosing Temp as the splitting variable at the first iteration. Consequently, when we classify the new tuple `<cool,high>`, C4.5 uses the rightmost leaf of the tree in Figure 2(b) and predicts "Yes." In the training set, however, there is much more evidence of "if Humid=high then No" than "if Temp=cool then Yes", therefore, it would make more sense to classify the input tuple as "No."

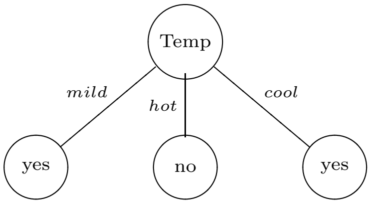
These two shortcomings penalize C4.5 when dealing with datasets that have a large number of uncorrelated many-valued variables[3] where each variable, when taken alone, has low predictive power. Under such circumstances the knowledge can be scattered among many rectangular portions of the input relation which are difficult to be learned by C4.5. As discussed in Section 5, C4.5 was not able to derive a meaningful tree from a database of a real-world application with a

[2] We used Release 8 of C4.5 available at `www.cse.unsw.edu.au/~quinlan` with all default settings.

[3] Intuitively, a "many-valued variable" is a variable that takes on values from a large domain.

Temp	Humid.	Play?
mild	dry	yes
mild	normal	yes
mild	dry	yes
hot	high	no
hot	high	no
hot	high	no
hot	high	no
hot	dry	no
hot	normal	no
cool	dry	yes

(a) The training set



(b) The tree produced by C4.5

Fig. 2: The Divide-and-Conquer approach of C4.5 fails to discover subsumed patterns

large number of many-valued attributes. This motivated us to develop the Noah rule discovery algorithm that performs a more exhaustive search than C4.5.

By running Noah over the dataset of Figure 1(a) we get the rule: "if Temp=hot and Humid=high then No (conf=1, supp=3)." Noah uses this rule when asked to classify the new case `<hot,high>` and correctly outputs "No" as the predicted class. From the dataset of Figure 2(a), Noah discovers, among others, the rule: "if Humid=high then No." When asked to classify the new tuple `<cool,high>`, Noah uses this rule and properly classifies the new tuple as "No."

Performing a multivariate search leads to a much larger consumption of computation time and memory, this may be prohibitive for large datasets. We remedied to this problem by exploiting effective pruning strategies and efficient data structures.

4 The NOAH algorithm

Noah is based on the well known level-wise approach used by many rule induction systems. This approach was first proposed by Agrawal *et al.* [1] in their *Apriori* algorithm for discovering association rules. In such an approach, rules are refined in a general-to-specific fashion. That is, rules are derived by progressively refining their pattern at each iteration. The main strength of this algorithm is its ability of pruning *infrequent* patterns in a hierarchical way (refer to [12] for a nice formalization of this problem).

The Noah algorithm is outlined in Figure 3. We first perform a reordering of the terms in the input relation (line 1 of the algorithm in Figure 3), which is described in detail in Section 4.1. The set R_k, containing all k-term rules, is initialized to $\emptyset$ at the beginning of each iteration (line 2). The set S_k (line 4) contains all possible k-patterns created from the current tuple t at the *k-th* iteration ($k = 1, 2, \dots$). In turn, for each pattern ρ of S_k the set T_k of all sub-patterns of ρ with cardinality $k-1$ is computed. The `if` statement of line 6 tests the existence of all such sub-patterns: If all elements of T_k exist in R_{k-1} (line 6) and a rule with pattern ρ does not already exist in R_k (line 7), then a new rule with pattern ρ is created and inserted into R_k (line 8). The class distribution of the rule with pattern ρ is then updated (line 9).

Once all tuples in the input set I have been visited, a set of k-term rules is contained in R_k. We then prune meaningless rules in R_k as follows: For each rule in R_k we first perform the minimum support pruning (line 10) as in *Apriori*. Then, as proposed by [2], we remove from R_k all rules having confidence greater than the $minConf$ threshold (line 11) and store them in R_{final} (line 12). Notice that Noah uses a different strategy from other induction systems, such as CN2 and Ripper, that always look for the most overall confident rules. In Noah, the user sets a lower bound for the rule confidence ($minConf$). Every rule whose confidence is greater than $minConf$ is considered satisfactory. While this strategy may penalize the overall accuracy of the system, it provides early stop of rule growing and thus reduces the computation complexity and improves the scalability. Furthermore, this allows the implementation of the pruning-ahead "confidence upper bound" strategy (discussed in Section 4.2). Similarly to the minimum support concept, the rationale to setup $minConf$ is application-dependent. Then, two other pruning techniques are invoked (lines 13 and 14). They are based on the "confidence upper bound" and "term dependency concepts, their details are discussed in Section 4.2 and 4.3, respectively.

After the pruning process, Noah starts a new iteration. This process continues as long as R_k contains rules to be refined. Since we never partition the input dataset in Noah, all rules are always induced from the entire training set. This alleviates the *small disjunct problem* [9, 7].

4.1 Terms Reordering

To optimize rule lookup, all terms from the training set are ordered according to their support. Thus, a new version of the input dataset is created where each term is replaced by a numerical id that corresponds to the sorted order. Such a *tokenized* representation enables Noah to implement a *bit index* structure to fast test existence of *conjunctions* of terms (line 6 of the algorithm in Fig. 3).

4.2 Pruning Rules by "Confidence Upper Bound"

Recall that in Noah the user specifies a minimum value for the rule confidence ($minConf$). Once the confidence of a rule is larger than $minConf$ the rule will not be further refined. This property allows us to prune rules ahead by estimating, after each iteration, whether or not a certain rule will ever satisfy the minimum confidence constraint.

In Noah, the rule confidence $\mathcal{C}(P)$ is computed as:

$$\mathcal{C}(P) = 1 - \frac{-\sum_i^k p_i \ln{(p_i)}}{\ln{(k)}} \qquad (1)$$

```
input  : Input Database $I$
output : A Set of Rules $R_{final}$

TermReordering( $I$);
$R_{final} \leftarrow \emptyset$;
$k \leftarrow 1$;
while $k = 1$ or $R_{k-1} \neq \emptyset$ do
    $R_k \leftarrow \emptyset$;
    foreach tuple $t \in I$ do
        $S_k \leftarrow$ {k-patterns from t};
        foreach pattern $\rho \in S_k$ do
            $T_k \leftarrow$ {(k-1)-patterns from $\rho$};
            if ($T_k \subseteq R_{k-1}$) or ($k = 1$) then
                if there is not a rule with pattern $\rho$ in $R_k$ then
                    $R_k \leftarrow R_k \cup$ { "if $\rho$ then [0,0,...,0]" };
                end
                increment $t.class$ of the rule whose pattern is $\rho$;
            end
        end
    end
    $R_k \leftarrow R_k / \{\rho \in R_k \mid supp(\rho) < minSupp\}$;
    $S \leftarrow \{\rho \in R_k \mid conf(\rho) \geq minConf\}$;
    $R_k \leftarrow R_k / S$;
    $R_{final} \leftarrow R_{final} \cup S$;
    Prune $R_k$ by Confidence Upper Bound (see Section 4.2);
    Prune $R_k$ by Term Dependency (see Section 4.3);
    increment $k$;
end
```

Fig. 3: The Noah algorithm

where P is the class probability distribution $(p_1, p_2, \ldots, p_k)$ and k is the number of classes. The numerator in (1) is the entropy of the class probability distribution. The $\ln(k)$ in the denominator is used to normalize the entropy so that its value falls in the range 0 to 1. For a given rule, the larger the value of $\mathcal{C}$ the more *confident* we are about that rule.

In the following we will restrict our study to the two-class case (i.e., $k = 2$). We first need to prove the following proposition.

Proposition 1 The confidence $\mathcal{C}$ of a given rule is monotonically increasing w.r.t. the absolute difference between p_1 and p_2 (where p_1 and p_2 are the probability of the first and second class, respectively).

Proof. We proceed by rewriting (1) in terms of $\Delta = \mid p_1 - p_2 \mid$. Let us first consider the case $p_1 \geq p_2$. We have:

(i) $p_1 + p_2 = 1$, for the axiom of probabilities.
(ii) $\Delta = p_1 - p_2$.

After some manipulations of these two equations, we have $p_1 = \frac{1+\Delta}{2}$ and $p_2 = \frac{1-\Delta}{2}$. By substituting p_1 and p_2 into (1) we rewrite $\mathcal{C}$ as function of Δ. For the two class case, we have:

$$\hat{\mathcal{C}}(\Delta) = 1 - \frac{-\frac{1+\Delta}{2}\ln(\frac{1+\Delta}{2}) - \frac{1-\Delta}{2}\ln(\frac{1-\Delta}{2})}{\ln(2)} \qquad (2)$$

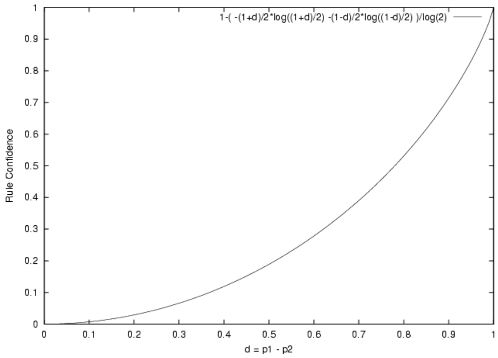


Fig. 4: Rule Confidence is a monotonically increasing function of $\mid p_1 - p_2 \mid$

```
1. Given a rule $r$, being $c_1$ and $c_2$ the population of the first and second class of $r$, respectively;
2. Let $d = c_1 + c_2 - minSupp$;
3. Let $c_{small} = min(c_1, c_2)$;
4. Let $c_{large} = max(c_1, c_2)$;
5. Let $c_{small} = c_{small} - d$;
6. if $c_{small} < 0$ then $c_{small} = 0$;
7. Let $tot = c_{small} + c_{large}$;
8. Let $p_1 = c_{small}/tot$;
9. Let $p_2 = c_{large}/tot$;
10. if $\mathcal{C}([p_1, p_2]) < minConf$, then drop the rule $r$;
```

Fig. 5: The CUB pruning algorithm

The sign of the first derivative of (2) is always positive in the interval $(0, 1)$, therefore $\hat{\mathcal{C}}$ is monotonically increasing. The plot in Fig. 4 shows the monotonicity of $\hat{\mathcal{C}}$ w.r.t. $p_1 - p_2$. In the same manner, we can prove the monotonicity of $\hat{\mathcal{C}}$ for the case of $p_1 < p_2$. □

In our pruning strategy we combine the results of this proposition with the well known property that rule support does not increase after each rule specialization. For example, say that $minSupp = 100$ and $minConf = 0.75$. Let us assume that after some iterations Noah found the rule r "if … then $[c_1 = 30, c_2 = 80]$." The confidence $\mathcal{C}$ of r is 0.59 and its support is 110. Thus, r covers only 10 tuples more than the specified $minSupp$ threshold. From Proposition 1 we know that the larger the difference between the two class probabilities, the higher is the rule confidence $\mathcal{C}$. Therefore, in the most optimistic case, r is specialized in such a way that 10 tuples with class attribute c_1 are not covered anymore. This maximizes the difference between the two class probabilities and, consequently, the confidence of r. Thus, we get the distribution $[c_1 = 20, c_2 = 80]$. The confidence based on this distribution is 0.72. We refer to such confidence value as the *Confidence Upper Bound* (CUB) of r. Now, since CUB is smaller then $minConf$, we drop r at this point as none of its possible specializations will have confidence larger than $minConf$ and support larger than $minSupp$ at the same time.

The algorithm of this pruning strategy for a two-class case is shown in Fig. 5. An important aspect of this pruning technique is that it does not introduce any loss of information, i.e., it pruned ahead rules which will be anyhow pruned later.

4.3 Pruning Rules by Term Dependency

This pruning strategy is based on the concept of *informational relevance* discussed by Pearl [14] and, in part, proposed by Bayardo [2] for *Apriori*. Recall that a proposition Z is said to be *conditionally independent* of Y given X if:

$$P(Z \mid X) = P(Z \mid X, Y) \tag{3}$$

The *conditional probability* of Z given X does not change after we become aware of the new proposition Y. In this case, we say that: given X, Y *is irrelevant* to Z.

The level-wise approach of Noah offers a natural framework for computing the probabilities of (3). Suppose that after the first iteration Noah finds the rule "if X then Z" where Z is the class distribution under the condition X. On the second iteration, Noah refines X by *and*-ing it with Y. If the class distribution of this new rule is still Z, then Y is irrelevant to Z given X. Thus, the rule "if X and Y then Z" provides the same information as "if X then Z." Therefore, Noah drops the rule "if X and Y then Z."

Example 1 Consider a medical database containing the variables Sex and Pregnant. Consider the rule: "if Pregnant=Yes then Z." Refining the pattern of this rule to include the term "Sex=Female" is a waste of time since someone who is pregnant is always a female. Thus, "Sex=Female" is irrelevant for "Z", once we know the fact that "Pregnant=Yes." □

Notice that "Pregnant=No" does not give us information on the sex of the individual, i.e., both men and women can be non-pregnant. Thus, there is no *functional dependency* between the two variables. In fact, the dependency applies only for certain configuration of these variables. Therefore, the case discussed in the previous example will not be captured by the well known notion of functional dependency—which has been extensively studied in databases.

Noah uses such dependencies among certain configurations of the input variables to cut the search space. We do not need to compare the entire distribution before and after a refinement, in fact, we only need to measure variations of the rule support for the given refinement. Noah relaxes (3) with an *almost equal* concept by means of the user-defined *minDepRatio* parameter. For example, say $minDepRatio = 0.9$ and consider a pattern ρ and a term y. If

$$\frac{supp(\rho \cup \{y\})}{supp(\rho)} > minDepRatio \tag{4}$$

then Noah discards the rule "if ρ and y then ...". In other words, if at least 90% of the tuples covered by ρ are also covered by $\rho \cup \{y\}$, then y is considered "irrelevant."

Once an irrelevance between a pattern ρ and a term y has been discovered, we also check whether ρ and y are *equivalent* as follows. If:

$$\frac{supp(\rho \cup \{y\})}{supp(y)} > minDepRatio \tag{5}$$

then also ρ is irrelevant given y. Thus ρ and y are equivalent. In other words, ρ is a necessary and sufficient condition for y. By summarizing, given a pattern ρ and a term y:

- if (4) holds and (5) does not hold, then y is irrelevant given ρ;
- if both (4) and (5) hold, then y is equivalent to ρ;

Table 1. Number of rules pruned by min-supp (ms) and term dependency (ti) per iteration on datasets from UCI repository (*minSupp* = 1%, *dependency-ratio=90%*)

Iter.N.	Vote		Heart		Ann		Hypo		Soybean		Adult		House		Tae		Process		Insurance	
↓	ms	td	ms	td	ms	td	ms	td	ms	td	ms	td	ms	td	ms	td	ms	td	ms	td
#1	0	0	134	0	932	0	772	0	2	0	22018	0	2408	0	0	0	110	0	0	0
#2	98	32	1374	147	9214	2509	8200	3800	478	2607	3845	433	117	381	0	67	432	2065	0	2062
#3	694	304	314	430	18	1	130	123	2766	1699	465	52	1	20	0	13	369	338	0	2201
#4	512	388	66	139			3	2	2161	3052	31	1					100	318	0	148
#5	446	461	3	17					541	1778	7	0					10	59	0	4
#6	116	154	0	2					65	299									0	1
#7	17	9							0	13										
Tot.→	1883	1348	1891	735	10164	2510	9105	3925	6013	9448	26366	486	2526	401	0	80	1021	2780	0	4416

In case y is equivalent to ρ Noah discards all rules discovered so far that contain y in their antecedent. This is because for each such a rule there will be another rule containing ρ, instead of y, that provide the same information.

Experimental results based on large application datasets showed that such pruning is very effective. This is because often databases need to be *de-normalized* prior to be processed by rule induction algorithms and such de-normalization creates many *dependences* among the terms in the databases.

We also tested this pruning strategy for ten datasets from the UCI repository using a minimum support of 1% and a dependency ratio of 90%. The number of rules pruned for each dataset by both the minimum support and the term-dependency pruning are summarized in Table 1. For each dataset (columns "Vote," "Heart," "Ann," etc. in Table 1) we report the number of rules pruned, at each iteration, by minimum support (columns "ms") and by term dependency (columns "td"). The number of rules pruned by the minimum support are reported just for comparison. For instance, for the "Vote" dataset, on the third iteration (row "#3"), 694 rules are pruned by the minimum support versus 304 rules pruned by term dependency. The last row shows the total number of rules pruned by each method.

Conversely to the "Confidence Upper Bound" earlier discussed, the term dependency pruning produces some loss of information. In fact, we drop a rule when its specialized version yields to a class distribution which is "close enough" to the more general version of the rule. The *minDepRatio* is set by the user according to the current application. This pruning can be easily *switched off* by setting *minDepRatio* to 1.

5 Application of Data Mining for Cross-Selling

In this section, we discuss the use of Noah for a marketing *cross-selling* application. In a cross selling activity, a company tries to sell additional products or services to its current customers. A fundamental task for a successful cross-selling activity is the accurate identification of the best prospects for a given product. Since the company acts among current customers, it has already a considerable amount of individual level data that may support such a task.

The Database. The database provided to us has a total of 41,400 records (number of customers) and 221 attributes. The total number of possible values of all the attributes combined together is 53,137. Furthermore, a large amount of missing and/or noisy values are presented in the dataset. We split the dataset into a training set of 27,703 tuples and a hold-out sample of 13,697 tuples. Almost all of the attributes are discrete; some of them have a large domain (e.g., "zip" and "city").

5.1 Data Mining with Noah for the Cross-Selling Problem

The company we are studying offers a portfolio of five possible services to its customers. We classified each customer either as "Single" (if he/she purchased only one service) or "Multiple" (if he/she purchased more than one service). Our goal is the accurate prediction of the correct class—"Multiple" or "Single"—of each customer. From a managerial perspective, we need to pinpoint those customers who are more inclined to have multiple services as they constitute the target of the cross-selling. Differently from common prediction problems, not only do we need to predict the class for each customer, we also need to associate a confidence measure to each prediction. Such confidence measures allow us to rank prospects in a *gain chart* form (discussed below) in order to target them differently. For example, we may decide to call over the phone (an expensive marketing process) prospects classified as "Multiple" with high confidence as opposed to mail a standard flyer (a cheaper marketing process) to those predicted as "Multiple" with low confidence. This allows the company to properly allocate available resources to maximize its return from the marketing campaign.

To address this problem, we first ran Noah on the input training set. It found a total of 4,135 rules—the training dataset was of 27,703 tuples, we set $minSupp = 0.5\%$ and $minConf = 0.75$. We then *classified* each customer in the hold-out sample by using the discovered rules. Notice that rules discovered by Noah are not mutually exclusive, therefore, multiple rules can cover a specific tuple during classification. We select the rule with the highest confidence in this case. We then labelled each customer in the hold-out with the best class suggested by the rule and the confidence of the rule.

Out of the entire hold-out sample, rules discovered by Noah covered a total of 6,224 customers (the rest are predicted by the most populated class, the *default* rule, computed over the original distribution). We arranged Noah predictions on a gain chart as follows: we created 10 (almost) equal-size clusters of hold-out customers (i.e., quantiles of about 10%). Each cluster contains customers whose prediction confidence falls within a specified range. In other words, we fixed the confidence range for each cluster in the gain chart to accommodate about 10% of the covered customers within that cluster. We then sorted those clusters according to the confidence range in a descending order. Then, for each cluster we computed the accuracy of our prediction for all customers belonging to that cluster. The resulting gain chart is shown in Table 2. For instance, for the cluster number 1, we set a confidence range of 0.850–1.000, the total number of customers whose associated prediction confidence falls in that range is

Table 2. The gain chart created using Noah

Cl. Rank	Conf. range	Cl. Size	Hits	Accuracy	%Pop	%Covered
1	0.850–1.000	646	581	89.94%	4.72%	10.38%
2	0.830–0.850	657	538	81.89%	4.80%	10.56%
3	0.818–0.830	679	572	84.24%	4.96%	10.91%
4	0.805–0.818	647	531	82.07%	4.72%	10.40%
5	0.795–0.805	642	509	79.28%	4.69%	10.31%
6	0.788–0.795	576	454	78.82%	4.21%	9.25%
7	0.777–0.788	601	460	76.54%	4.39%	9.66%
8	0.766–0.777	656	499	76.07%	4.79%	10.54%
9	0.758–0.766	592	431	72.80%	4.32%	9.51%
10	0.750–0.758	528	386	73.11%	3.85%	8.48%

646 (column "Cl. Size"), of which 581 (column "Hits") are correctly predicted. This means an accuracy of 89.94% (column "Accuracy"). The column "%Pop" contains the proportion of the number of customers in the cluster over the total size of the hold-out sample ($\frac{\text{Cl. Size}}{13697}$). Whereas "%Cover" is computed over the total number of covered customers ($\frac{\text{Cl. Size}}{6224}$). As expected, the accuracy of each cluster worsens as the confidence range goes down. Notice the anomaly of the $2nd$ cluster whose accuracy drops to 81.89%, which is worse than the $3rd$ and $4th$ cluster.

The overall results were deemed very satisfactory by marketing domain experts. As already mentioned, the actual utilization of such findings consists of a set of different marketing strategies that are applied to the different clusters depending on the cluster accuracy. Since we are working with current customers, each of them can be precisely pinpointed (i.e., his/her phone number, address, etc.).

5.2 Experimental Results using Other Mining Algorithms

We performed our experiments on a Linux OS with a 266 MHz Pentiun PC with 128 Mbytes of physical memory and 128 Mbytes of virtual memory. Our experiments revealed that our application dataset is intractable for CN2[4], Ripper[5], Apriori[6], and CBA[7] as all of them failed in their execution for lack of memory.

On the other hand, C4.5 was able to complete its task but the discovered decision tree was very poor. We first tried C4.5 with the original dataset. C4.5 discovered a decision tree with 54, 575 nodes which, after pruning, was reduced to a single-node tree—that corresponds to always predict the most populated class. After *massaging* the input training set, e.g., removing some many-valued variables and/or converting others to different types, C4.5 was only able to produce the two-level tree shown in Figure 6. Basically, C4.5 identified only the variable "FirstService" which stores the first service (out of five available)

[4] Downloaded from `http://www.cs.utexas.edu/users/pclark/software.html`.

[5] To be requested at `http://www.research.att.com/~wcohen/ripperd.html`

[6] Downloaded from `http://fuzzy.cs.uni-magdeburg.de/~borgelt`.

[7] Downloaded from `http://www.comp.nus.edu.sg/~dm2`.

Table 3. The results of applying C4.5 to the application dataset

Results from training set (27703 items)				Results from validation set (13697 items)			
Before Pruning		After Pruning		Before Pruning		After Pruning	
Tree Size	Error	Tree Size	Error	Tree Size	Error	Tree Size	Error
58144	3183 (11.5%)	6	8747 (31.6%)	58144	5444 (39.7%)	6	(4321) 31.5%

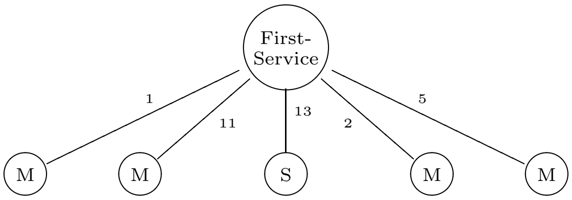


Fig. 6: The tree generated from C4.5 from the application database

each customer subscribed to. The classification accuracy results of applying C4.5 are shown in Table 3. For the training set, the error before pruning is 11.5% over 58,144 tree nodes; the error, after pruning, increased to 31.6% over only six nodes. This is a symptom of the complexity of the input dataset whose knowledge was initially captured by an over-fitted tree with a large number of poorly populated leaves. This tree was then pruned down to a much smaller tree with a much larger error. The over-fitting is also witnessed by the opposite behavior on the validation set: the unpruned tree produces a larger error than the pruned tree.

A gain chart (Table 4) can be drawn based on the output of C4.5. It can be obtained by attaching to each predicted class the accuracy of the corresponding leaf in the decision tree. Comparing Tables 2 and 4, we note that the output from Noah is superior in many aspects to the one from C4.5. For instance, with C4.5, we cannot partition customers in more than five clusters since that is the total number of leaf nodes in the generated tree. Furthermore, we cannot create the quantile representation of the gain chart, as we did with Noah. This reduces the flexibility for cross-selling planning. Accuracy is in general higher for Noah. Noah accuracy over the best 4,448 customers[8] is 82.24% (this is obtained by averaging the accuracy of clusters 1 to 7). Whereas, the accuracy for the best 4,195 customers in C4.5 (i.e., the ones belonging to cluster 1) is 76.4%.

Comparison over the entire hold-out sample. In Table 2 we only reported the customers who are covered by some *good* rule discovered by Noah (i.e., rules with confidence greater than the $minConf$ threshold set to 0.75). They are a total of 6,224 customers from the hold-out sample. The rest of them, i.e., $13697 - 6224 = 7473$, are predicted by low confidence rules and/or the *default* rule. In C4.5 there is no concept of default rules as all the cases are classified by some leaf. Therefore, for the sake of completeness, we now discuss the performance of C4.5 and Noah

[8] By "best customer" here we mean a customer whose class we can accurately predict.

Table 4. The gain chart created using C4.5

Cl. Rank	Leaf Conf.	Cl. Size	Hits	Accuracy	%Pop	%Covered
1	0.76	4195	3205	76.40%	30.63%	30.63%
2	0.74	1321	972	73.58%	9.64%	9.64%
3	0.67	3069	2055	66.96%	22.41%	22.41%
4	0.63	2656	1681	63.29%	19.39%	19.39%
5	0.61	2456	1463	59.57%	17.93%	17.93%

over the entire hold-out sample (i.e., $13,697$ records). Noah performs slightly better than C4.5 over the entire hold-out sample. In fact, C4.5 reports an error of $4,321$ cases (see Table 3) which yields an accuracy of $\frac{13697-4321}{13697} = 68.4\%$ whereas Noah misclassifies a total of $4,040$ cases which yields an accuracy of $\frac{13697-4040}{13697} = 70.5\%$. It is important to note that the accuracy lower bound (or *baseline*) of the dataset is 63.9%, that is, if we always guess "Multiple" (i.e., the most populated class in the training set) we correctly predict 63.9% of the cases in the hold-out sample (any useful classifier has to beat this value). Thus, the percentage improvement over the baseline is $\frac{68.4-63.9}{63.9} = 7\%$ for C4.5 versus the $\frac{70.5-63.9}{63.9} = 10.3\%$ of Noah.

For this specific application, an important contribution of Noah is the possibility of creating the gain chart shown in Table 2 which properly supports the cross-selling activity. Basically, with Noah we are able to locate those portions of the dataset where an accurate prediction is possible, that is, where some robust knowledge can be elicited. This matches with the cross-selling problem where we do not need to target the entire population of customers. With Noah we are able to predict (clusters of) customers with accuracy up to almost 90% (see Table 2). C4.5 is far from that level and its outcome was considered of no interest by domain experts.

In terms of learning time, C4.5 performs much faster than Noah. C4.5 took about 8 minutes to complete its task[9] whereas Noah completed in about 4 hours. The main reasons are: Noah performs a multivariate search for rules, it never partitions the dataset and always keeps it as an external file.

6 Related Work

Both KDS [8, 6] and CBA [11] are very closely related to our work as they mine classification rules in a *Apriori* [1] style. The focus of KDS is for the SQL implementation of the learning procedure. CBA provides a sophisticated classification procedure. An extensive comparison of accuracy performance with C4.5 on a set of UCI datasets is discussed in [11]. Both KDS and CBA do not treat the problem of mining datasets with large number of attributes. Basically, Noah is an extension of KDS to handle large number of many-valued attributes.

Bayardo *et al.* [3] discuss an extension of *Apriori*, called "Dense-Miner," to discover classification rules. They introduce the concept of "minimum improve-

[9] We used all standard parameters when running C4.5.

ment" which, basically, discards rules that provide only marginal confidence improvement (or just deterioration) over a more general version of the rule. They also propose a pruning by "confidence upper bound," similar to the one we described. However, they use an "association form" for the classification rules: $\rho \Rightarrow y$, where ρ is a pattern and y is a class term. Their definition of rule confidence is $\mathcal{C} = \frac{supp(\rho \cup \{y\})}{supp(\rho)}$ which is different from the one used in Noah, therefore the proposed solutions are different.

Our "term dependency" pruning resembles closely the "(near) equivalence" strategy proposed by Bayardo [2]. However, in Noah we extended it with the concept of "equivalence" between terms.

Lent *et al.* [10] also propose a classifier based on *Apriori*-like association rules. They propose an algorithm to cluster *similar* association rules into a more descriptive rule. They however limit their approach to only two terms in the rule antecedent.

SLIQ [13], SPRINT [16], and MIND [17] induce classification trees from databases residing on disk. They are based on the univariate and divide-and-conquer principles of C4.5 discussed earlier. Thus, they will exhibit similar drawbacks as of C4.5.

7 Conclusions

In this paper we presented an algorithm to mine classification rules from datasets with a large number of many-valued attributes. In such datasets knowledge may be scattered among many uncorrelated rectangular portions of the input dataset. The univariate and divide-and-conquer approach of C4.5 may fail in discovering knowledge under such circumstances. We presented an algorithm, Noah, that discovers classification rules by following a multivariate approach. The combinatorial explosion related to the multivariate search is controlled by optimized data structures and efficient pruning strategies. These strategies are quite general and may be exploited by other learning algorithms as well. We have successfully applied Noah to a real application of cross-selling marketing based on a dataset of 41,400 records and 221 attributes. C4.5 failed in discovering useful knowledge from this dataset. Applying the same dataset on other rule discovery algorithms, such as CN2, Ripper, Apriori, and CBA, failed due to a lack of memory. Our future work agenda includes an improved classification procedure and the application of Noah to other large dataset domains. Noah does not require the dataset to reside in main memory during learning. This provides a basics for a tighter integration of Noah with DBMS which is currently under study.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press, 1996.

2. R.J. Bayardo. Brute-force mining of high-confidence classification rules. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*. AAAI Press, 1997.
3. R.J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proc. of the 15th Int'l Conf. on Data Engineering*, pages 188–197, 1999.
4. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
5. W.W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI-96*. AAAI press/ The MIT press, August 1996.
6. L. G. Cooper and G. Giuffrida. Turning datamining into a management science tool: New algorithms and empirical results. *Management Science*, 2000 (To appear).
7. P. Domingos. Linear-time rule induction. In E. Simoudis, J. W. Han, and U. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 96. AAAI Press, 1996.
8. G. Giuffrida, L. G. Cooper, and W. W. Chu. A scalable bottom-up data mining algorithm for relational databases. In *10th International Conference on Scientific and Statistical Database Management (SSDBM '98)*, Capri, Italy, July 1998. IEEE Publisher.
9. R.C. Holte, L.E. Acker, and B.W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, (MI), 1989. Morgan Kaufmann.
10. B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE '97)*, Birmingham, UK, 1997.
11. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In R. Agrawal, P. Storloz, and G. Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, page 80. AAAI Press, 1998.
12. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1, November 1997.
13. M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. *Lecture Notes in Computer Science*, 1057, 1996.
14. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.
15. J. R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.
16. J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB 1996*, Mumbai (Bombay), India, September 1996. Morgan Kaufmann.
17. M. Wang, B. Iyer, and J. S. Vitter. Scalable mining for classification rules in relational databases. In *Proceedings of International Database Engineering and Application Symposium (IDEAS'98)*, Cardiff, Wales, U.K., July 1998.