

Decision Rule Extraction from Binary Data Using Winning Combinations

Yuval Cohen, The Open University of Israel, Department of Management and Economics, Raanana, Israel.
yuvalco@openu.ac.il

Arik Sadeh, HIT Holon Institute of Technology, Department of Management of Technology, Holon, Israel.
sadeh@hit.ac.il

Arie Ben-David, HIT Holon Institute of Technology, Department of Management of Technology, Holon, Israel. abendav@hit.ac.il

ABSTRACT

This paper presents a new approach for making inferences about binary data. The objective is to determine rules that lead to a certain result. The method consists of four phases: in the first phase, the data is processed into a binary format of a Truth Table; in the second phase, rules are defined by utilizing an algorithm that minimizes Boolean functions (by Quine and McCluskey); in the third phase, the rules are checked and filtered; and in the fourth phase, simple rules that involve one to two features are presented.

Keywords: Decision Support Systems, Decision Analysis, Rule Extraction, Knowledge Acquisition, Data Mining, Rules Extraction from Binary Data

1. Introduction

The importance of rule extraction in management science can be illustrated through its application areas such as credit risk evaluation, granting loans, classifying clients in insurance fraud detection, and finding winning combinations from past history. For example, Baesens et al. (2003) and Steenakers (1989) mention that credit-risk evaluation is a very important management science problem. Though great advances have been achieved in this area, the search for techniques for finding such rules is as fervent as ever. The need to extract knowledge from data has spawned increasing efforts in trying to infer rules from databases (Andrews et al. 1995). For example, over the last two decades, the fields of neural networks and data mining have grown considerably.

The current paper attempts to take a step forward in this direction. It offers a new approach in dealing with dichotomous data (fields with one of two values). Specifically, it characterizes combinations of features that lead to one of two results, which we call either a 'success' or a 'failure'.

The current paper presents a technique for rule extraction with three major constraints that differentiates it from general data mining techniques:

1. All the data fields are dichotomous (0/1 values)
2. The population of records is classified into two groups: "Winning" =1, and "Losing"=0
3. The rules always associate records with the group defined as '1' (the "Winning" group)

The presented approach finds the rules characterizing the desired combinations and expresses these rules in the most efficient way. In the business world, various phenomena could be classified as either a success or a failure. For example, a success could be when a customer in a supermarket purchases a bottle of wine, when an entrepreneur receives a loan, or when corporate sales grow over 50%. In other cases, phenomena could be classified into two values with no clear winner, for example, a population classified into patients under 21 years of age and patients over 21 years of age. The Winning group in this latter case is chosen either based on our interest in the group, or arbitrarily.

The question of interest is: "What are the rules that lead to a success or a failure?" In other words, what variables are associated with success?

For example, if the purpose is to characterize customers that purchase wine, the question is: what variables are associated with these customers? The variables could be the customer's gender, the customer's age, other products that were purchased, the time of day, the total bill amount in dollars, etc. It is, however, important to note that while gender or a product purchase is a 0/1 variable by nature, other variables are not. Thus, we have to segment the other variables into 0/1 groups by "inventing" a threshold. For example, customers under 30 years of age and over 30 years of age, or bills totaling under \$100 and over \$100.

Let us look at another case where a patient clinic has to allocate time slots for patients who schedule appointments with the doctors. The appointments could be either of 15 or 30 minutes' duration. The clinic management is interested in categorizing the patients into two groups; asking the patients several yes/no questions should help to determine whether to allocate 15 minutes or 30 minutes for each appointment. Dichotomous variables could be whether the patient is a smoker or a non-smoker, whether body temperature is at least 2 degrees above normal, whether the patient is over 60 or not, etc. It is important to note that in such cases, the combination of values is of crucial importance. For example, the combination of a smoker who is over 60 years of age could trigger classification into a 30-minute appointment with the doctor, whereas a young smoker and an older non-smoker could each be allocated a 15-minute appointment.

Since in the current paper we constrained ourselves to binary variables, the rules could be expressed as combinations (or strings) of ones and zeros of the corresponding variables. For example, if we are trying to characterize beer consumers, using their smoking habits, age, and gender, it is reasonable to define the following variables:

X= smoker/non-smoker (smoker = 1, non-smoker = 0).

Y =gender (male=1, female=0),

Z=age over or under 30 (under=1, over=0).

Thus, if we found that all male smokers under 30 in the sample are beer consumers, it could be expressed as (X=1, Y=1, Z=1), **111** or simply **XYZ**. When a rule contains a zero, we use the NOT=' sign (e.g., X NOT=X'). For example, if we try to characterize healthy people, we may find that all of the non-smokers under 30 are healthy (X=0 and Z=1), expressed as X'Z. Hence, X'Z represents the rule that has been found.

When there are several "success" rules, the rules are expressed using the "+" sign, serving as the Boolean "or". For example, if it is found that all organic food buyers are either women or non-smokers, using the classification above, the organic food buyers are characterized as either X=0, Y=0, or simply X' + Y'. This expression is the rule we were looking for.

However, the process by which the desired rule is discovered is really the important part. This is described through a four-phase mechanism that identifies rules from binary data.

The rest of the paper is arranged as follows: Section 2 discusses some related literature; Section 3 describes the phases in subsections 3.1 to 3.4.; Section 4 presents a case study, which is then used in subsections 4.1 to 4.4 to illustrate the four phases, respectively.

2. Related Literature

Forming a decision tree is a learning technique that generates a set of rules for integer and binary data, but this technique still requires an extensive set of training examples (Mitchell 1997, Russell and Norving 1995, and Winston 1992). The problem tackled in this paper has also been tackled in Sorensen and Janssens (2003) via binary trees. However, rule extraction, according to binary trees, does not guarantee that all rules are good and, like other trees, the rules depend on the order in which the tree is developed. The ID3 and C4.5 algorithms used / proposed by Quinlan (Quinlan 1986 and Quinlan 1993) serve as good examples of learning algorithms that are suitable for building rules. However, Quinlan's algorithm requires large amounts of data for the learning process and cannot cope with erroneous or missing data. A very critical view of the above methods appears in Quine (1952).

Some implementations of classifying and characterizing desired combinations of attributes is shown in the literature. Credit-risk evaluation for granting loans based on the client characteristics is discussed in Baesens et al. (2003), Steenackers and Goovaerts (1989), and Capon (1982). Viaene, Derring, Baesens and Dedene (2002) deal with classifying customers for insurance fraud detection.

The Logit model (Allison 1991, Allison 1995, Greene 2000, Long and Freese 2001, and Long 1997) is the most common and well-known regression-based approach for discrete and binary data. While it cannot

deal with the effect of combinations, it does find the effect of each single independent variable (the main effects). Logit differs from regular regression by handling data in which the dependent variable is binary or even discrete ordinal; the independent variables can be either continuous or categorical. The main idea is to find a relationship between predetermined values of independent variables and the probability that the dependent variable is a success (or that it is a failure). The Logit model utilizes a regression procedure and maximum likelihood principle to estimate the main effect of each independent variable. However, the Logit model cannot deal with the effect of combinations of variables; thus, it is not suitable for the case studied in this paper.

In the current paper and in some of the previous methods, forming Winning rules is based on the "general principle of inductive learning often called *Ockham's razor: The most likely hypothesis is the simplest that is consistent with all observations.*" (Russel and Norvig 1995 p. 534). An Ockham algorithm is "an algorithm that is capable of finding a consistent hypothesis that achieves a significant compression of the data it represents" (Russel and Norvig 1995, p. 560). This paper utilizes an Ockham algorithm used / proposed by Quine (1955) and McCluskey and Schorr (1962). The algorithm is discussed and explained thoroughly in Chapter 4 of Kohavi (1978). This algorithm is a generalization of Karnaugh maps devised by Karnaugh (1953) for small problems. It is not surprising that Karnaugh's map method is also an Ockham algorithm.

3. Proposed Solution Methodology

The methodology of the proposed technique is divided into four phases:

1. Obtain the data and construct a Truth Table
2. Form Winning Rules using algorithms used / proposed by Quine (1955) and McCluskey and Schorr (1962)
3. Filter outliers and non-efficient elements
4. Check for main effects and effects of all pairs of combinations

Phase 1 involves obtaining the data and constructing the Truth Table. Obtaining the data may involve sampling. The Truth Table tells us what combinations have proven to be successful and what combinations have been failures.

Phase 2 forms rules by describing the Truth Table as compactly as possible. This is achieved by using algorithms proposed by Quine (1955) and McCluskey and Schorr (1962), which minimize logical functions in general.

Phase 3 implements a filter based on the Pareto Principle to eliminate inefficient rules from Phase 2.

Phase 4 checks for rules that cover multiple instances and which may have been rejected in Phase 2 due to either exceptions or a lack of data.

Each of the phases will be discussed in further detail below.

3.1. Phase 1: Construction of a Truth Table

The purpose of Phase 1 is to classify the observed data records into one of two groups (Winning or Losing). If the data comes from a mechanical or digital system where the same inputs always result in the same outputs, the Phase 1 is fairly simple:

Phase 1 for a fully determined system

Stage 1: Obtain the data

Stage 2: Process the data into dichotomous values (using a threshold when necessary)

Stage 3: Construct a Truth Table

Missing combinations in the data are spots of uncertainty and in typical conservative treatment should be considered as losing combinations (just to be on the safe side). The following is an illustration of a very simple Truth Table:

Table 1. Truth Table for the rule (if X=1 and Y=1, then Z=1) or, in short, XY

Input		Output
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

If we consider purchase of a product, human reaction, and most business phenomena - the same input could result in different outputs. When such a system is involved, some preparatory stage must precede the construction of the Truth Table, and the procedure is altered as follows:

Phase 1 for a system with an uncertain response

- Step 1: Obtain the data
- Step 2: Process the data into dichotomous values (using a threshold when necessary)
- Step 3: Construct a frequency table with the following fields:
 - Binary combination, # of successes, # of failures,
 - and % of successes: # of successes / (# of successes + # of failures)
- Step 4: Classify Winning binary combinations of data using Threshold Values of "% of successes" and "# of successes".
- Step 5: Construct a Truth Table

3.2. Phase 2: Forming Winning Rules - Using algorithms proposed by Quine and McCluskey

The Truth Table contains every combination that was classified as a success. For k dichotomous attributes, the Truth Table has 2^k entries. This is a very long and inefficient way of describing the rules of success or failure. Moreover, it grows exponentially. We need better rules that will capture the common features of large groups of combinations. Like other methods, the proposed method is based on the Ockham's razor philosophy: ***The most likely hypothesis is the simplest that is consistent with all observations.***" (see Section 2: Related Literature). Thus, an Ockham-type algorithm, such as those proposed by Quine and McCluskey has been adopted by the authors. This algorithm comes from the realm of digital design and, when applied to the Truth Table, generates a minimal Boolean function for that table. A minimal Boolean function of a Truth Table is a function that describes the table using a minimal number of terms. The Quine-McCluskey algorithm is an Ockham algorithm, since a minimal Boolean function is, by definition, the simplest function. Minimizing Boolean functions include the pursuit of powerful rules with the fewest possible variables. This could be explained in a more intuitive way as follows:

Suppose, for example, that we seek combinations of characteristics that are associated with lung patients who have developed cancer. Furthermore, suppose that all these combinations contain cigarette smokers and that there are only very few smokers who have not developed cancer. Classifying smoking as the rule for cancer is the most efficient rule in this case. Smoking is one variable. Rules based on a single variable are the most efficient way to describe large groups. In general, rules with pairs of variables are less efficient than rules with one variable, since they cover fewer success combinations. It follows that rules with three variables cover less combinations than rules with two variables, and so on. In general, the rule is stronger and more efficient when it contains fewer variables.

In the early days of digital design, logical variables were constructed using physical gates and vacuum bulbs. Digital designers were struggling to minimize the cost of representing a Truth Table. The only way to do this was to minimize logical functions. While small functions with up to five variables were minimized using the prevalent Karnaugh maps (Karnaugh 1953), the algorithm for the general case is much less known and combines two algorithms: the first by Quine and the second by McCluskey into one framework. The framework and the details of the algorithm along with examples and discussion are presented in Kohavi (1978).

3.3. Phase 3: Filtering outliers and non-efficient elements

The result of Phase 2 is a set of rules that exactly matches the original Truth Table. While this is incredibly more compact than a Truth Table, it may still have quite a few specific rules for isolated cases. Like most real world systems, the Pareto Rule holds true in this case too. Namely, 20% of the rules describe 80% of the success cases. Moreover, the desirability of keeping a rule that describe less than 1% of the cases is usually low enough to ignore this rule altogether. This brings us to Phase 3, where non-desirable rules are filtered out.

In Phase 3, we must determine a level of rule acceptance. Say we chose the rule acceptance level to be 5%; then, any rule that does not describe at least 5% of the cases is filtered out.

While Phase 3 filters out inefficient rules, it may happen that a very efficient rule with only one or two variables has not been revealed so far. For this to happen, it is enough that one combination (or more) belonging to the rule is missing, or was not recorded by mistake. These cases are treated in Phase 4.

3.4. Phase 4: Checking for main effects and effects of all pairs of combinations

Main effects are rules of a single variable (that is, a single attribute). In this paper's context, main effects are powerful rules meaning that the variable value alone already determines the result. In this category of powerful rules we also include the effects of a pair of attributes (meaning that the combination of the pair of attribute values alone, already determines the result.)

Many efficient rules could be missing due to missing data and exceptions. Phase 4 checks for such cases and amends them. This phase is relatively simple. Checking for main effects only requires that we collect data for each variable: (1) the % of successes when the variable is one, and (2) the % when the variable is zero. Exceeding a predetermined percentage (e.g., 95% successes when the variable is 1) would inaugurate a new rule (with the predetermined 95% accuracy). We also check for each pair of variables and their success/failure percentages. It is important to note that the complexity of checking combinations grows immensely, and the power of the resultant rules drops significantly, together with the growing number of attributes. Therefore, we do not recommend going beyond pairs for this brute force enumeration.

3.4.1. Phase Four Complexity

In this section, we show how Phase 4's complexity grows very quickly together with the number of variables/attributes in a rule. Phase 4 is based on enumeration. If the number of dichotomous characteristics we use is k , then there are 2^k possible combinations (or rows in the Truth Table). Each one is either a success or a failure. From our sample, we may find values for only part of these and the missing values are conservatively considered as failures. We split the data records into two groups: success and failure. Then, for each group of records, and for each variable, we count both the zeros and ones of the specific variables for all relevant combinations. Thus, we process $k \cdot 2^k$ elements for single variables.

For pairs of variables, the complexity is greater: once again, we form the two groups for success and failure; for each pair of variables, we compute the success and failure percentages for each of the four combinations: 00,01,10,11. Finding a percentage that is over a threshold yields a rule. The number of pairs is: $k(k-1)$. Thus, we have to process 2^k possible combinations for each pair. Therefore, the complexity of the two stages is $(2^k)k(k-1)$ times, which is $O(k^2)(2^k)$.

This tells us that Phase 4 can only work for either a medium or small number of fields. For example, for $k=20$, the complexity is $O(20^2)(2^{20}) = 419,340,400$, which is still a plausible task for a PC. Although k rarely exceeds 30 in the problems we examined, there may be others where k is much greater. As k increases, the solution rapidly becomes intractable or a job for supercomputers.

4. A Case Study

Before planning a new facility for treating heart attack patients, an (undisclosed) HMO is interested in characterizing most of its patients so as to ensure the facility's suitability level for its patients. In other words, the HMO seeks out popular combinations that characterize heart attack patients. Success, in this case, means having higher than average percentages of heart attack patients. Since the HMO we examined insisted that the real data not be disclosed, we had to use contrived data in the current case study.

Phases 1, 2 and 4 are best illustrated on a small case study, whereas illustrating the advantages of Phase 3 requires a larger case. Since the chosen case study is small, it will be used to illustrate Phases 1,2, and 4. Phase 3 will be dealt with separately. The study consists of four binary characteristics ($k=4$) used for characterizing the population of heart attack patients. The steps leading to a rule stating the combinations that characterize heart attacks patients are presented below.

4.1. Phase 1 (for characterizing risk of heart attack)

Step 1: Obtain the data: Most HMOs around the globe possess the following patient data: patient's age, gender, height, weight, and whether or not s/he smokes. Most importantly, they also have the number of the patient's heart attacks. Thus, after collecting one thousand heart attack records from a local HMO, we are now ready for Step 2.

Step 2: Process the data into dichotomous values. Gender and smoker/non-smoker factors have only two possible values; thus, they are dichotomous by nature. The combination of height and weight could be categorized into obese or not obese (according to medical definitions). For age, a threshold is necessary to make it dichotomous. We shall use 50 years of age as our threshold. Finally, we will differentiate between zero heart attacks (0) and above zero (1).

We will use the following definitions:

$a=1$, if a candidate is obese (according to medical definitions), and zero otherwise.

b=1, if a candidate is male, and zero if a candidate is female
 c=1, if a candidate is elderly (say, above 50), and zero otherwise
 d=1, if a candidate smokes, and zero otherwise.

Step 3: Construct a frequency table with Binary combination, # of successes, # of failures, and % of successes=(# of successes / (# of successes + # of failures))

Since we have k=4 dichotomous attributes, we have 2⁴=16 possible combinations. For each combination, the number of heart-attack patients is counted and the percentage is calculated as follows:

Table 2: Success frequency table for all combinations

Binary Combination				# of successes	% of successes	Using a 7% Threshold	
Obesity 1=obese 0=not obese	Gender 1=male 0=female	Age 1=">50" 0="<=50"	Smoking 1=smoker 0=non-smoker				Number of Heart Attack Cases
0	0	0	0	2	0.2%	0	
0	0	0	1	1	0.1%	0	
0	0	1	0	6	0.6%	0	
0	0	1	1	16	1.6%	0	
0	1	0	0	5	0.5%	0	
0	1	0	1	28	2.8%	0	
0	1	1	0	49	4.9%	0	
0	1	1	1	46	4.6%	0	
1	0	0	0	10	1.0%	0	
1	0	0	1	36	3.6%	0	
1	0	1	0	182	18.2%	1	
1	0	1	1	121	12.1%	1	
1	1	0	0	44	4.4%	0	
1	1	0	1	104	10.4%	1	
1	1	1	0	148	14.8%	1	
1	1	1	1	202	20.2%	1	
Total	8	8	8	8	1000	100%	5

Step 4: Extract rules (classify the "winning" binary combinations) using threshold values of "% of successes" and "# of successes".

In a case of total randomness, each of the 16 combinations would average 1/16 of the observations or close to 6.25% of the cases. Therefore, the determined threshold should exceed this figure. In the present case study, we set the threshold at 7% (or 70 observations).

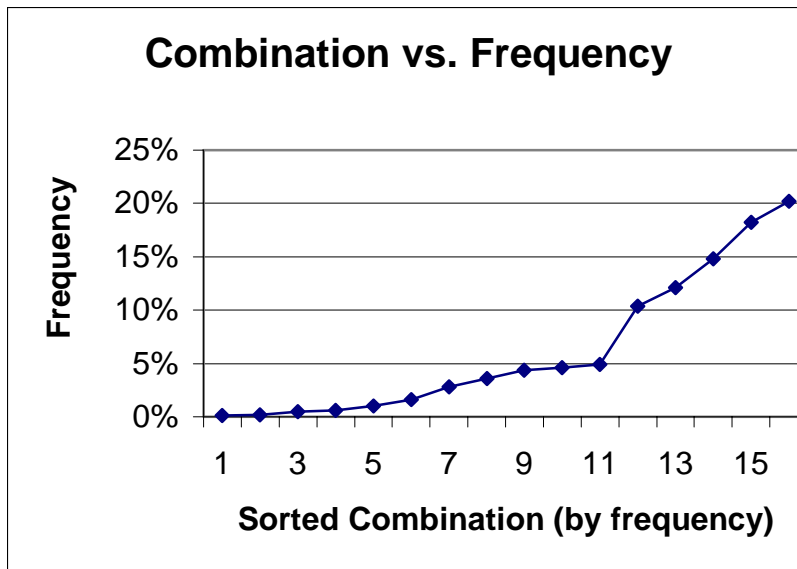
Setting the threshold is a somewhat arbitrary decision based on the analyst's discretion. However, in many cases, the Pareto Rule may work, where we set the threshold to differentiate between the higher 20% (or so) and the rest of the pack.

It is often useful to look at a chart that shows the combinations vs. their frequency before determining the threshold. The chart for the current case study is shown below. The combinations were categorized by their frequency before constructing the chart.

Stage 5: Construct a Truth Table based on Table 2 and the threshold

The last column of Table 2 is calculated by simply converting all values on the right side of Table 2 that are under the threshold to zero and those above the threshold to one (remember that the threshold is 7%).

Figure 1: Combinations sorted by frequency for the case study
 (Note that the last 4 points are responsible for the majority of the area below the curve.)



4.2. Phase 2: Extracting Rules - Using algorithms proposed by Quine and McCluskey

The aim of this phase is to form rules based on the Truth Table. We applied the method proposed by Quine and McCluskey, which is explained thoroughly in Kohavi (1978) to obtain the following rules: Heart attack patients are characterized by the following popular combinations:

- 1) **Obese, Elderly**
- 2) **Obese, Male, Smoker**

The mathematical notation for the rules is as follows:

The letters represent dichotomous characteristics/fields: A=Obesity, B=Gender, C=Age, D=Smoking, and Y= Success. Also, we use \cup for the Boolean "or" and \cap for the Boolean "and".

Writing the letter means that its value is one; adding the apostrophe (') to the letter means that its value is zero. For example, "A" represents A=1; adding 'the apostrophe (') as in "A'", represents A=0. Thus, the rule is: $Y = (A \cap C) \cup (A \cap B \cap D)$.

It is customary to replace \cup by the "+" sign and treat \cap as a multiplication resulting in:

$$Y = (A \cap C) \cup (A \cap B \cap D) = AC + ABD \tag{1}$$

Since this is a relatively small problem (with $k < 6$), the same results of the Quine and McCluskey algorithm could be achieved and verified using a Karnaugh map - see Figure 2 below and the following explanation.

Figure 2: Karnaugh map for the case study. The circles correspond to the rules.

		A'B'	A'B	AB	AB'
		00	01	11	10
C'D'	00	0	0	0	0
C'D	01	0	0	1	0
CD	11	0	0	1	1
CD'	10	0	0	1	1

Diagram description: In the Karnaugh map above, a circle labeled 'ABD' encloses the 1s in the AB' column (rows 01 and 11). An oval labeled 'AC' encloses the 1s in the AB and AB' columns (rows 11 and 10).

$$Y = AC + ABD = (1) \text{ Obese, Old. Or } (2) \text{ Obese, Male, Smoking}$$

A Karnaugh map is a graphical tool for minimizing Boolean functions. A Karnaugh map is composed of a matrix, in which each entry corresponds to a single combination. The value of each entry is binary (1 or 0, for success or failure, respectively).

Forming the matrix starts with splitting the binary attributes of the problem into two separate groups: a variable group for the columns and the rest of the attributes for the rows. For example, in Figure 2, four binary attributes: A, B, C, D are divided into A, B for the columns, and C, D for the rows. Thus, columns represent combination values of A and B, while rows represent combination values of C, D. Each column corresponds to a single combination of values for A and B (one of: 00,01,10,11). Each row corresponds to a single combination of values for C and D (one of: 00,01,10,11). The value of each entry in the Karnaugh map (in Figure 2) is the result (success/failure) of the value combination formed by both the column and the row. For example, the upper left corner has the first column corresponding to AB=00, while the first row corresponds to CD=00. Hence, the upper left corner entry corresponds to A'B'C'D'=0000. Its value is zero corresponding to a failure. As another example, the lower right corner corresponds to AB'CD'=1010; its value "1" represents a success.

The column and row state combinations are ordered so that between any pair of neighboring columns (or rows), there would be only one change of one field (bit). For example, between (0,0) and (0,1) there is one bit change. An example of an illegal adjacency is (0,1) and (1,0), since there are two bit changes between the two. Thus, Karnaugh maps can handle only up to 5 attributes (for more on Karnaugh maps, see Kohavi 1978). Note that Quine and McCluskey's method minimizes the binary function in general for any number of attributes.

In Figure 2, the rows represent all of the value combinations of attributes A and B, while the columns represent the combinations of attributes C and D. Circling the largest groups of "1", which cover all of the "1"s and nothing but the "1"s, provide the desired rules. For example, circling the four "1"s at the bottom-right corner, corresponds to the common values of all entries in this group: in this case, A=1, C=1 (short for A∩C). It is important to note that the circled groups (rules) can only have 2, 4 or 8 entries, and that fully contained circles are dominated by the bigger group, which is also the more general rule.

4.3. Phase 3: Filtering out rules

The filtering process is described using the following procedure:

I. Make a list of the rules, compute their corresponding percentage of the original success cases:

$$A. \text{ Original successes: } \quad P(AC)=18.2+12.1+14.8+20.2=65.3\%, \quad (2)$$

$$P(ABD)=20.2+10.4=30.6\%, \quad (3)$$

(*note that ABCD overlaps AC)

$$\text{Total_Successes:} \\ = P(AC)+P(ABD)-P(ABCD) = AC+ABC'D=18.2+12.1+14.8+20.2+10.4 = 75.7\% \quad (4)$$

$$B. \text{ Percent of original successes: } \quad AC: 65.3/75.7=86\% \quad (5)$$

$$ABD: 30.6/75.7=40\% \quad (6)$$

II. Sort the list according to the percentage (the current case study is too small, so it has already been sorted).

III. Determine the coverage level - the percentage of successes that you would like to describe with rules. To illustrate this, we shall describe two cases: (1) 85% coverage and (2) 95% coverage.

IV. Follow the steps of the loop below:

- A) From the remaining rules, choose the rule with the greatest percentage.
- B) Increase the "Success Coverage" by the chosen rule percentage.
- C) Reduce the percentage of the remaining rules by their overlap with the chosen group of rules.
- D) If the "Success Coverage" exceeds the threshold indicated in Step III, stop; otherwise, delete the chosen rule from the list of remaining rules and go to A.

For example:

- The AC rule satisfies the 85% threshold(it covers 86%>85% of the successes).
- The AC rule is insufficient for the 95% threshold (86%<95%). Thus, we choose the next rule, ABD and reach 100% coverage of the successes (which must be satisfactory).

4.4. Phase 4: Revisiting rules of single attributes (main effects) and rules of attribute pairs.

In this phase, we sequentially revisit the percentage of cases for each attribute or pair of attributes and decide which of them may have been ignored due to minor inconsistencies that can be tolerated. The current case study is used for illustrating Phase 4, as discussed below.

4.4.1. Revisiting main effects (rules of a single attribute) in the case study

Since there are four attributes in the current case study, there are eight rules of single attribute to consider:

A, A', B, B', C, C', D and D' (explicitly: A=1, A=0, B=1, B=0, C=1, C=0, D=1, D=0)

For example, consider rule A, meaning A=1 (Obese) in our case study. From Tables 2 and 3, this rule covers: 10+36+182+121+44+104+148+202=847 cases (84.7% of the population). However, this rule is inconsistent in the following cases (see Tables 2 and 3): AB'C'D' (10 cases), ABC'D' (36 cases), and AB'C'D (44 cases). Therefore, using the rule A=1 (Obese), has the following probability of error (using Table 2):

$$P(\text{Error}|A) = (\text{AB'C'D', ABC'D', AB'C'D cases}) / (\text{A cases}) = 90/847 = 0.106 = \mathbf{10.6\%} \quad (7)$$

If 10.6% error can be tolerated, it must still be weighed against the simplicity it brings (i.e., the number of rules it saves). In this case, it saves one rule by replacing the two rules: AC, ABD. Thus, we have to weigh 10.6% error against saving one rule.

A second example is the rule A', meaning A=0 (not obese). All A' entries are zeros, hence:

$$P(\text{Error}|A') = (\text{A' cases}) / (\text{A' cases}) = 153/153 = \mathbf{1} \quad (8)$$

These computations and decisions are repeated for all 8 potential rules.

4.4.2. Revisiting effects of all attribute pairs:

Since there are four binary attributes (A,B,C,D) in the current case study, we have to consider the following 24 pairs of combinations: AB, A'B, AB', A'B', AC, A'AC', A'C', AD, A'D, AD', A'D', BC, B'C, BC', B'C', BD, B'D, BD', B'D', CD, C'D, CD', C'D'.

Since AC is already part of the rules:

For each pair, the computations are analogous to the computations of the single attribute.

For example, consider the rule AD for the current case study. Like all rules of attribute pairs, it has four combinations:

1. ABCD, meaning A=1,B=1,C=1,D=1 -with 202 cases - a success
2. AB'CD, meaning A=1,B=0,C=1,D=1 - with 121 cases - a success
3. ABC'D, meaning A=1,B=1,C=0,D=1, - with 104 cases - a success
4. AB'C'D, meaning A=1,B=0,C=0,D=1 - with 36 cases - a failure

Thus, using this rule has the following error probability:

$$P(\text{Error}|AD) = (\text{AD \# of failure cases}) / (\text{\# of AD cases}) = 36/(202+121+104+36) = 36/463 = 7.8\%$$

The rule AD can replace ABD so the only benefit gained from using it is the ability to ignore attribute B (B is gender). The decision-makers then have to decide whether or not to ignore the gender factor and have 7.8% error probability, or to eliminate this error probability by considering the gender factor.

5. The effect of having more data fields (more characteristics/attributes)

Let us define *k* as the number of attributes in the problem. As the problem becomes bigger, *k* grows and the consequences are as follows:

- In Phase 1: The number of combinations (rows in the Truth Table) is 2^k. This is an exponential growth.
- In Phase 2: If *k*>5, a Karnaugh map can no longer describe it. Instead, the Quine and McCluskey algorithm must be applied. However, the complexity of Quine and McCluskey's algorithm grows exponentially with *k*.
- In Phase 3: As *k* grows, the number of rules also grows considerably, and filtering out rules becomes more of an issue. When it comes to human decision, too many rules complicate things, and we may be willing to trade the exactness of describing successes for simplicity.
- In Phase 4: The number of single attribute computations is 2^k (each of the *k* attributes can be either 1 or 0). The number of attribute pair computations is the multiplication of all the value combinations of the pair (2²) by the number of pairs: (k-1)+(k-2)+...+1 = ((k-1)*k)/2. For example, in 4.4.2 above, the number of pairs is:

$$(2^2)^* ((k-1)*k)/2 = (2^2)^* ((4-1)*4)/2 = 4*(3*4)/2 = 4*12/2 = 4*6 = 24 \quad (9)$$

Overall, the number of computations is proportional to k^2 ($O(k^2)$). However, each single computation directly depends on the number of combinations and thus, grows exponentially with k .

6. Conclusion

This paper presents a new approach for rule extraction in dichotomous or binary databases. Each rule describes a set of combinations necessary to achieve the desired output. The approach is based on four phases and utilizes the fact that some data could easily be transformed into binary data. In the second phase, a Boolean function minimization algorithm proposed by Quine and McCluskey is adopted for Boolean logic function minimization. However, without the other three phases, Boolean function minimization alone does not suit decision-making. While the example in the current paper is small, the method is very efficient when applied to much bigger systems.

The current paper may lead to future work in data mining, in general, and in rule extraction, in particular. The naive method of choosing the threshold for success and failure can be replaced by optimization techniques using mathematical programming. Future research is still needed to improve the processes in each phase of the methodology described above in the current paper. An interesting extension could be generalization of the QM method for non-binary data.

REFERENCES

- Allison, P. D. (1991) Logistic Regression Using the SAS System: Theory and Application. SAS Institute Cary, NC.
- Allison, P. D. (1995) Survival Analysis Using the SAS System: A Practical Guide. SAS Institute Cary, NC.
- Andrews, R. J. Diederich, and A. B. Tickle, (1995) A survey and critique of techniques for extracting rules from trained neural networks, Knowledge Based Systems 8, No. 6.
- Baesens, B. R. Setino, C. Mues, J. Vanthienen, (2003) Using Neural Network Rule Extraction and decision tables for credit-risk evaluation, Management Science, 49, No. 3.
- Bartree, T. C. (1961) Computer design of multiple output logical networks, IRE Transactions on Electronics Computers, EC-10, No.1.
- Capon, N. (1982) Credit scoring systems, a critical analysis, Journal of Marketing, 46.
- Chelst, K. (1998) Can't See the Forest Because of the Decision Trees: A Critique of Decision Analysis in Survey Texts. Interfaces, 28, No. 2.
- Greene, W. H. (2000) Econometric Analysis (Fourth edition), Prentice Hall.
- Karnaugh, M. (1953) The map method for synthesis of combinatorial logic circuits, Transactions of AIEE, 72, No. 9.
- Kohavi, Z. (1978) Switching and Finite Automata Theory (2nd ed.), McGraw Hill.
- Long, S. J. and J. Freese (2001) Regression Models for Categorical Dependent Variables Using STATA, STATA Press, College Station, TX,.
- Long, S. J. (1997) Regression Models for Categorical and Limited Dependent Variable, Advanced Quantitative Techniques in the Social Sciences, Sage Publications.
- McCluskey, E. J. (1956) Minimization of Boolean functions, Bell System Technical Journal, 35, No. 6.
- McCluskey, E. J. and H. Schorr (1962) Essential multiple-output prime implicants in mathematical theory of automata, Proceedings of the Polytechnique Institute Brooklyn Symposium, 12.
- Mitchell, T. (1977) Decision tree learning, in: T. Mitchell, Machine Learning, (McGraw-Hill).
- Quine, W. V. (1952) The problem of simplifying truth functions, American Mathematics Monthly, 59, No. 8.
- Quine, W. V. (1955) A way to simplify truth functions. American Mathematics Monthly, 62 No.9.
- Quinlan, J. R. (1986) Induction of decision trees, Machine Learning, (1986).
- Quinlan, J. R. (1993) C4.5 programs for machine learning, Morgan Kaufman, Chambery, France
- Russell, S. and P. Norvig (1995) Artificial intelligence a modern approach, Prentice Hall.
- Sörensen, K. and G. K. Janssens, (2003) Data mining with genetic algorithms on binary trees, European Journal of Operational Research, 151, No. 2.
- Steenackers, A. and M. J. Goovaerts (1989) A credit scoring model for personal loan, Insurance: Math Economics, 8.
- Viaene, S., D. A. Derrig, B. Baesens, and G. Dedene (2002) A comparison of state of the art classification techniques for the auto-mobile insurance claim fraud detection, The Journal of Risk and Insurance, 69, No. 3.
- Veitch, E. W. (1952) A Chart method for simplifying truth functions, Proceedings of ACM.
- Winston, P. (1992) Learning by building identification trees, in: P. Winston, Artificial Intelligence, Addison-Wesley Publishing Company.