2008

Data exploration by using the monotonicity property

Hongyi Chen
Louisiana State University and Agricultural and Mechanical College

DATA EXPLORATION BY USING THE MONOTONICITY PROPERTY

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in System Science

in

The Department of Computer Science

by
Hongyi Chen
B.E., Sichuan University, 2001
August, 2008

We are drowning in information,
but starving for knowledge!

----John Naisbitt

ACKNOWLEDGMENTS

At this opportunity the author wishes to express her most sincere gratitude and appreciation to Dr. Evangelos Triantaphyllou of the Computer Science Department for his valuable guidance and genuine interest as research advisor and chairman of the examination committee. His encouragements have kept me going even in difficult times, when the research seemed to go nowhere.

Deep appreciation is also extended to Dr. Jianhua Chen of the Computer Science Department and Dr. Warren Liao of the Industrial Engineering Department for serving as my committee members. Their comments have improved the quality of this research.

Finally, I am forever grateful for the supports from my parents, my boyfriend, my roommates and all my college friends throughout my study at LSU. They have all contributed to this thesis through ideas, encouragement, and support. Without their help, my study will not be as enjoyable and memorable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Dealing with different misclassification costs has been a big problem for classification. Some algorithms can predict quite accurately when assuming the misclassification costs for each class are the same, like most rule induction methods. However, when the misclassification costs change, which is a common phenomenon in reality, these algorithms are not capable of adjusting their results. Some other algorithms, like the Bayesian methods, have the ability to yield probabilities of a certain unclassified example belonging to given classes, which is helpful to make modification on the results according to different misclassification costs. The shortcoming of such algorithms is, when the misclassification costs for each class are the same, they do not generate the most accurate results.

This thesis attempts to incorporate the merits of both kinds of algorithms into one. That is, to develop a new algorithm which can predict relatively accurately and can adjust to the change of misclassification costs.

The strategy of the new algorithm is to create a weighted voting system. A weighted voting system will evaluate the evidence of the new example belonging to each class, calculate the assessment of probabilities for the example, and assign the example to a certain class according to the probabilities as well as the misclassification costs.

The main problem of creating a weighted voting system is to decide the optimal weights of the individual votes. To solve this problem, we will mainly refer to the monotonicity property. People have found the monotonicity property does not only exist in pure monotone systems, but also exists in non-monotone systems. Since the study of the monotonicity property has been a huge success on monotone systems, it is only natural to apply the monotonicity property to non-monotone systems too.

This thesis deals only with binary systems. Though such systems hardly exist in practice, this treatment provides concrete ideas for the development of general solution algorithms.

After the final algorithm has been formulated, it has been tested on a wide range of randomly generated synthetic datasets. It has also been compared with other existing classifiers. The results indicate this algorithm performs both effectively and efficiently.

CHAPTER 1. INTRODUCTION

1.1 Data Mining and Classification Algorithms

There are many definitions for data mining. One of the most popular definitions is: data mining, also known as Knowledge Discovery in Databases (KDD), is "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" (Frawley, et al., 1992). That is to say, data mining is nothing but "the science of extracting useful information from large data sets or databases" (Hand, et al., 2001).

A central problem in data mining is how to assign data examples to one of several predefined classes and infer some key patterns. This kind of problem in data mining is referred to as the classification problem. An algorithm designed to solve this type of problem is consequently called classification algorithm, or classifier.

Usually an analyst will be given a set of examples whose classes are already known. Then the analyst will need to classify some new examples, based on the knowledge he/she has learned from the given examples. It is a pervasive problem that encompasses many diverse applications. With the help of a classifier, analysts can identify different classes of customers, behaviors, or activities, and take actions based on their classes.

For instance, a doctor may have a pile of past medical records of a certain disease, and a new patient to be seen. After taking measurements of the new patient, the doctor needs to decide whether this new patient has the disease or not. With the assistance of a classifier, the doctor can make use of all the past records to obtain knowledge about the pattern of the existence of that disease. Thus, the pattern inferred from the given records provides a predictive model. This predictive model will help the doctor to decide if the patient is sick or not, it might even tell the doctor at what stage the disease is. Therefore, the doctor can decide what kind of action should be taken next, maybe an operation needs to be performed, or a more conservative treatment should be given.

Credit card companies have been using classifiers to detect potentially fraudulent credit card transactions. Since credit card companies have the records of their customers' past behavior, as well as those fraud activities that have been found out in the past, a classifier can be applied to analyze these data and yield a predictive model. Next time when a credit transaction is executed, the transaction and all its data elements describing this transaction are analyzed to determine whether or not the new transaction is a potentially fraudulent charge. Apparently this is of great interest to those companies, because it can help the credit card issuers, along with their customers, to decrease and mitigate losses due to fraudulent charges.

Manufacturing companies can use classifiers not only to identify the defective products, but also to find out the possible cause or characteristics of defective products. For example, on what day of a week or what time of a day produced the most defective products, and what components were used and which individuals were working on the assembling line. By understanding these characteristics, changes can be made to the manufacturing process to improve the quality of the products being produced. High-quality products lead to improved reputation of the organization within its industry and help to drive sales. In addition, profitability improves through the reduction of returned materials and field service calls.

A few practical uses of classifiers have been highlighted here. In fact, there are many more other ways that classification algorithms can be applied to provide great insight into business analysis, science projects or medical research. All in all, the valuable knowledge that classification algorithms can provide is patterns or events that one may not see through his/her eyes. As data storage technology advances and information systems continue endlessly to collect and process data, a treasure is amassing and is waiting to be discovered.

1.2 Misclassification Costs

A big problem in classification is how to evaluate the performance of a classifier. The misclassification rate, i.e., the number of examples being misclassified, is certainly an important criterion. In real life applications, however, mere misclassification rate is simply not good enough of a criterion to decide how a classifier performs. More often than not, we also need to incorporate a concept called misclassification cost (Michie, 1994).

Think of the case of medical diagnostics. Imagine two doctors that both sort out tumors into the "benign" and "malicious" classes (Triantaphyllou, 2008). Suppose the first doctor is correct on 98% cases, and the second one is only correct on 90% cases. But the first one misdiagnoses all malicious cases, while the second only makes mistakes on benign cases. Which doctor is considered better? Despite of the fact that the first doctor has lower rate of misclassification, the second doctor is the one considered to give better medical advice. This is because of different costs of misclassification. In our scenario, there are two types of misclassification:

- assign a truly "benign" case to the "malicious" class;
- assign a genuine "malicious" case to the "benign" class.

Whereas the first type of misclassification is just annoying, the second type could be disastrous. In other words, the misclassification cost of the first type is relatively low, while the misclassification cost of the second type could be too high to afford. If the two doctors are two classifiers that we are evaluating, clearly, one would rather bias the classifier which makes fewer errors of the second type, even if that means making more mistakes of the first type. Of course, even though the cost of the second type of

misclassification is much higher, we cannot go to the other extreme to announce every case to be "malicious" just to be on the safe side. Such a classifier is practically of no use and such a doctor will lose credit from his patients. A good classifier should be able to tell a "benign" case when it reaches a certain confidence level.

The same situation happens when the government tries to decide whether or not to issue a hurricane alert. They, too, face two types of errors:

- issue a hurricane alert but nothing really happens;
- fail to issue an alert before the hurricane really comes.

The first type of misclassification will cause people to move around in vain, while the second type could cost people's lives. So, there again, the two types of misclassification have different costs. We certainly do not want to sacrifice people's lives. However, we also do not want to issue an alert every time there is only a slight chance of having a hurricane, because the cost of having people evacuate everyday is also high. It is a huge waste of time and money. Therefore, we must find a balance between the two types of misclassification rates, and the two types of rates are in every way related to their misclassification costs.

For a credit card company, a classifier is expected to help make the decisions that will minimize the risk while trying to issue credit cards to more people. In such decisions, they take the risks of:

- offering someone a credit card but the card holder is not capable of paying off;
- not offering someone a credit card but lose a potential customer.

In this case, the misclassification costs may vary from company to company. There are some risk-taking companies; while there are also some conservative ones. In such circumstances, we would prefer to have a classifier that can deliver an assessment of probabilities, i.e., the probabilities of a certain applicant belonging to given classes. Then the decision is left to the companies themselves. For instance, a classifier may announce some applicant to be 35% likely of being financially capable, and 65% likely of being non-capable. Then the companies will decide themselves whether or not to issue this applicant a card, based on their own judgment of misclassification costs.

After examining the above illustrating cases, we can see that the root to the problem of different misclassification costs is how to find a balance between overfitting and overgeneralization. Overfitting is the situation where a classifier learns very well on the training examples but makes poor predictions on new test examples (Mitchell, 1997). An extreme case of overfitting is when a classifier only accepts examples that are exactly the same as the given examples and rejects all the new examples. Such a classifier is not capable of making any predictions at all. On the other hand, overgeneralization, also known as underfitting, is exactly the opposite of overfitting. It will classify too many examples even if it does not have adequate information of how to classify. An extreme

case of overgeneralization is when a classifier accepts all new examples to be in a certain class. This classifier cannot be very useful in practical situations either.

1.3 Importance of This Problem

As mentioned above, dealing with different misclassification costs is an important problem in classification. Some classifiers can predict quite accurately when assuming the misclassification costs for each class are the same. This is the case of most rule induction methods. However, when the misclassification costs for different classes take different values, which is a common phenomenon in reality, these classifiers are not capable of adjusting their results.

Some other classifiers, like the Bayesian methods, have the ability to yield an assessment of probabilities for each unclassified example. This assessment of probabilities is a tuple of numbers that indicate the probability of a certain example belonging to each class. It can help us to make modification on the results according to the change of misclassification costs. It is also useful to decide where the balance between overfitting and overgeneralization is. However, the shortcoming of such classifiers is, though it can yield an assessment of probabilities, comparing to other classifiers, they have a relatively high misclassification rate.

Therefore this thesis attempts to incorporate the merits of both kinds of classifiers into one, i.e., to develop a new classifier which can both predict relatively accurately, and calculate an assessment of probabilities at the same time.

Next is the layout of this thesis. Some notation and an illustrative problem will be described in Chapter 2. Then some background information and the relevant literature will be reviewed in Chapter 3. In Chapter 4, some new notation are introduced, and in Chapter 5, the proposed methodology for solving this problem is presented. There will be 24 different schemes for the final algorithm. In Chapter 6, after describing the process of generating synthetic datasets, the 24 proposed schemes will be compared on 10 randomly generated synthetic datasets. On analyzing the results, the scheme that has performed the best will form the new algorithm, called the M^* algorithm. In Chapter 7, the pseudo code of M^* algorithm and its complexity analysis will be given. Next, an extensive illustrative example will be elaborated in Chapter 8. In Chapter 9, we will compare the M^* algorithm to other 75 existing classifiers, and analyze the results. Chapter 10 will explain the rationale of the newly developed algorithm, and Chapter 11 will give the concluding remarks. In the last, Chapter 12 will present a few directions of future research.

CHAPTER 2. NOTATION AND ELEMENTARY PROBLEM DESCRIPTION

2.1 Notation

2.1.1 The Binary System

Suppose we use attributes x_1, x_2, x_3, … to describe an example, and each attribute only takes two values. For instance, attribute "sex" takes values {male, female}, attribute "humidity" takes values {high, low}, and attribute "cancer" takes values {yes, no}. Then these attributes are called binary attributes. Since binary attributes take only two values, we can always arbitrarily assign one value to be "1", and the other to be "0". Suppose we have n such binary attributes $<x_1, x_2, x_3, \ldots, x_n>$ for each example. Since every attribute takes 2 values, then n attributes can define a total of 2^n examples. Now imagine each of the 2^n examples is attached with another binary attribute, c. This attribute defines which class the corresponding example belongs to, and hence it will be called the class attribute. The two values of the class attribute c can also be represented by "1" and "0". Moreover, if an example has class attribute "1", we will say the example belongs to class "1", or, the example is positive. Similarly, if an example has class attribute "0", we will say the example belongs to class "0", or, the example is negative. All these 2^n examples, together with their class attribute c, form a binary system, and this binary system has dimension n.

Apparently, in a deterministic setting, there exists a mapping f between the 2^n examples and the class attribute c, that is, a mapping f: $\{0,1\}^n \rightarrow \{0,1\}$. This mapping, or pattern, is exactly the knowledge we want to find out, normally it is not known to us. Provided with some classified examples, i.e., examples whose classifications are already known to us, we hope our classifier can simulate this mapping. Note that every mapping of f: $\{0,1\}^n \rightarrow \{0,1\}$ can be represented as a Boolean function, and every Boolean function can be transformed into conjunction normal form (CNF) or disjunction normal form (DNF) (Koppelberg, 1989). In this thesis, we will use CNF to represent the hidden mapping of each application system.

To get a sense of what a classification problem would be like, here is an illustration. Suppose in a binary system of dimension 3, each example is described by a tuple $<x_1, x_2, x_3>$. We are given four classified examples, two positive examples <1, 1, 1>, <1, 1, 0>, and two negative examples <0, 0, 1>, <0, 0, 0>. Since this binary system has dimension 3, the system has a total of $2^3 = 8$ examples. We already know the classification of half of them. We still want to know the classification of the rest. Suppose the hidden mapping f of this binary system is $f = (x_1 \vee x_2) \wedge (x_2 \vee x_3)$. The Boolean function's value is the class attribute's value of the corresponding examples, then the actual classifications for the rest of the four unclassified examples <1, 0, 1>, <0, 1, 1>, <1, 0, 0>, <0, 1, 0> should be 1, 1, 0, 1, respectively. Suppose a classifier infers these unclassified examples have classifications 1, 1, 0, 0, respectively. Then this classifier gets 3 out 4 classifications

correct. It misclassifies the last example <0, 1, 0>, which is predicted to be negative, while its actual classification should be positive.

Throughout this thesis, we will mainly discuss binary systems. Though pure binary systems are rare in practice, this is not a real limitation to the application of our discussion, because any non-binary system, no matter what its data type is, can be easily transferred into a binary one, through a binarilization process (Bartnikowski, et al., 2006). In fact, binary systems are the basis of all other systems. The treatment of binary systems can provide concrete and fundamental insights into the development of non-binary systems. Since binary systems are the simplest models, we can easily study their main properties without being distracted by other factors.

2.1.2 The Probability Pair (p_1, p_0)

The probability pair (p_1, p_0) of an example of a binary system is the probabilities of the corresponding example belong to both classes. To be more specific, p_1 is the probability of the corresponding example belonging to class "1", while p_0 is the probability of the corresponding example belonging to class "0". Since a binary system has only two classes, the probability pair must satisfy the following condition:

$$p_1 + p_0 = 1$$

For instance, examples v_1 through v_5 are assumed to have the probability pairs shown in the following table:

Table 2.1 Sample Probability Pairs.

Example	Probability Pair
v_1	(0.9, 0.1)
v_2	(0.7, 0.3)
v_3	(0.55, 0.45)
v_4	(0.5, 0.5)
v_5	(0.4, 0.6)

Apparently, v_1, v_2, v_3 have higher probability of belonging to class "1"; v_4 has equal probability of belonging to either class; while v_5 is more likely to belong to class "0".

Moreover, if we classify v_1, v_2, v_3 to be positive, then v_1 is more likely to be correct, and v_3 is more likely to be wrong. Therefore if we are trying to shrink class "1", we will drop v_3 first, and keep v_1 last.

2.1.3 The Misclassification Probability Pair (p_0, p_1)

Another implication of the probability pair is, if we decide v_1 to be class "1", it is 90% likely that we have made the correct decision. There is still 10% chance that we have

misclassified it. On the other hand, if we decide v_1 to be class "0", it is 90% probable that we made the wrong decision.

In general, if we classify an example to be class "1", $(1 - p_1)$ represents the probability that the corresponding example is misclassified; if we classify an example to be class "0", $(1 - p_0)$ represents the probability that the corresponding example is misclassified.

Therefore, we can say that the pair $((1 - p_1), (1 - p_0))$, or (p_0, p_1), is the pair of the misclassification probabilities if the corresponding example is classified to class "1" and "0", respectively.

2.1.4 The Misclassification Costs (c_1, c_0)

Now, suppose we have a pair of misclassification costs (c_1, c_0), where c_1 is the cost of an example being misclassified to be of class "1", or the cost of a false-positive case, while c_0 is the cost of an example being misclassified to be of class "0", or the cost of a false-negative case.

Misclassification costs can be any real numbers, such as (4.5, 0.5). But we can always normalize these numbers so that,

$$c_1 + c_0 = 1, \text{ and } c_1, c_0 \geq 0.$$

This is valid because in the end what really matters is the ratio between c_1 and c_0. Hence (4.5, 0.5) means a false-positive costs 9 times more of a false-negative. Thus the misclassification costs can be normalized to (0.9, 0.1) according to the two above conditions. From now on, we will assume to use normalized misclassification costs.

2.1.5 The Expected Misclassification Costs (s_1, s_0)

If we take the Cartesian product of the misclassification probabilities (p_0, p_1) and the misclassification costs (c_1, c_0), the result will be the expected misclassification costs of classifying a certain example to a certain class (s_1, s_0). In other words,

$$(s_1, s_0) = (p_0, p_1) \times (c_1, c_0) = (p_0 \times c_1, p_1 \times c_0)$$

To be more specific, for example v, if it has misclassification probability pair (p_0, p_1), and misclassification costs (c_1, c_0), then $s_1 = p_0 \times c_1$ represents the expected misclassification cost of assigning v to be class of "1", and $s_0 = p_1 \times c_0$ represents the expected misclassification cost of assigning v to be of class "0". This is the case because if we classify v to be of class "1", it is p_1 probable that we have correctly classified it, then there will be no cost of misclassification. On the other hand, it is p_0 probable that we have incorrectly classified it. Then since the misclassification cost of a false-positive is c_1, the expected misclassification cost of classifying v to be of class "1" is $p_0 \times c_1$. The same

reasoning applies to the case when v is classified to be of class "0", the expected misclassification cost should be $p_1 \times c_0$.

The notion of expected misclassification costs combines the two most important factors, the misclassification probability pair and the misclassification costs. It shows us the expected costs of classifying an example to each class. In our algorithm, we will try to assign examples to a certain class so that the expected misclassification cost will be minimum, and generally the actual cost of misclassification will turn out to be minimized as well.

For instance, for example v, if we have expected misclassification costs (s_1, s_0) = (0.1, 0.9), it means assigning v to class "1" has a much lower expected cost than assigning it to class "0". Therefore in our final result, example v will be assigned to class "1" according to the comparison of s_1 and s_0.

Now that we are acquainted with the notions of the probability pair (p_1, p_0), the misclassification probability pair (p_0, p_1), the misclassification costs (c_1, c_0), and the expected misclassification costs (s_1, s_0), we will examine some scenarios to get a better insight of how these factors will affect our classification process.

Suppose we are provided with some classified examples, and five new unclassified examples v_1 to v_5. Through our calculation, we get the probability pairs of the five unclassified examples as shown in Table 2.1. Now assume we have misclassification costs of (0.5, 0.5), the results are shown in the following table.

Table 2.2 Classification of Table 2.1 When Misclassification Costs (c_1, c_0) are (0.5, 0.5).

Ex	Probability Pair (p_1, p_0)	Misclassification Probability Pair (p_0, p_1)	Expected Misclassification Costs (s_1, s_0)	Class
v_1	(0.9, 0.1)	(0.1, 0.9)	(0.05, 0.45)	1
v_2	(0.7, 0.3)	(0.7, 0.3)	(0.35, 0.15)	1
v_3	(0.55, 0.45)	(0.45, 0.55)	(0.225, 0.275)	1
v_4	(0.5, 0.5)	(0.5, 0.5)	(0.25, 0.25)	1/0
v_5	(0.4, 0.6)	(0.6, 0.4)	(0.3, 0.2)	0

The misclassification costs of (0.5, 0.5) means a false-positive costs the same as a false-negative. This is the case, for instance, when we are trying to guess someone's gender. Either kind of mistake costs equally. Since the misclassification costs are the same, choosing the class that yields the smaller expected misclassification cost is the same as choosing the class that has a lower misclassification probability, or as choosing the class that has a higher probability. So we will assign an example to the class that it is more

probable to belong to. Therefore v_1, v_2, v_3 are assigned to class "1", while v_5 is assigned to class "0".

A special case here is v_4, whose expected misclassification costs are the same (0.25, 0.25). In this situation, there are two ways to settle this problem. We can refer to a random function and assign whatever class the random function generates for this example, or we can mark this example as an "undecidable" one (not to be confused with the unclassified ones). In the algorithm we discuss later, we will use the first method, a random function, to decide a tied case.

In the second scenario, for the same set of examples, we have different misclassification costs, say (0.8, 0.2). As it can be seen in Table 2.3 that now the result is very different from what we get from Table 2.2, where we have misclassification cost (0.5, 0.5). The reason is that now we have a much higher cost of a false-positive than a false-negative. Thus, we want to classify more examples to be negative to reduce the false-positives. But which examples should we change their classes to negative and which examples we still want to keep as positive? That will need the help of the probability pair. For v_2 and v_3, which previously were classified as positive, their expected misclassification cost to be a positive are now higher than their expected misclassification cost to be a negative, they will change their classes to be negative. However, for v_1, since it is highly probable to be positive, its expected misclassification cost to be a positive is still smaller than its expected misclassification cost to be a negative, v_1 will remain to be classified as positive.

Table 2.3 Classification of Table 2.1 When Misclassification Costs (c_1, c_0) are (0.8, 0.2).

Ex	Probability Pair (p_1, p_0)	Misclassification Probability Pair (p_0, p_1)	Expected Misclassification Costs (s_1, s_0)	Class
v_1	(0.9, 0.1)	(0.1, 0.9)	(0.08, 0.18)	1
v_2	(0.7, 0.3)	(0.7, 0.3)	(0.56, 0.06)	0
v_3	(0.55, 0.45)	(0.45, 0.55)	(0.36, 0.11)	0
v_4	(0.5, 0.5)	(0.5, 0.5)	(0.4, 0.1)	0
v_5	(0.4, 0.6)	(0.6, 0.4)	(0.48, 0.08)	0

The third scenario is when the misclassification costs are (0.2, 0.8), which is the opposite situation of case 2. Now a false-positive costs a lot less than a false-negative.

Table 2.4 Classification of Table 2.1 When Misclassification Costs (c_1, c_0) are (0.2, 0.8).

Ex	Probability Pair (p_1, p_0)	Misclassification Probability Pair (p_0, p_1)	Expected Misclassification Cost (s_1, s_0)	Class
v_1	(0.9, 0.1)	(0.1, 0.9)	(0.02, 0.72)	1

(table con'd.)

v_2	(0.7, 0.3)	(0.7, 0.3)	(0.14, 0.24)	1
v_3	(0.55, 0.45)	(0.45, 0.55)	(0.09, 0.44)	1
v_4	(0.5, 0.5)	(0.5, 0.5)	(0.1, 0.4)	1
v_5	(0.4, 0.6)	(0.6, 0.4)	(0.12, 0.32)	1

After comparing the expected misclassification costs of each class, we will choose the class that costs less. The results are shown in Table 2.4, as we can see, all five examples are now classified to be positive.

From the above three scenarios, we also get a glimpse of the classification process, which is as follows:

- get the probability pair (p_1, p_0) of each unclassified example;
- reverse the probability pair (p_1, p_0) to get the misclassification probability pair (p_0, p_1) of each unclassified example;
- take the Cartesian product of the misclassification probability pair (p_0, p_1) and the misclassification costs (c_1, c_0) to get the expected misclassification costs (s_1, s_0) of each unclassified example;
- compare s_1 and s_0, find the corresponding class that has the smaller expected misclassification cost for each unclassified example.

Since the misclassification costs (c_1, c_0) are provided by each application system, all we need to calculate is the probability pair (p_1, p_0) of each unclassified example, then we can get the misclassification probability pair (p_0, p_1) as well as the expected misclassification costs (s_1, s_0), and hence the class of each unclassified example. Therefore this thesis aims at finding a classifier which can calculate the probability pair (p_1, p_0) for each unclassified example.

2.2 Elementary Problem Description

Assume there is a binary system B that has dimension n, and available are two sets, T_1 and T_2, of binary examples of system B. T_1 is called the training data and T_2 is called the testing data. As suggested by their names, the training data T_1 will be used to feed the classifier and build a predictive model, then the testing data T_2 will be used to evaluate the performance of that classifier.

Other parameters are as follows: m_1 is the total number of examples in T_1; m_1^+ is the number of positive examples in T_1; m_1^- is the number of negative examples in T_1; m_2 is the total number of examples in T_2; m_2^+ is the number of positive examples in T_2; and m_2^- is the number of negative examples in T_2. Apparently, the following equations are true:

$$m_1 = m_1^+ + m_1^-, \text{ and } m_2 = m_2^+ + m_2^-.$$

Suppose a classifier has generated results for certain training data T_1 and testing data T_2. If we compare the predicted class of each example in testing data T_2 with its actual class, we will get the number of false-positive R_1, and the number of false-negative R_0. Suppose the misclassification costs are (c_1, c_0). Then the actual cost of false-positive is the product of $R_1 \times c_1$, the actual cost of false-negative is the product of $R_0 \times c_0$. If we sum up both products, we will the actual total cost C,

$$C = R_1 \times c_1 + R_0 \times c_0$$

Therefore, the goal of this thesis is to formulate a new classifier that has the ability to generate a probability pair as well as a predicted class for every unclassified example in testing dataset T_2, such that the actual total cost C will be the minimum.

The way we evaluate the performance of different classifiers will be,

- run different classifiers on training data T_1 and testing data T_2;
- compare the predicted class of each example in testing data T_2 with its actual class, count the numbers of false-positive R_1, and the number of false-negative R_2;
- multiply the number of false-positive by the cost for false-positive, and the number of false-negative by the cost for false-negative respectively, then sum up both products, i.e., get the value of the actual total cost $C = (R_1 \times c_1) + (R_2 \times c_2)$.
- Compare the value of C for different classifiers.

We will repeat this process for different training datasets and testing datasets.

Since we would like to compare our new classifier with other existing classifiers, and not all classifiers have the ability to adjust to the change of misclassification costs, for now we will assume the misclassification costs take the same value, i.e., (c_1, c_0) = (0.5, 0.5).

Let us use an illustrative problem to see our classification process. Suppose we have a binary system of dimension $n = 6$. The hidden Boolean function of this binary system is assumed to be given by,

$$f = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_4 \vee \overline{x_5} \vee x_6)$$

As mentioned before, we will use CNF to represent the mapping between $\{1,0\}^n \rightarrow \{1,0\}$, and this is not a limitation to the representation.

Since this binary system has dimension 6, it means we have a total of $2^6 = 64$ examples. Among them, 32 examples (i.e., half of them) are classified, or are the training data. There are 20 positive examples and 12 negative examples. In other words, $m_1 = 32$, $m_1^+ = 20$, and $m_1^- = 12$. The remaining 32 examples are the testing data. We will treat them as the unclassified examples before we get their predicted classes. After we run the classifiers on the testing data, we will refer to the hidden Boolean function f, and find out the actual classes of the testing data. In such a way, we can compare the predicted class

with the actual class of each example in the testing data. There are 32 examples in T_2, 20 of them are positive, and the remaining 12 of them are negative, i.e., $m_2 = 32$, $m_2^+ = 20$, and $m_2^- = 12$. Note that m_1 and m_2, m_1^+ and m_2^+, m_1^- and m_2^- happen to be the same numbers, this is not usually the case in general situations. However, if the training data and the testing data are selected randomly, the corresponding parameters do tend to be proportioned. That is:

$$\frac{m_1}{m_2} \approx \frac{m_1^+}{m_2^+} \approx \frac{m_1^-}{m_2^-}$$

All the 64 examples are shown below. T_1^+ is the set of classified positive examples, T_1^- is the set of classified negative examples, T_2^+ is the set of unclassified positive examples, T_2^- is the set of unclassified negative examples.

$$T_1^+ = \begin{bmatrix} 1&1&1&1&1&1 \\ 1&1&1&1&0&1 \\ 1&1&0&1&1&1 \\ 1&0&1&1&1&1 \\ 1&1&1&0&1&0 \\ 1&0&1&1&1&0 \\ 1&0&1&1&0&1 \\ 1&1&0&0&1&1 \\ 0&1&0&1&1&1 \\ 1&1&0&1&0&0 \\ 1&0&0&1&1&0 \\ 1&0&1&0&0&1 \\ 1&0&0&1&0&1 \\ 0&1&0&0&1&0 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&1 \\ 1&0&0&0&0&1 \\ 0&1&0&1&0&0 \\ 0&1&0&0&0&0 \\ 0&0&0&0&0&0 \end{bmatrix}, \quad T_1^- = \begin{bmatrix} 1&1&1&0&0&1 \\ 0&1&1&0&1&1 \\ 0&1&1&1&0&0 \\ 1&0&1&0&1&0 \\ 0&0&1&1&1&0 \\ 0&0&1&1&0&1 \\ 0&0&1&0&0&1 \\ 0&0&1&0&1&0 \\ 1&0&0&0&1&0 \\ 0&1&1&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&0 \end{bmatrix}$$

$$T_2^+ = \begin{bmatrix} 1&1&1&1&1&0 \\ 1&1&1&1&0&0 \\ 1&1&0&1&1&0 \\ 1&1&0&1&0&1 \\ 1&0&1&0&1&1 \\ 1&0&0&1&1&1 \\ 1&0&1&1&0&0 \\ 0&1&0&1&1&0 \\ 1&1&0&0&0&1 \\ 0&1&0&1&0&1 \\ 1&0&0&0&1&1 \\ 0&0&0&1&0&1 \\ 0&1&0&0&0&1 \\ 0&0&0&1&1&0 \\ 1&0&0&1&0&0 \\ 1&0&1&0&0&0 \\ 1&1&0&0&0&0 \\ 1&0&0&0&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&0&1 \end{bmatrix}, \quad T_2^- = \begin{bmatrix} 1&1&1&0&1&1 \\ 0&1&1&1&1&1 \\ 0&1&1&1&1&0 \\ 0&1&1&1&0&1 \\ 0&0&1&1&1&1 \\ 1&1&1&0&0&0 \\ 1&1&0&0&1&0 \\ 0&1&1&0&1&0 \\ 0&1&1&0&0&1 \\ 0&0&1&0&1&1 \\ 0&1&0&0&1&0 \\ 0&0&1&1&0&0 \end{bmatrix}$$

Throughout this thesis, this illustrative problem will be used to highlight some key concepts. After we have formulated the classifier, the illustrative problem will be analyzed in Chapter 8.

CHAPTER 3. LITERATURE REVIEW

3.1 Review of Some Well Known Classifiers

As Michie's book (1994) suggests, there has been a lot of different approaches to classification. But all these approaches can be categorized into three groups, which are statistical methods, machine learning methods, and neural networks. The author will briefly describe all three categories and introduce some representative classifiers of each category.

3.1.1 Statistical Methods

Statistical approaches are generally characterized by having an explicit underlying probability model, which provides a probability of being in each class rather than simply a classification. There are several subcategories of statistical methods, such as classical statistical methods, Bayesian methods, *k*NN methods, and so on. In this section we will mainly introduce Bayesian methods and *k*NN methods. Because Bayesian methods provide the basis for learning algorithms that directly manipulate probabilities and *k*NN methods have a resemblance, in some way, to the new classifier in this thesis. A thorough treatment of statistical procedures is given in (McLachlan, 1992), (Schalkoff, 1992), and (O'Hagan, 2003).

3.1.1.1Bayesian Methods

All Bayesian methods are based on Bayes' theorem,

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

In this theorem, *D* is the training data, and *h* is a hypothesis. *P*(*h*) denotes the initial probability that hypothesis *h* holds, before we have observed the training data. *P*(*h*) is often called the prior probability of *h* and may reflect any background knowledge we have about the chance that *h* is a correct hypothesis. If we have no such prior knowledge, then we might simply assign the same prior probability to each candidate hypothesis. *P*(*D*) denotes the prior probability that training data *D* will be observed. *P*(*h*|*D*) denotes the probability of observing data *D* given some world in which hypothesis *h* holds. In this scenario we are interested in probability *P*(*h*|*D*) that *h* holds given the observed training data *D*. *P*(*h*|*D*) is called the posterior probability of *h*, because it reflects our confidence that *h* holds after we have seen the training data *D*. Notice the posterior probability *P*(*h*|*D*) reflects the influence of the training data *D*, in contrast to the prior probability *P*(*h*), which is independent of *D*. Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability *P*(*h*|*D*), from the prior probability *P*(*h*), together with *P*(*D*) and *P(D*|*h*).

Important Bayesian methods include Naïve Bayes (John, et al., 1995), Naïve Bayes Simple (Duda, et al., 1973), AODE Bayes (Webb, et al., 2005), AODEsr (Zhang et al.,

2006), HNB Bayes (Zhang, et al., 2005), NaiveBayesUpdateable (John, et al., 1995), and WAODE (Jiang, et al., 2006).

In Bayesian methods, each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example. Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian methods, prior knowledge is provided by asserting a prior probability for each candidate hypothesis, and a probability pair over observed data for each possible hypothesis. Bayesian methods can accommodate hypotheses that make probabilistic predictions. New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities. Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions. A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case.

For more information on Bayesian methods, refer to (Jensen, 1996), (Berger, 1999), (Bolstad, 2004), (Gelman, 2003), (Lee, 1997), and (Winkler, 2003).

3.1.1.2 *k*-Nearest Neighbor (*k*NN) Methods

The *k*-nearest neighbor algorithm is amongst the simplest of all data mining algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors. *k* is a positive integer, typically small. If $k = 1$, then the object is simply assigned to the class of its nearest neighbor. In two-class systems, it is helpful to choose *k* to be an odd number as this avoids tied votes.

In order to identify neighbors, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidean distance, though other distance measures, such as the Manhattan distance could in principle be used instead. The *k*-nearest neighbor algorithm is sensitive to the local structure of the data. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

There are a number of ways to classify the new vector to a particular class. One of the most used techniques is to predict the new vector to the most common class amongst the *k*-nearest neighbors. A major drawback to using this technique to classify a new vector to a class is that the classes with the more frequent examples tend to dominate the prediction of the new vector, as they tend to come up in the *k*-nearest neighbors when the neighbors are computed due to their large number. One of the ways to overcome this problem is to take into account the distance of each *k*-nearest neighbor with the new vector that is to be classified and predict the class of the new vector based on these distances.

The best choice of *k* depends upon the data; generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good *k* can be selected by various heuristic techniques, for example, cross validation.

The naive version of the algorithm is easy to implement by computing the distances from the test sample to all stored vectors, but it is computationally intensive, especially when the size of the training set grows. Many nearest neighbor search algorithms have been proposed over the years; these generally seek to reduce the number of distance evaluations actually performed. Some optimizations involve partitioning the feature space, and only computing distances within specific nearby volumes.

Important *k*NN methods include a branch and bound *k*NN rule (Fukunaka, et al., 1975), a condensed *k*NN rule (Hart, 1968), a reduced *k*NN rule (Gates, 1972), an edited *k*NN rule (Hand, et al., 1978), and KStar (Cleary, 1995).

For more information on *k*NN methods, refer to (Devijver, et al., 1982), (Davies, 1988), (Todeschini, 1989), (Aha, et al., 1991), (Scott, 1992), (McLachlan, 1992), (Dasarathy, 1991), and (Shakhnarovish, 2005).

3.1.2 Neural Networks

Generally, neural networks consist of layers of interconnected nodes, each node producing a non-linear function of its input. The input to a node may come from other nodes or directly from the input data. Also, some nodes are identified with the output of the network. Therefore, the complete network represents a very complex set of interdependencies which may incorporate any degree of nonlinearity, allowing very general functions to be modeled.

In the simplest networks, the output from one node is fed into another node in such a way as to propagate information through layers of interconnecting nodes. More complex behavior may be modeled by networks in which the final output nodes are connected with earlier nodes, and then the system has the characteristics of a highly nonlinear system with feedback. It has been argued that neural networks mirror to a certain extent the behavior of networks of neurons in the brain.

Neural network approaches combine the complexity of some of the statistical techniques with the machine learning objective of imitating human intelligence. However, this is done at a more “unconscious” level and hence there is no accompanying ability to make learned concepts transparent to the user.

To formulate a new network, the first step is to design a specific network architecture that includes a specific number of layers each consisting of a certain number of neurons. The size and structure of the network needs to match the nature of the investigated phenomenon. Because the latter is obviously not known very well at this early stage, this task is not easy and often involves multiple trials and errors.

The new network is then subjected to the process of training. In that phase, neurons apply an iterative process to the number of inputs to adjust the weights of the network in order to optimally predict the sample data on which the training is performed. After the phase of learning from an existing data set, the new network is ready and it can then be used to generate predictions.

The resulting network developed in the process of learning represents a pattern detected in the data. Thus, in this approach, the "network" is the functional equivalent of a model of relations between variables in the traditional model building approach. However, unlike in traditional models, in the "network," those relations cannot be articulated in the usual terms used in statistics or methodology to describe relations between variables. Some neural networks can produce highly accurate predictions; they represent, however, a typical a-theoretical (one can say, "a black box") research approach. That approach is concerned only with practical considerations, that is, with the predictive validity of the solution and its applied relevance and not with the nature of the underlying mechanism or its relevance for any theory of the underlying phenomena.

One of the major advantages of neural networks is that, theoretically, they are capable of approximating any continuous function, and thus the researcher does not need to have any hypotheses about the underlying model, or even to some extent, which variables matter. An important disadvantage, however, is that the final solution depends on the initial conditions of the network, and, as stated before, it is virtually impossible to "interpret" the solution in traditional, analytic terms, such as those used to build theories that explain phenomena.

For more information on neural networks, see (Hertz, et al., 1991), (Bhagat, 2005), (Bishop, 1995), (Duda, et al., 2001), and (Egmont-Petersen, et al., 2002).

3.1.2.1 Multilayer Perceptron Networks

The most widely used neural classifier today is the Multilayer Perceptron (MLP) network (Haykin, 1998) which has also been extensively analyzed and for which many learning algorithms have been developed. An MLP is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard linear perceptron in that it uses three or more layers of neurons with nonlinear activation functions, and is more powerful than the perceptron in that it can distinguish data that is not linearly separable, or separable by a hyperplane.

Learning occurs in the perceptron by changing connection weights (or synaptic weights) after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron. The complexity of the MLP network can be changed by varying the number of layers and the number of units in each layer. Given enough hidden units and enough data, it has been shown that MLPs can approximate virtually any function to any desired accuracy. MLPs are valuable tools in problems when one has little or no knowledge about the form of the relationship between input vectors and their corresponding outputs.

3.1.2.2 Radical Basis Function Networks

Radical Basis Functions (RBF) are powerful techniques for interpolation in multidimensional spaces. An RBF is a function which has built into a distance criterion with respect to a center. RBF networks have two layers of processing: In the first, input is mapped onto each RBF in the hidden layer. The RBF chosen is usually a Gaussian. In classification problems, the output layer is typically a sigmoid function of a linear combination of hidden layer values, representing a posterior probability. Performance is often improved by shrinkage techniques, known as ridge regression in classical statistics and known to correspond to a prior belief in small parameter values (and therefore smooth output functions) in a Bayesian framework (Buhmann, 2003) (Yee, 2001).

RBF networks have the advantage of not suffering from local minima in the same way as MLP. This is because the only parameters that are adjusted in the learning process are the linear mapping from the hidden layer to the output layer. Linearity ensures that the error surface is quadratic and therefore has a single easily found minimum. In classification problems the fixed non-linearity introduced by the sigmoid output function is most efficiently dealt with using iteratively re-weighted least squares.

RBF networks have the disadvantage of requiring good coverage of the input space by radical basis functions. RBF centers are determined with reference to the distribution of the input data, but without reference to the prediction task. As a result, representational resources may be wasted on areas of the input space that are irrelevant to the learning task. A common solution is to associate each data point with its own center, although this can make the linear system to be solved in the final layer rather large, and requires shrinkage techniques to avoid overfitting.

3.1.3 Machine Learning Methods

As a broad class of techniques can come under this heading, this thesis will mainly introduce two types of machine learning methods. These are decision tree methods and rule induction methods. For a thorough understanding, refer to (Weiss, et al., 1991), (Alpaydın, 2004), (Bishop, 2007), and (Mitchell, 1997).

3.1.3.1 Decision Tree Methods

The problem of constructing a decision tree can be expressed recursively. First, select an attribute to place at the root node and make one branch for each possible value. This splits up the example set into subsets, one for every value of the attribute. Now the process can be repeated recursively for each branch, using only those instances that actually reach the branch. If at any time all instances at a node have the same classification, stop developing that part of the tree.

Decision tree induction is a nonparametric approach for building classification models. In other words, it does not require any prior assumptions regarding the type of probability pairs satisfied by the class and other attributes.

Finding an optimal decision tree is an NP-complete problem. Many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm might use a greedy, top-down, or a recursive partitioning strategy for growing a decision tree.

Decision trees, especially smaller-sized trees, are relatively easy to interpret. The accuracy of the trees is also comparable to other classification techniques for many simple data sets. The presence of redundant attributes, i.e., attributes that are strongly correlated with another attribute, does not adversely affect the accuracy of decision trees. One of the two redundant attributes will not be used for splitting once the other attribute has been chosen. However, if the data set contains many irrelevant attributes, i.e., attributes that are not useful for the classification task, then some of the irrelevant attributes may be accidently chosen during the tree growing process, which results in a decision tree that is larger than necessary. Feature selection techniques can help to improve the accuracy of decision trees by eliminating the irrelevant attributes during preprocessing.

Important Decision Tree algorithms include ADTree (Freund, et al., 1999), BFTree (Friedman, et al., 2000) (Shi, 2007), Id3 (Quinlan, 1986), J48 (Quinlan, 1993), J48graft (Webb, 1999), LMT (Landwehr, et al., 2005) (Sumner, et al., 2005), NBTree (Kohavi, 1996), Random Forests (Breiman, 2001), and Simple Cart (Breiman, et al., 1984).

3.1.3.2 Rule Induction Methods

Rule induction considers each class in turn and seeks a way for covering all instances in it, at the same time excluding instances not in the class. This is called a *covering* approach because at each stage one identifies a rule that "covers" some of the instances. Covering algorithms operate by adding tests to the rule that is under construction, always trying to create a rule with maximum accuracy. While a decision tree algorithm chooses an attribute to maximize the separation between the classes (using some information gain criterion), the covering algorithm chooses an attribute-value pair to maximize the probability of the desired classification.

This kind of learning has been examined intensively. Here is just a few references, (Kovalerchuk, et al., 1995), (Kovalerchuk, et al., 1996), (Nieto Sanchez, et al., 2001), (Triantaphyllou, et al., 1996a), and (Triantaphyllou, et al., 1996b).

Many rule induction methods can achieve high accuracy. The disadvantage is that most of them do not have the ability to adjust to the change of misclassification cost.

Important rule induction algorithms include the Decision Table Method (Kohavi, 1995), JRip (Cohen, 1995), NNge (Martin, 1995) (Roy, 2002), OneR (Holte, 1993) PART (Frank, et al., 1998), Prism (Cendrowska, 1987), Ridor (Gaines, et al., 1995), and a B&B approach (Triantaphyllou, 1994).

3.2 Weka: A Library of Data Mining Algorithms

Weka, short for Waikato Environment for Knowledge Analysis, is a collection of data mining algorithms and data preprocessing tools. It includes virtually all the algorithms described above. It is designed so that one can quickly try out existing methods on new datasets in flexible ways. It provides extensive support for the whole process of experimental data mining, including preparing the input data, evaluating learning schemes statistically, and visualizing the input data and the result of learning. Besides a wide variety of learning algorithms, it includes a wide range of preprocessing tools. This diverse and comprehensive toolkit is accessed through a common interface so that its users can compare different methods and identify those that are most appropriate for the problem at hand. A more detailed guide of using weka can be found in (Kaufmann, 2005).

Most algorithms we described above are implemented in Weka. In this thesis, the author will compare a newly developed algorithm with all the other applicable classification algorithms provided in Weka. A Weka 3.5.7 version will be used in this thesis.

CHAPTER 4. FORMAL PROBLEM DESCRIPTION

4.1 Binary System and Partially Ordered Set

4.1.1 Partially Ordered Set (Poset)

In general, a partially ordered set, or poset, consists of a set S together with a partial order, such as "greater than or equal to" ($\succcurlyeq$) defined on the set. The partial order "$\succcurlyeq$" should satisfy the following conditions,

> Definition 4.1 General Definition of the Non-strict Partial Order "$\succcurlyeq$":
>
> For any a, b, and c in set S, we have that:
> $a \succcurlyeq a$ (reflexivity);
> if $a \succcurlyeq b$ and $b \succcurlyeq a$ then $a = b$ (anti-symmetry);
> if $a \succcurlyeq b$ and $b \succcurlyeq c$ then $a \succcurlyeq c$ (transitivity property).
> Then, "$\succcurlyeq$" is a non-strict partial order; the set S combined with "$\succcurlyeq$" is a partially ordered set.

Sometimes, for convenience, we will also use the expression $b \preccurlyeq a$ (a, $b \in S$), which reads "b is less than or equal to a". This means exactly the same as $a \succcurlyeq b$ (a, $b \in S$). The two expressions can be seen as a pair of opposite relations, just as their analogous arithmetic operations "$\leq$" and "$\geq$".

The partial order defined above is called a non-strict partial order. In some other occasions, we need to use the strict partial order "greater than" ($\succ$), which is defined as,

> Definition 4.2 General Definition of the Strict Partial Order "$\succ$":
>
> For any a, b, and c in P, we have that:
> $\neg(a \succ a)$ (irreflexivity);
> if $a \succ b$ then $\neg(b \succ a)$ (asymmetry);
> if $a \succ b$ and $b \succ c$ then $a \succ c$ (transitivity).
> Then, "$\succ$" is a strict partial order; when the set S combined with "$\succ$" is a partially ordered set.

Similarly, we sometimes use its opposite relation "less than" ($\prec$) at our convenience.

There is a one-to-one correspondence between all non-strict and strict partial orders. This means if one of the two relations is defined, the other is determined too.

> Definition 4.3 An Alternative Definition of the Strict Partial Order "$\succ$" (Based on the Non-Strict Partial Order "$\succcurlyeq$"):
>
> If "$\succcurlyeq$" is a non-strict partial order, then its corresponding strict partial order "$\succ$" is,
>
> $$a \succ b, \quad \text{iff} \quad a \succcurlyeq b \text{ and } a \neq b.$$

On the other hand,

> Definition 4.4 An Alternative Definition of the Non-Strict Partial Order "$\succcurlyeq$" (Based on the Strict Partial Order "$\succ$"):
>
> If "$\succ$" is a strict partial order, then its corresponding non-strict partial order "$\succcurlyeq$" is,
>
> $$a \succcurlyeq b, \quad \text{iff} \quad a \succ b \text{ or } a = b.$$

A poset can often be visualized through a directed Hasse diagram. For example,

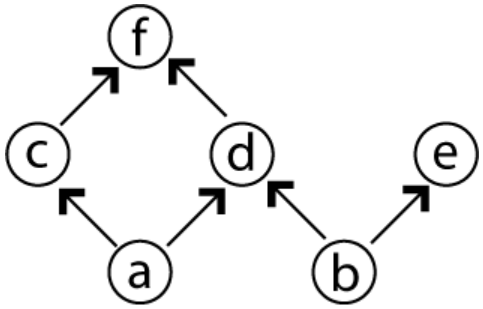

Figure 4.1 An Example of a Poset.

In the above diagram, each vertex corresponds to an example and each directed edge represents a partial relation. For instance, the directed edge (a, c) denotes $a \succ c$ or $c \prec a$. Partial relations that are transitively implied by other relations are not shown in the diagram. For example, there is no directed edge between a and f, but their relation $a \succ f$ is implied by relations $a \succ c$ and $c \succ f$.

4.1.2 Poset Expression of Binary Systems

A partial order "$\succcurlyeq$" on binary systems is defined as follows,

> Definition 4.5 Definition of the Non-strict Partial Order "$\succcurlyeq$" on Binary Systems:
>
> Let $v_1 = (x_{11}, x_{12}, x_{13}, \ldots, x_{1n})$, and $v_2 = (x_{21}, x_{22}, x_{23}, \ldots, x_{2n})$. Then we say
>
> $$v_1 \succcurlyeq v_2, \text{ iff } \quad x_{1i} \geq x_{2i} \ \forall \ i = 1, 2, 3, \ldots, n.$$

It can easily be proved that the relation defined above satisfies all three conditions mentioned by the general definition of a non-strict partial order in Definition 4.2.

Accordingly, a strict partial order "$\succ$" on binary systems is defined as follows,

> Definition 4.6 Definition of a Strict Partial Order "$\succ$" on Binary Systems:
>
> For two examples v_1 and v_2 of the same dimension,
>
> $$v_1 \succ v_2, \quad \text{iff} \quad v_1 \succcurlyeq v_2 \text{ and } v_1 \neq v_2.$$

Note that, unlike the general case of a strict partial relation "$\succ$" that has two sets of definitions, the strict partial relation "$\succ$" on binary systems does not have an alternative definition like "For $v_1 = (x_{11}, x_{12}, x_{13}, \ldots, x_{1n})$, and $v_2 = (x_{21}, x_{22}, x_{23}, \ldots, x_{2n})$, we say $v_1 \succ v_2$ iff $x_{1i} > x_{2i}$, $\forall\, i = 1, 2, 3, \ldots, n$." A simple example is that if $v_1 = <111>$, and $v_2 = <110>$, then v_1 and v_2 satisfy Definition 4.6, therefore $v_1 \succ v_2$. However, v_1 and v_2 does not satisfy the above quoted statement.

As mentioned before, there exist two opposite orders "$\preccurlyeq$", "$\prec$" to each of the above, respectively. For instance, <1110> is greater than or equal to examples <1110>, <1100>, and <0010>. On the other hand <0100> is less than or equal to examples <0100>, <1100>, and <0111>.

A binary system when it is combined with a partial order forms a poset. We say it is a poset on $\{1, 0\}^n$, which can be depicted by a diagram shaped like a diamond. Figure 4.2 shows the poset diagram when $n = 4$.

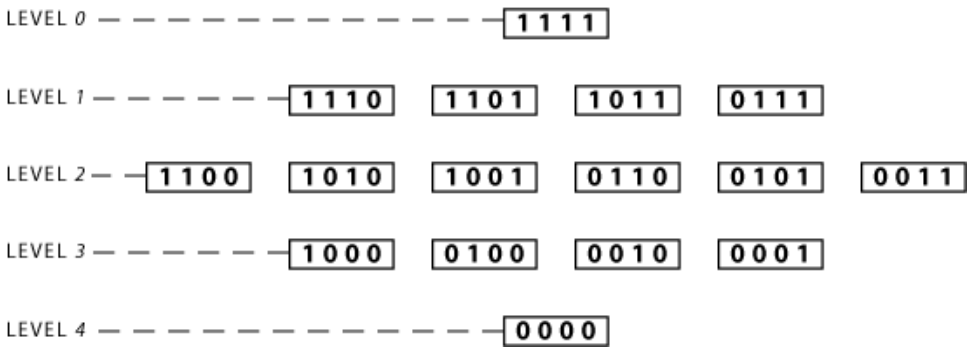

Figure 4.2 A Poset on $\{1, 0\}^4$.

Because of the special characteristics of the set $\{1, 0\}^n$, we organize the poset in different levels. Two examples are at the same level if they have the same number of 1s, and hence, the same number of 0s. A binary system of dimension n, or a poset on $\{1, 0\}^n$, have n+1 levels. From top to bottom, we call them level 0, level 1,…, and up to level n+1, according to the number of 0s in the examples of that level. Note that $v_1 \succ v_2$ only if v_1 is at some level above v_2.

Since the partial relations between two examples can be specifically decided by the partial order definitions, we do not draw those directed edges as in Figure 4.1 anymore. Suppose we are provided with some classified examples. Then we color an example blue if it is positive, red if it is negative. Thus, we can have the following diagram:

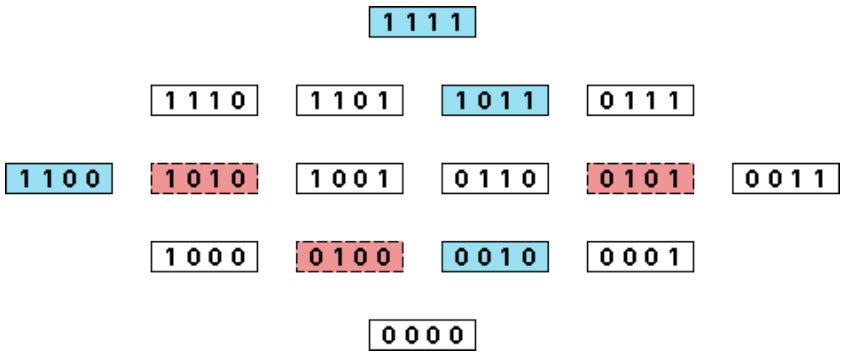

Figure 4.3 A Poset on $\{1, 0\}^4$ with some hypothetically classified examples.

While the colored examples are classified, the blank examples represent unclassified ones. From now on, we will use this kind of poset diagrams to illustrate the binary systems we have. Since we assume we are always provided with some classified examples, we will omit the part "with some classified examples" in the caption, and simply call them posets on $\{1, 0\}^n$.

4.2 Binary Systems and the Monotonicity Property

4.2.1 The Monotonicity Property

In general, the definition of the monotonicity property is given as follows:

Definition 4.8 General Definition of the Monotonicity Property:

For a function $f: S \to \mathbb{R}$ defined on a poset S,

if for any $v_1, v_2 \in S$, we have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \geq f(v_2)$

then we say function f is increasingly monotone on S.

If for any $v_1, v_2 \in S$, we have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \leq f(v_2)$

then we say function f is decreasingly monotone on S.

In both cases function f is called monotone on S, or equivalently, function f has the monotonicity property on S, and the set S is called a monotone set.

As stated in the book (Triantaphllou, 2008), the monotonicity property is "inherently frequent in applications", "simple and intuitive", and "can represent relatively complex knowledge and still be validated". Throughout this thesis, we will be mainly exploring this property on binary systems.

4.2.2 The Monotonicity Property of Binary Systems

Since we already have a partial relation "$\succcurlyeq$" defined on $\{1,0\}^n$, we can now define the monotonicity property on binary systems. Making a small revision to Definition 4.8, we get a definition as follows:

Definition 4.9 Definition of the Monotonicity Property on Binary Systems:

For a function $f: \{1,0\}^n \rightarrow \{1,0\}$ defined on a binary system *B*,

if for any $v_1, v_2 \in \{1,0\}^n$, we have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \geq f(v_2)$

then we say function *f* is increasingly monotone on $\{1,0\}^n$.

If for any $v_1, v_2 \in \{1,0\}^n$, we have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \leq f(v_2)$

then we say function *f* is decreasingly monotone on $\{1,0\}^n$.

In both cases function *f* is called monotone on $\{1,0\}^n$, or equivalently, function *f* has the monotonicity property on $\{1,0\}^n$, and binary system *B* is called a monotone binary system.

Definition 4.9 is not only the definition of the monotonicity property on binary systems, but also the definition of monotone binary systems. We already know that, if a Boolean function can be transformed into a CNF/DNF that has no negated variables, then this Boolean function is a monotone function (Koppelberg, 1989). The following is an example of a Boolean monotone function,

$$f = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$$

If a monotone function is defined on a binary system $\{1,0\}^n$, then the binary system is a monotone binary system. For instance, when all the classes of all the examples are known, a binary system $\{1,0\}^4$ with the above monotone function has a poset diagram as follows:

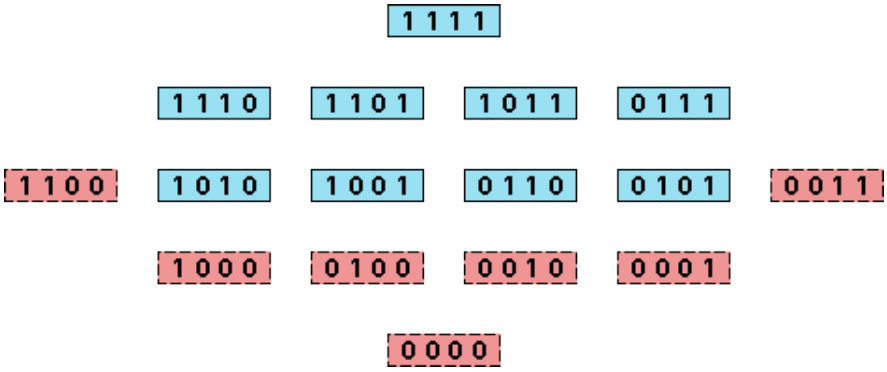

Figure 4.4 A Poset on $\{1,0\}^4$ with monotone function $f = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$.

It can be seen from this diagram, that the entire binary system can be separated into two parts. One part is all the examples which are less than or equal to <1111> and greater than or equal to either one of <1010>, <1001>, <0110>, and <0101>. The other part is all the examples which are less than or equal to either one of <1100> and <0011>, and greater than or equal to <0000>. The first part is all examples of class "1", while the second part is all examples of class "0". In fact, all monotone binary systems can be separated in such a way into two parts, where each part is of the same class.

So far we have defined monotone binary systems. But what happens with non-monotone binary systems? Let us look at the following real function as an analogy.

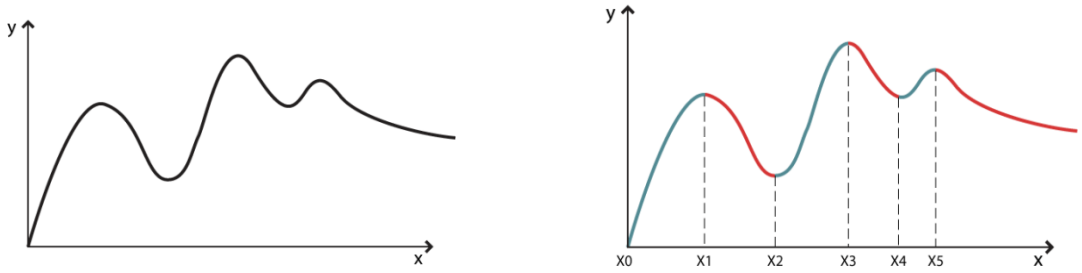

Figure 4.5 The Partial Monotonicity Property on a Non-Monotone Real Function.

The left function is apparently non-monotone, because *y* goes up and down with *x* increasing. However on the right, the same function appears to be monotone, or has the monotonicity property, on sub-intervals $[x_0, x_1]$, $[x_1, x_2]$, $[x_2, x_3]$, and so on. This observation inspires us that even if the monotonicity property cannot be found on the whole system, we can always break a system into several sub-systems that are monotone. Based on this thought, we can define the monotonicity property on the subsets of non-monotone binary systems.

Definition 4.10 Definition of the Monotonicity Property on Subsets of a Non-Monotone Binary System:

For a function f: $\{1,0\}^n \rightarrow \{1,0\}$ defined on a binary system B, if there exists a subset S of B such that,

for any $v_1, v_2 \in S$ have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \geq f(v_2)$

or, for any $v_1, v_2 \in S$ have $v_1 \succcurlyeq v_2 \Longrightarrow f(v_1) \leq f(v_2)$

Then we say function f is monotone on subset S, or equivalently, function f has the monotonicity property on subset S.

4.2.3 Some Important Implications of the Monotonicity Property

- Any binary system can be entirely separated into a certain number of monotone subsets, though the monotone subsets could be very small.

- In a monotone subset of a binary system, for a certain unclassified example *z*, if the examples that are greater than and less than *z* are all positive, then *z* is positive; if the examples that are greater than and less than z are all negative, then *z* is negative.

4.3 A Weighted Voting System

A weighted voting system contains a series of rules for valid voting. It allows voters, the classified examples in our case, to vote between two classes with regard to a target

example. Based on the results of voting, we can calculate the evidence of the unclassified example belonging to each class, and hence, the probabilities the unclassified example belongs to each class.

The most important two questions in deciding the voting rules are,

- "Which example gets to vote?"
- "What are the weights of each vote?"

We will answer the above two questions through the process for developing the new classifier.

4.4 The Poset Description of the Illustrative Problem

For the convenience of future discussion, there is a need to re-describe the illustrative problem mentioned in Chapter 2 in terms of a poset diagram.

Recall that the illustrative problem has dimension $n = 6$, a total of $2^6 = 64$ examples, 20 classified positive examples, 12 classified negative examples, and 32 unclassified examples. As before, the blue colored examples are classified positive, the red colored examples are classified negative, and the blank examples are the unclassified ones. To distinguish classified and unclassified examples, we will not color the unclassified examples in the diagram. Therefore, the poset diagram for this illustrative problem is as follows:

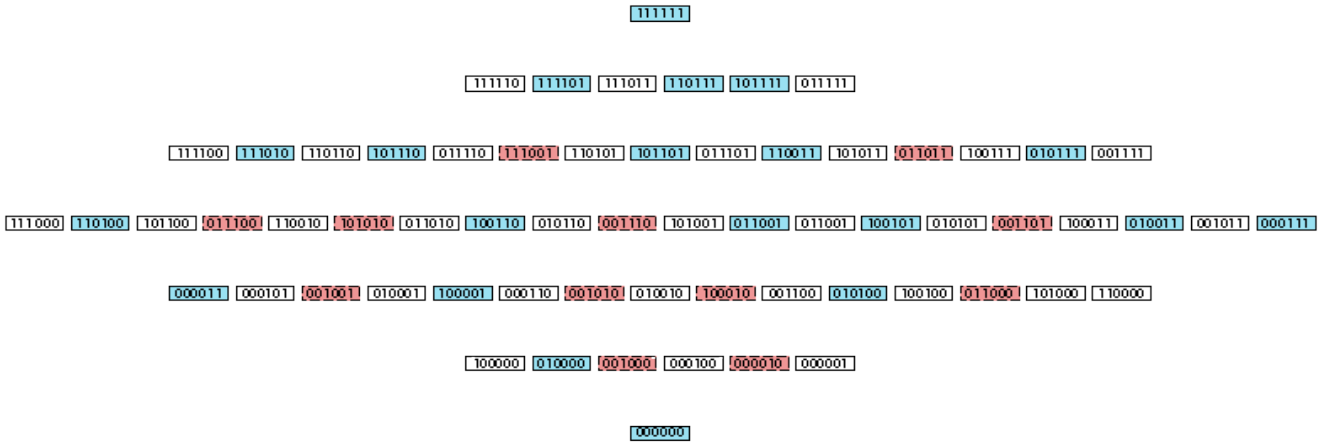

Figure 4.6 The Poset Diagram of the Illustrative Problem.

To learn the classification of the unclassified examples, we will refer to the hidden function

$$f = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_4 \vee \overline{x_5} \vee x_6).$$

CHAPTER 5. PROPOSED METHODOLOGY

5.1 Different Types of Examples

To answer the first important question in building a voting system, "which example gets to vote?", we will need to define some different types of examples with regard to the example we are trying to classify, also to be referred as the target example *z*. We will use the illustrative problem to describe these different types of examples.

5.1.1 The Ordered Relation

Suppose the first unclassified example <111110> is the target example *z*. We can take *z*, as well as all the examples that are greater than or less than *z*, from the original poset diagram, and arrange them into a different diagram as follows:

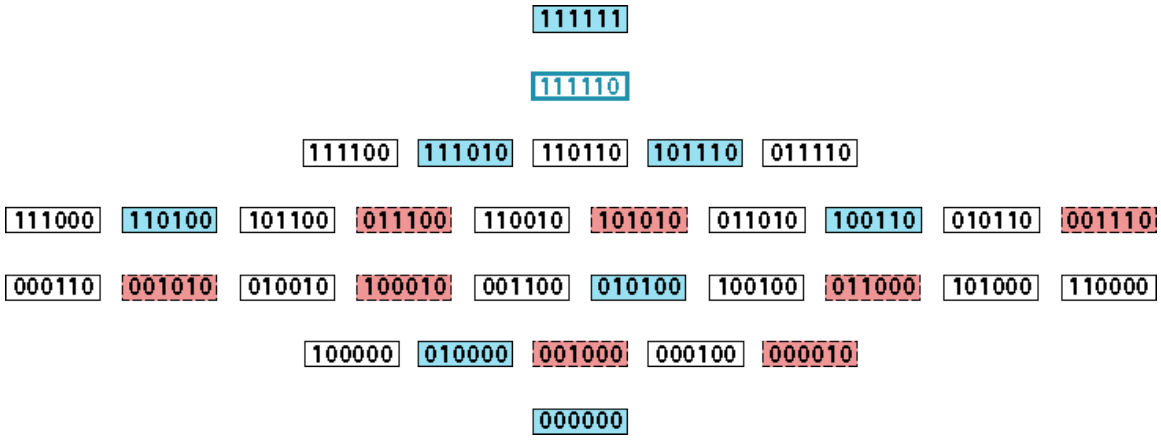

Figure 5.1 All the Examples in Ordered Relation with <111110>.

To distinguish the target example *z*, we use a colored frame. Here a blue frame denotes that this example is the target example and its actual class should be "1". Similarly, if we have a red framed example, it denotes that this example is the target example and its actual class should be "0".

The rest of the examples in Figure 5.1 are the ones that are either greater than or less than the target example *z*, or as we will call them from now one, they are in an ordered relation with *z*.

> Definition 5.1 Definition of an Ordered Relation:
>
> In a binary system, for any two examples *x* and *y*, if
>
> $$x \succ y \quad \text{or} \quad x \prec y$$
>
> then we say *x* and *y* are in an ordered relation.

A useful concept related to the ordered relation is the concept of a path. Imagine there were invisible edges between adjacent levels depicting the relation of "$\succ$" (or "$\prec$"). Then there must exist one or more path(s), i.e., a sequence of examples, between two examples that are in ordered relation.

> Definition 5.2 Definition of a Path:
>
> In a binary system, if $v_1, v_2, v_3, \ldots, v_k$ is a sequence of examples such that,
> v_i is one level above v_{i+1}, and $v_i \succ v_{i+1}$, $\forall\, i = 1, 2, \ldots, k-1$
> then we say $v_1, v_2, v_3, \ldots, v_k$ is a path between v_1 and v_k, and the path has length $k-1$.

Note that, whenever two examples are in an ordered relation, there must exist at least one path between them. Moreover, for any two ordered examples, every path(s) between them must have the same length. This is true because every example in a path is one level above its previous example. Therefore the length of every path between v_1 and v_k is equal to the difference of the levels of v_1 and v_k.

> Definition 5.3 Definition of Distance of Two Ordered Examples:
>
> In a binary system, if example x is at level i, y is at level j, and $x \succ y$, then the distance d of x and y is the length of the path(s) between x and y, or
> $d = |i - j|$.

5.1.2 Direct and Indirect Relations

When we take a closer look at pairs of examples in an ordered relation, we realize that some of them may have a different property than the others. For instance, in Figure 5.1, let us consider the classified example <100110> and the target example <111110>. There are two paths of length 2 between the two examples, namely,

path 1: 100110 - 110110 - 111110
path 2: 100110 - 101110 - 111110

Notice that both paths contain only positive examples and unclassified examples (the target example is considered to be an unclassified example).

Next, we will check examples <010100> and the target example <111110>. There are six paths of length 3 between the two, namely,

path 1: 010100 - 110100 - 111100 - 111110
path 2: 010100 - 110100 - 110110 - 111110
path 3: 010100 - 011100 - 111100 - 111110
path 4: 010100 - 011100 - 011110 - 111110

path 5: 010100 - 010110 - 110110 - 111110

path 6: 010100 - 010110 - 011110 - 111110

Intuitively, unlike the previous case, we find that there are some red examples present in the paths. In other words, there exist some paths which contain not only positive examples and unclassified examples but also negative examples.

We will call that example <100110> in the former case is in direct relation with the target example *z*; and example <010100> in the latter case is in indirect relation with the target example *z*. Moreover, the negative example <011100> in the paths of the latter case is called a *cutting* example.

> Definition 5.4 Definition of a Direct Relation:
>
> For a classified example *v* and a target example *z*, if every path connecting *v* and *z* contains only examples of the same class of *v* and unclassified examples, then *v* is in direct relation with *z*.

> Definition 5.5 Definition of an Indirect Relation and a Cutting Example:
>
> For a classified example *v* and a target example *z*, if there exists at least one path connecting *v* and *z* which contains example(s) of the opposite class of *v*, then *v* is in indirect relation with *z*. The example(s) of the opposite class is/are called cutting example(s).

Therefore, according to the previous definition, the negative example <011100> is in direct relation with *z*. This is the case because every path between <011100> and *z* contains only negative examples and unclassified examples. The negative example <101010> is in indirect relation with *z*, since there are two cutting examples of the opposite class of <101010>, namely positive examples <111010> and <101110>.

Note that we do not need to find out all the cutting examples. For a classified example *v*, as soon as we find one cutting example in any path between *v* and the target example *z*, we can decide that *v* is in indirect relation with *z*.

Another notion is that both classified and unclassified examples can be considered in some ordered relation with the target example. However, examples in direct/indirect relations with the target example can only be classified examples. It is meaningless to talk about unclassified examples in terms of direct/indirect relations.

5.1.3 Examples at the Same Level

The examples that are at the same level of the target example *z* clearly cannot be in ordered relation with *z*, and hence not in direct or indirect relation with *z*. Even though this thesis attempts to build a voting system based on the monotonicity property, hence the ordered relation, we still want to discuss the possibility that examples at the same level of *z* might have an impact on deciding the class of *z*. Therefore, these examples will form another type of relation with the target example. For instance, if <111110> is the target example, then the examples at the same level include <111101>, <111011>, <110111>, <101111>, and <011111>. All these examples have the same number of 1's and 0's.

5.1.4 The Rest of the Examples

With regard to a certain target example *z*, there could still be some examples that do not fall in any of the above categories. These are examples neither in ordered relations with *z*, nor at the same level of *z*. All these examples will form another type of relation, termed "the rest of the examples". For instance, if <111110> is the target example, then examples <101011>, <000111>, and <100001> are in the group of "the rest of examples".

We have described several types of relations with the target example *z*. Once we have specified *z*, we can categorize all the classified examples into different types.

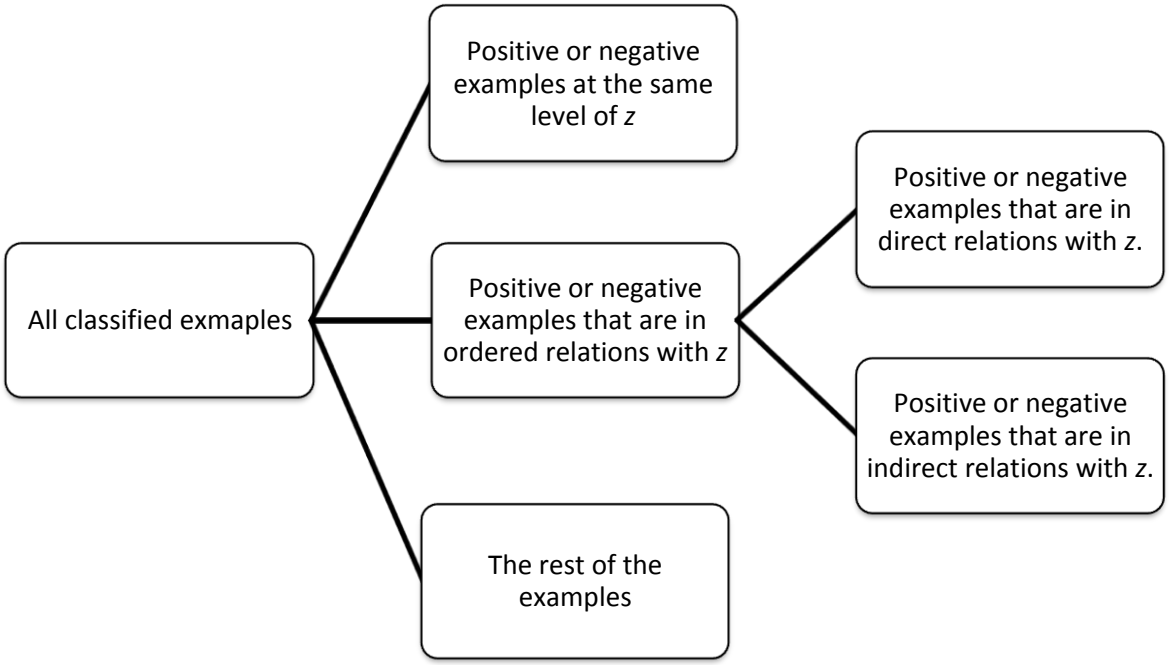

Figure 5.2 Different Types of Relations with the Target Example.

The above figure can also be shown in a poset diagram. In Figure 5.3 shown below, the space between the dashed straight lines depicts all the examples at the same level as *z*.

The space within the dashed contour depicts all the examples in ordered relations with *z*. Ordinarily, all these ordered examples can be arranged in a shape of a sandglass, or two diamonds, as shown in the following figure. The arrows in the contour denote a path, or equivalently, an ordered relation.

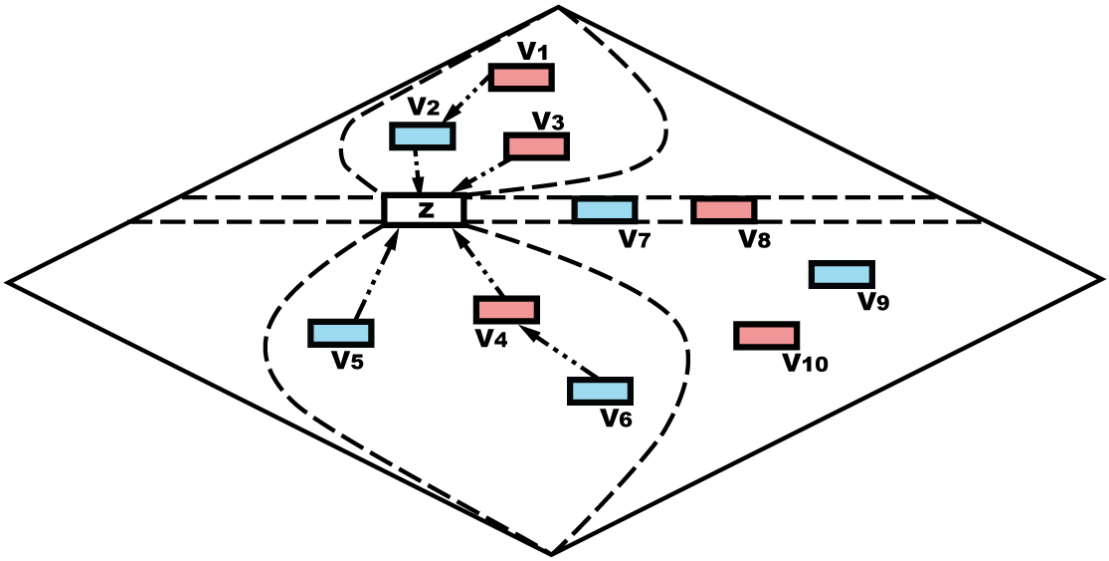

Figure 5.3 Different Types of Examples in a Poset Diagram.

From Figure 5.3, we can see that

v_1, v_2, v_3, v_4, v_5, v_6 are in ordered relations with *z*.
v_2, v_3, v_4, v_5 are in direct relations with *z*.
v_1 is in an indirect relation with *z*, with v_2 as one of the cutting examples.
v_6 is also in an indirect relation with *z*, with v_4 as one of the cutting examples.
v_7, v_8 are at the same level as *z*.
v_9, v_{10} are in the group called "the rest of the examples".

5.2 Twenty-Four Weighting Schemes

Now that we have identified different types of relations with the target example, we will proceed to solve the second important question in building a voting system. That is, "what are the appropriate weights for each vote?" To be more specific, we have four types of examples: examples in direct relations, examples in indirect relations, examples at the same level, and the rest of the examples. We will examine these four types of examples one by one, and discuss some key options in developing weighting schemes for each type.

5.2.1 Weights for Examples in Direct Relations

This is the main type of examples and may have the most impact on determining the class of the target example *z*. These are the examples that are either greater than or less than *z*,

with no examples of the other class in the paths leading to *z*. As shown in Figure 5.4, examples v_1 to v_7 are all in a direct relation with *z*.

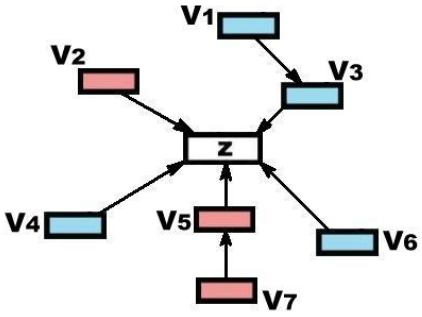

Figure 5.4 Examples in Direct Relation with the target example *z*.

The assumption here is that the target example *z* might be in a small subset of the binary system which exhibits the monotonicity property locally. Therefore, if the examples that are greater than and less than *z* are all positive, then *z* is most probably positive. If the examples that are greater than or less than z are all negative, then *z* is most probably negative. In practice, often times exist both positive and negative examples in direct relations with *z*. Therefore, we need to weigh the evidence of *z* belonging to each class.

Intuitively, we assume the closer an example is to the target example *z*, the more likely is for *z* to belong to the same class as that example. This means that an example at the most adjacent level of *z* should have the most weight of its vote. The farther level an example is from *z*, the less weight the example's vote should have. This implies that the weights we are looking for should be a function of the distance between a voting example and *z*.

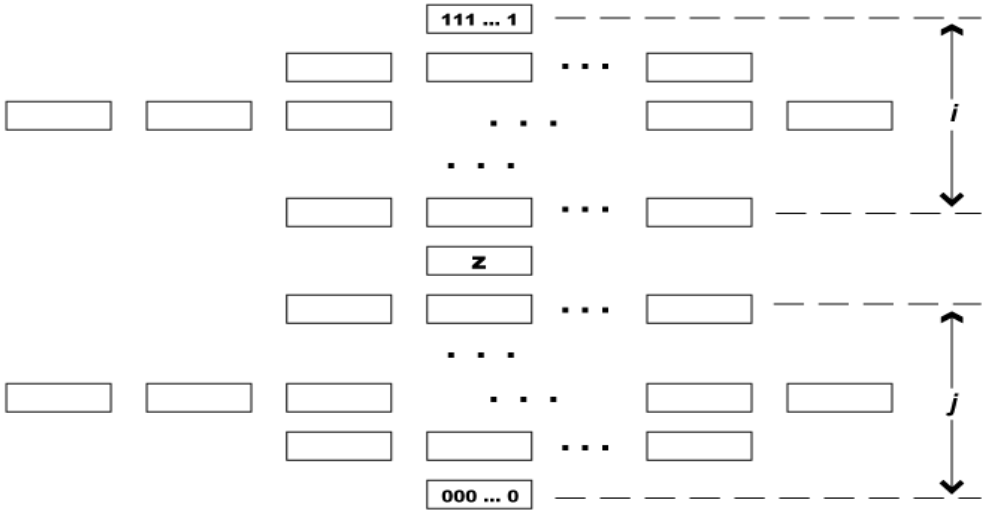

Figure 5.5 Examples in Ordered Relations with the target example *z*.

Based on the above criterion, we need to find the relation between the weights and the distances. Figure 5.5 shows all the examples in ordered relations with *z*. They all can be in direct relation or indirect relation with *z*. Thus we will later assign weights to these examples based on the layout of this diagram.

We have two key options on the way the weights will change with regard to the distance. The first key option is that the weights change linearly. We will let the examples at the farthest level have weight of 1, and then the weights of examples at the next closer level increase by one and so on. Also we will make sure examples at the two levels that are above and below *z* have the same weights if the examples are of the same distance to *z*.

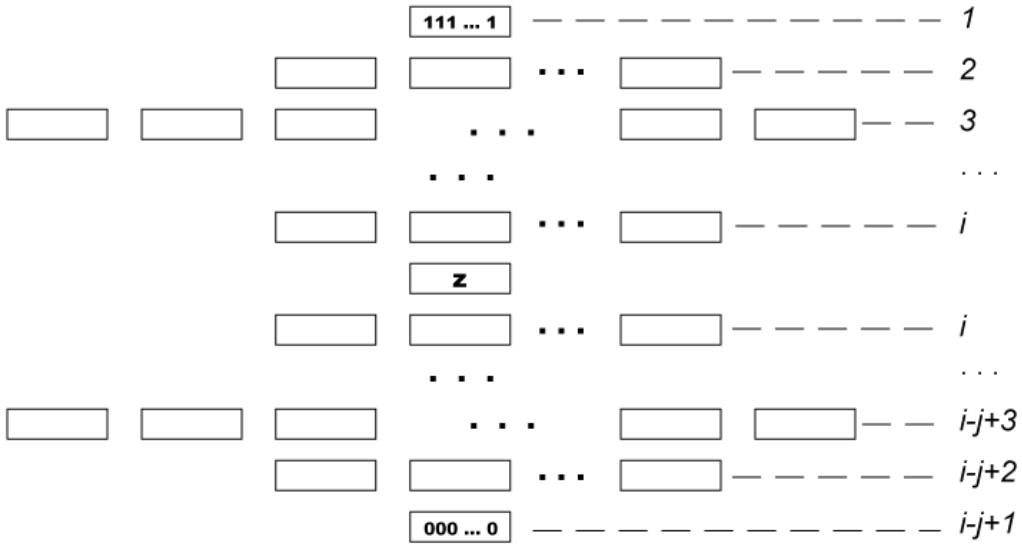

Figure 5.6-1 Weights of Examples in Ordered Relation Changing Linearly (when $i \geq j$).

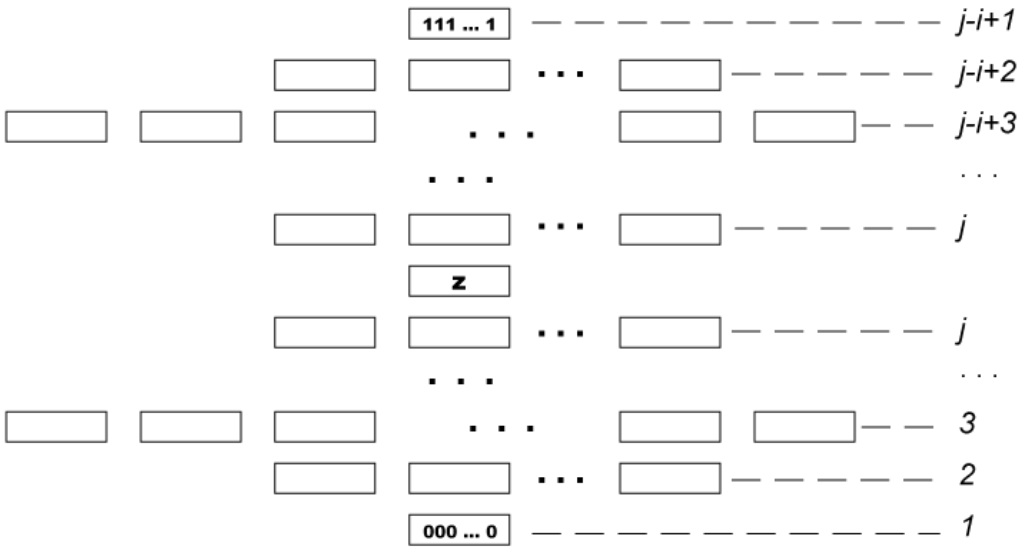

Figure 5.6-2 Weights of Examples in Ordered Relation Changing Linearly (when $i < j$).

In Figures 5.6-1 and 5.6-2, the weights of examples are shown on the right hand side to each level. Obviously, examples at the same level are assigned the same weight.

In the second option the weights will change exponentially. Again, we will let the examples at the farthest level have weights of $2^0 = 1$, and then the exponent of the weights of examples at the next farthest level will increase by one, i.e., 2^1, 2^2, 2^3, and so on. We will make the examples at the two levels that are above and below *z* to have the same weights if the examples are of the same distance to *z*.

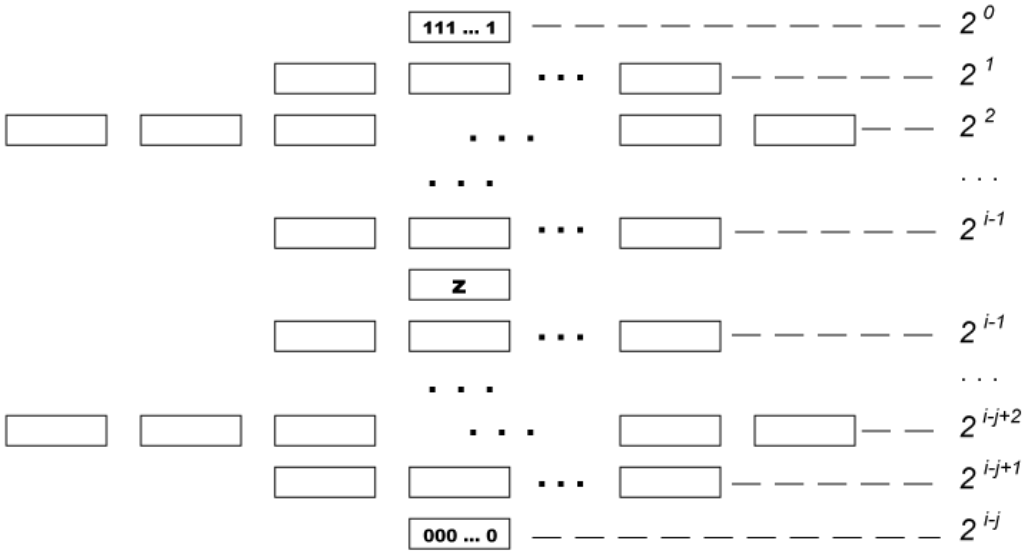

Figure 5.7-1 Weights of Examples in Direct Relation Changing Exponentially (when $i \geq j$).

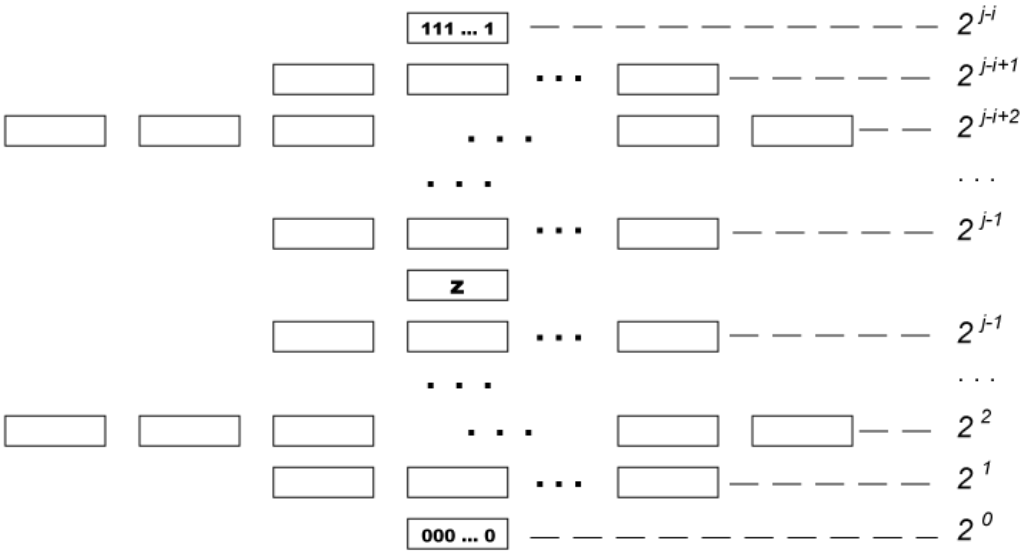

Figure 5.7-2 Weights of Examples in Direct Relation Changing Exponentially (when $i < j$).

In summary, there are two key options in calculating the weights of "voting" examples in direct relations with the target example *z*:

1) The weights change linearly with regard to the distance of a "voting" example to *z* (refer to Figures 5.6-1 and 5.6-2);
2) The weights change exponentially with regard to the distance of a "voting" example to *z* (refer to Figures 5.7-1 and 5.7-2).

5.2.2 Weights for Examples in Indirect Relations

Examples in indirect relations are the examples that are greater than or less than the target example *z*, but have one or more cutting example(s) of the opposite class in path(s) leading to *z*. As shown in Figure 5.8, v_1, v_2 and v_5 are all examples in indirect relations with *z*, and v_3 and v_4 are the corresponding cutting examples.

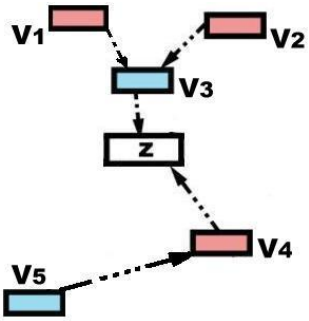

Figure 5.8 Examples in Indirect Relations with the target example *z*.

This type of examples may or may not have an impact on determining the class of the target example. Therefore, the first key option is for the weights of such examples to be equal to 0. In other words, these examples do not play any role in the voting system.

If these examples do play a role in the voting system, their weights will be decided in relation to the weight options of the direct relation type. This is because the importance of examples in indirect relation cannot be stronger than that of examples in direct relations.

Therefore, if the weights of examples in direct relations change linearly, the weights of examples in indirect relations cannot change exponentially. In this situation, the weights can change linearly or to be equal to some constant. If the weights of examples in indirect relations change linearly, then they will take the same weights as in Figures 5.6-1 and 5.6-2. If the weights are a constant number, we will use the constant 1 as their value.

If the weights of examples in direct relations change exponentially, then the weights of examples in indirect relations can change exponentially as well, or change linearly. Taking a constant number is also an option, but comparing to the increases of the exponential scheme, constant numbers are considered relatively insignificant. Therefore,

taking a constant number is not a key option of the weighting schemes to be explored. If the weights change linearly, the weights are the same as in Figures 5.6-1 and 5.6-2. If the weights change exponentially, the weights are the same as in Figures 5.7-1 and 5.7-2.

In summary, there are four key options for the weights of examples in indirect relations with the target example:

1) The weights are all equal to 0;
2) The weights are a constant number (i.e., all equal to 1), when the weights of example in direct relations change linearly;
3) The weights change linearly with regard to the distance of a "voting" example to *z*, when the weights of example in direct relations change linearly or exponentially (refer to Figures 5.6-1 and 5.6-2);
4) The weights change exponentially with regard to the distance of a "voting" example to *z*, when the weights of examples of direct relations change exponentially (refer to Figures 5.7-1 and 5.7-2).

5.2.3 Weights for Examples at the Same Level

The weights for examples at the same level might not have a role in the voting systems either, which is the first key option. If they have, however, the weights will depend on the weighting scheme of examples in direct relations. If the weights of examples in direct relations change exponentially or linearly, the weights of examples at the same level will also change exponentially or linearly, but with a different weighting scheme.

Since each level has a different number of examples, it is reasonable to think that the more examples *z* has at the same level, the less weight each example will have; while the less examples *z* has at the same level, the higher weight each example will have. Therefore, it is assumed that examples at the middle level have the lowest weight, and the other weights increase linearly or exponentially when we move farther from the middle.

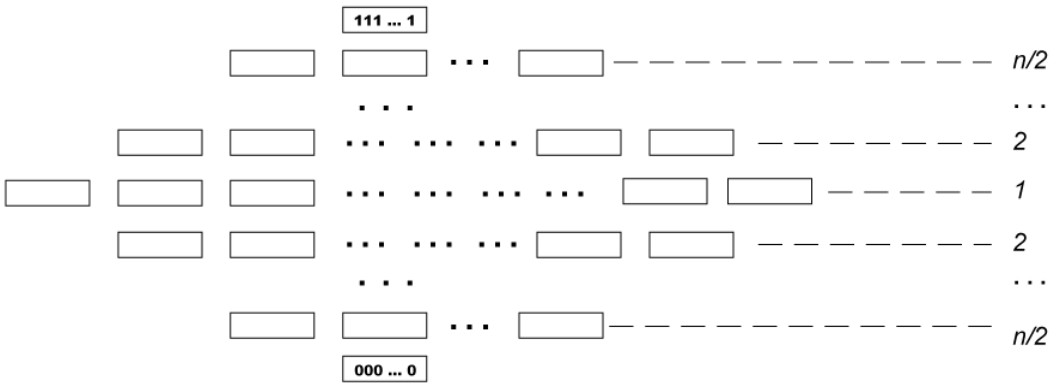

Figure 5.9-1 Weights of Examples at the Same Level Changing Linearly (when *n* is even).

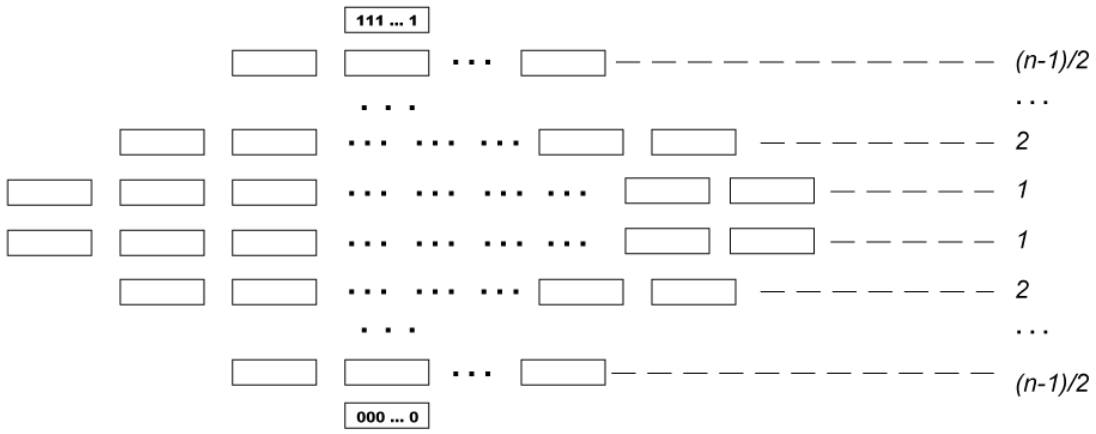

Figure 5.9-2 Weights of Examples at the Same Level Changing Linearly (when *n* is odd).

The previous figures show the positions that the target example might appear in, and hence the positions where the examples at the same level might appear in. The top and bottom level do not have a weight. This is because if the target example happens to be at these two levels, there will be no other examples at the same level.

Figures 5.9-1 and 5.9-2 show the second option of the weighting schemes of examples at the same level, having *n* being even and odd numbers respectively. Assume examples at the middle level have the lowest weights (=1), and increase linearly (by 1), when the distance to the middle goes up.

Similarly, we have a third option of the weighting schemes of examples at the same level. It is assumed that examples at the middle level have the lowest weights, i.e., $2^0 = 1$, and increase exponentially, i.e., the exponent increases by 1, when the distance to the middle goes up. Figures 5.10-1 and 5.10-2 show the third option. There are also two scenarios as *n* can be even or an odd number.

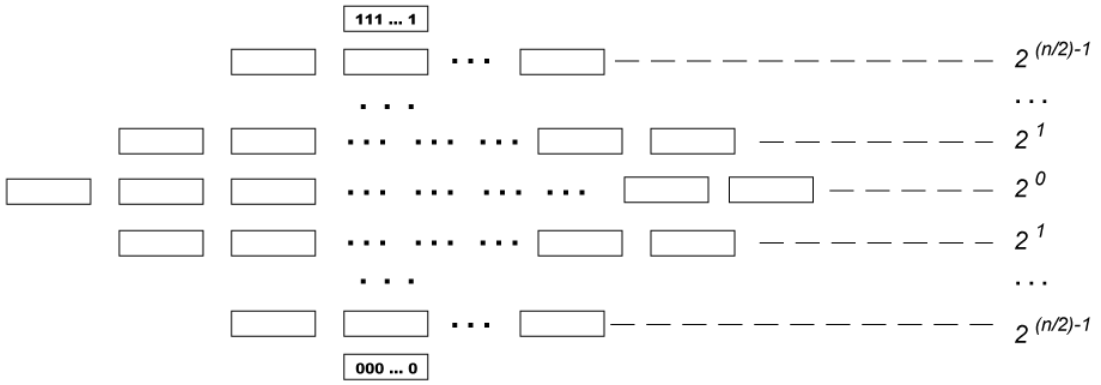

Figure 5.10-1 Weights of Examples at the Same Level Changing Exponentially (when *n* is even).

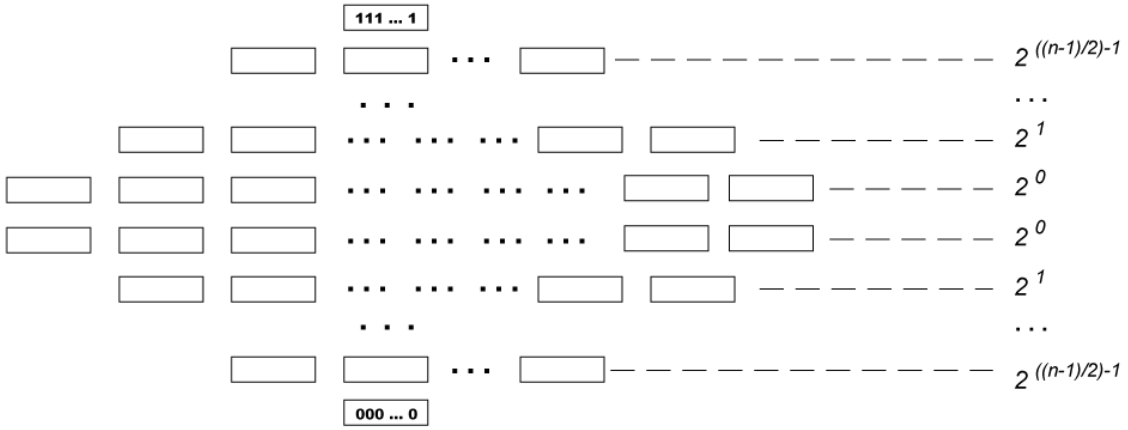

Figure 5.10-2 Weights of Examples at the Same Level Changing Exponentially (when *n* is odd).

In summary, there are three key options of the weights of examples at the same level as the target example *z*:

1) The weights are all equal to 0;
2) The weights change linearly with regard to the distance of an example to *z*, when weights of example of direct relation change linearly (refer to Figures 5.9-1 and 5.9-2);
3) The weights change exponentially with regard to the distance of an example to *z*, when weights of example of direct relation change exponentially (refer to Figures 5.10-1 and 5.10-2).

5.2.4 Weights for the Rest of the Examples

This is the easiest type to decide. In this thesis, we assume that the other examples have no or very little impact on determining the class of the target example. Therefore, the weights are 0 for the rest of the examples. In summary, there is only one key option of the weights of the rest of the examples:

1) The weights are equal to 0.

To put the key options for weights of the four types of examples together, we derive the following table:

Table 5.1 Key Options for Weights of the Different Types of Examples.

Type of Examples	Key Options for Weights
Direct Relations	1) The weights change linearly with regard to the distance of a "voting" example to *z* (refer to Figures 5.6-1 and 5.6-2); 2) The weights change exponentially with regard to the distance of a "voting" example to *z* (refer to Figures 5.7-1 and 5.7-2)

(table con'd)

Indirect Relations	1) The weights are equal to 0; 2) The weights are a constant number (i.e., all equal to 1), when the weights of examples in direct relations change linearly; 3) The weights change linearly with regard to the distance of a "voting" example to *z*, when the weights of examples in direct relations change linearly or exponentially (refer to Figures 5.6-1 and 5.6-2); 4) The weights change exponentially with regard to the distance of a "voting" example to *z*, when the weights of examples of direct relations change exponentially (refer to Figures 5.7-1 and 5.7-2).
Examples at the Same Level	1) The weights are equal to 0; 2) The weights change linearly with regard to the distance of an example to *z*, when weights of examples of direct relation change linearly (refer to Figures 5.9-1 and 5.9-2); 3) The weights change exponentially with regard to the distance of an example to *z*, when weights of examples of direct relation change exponentially (refer to Figures 5.10-1 and 5.10-2).
The Rest of the Examples	1) The weights are equal to 0.

Aside from dealing with the four types of examples, there is another possible action we might need to take. Imagine the situation that we are given 100 positive examples, and 10 negative examples. For a certain target example *z*, 20 positive examples and 10 negative examples participate in voting. Even though there are more positive examples voting, it is just 20% of all the positive examples. On the other hand, though the number of the voting negative examples is smaller, all of them have participated in the voting. Hence it is possible that the weights of the negative examples should be somewhat higher than that of the positive examples. Therefore we might want to multiply a factor to each vote. In such a way, we can modify on the weights of positive and negative votes.

Suppose that after calculating the votes of the four types of examples, we have (V_1, V_0), where:

$$V_1 = \sum \text{the weights of positive "voting" examples}$$

$$V_0 = \sum \text{the weights of negative "voting" examples}$$

We need to multiply some factors to V_1 and V_0, respectively.

There are two possibilities for the values of these factors. Each possibility also has two factors, one for modifying the positive vote, the other for modifying the negative vote. In the first possibility, these factors are equal to (1, 1), meaning we do not change the

calculated positive or negative votes. In the second possibility, these factors are equal to (F_1, F_0), where:

$$F_1 = \frac{\text{the number of participating positive examples}}{\text{the total number of all positive examples}}$$

$$F_0 = \frac{\text{the number of participating negative examples}}{\text{the total number of all negative examples}}$$

When we take the Cartesian product of (V_1, V_0) and the factors (1, 1) or (F_1, F_0), we will get the final positive and negative votes, denoted by PositiveVote and NegativeVote, respectively.

When combining the information in Table 5.1 and the option of multiplying by these factors, we get the following table of all possible key weighting schemes:

Table 5.2 Twenty-Four Weighting Schemes.

Scheme	Direct Relation	Indirect Relation	Examples on the Same Level	Factors
A	linear	0	0	(1, 1)
B	linear	0	0	(F_1, F_0)
C	linear	0	linear	(1, 1)
D	linear	0	linear	(F_1, F_0)
E	linear	1	0	(1, 1)
F	linear	1	0	(F_1, F_0)
G	linear	1	linear	(1, 1)
H	linear	1	linear	(F_1, F_0)
I	linear	linear	0	(1, 1)
J	linear	linear	0	(F_1, F_0)
K	linear	linear	linear	(1, 1)
L	linear	linear	linear	(F_1, F_0)
M	exponential	0	0	(1, 1)
N	exponential	0	0	(F_1, F_0)
O	exponential	0	exponential	(1, 1)
P	exponential	0	exponential	(F_1, F_0)
Q	exponential	linear	0	(1, 1)
R	exponential	linear	0	(F_1, F_0)
S	exponential	linear	exponential	(1, 1)
T	exponential	linear	exponential	(F_1, F_0)
U	exponential	exponential	0	(1, 1)
V	exponential	exponential	0	(F_1, F_0)
W	exponential	exponential	exponential	(1, 1)
X	exponential	exponential	exponential	(F_1, F_0)

Finally, the probability pair (p_1, p_0) is calculated by

$$p_1 = \frac{PositiveVote}{PositiveVote + NegativeVote}$$

$$p_0 = \frac{NegativeVote}{PositiveVote + NegativeVote}$$

where PositiveVote + NegativeVote $\neq 0$.

If PositiveVote + NegativeVote = 0, then (p_1, p_0) = (0.5, 0.5).

CHAPTER 6. COMPUTATIONAL EXPERIMENTS AND ANALYSIS OF THE RESULTS

According to the twenty-four weighting schemes described in the previous chapter, we will next test these weighting schemes on some synthetic datasets. Next we will try to determine which scheme gives the best result, and that scheme will lead to the final algorithm.

6.1 Data Preparation

Since there are not so many real binary datasets available in practice, we need to generate some synthetic datasets. Consequently, we want to make these datasets as random as possible. The following are the steps that are taken to prepare unbiased random datasets.

For the first step we use a Random Number Generator (RNG) developed by (Devilly, 2004) to generate random decimal numbers. When n is given, this generator can provide us with a certain amount, say m, $1 \leq m \leq 2^n$, of decimal numbers within the range $[0, 2^n - 1]$. Notice that these decimal numbers must not have duplicates. In this thesis we will generate 10 such datasets, five of them with n = 10, and the other five of them with n = 15. For the datasets with n = 10, we will generate m = 500 decimal numbers. For the datasets with n = 15, we will generate m = 8,000 decimal numbers. Figure 6.1 shows Decimal Dataset 1 with n = 10, and m = 500, i.e., 500 decimal numbers within the range [0, 1,023].

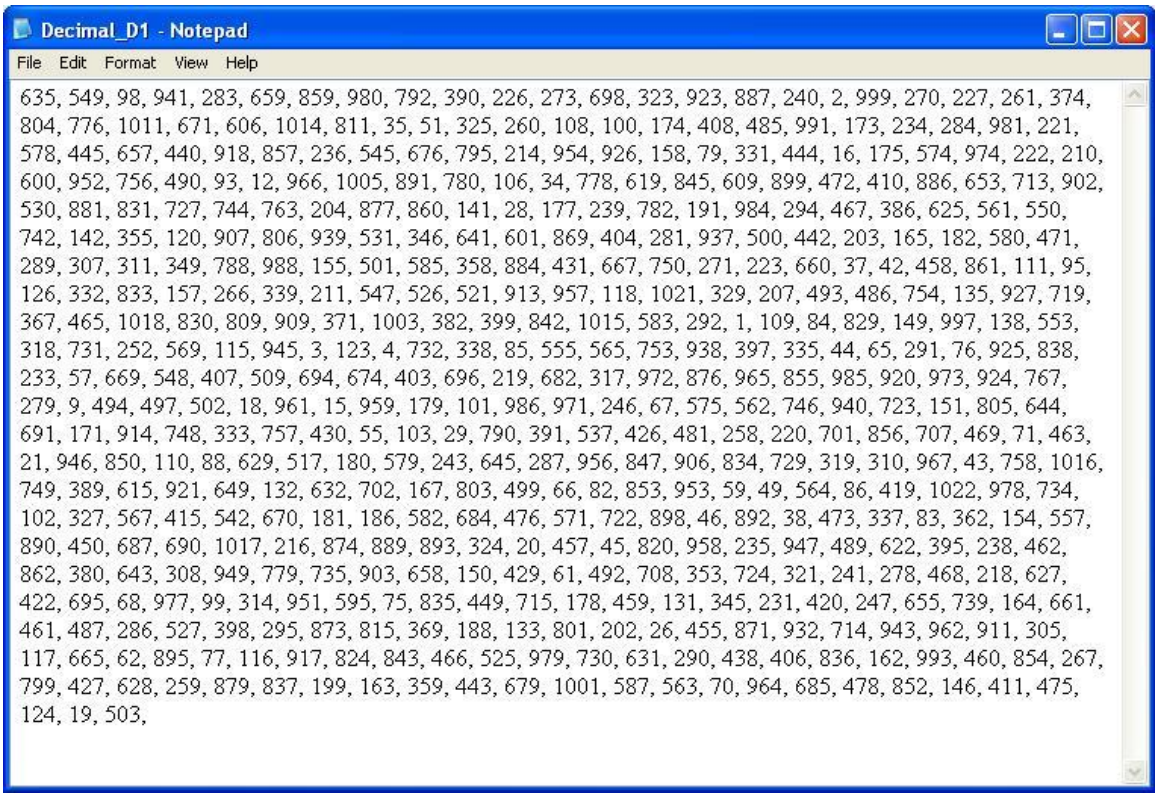

Figure 6.1 Data Preparation Step 1: Decimal Dataset 1.

For the second step we will generate 10 random Boolean functions in the conjunctive normal form (CNF), one for each dataset. We will make sure these Boolean functions vary on the number of clauses, and length of the longest/shortest clauses. We will also make sure that these Boolean functions have used up all the attributes that are available. That is, for a dataset with $n = 10$, we do not use a function like the following,

$$f = (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4)$$

where only 4 out of 10 attributes are used. Instead, we want to use all attributes from x_1 to x_{10}, for instance, a function such as this one:

$$f = (x_1 \vee x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_6} \vee x_7 \vee \overline{x_8} \vee x_9 \vee \overline{x_{10}})$$

The situation of irrelevant attributes does happen sometimes in practice. However, we will not discuss that situation in the scope of this thesis.

In particular, the Boolean function we will use for the first dataset is,

$$f_1 = (\overline{x_1} \vee \overline{x_2} \vee x_5 \vee x_8 \vee \overline{x_9}) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4} \vee x_7 \vee \overline{x_{10}}) \wedge (x_2 \vee x_3 \vee x_6 \vee x_{10}) \wedge (x_3 \vee \overline{x_5} \vee x_7 \vee \overline{x_9}) \wedge (x_4 \vee \overline{x_6} \vee \overline{x_8} \vee \overline{x_9} \vee x_{10})$$

For the third step we will use the decimal datasets generated in the first step and their corresponding Boolean functions generated in the second step to create datasets of classified binary examples. Next we will take Decimal Dataset 1 and Boolean function f_1, for instance, to do the following:

- Take the first decimal number in Decimal Dataset 1, which is 635 (see Figure 6.1), and convert it to an n-digit binary number. After the conversion, we get the binary number 1001111011. Note, for some small decimal numbers which do not need n-digit-long binary expressions, we will add 0s to the left, so that every binary number has exactly n digits.
- For each binary number, assume the leftmost digit is the value of attribute x_1, the second leftmost digit is the value of attribute x_2, and so on. Then these binary numbers can be seen as binary examples of dimension n. For instance, the binary number 1001111011 becomes binary example <1001111011>, with attributes $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, …, $x_{10} = 1$.
- Feed the values of attributes to the corresponding Boolean function, and get the classification of each binary example. For instance, feeding <1001111011> to f_1 will generate the classification value "0". To distinguish the classification value from the values of the binary attributes, we will put classification value "1" as "yes", and "0" as "no".
- Take the next decimal number and repeat this process.

The binary Dataset 1 in Figure 6.2 was derived from the Decimal Dataset 1 and Boolean function f_1.

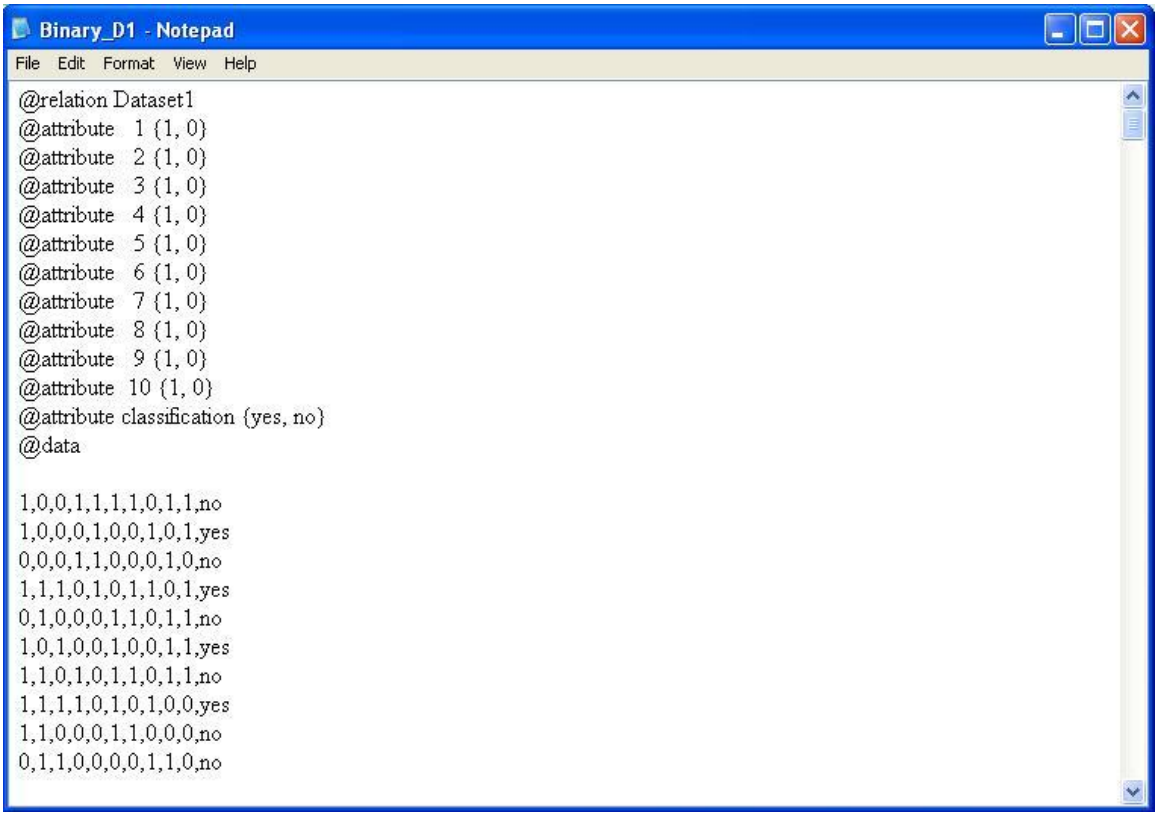

Figure 6.2 Data Preparation Step 3: Binary Dataset 1.

Note, as shown in Figure 6.2, our binary datasets are written in Weka ARFF file format, so these dataset files can later be used on Weka. The ARFF format has two sections. In the first section is the Header information, and in the second is the Data information (Witten, 2005). The Header section of the ARFF file contains the name of the relation, a list of the attributes, and their types. The Data section has all the examples and classifications that we have determined in the third step.

In the fourth step we separate each dataset into two sets. We use the first 80% examples to be training data T_1 and the rest 20% examples to be testing data T_2. To be specific, for the datasets with m = 500, the first 400 examples form the training data, and the rest 100 examples form the testing data. For the other five datasets with m = 8,000, take the first 6,400 examples form the training data and the rest 1,600 examples form the testing data. Figures 6.3.1 and 6.3.2 show the Training Dataset 1 and the Testing Dataset 1, respectively, after splitting Binary Dataset 1.

With the above four steps, we get 10 pairs of training datasets and testing datasets. Now we can start with our journey on algorithm developing.

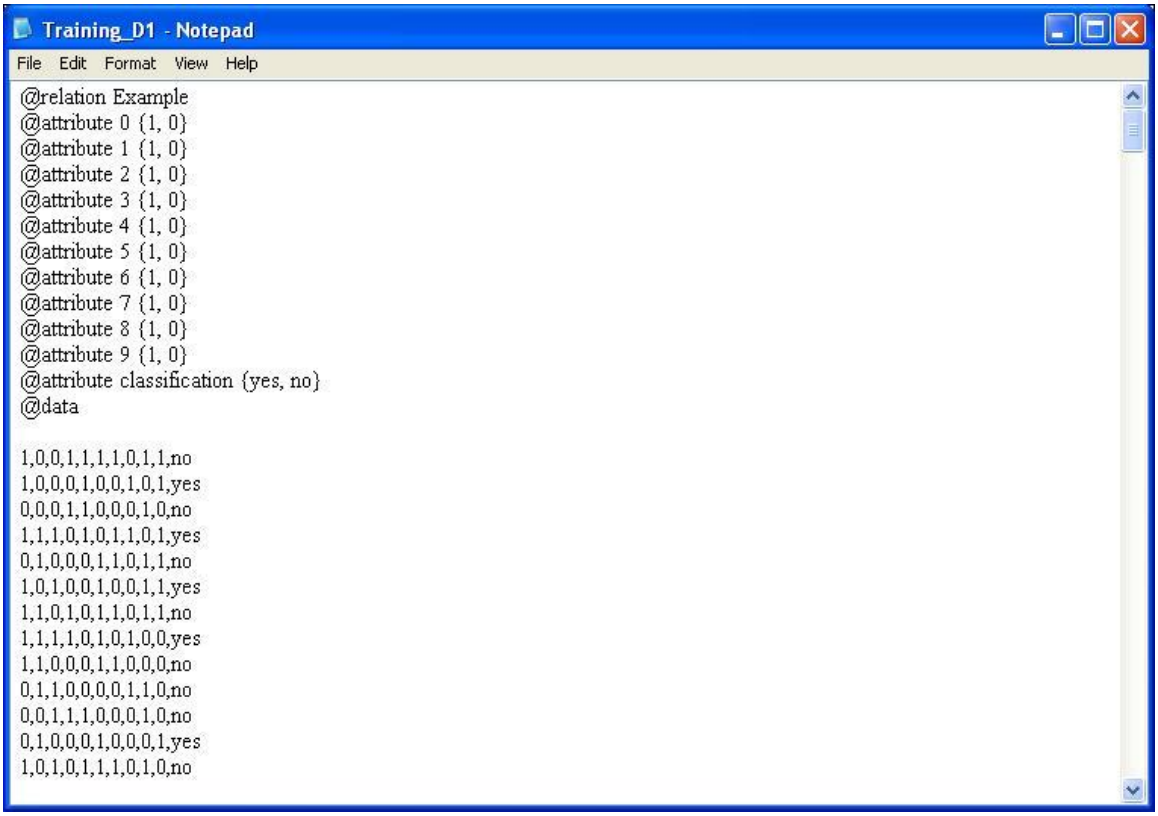

Figure 6.3-1 Data Preparation Step 4: Training Dataset 1.

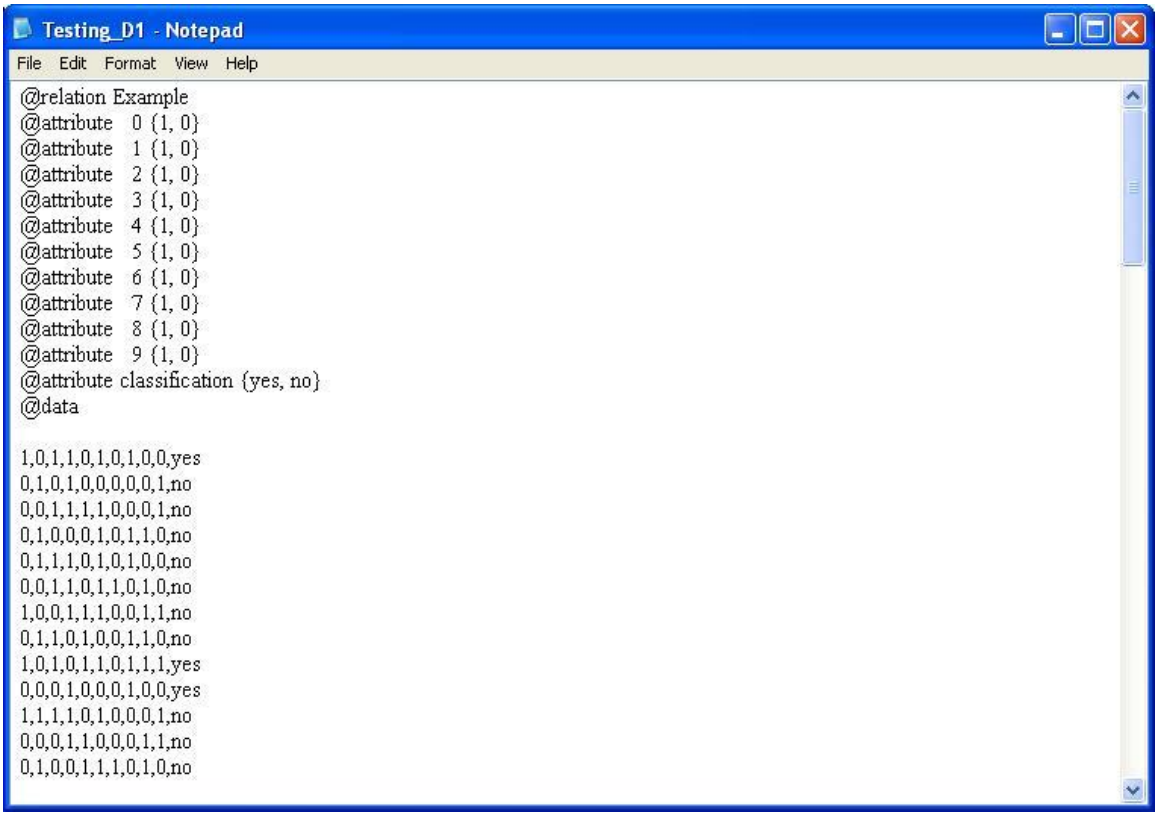

Figure 6.3-2 Data Preparation Step 4: Testing Dataset 1.

6.2 Test Results

Since we need to compare with other existing classifiers later, and not all classifiers have the ability to adjust to the change of misclassification costs, we will assume the misclassification costs take the same value, i.e., $(c_1, c_0) = (0.5, 0.5)$.

Dataset 1:

We have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 391$; $m_1^- = 109$; $m_2 = 100$; $m_2^+ = 78$; $m_2^- = 23$.

$$f_1 = (\overline{x_1} \vee \overline{x_2} \vee x_5 \vee x_8 \vee \overline{x_9}) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4} \vee x_7 \vee \overline{x_{10}}) \wedge (x_2 \vee x_3 \vee x_6 \vee x_{10}) \wedge (x_3 \vee \overline{x_5} \vee x_7 \vee \overline{x_9}) \wedge (x_4 \vee \overline{x_6} \vee \overline{x_8} \vee \overline{x_9} \vee x_{10})$$

Table 6.1-1 Test Results of Weighting Schemes on Dataset 1.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	77	15	92	1	7	8
B	76	15	91	2	7	9
C	78	4	82	0	18	18
D	77	7	84	1	15	16
E	77	13	90	1	9	10
F	78	14	92	0	8	8
G	78	1	79	0	21	21
H	78	9	87	0	13	13
I	78	8	86	0	14	14
J	65	14	79	13	8	21
K	78	0	78	0	22	22
L	67	15	82	11	7	18
M	77	15	92	1	7	8
N	74	16	90	4	6	10
O	78	10	88	0	12	12
P	78	14	92	0	8	8
Q	73	14	87	5	8	13
R	72	14	86	6	8	14
S	77	8	85	1	14	15
T	76	11	87	2	11	13
U	78	12	90	0	10	10
V	75	7	82	3	15	18
W	78	6	84	0	16	16
X	75	10	85	3	12	15

In the above table,

- "Type" -- is the different types of weighting schemes;

- "CorrectPos" -- is the number of correctly classified positive examples (meaning the actual class is positive and the predicted class is also positive);
- "CorrectNeg" -- is the number of correctly classified negative examples (meaning the actual class is negative and the predicted class is also negative);
- "Correct" -- is the number of all correctly classified examples (meaning the actual class and the predicted class are consistent);
- "WrongPos" -- is the number of misclassified positive examples (meaning the actual class is positive but the predicted class is negative);
- "WrongNeg" -- is the number of misclassified negative examples (meaning the actual class is negative but the predicted class is positive);
- "Wrong" -- is the number of all misclassified examples (meaning the actual class and the predicted class are not consistent).

Clearly, in the previous table, the following equations exist:

"CorrectPos" + "CorrectNeg" = "Correct"…….……(1)

"WrongPos" + "WrongNeg" = "Wrong"…….(2)

"CorrectPos" + "WrongPos" = m_2^+…….…(3)

"CorrectNeg" + "WrongNeg" = m_2^-…...……..…(4)

where m_2^+ is the number of the actual positive examples in the testing data;

m_2^- is the number of the actual negative examples in the testing data.

From Table 6.1-1, we can rank these weighting schemes according to the increasing order of the number of misclassified examples.

Table 6.1-2 Ranks of Weighting Schemes on Dataset 1.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	8	1	*I*	14	6	*Q*	13	5
B	9	2	*J*	21	10	*R*	14	6
C	18	9	*K*	22	11	*S*	15	7
D	16	8	*L*	18	9	*T*	13	5
E	10	3	*M*	8	1	*U*	10	3
F	8	1	*N*	10	3	*V*	18	9
G	21	10	*O*	12	4	*W*	16	8
H	13	5	*P*	8	1	*X*	15	7

From Table 6.1-2, we can see that, for Dataset 1, some weighting schemes perform better, while some do not do that well. Schemes *A*, *F*, *M*, and *P* perform the best on Dataset 1. They all correctly classified 92 examples and misclassified 8 examples. Schemes *B*, *E*, *N*, *U* misclassified 9 or 10 examples. These are acceptable results as well. At the same time, schemes *C*, *D*, *G*, *V*, and *W* rank at the bottom. They might not be the candidates of our final algorithm.

However, performing the best on one dataset does not guarantee a scheme is the optimal one. On the other hand, even the best scheme might not perform the best on all datasets. Therefore we would like to test these 24 weighting schemes on the rest 9 datasets, so we can get the ranks of the schemes on each datasets.

Dataset 2:

We have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 325$; $m_1^- = 175$; $m_2 = 100$; $m_2^+ = 60$; $m_2^- = 40$.

$$f_2 = (\overline{x_1} \vee x_8 \vee \overline{x_9}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_7 \vee \overline{x_8}) \wedge (x_2 \vee x_5 \vee x_6 \vee x_{10}) \wedge (\overline{x_3} \vee \overline{x_5} \vee \overline{x_7}) \wedge (x_4 \vee \overline{x_6} \vee \overline{x_7} \vee \overline{x_9} \vee x_{10}) \wedge (\overline{x_5} \vee \overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}})$$

Table 6.2-1 Test Results of Weighting Schemes on Dataset 2.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	57	32	89	3	8	11
B	50	31	81	10	9	19
C	59	13	72	1	27	28
D	58	21	79	2	19	21
E	57	28	85	3	12	15
F	58	30	88	2	10	12
G	59	11	70	1	29	30
H	59	26	85	1	14	15
I	49	21	70	11	19	30
J	35	29	64	25	11	36
K	59	4	63	1	36	37
L	39	29	68	21	11	32
M	58	33	91	2	7	9
N	52	32	84	8	8	16
O	59	26	85	1	14	15
P	59	26	85	1	14	15
Q	44	29	73	16	11	27
R	45	31	76	15	9	24
S	58	21	79	2	19	21
T	56	26	82	4	14	18
U	57	28	85	3	12	15
V	54	18	72	6	22	28
W	59	21	80	1	19	20
X	56	24	80	4	16	20

Table 6.2-2 Ranks of Weighting Schemes on Dataset 2.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	11	2	*I*	30	13	*Q*	27	11
B	19	7	*J*	36	15	*R*	24	10
C	28	12	*K*	37	16	*S*	21	9
D	21	9	*L*	32	14	*T*	18	6
E	15	4	*M*	9	1	*U*	15	4
F	12	3	*N*	16	5	*V*	28	12
G	30	13	*O*	15	4	*W*	20	8
H	15	4	*P*	15	4	*X*	20	8

Dataset 3:

We have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 228$; $m_1^- = 272$; $m_2 = 100$; $m_2^+ = 43$; $m_2^- = 57$.

$$f_3 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (x_1 \vee x_7 \vee \overline{x_8}) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_8 \vee x_9 \vee x_{10}) \wedge$$
$$(\overline{x_2} \vee \overline{x_9}) \wedge (\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_8 \vee \overline{x_9}) \wedge (x_3 \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee \overline{x_{10}})$$

Table 6.3-1 Test Results of Weighting Schemes on Dataset 3.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	40	56	96	3	1	4
B	38	53	91	5	4	9
C	37	53	90	6	4	10
D	35	50	85	8	7	15
E	39	53	92	4	4	8
F	40	53	93	3	4	7
G	36	54	90	7	3	10
H	38	55	93	5	2	7
I	33	52	85	10	5	15
J	33	50	83	10	7	17
K	31	52	83	12	5	17
L	36	52	88	7	5	12
M	40	56	96	3	1	4
N	39	53	92	4	4	8
O	40	55	95	3	2	5
P	38	52	90	5	5	10
Q	38	52	90	5	5	10
R	38	52	90	5	5	10
S	34	50	84	9	7	16
T	36	52	88	7	5	12
U	38	53	91	5	4	9

(table con'd)

V	31	49	80	12	8	20
W	36	53	89	7	4	11
X	34	52	86	9	5	14

Table 6.3-2 Ranks of Weighting Schemes on Dataset 3.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	4	1	*I*	15	10	*Q*	10	6
B	9	5	*J*	17	12	*R*	10	6
C	10	6	*K*	17	12	*S*	16	11
D	15	10	*L*	12	8	*T*	12	8
E	8	4	*M*	4	1	*U*	9	5
F	7	3	*N*	8	4	*V*	20	13
G	10	6	*O*	5	2	*W*	11	7
H	7	3	*P*	10	6	*X*	14	9

Dataset 4:

We have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 206$; $m_1^- = 294$; $m_2 = 100$; $m_2^+ = 36$; $m_2^- = 64$.

$$f_4 = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_6 \vee \overline{x_7} \vee x_8 \vee x_9 \vee x_{10}) \wedge (x_1 \vee \overline{x_4} \vee \overline{x_5} \vee x_8) \wedge (x_2 \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_2} \vee \overline{x_9}) \wedge (\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_8 \vee \overline{x_9}) \wedge (x_3 \vee \overline{x_6} \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_4} \vee x_8)$$

Table 6.4-1 Test Results of Weighting Schemes on Dataset 4.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	35	64	99	1	0	1
B	35	61	96	1	3	4
C	35	64	99	1	0	1
D	32	61	93	4	3	7
E	33	64	97	3	0	3
F	35	64	99	1	0	1
G	32	64	96	4	0	4
H	35	64	99	1	0	1
I	29	61	90	7	3	10
J	29	51	80	7	13	20
K	26	63	89	10	1	11
L	32	54	86	4	10	14
M	36	64	100	0	0	0
N	35	62	97	1	2	3
O	36	64	100	0	0	0

(table con'd)

P	36	62	98	0	2	2
Q	33	60	93	3	4	7
R	34	61	95	2	3	5
S	31	61	92	5	3	8
T	35	62	97	1	2	3
U	34	63	97	2	1	3
V	29	57	86	7	7	14
W	34	63	97	2	1	3
X	34	61	95	2	3	5

Table 6.4-2 Ranks of Weighting Schemes on Dataset 4.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	1	2	*I*	10	9	*Q*	7	7
B	4	5	*J*	20	12	*R*	5	6
C	1	2	*K*	11	10	*S*	8	8
D	7	7	*L*	14	11	*T*	3	4
E	3	4	*M*	0	1	*U*	3	4
F	1	2	*N*	3	4	*V*	14	11
G	4	5	*O*	0	1	*W*	3	4
H	1	2	*P*	2	3	*X*	5	6

Dataset 5:

We have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 300$; $m_1^- = 200$; $m_2 = 100$; $m_2^+ = 70$; $m_2^- = 30$.

$$f_5 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_8 \vee x_{10}) \wedge (x_2 \vee \overline{x_4} \vee \overline{x_7} \vee \overline{x_9}) \wedge$$
$$(\overline{x_2} \vee x_5 \vee \overline{x_9}) \wedge (\overline{x_2} \vee \overline{x_6} \vee x_7 \vee x_8 \vee \overline{x_{10}}) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_5} \vee x_7 \vee x_9) \wedge$$
$$(\overline{x_3} \vee \overline{x_5} \vee x_8 \vee \overline{x_{10}}) \wedge (x_4 \vee x_5 \vee \overline{x_7} \vee x_9 \vee x_{10}) \wedge (\overline{x_5} \vee \overline{x_6} \vee x_9 \vee \overline{x_{10}})$$

Table 6.5-1 Test Results of Weighting Schemes on Dataset 5.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	65	27	92	5	3	8
B	66	24	90	4	6	10
C	65	24	89	5	6	11
D	63	25	88	7	5	12
E	67	24	91	3	6	9
F	67	25	92	3	5	8
G	66	21	87	4	9	13
H	66	24	90	4	6	10

(table con'd)

I	61	16	77	9	14	23
J	57	16	73	13	14	27
K	69	13	82	1	17	18
L	59	19	78	11	11	22
M	65	27	92	5	3	8
N	67	25	92	3	5	8
O	66	27	93	4	3	7
P	65	28	93	5	2	7
Q	64	23	87	6	7	13
R	65	22	87	5	8	13
S	62	22	84	8	8	16
T	64	24	88	6	6	12
U	66	21	87	4	9	13
V	63	18	81	7	12	19
W	67	20	87	3	10	13
X	64	22	86	6	8	14

Table 6.5-2 Ranks of Weighting Schemes on Dataset 5.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	8	2	*I*	23	13	*Q*	13	7
B	10	4	*J*	27	14	*R*	13	7
C	11	5	*K*	18	10	*S*	16	9
D	12	6	*L*	22	12	*T*	12	6
E	9	3	*M*	8	2	*U*	13	7
F	8	2	*N*	8	2	*V*	19	11
G	13	7	*O*	7	1	*W*	13	7
H	10	4	*P*	7	1	*X*	14	8

Dataset 6:

We have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}763$; $m_1^- = 3{,}237$; $m_2 = 1{,}600$; $m_2^+ = 999$; $m_2^- = 601$.

$$f_6 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee \overline{x_4} \vee x_7 \vee x_{11}) \wedge (\overline{x_1} \vee \overline{x_8} \vee x_9) \wedge$$
$$(x_2 \vee \overline{x_4} \vee \overline{x_6} \vee \overline{x_{12}} \vee x_{13}) \wedge (\overline{x_2} \vee x_5 \vee \overline{x_9} \vee x_{10} \vee x_{13} \vee \overline{x_{15}}) \wedge$$
$$(\overline{x_3} \vee \overline{x_7} \vee x_9 \vee x_{12} \vee \overline{x_{14}}) \wedge (\overline{x_4} \vee x_6 \vee \overline{x_{10}} \vee x_{11}) \wedge (x_6 \vee \overline{x_7} \vee \overline{x_8} \vee x_{12}) \wedge$$
$$(\overline{x_7} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (x_8 \vee \overline{x_9} \vee x_{10} \vee \overline{x_{12}}) \wedge$$
$$(\overline{x_9} \vee x_{10} \vee x_{11} \vee x_{12} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15})$$

Table 6.6-1 Test Results of Weighting Schemes on Dataset 6.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	967	546	1513	32	55	87
B	966	480	1446	33	121	154
C	998	239	1237	1	362	363
D	978	406	1384	21	195	216
E	977	497	1474	22	104	126
F	977	541	1518	22	60	82
G	998	172	1170	1	429	430
H	979	515	1494	20	86	106
I	965	226	1191	34	375	409
J	942	264	1206	57	337	394
K	994	31	1025	5	570	575
L	950	380	1330	49	221	270
M	976	556	1532	23	45	68
N	972	495	1467	27	106	133
O	979	539	1518	20	62	82
P	981	552	1533	18	49	67
Q	965	436	1401	34	165	199
R	972	468	1440	27	133	160
S	980	362	1342	19	239	258
T	982	535	1517	17	66	83
U	972	434	1406	27	167	194
V	980	229	1209	19	372	391
W	975	397	1372	24	204	228
X	978	444	1422	21	157	178

Table 6.6-2 Ranks of Weighting Schemes on Dataset 6.

Type	Wrong	Rank	Type	Wrong	Rank	Type	Wrong	Rank
A	87	5	*I*	409	21	*Q*	199	13
B	154	9	*J*	394	20	*R*	160	10
C	363	18	*K*	575	23	*S*	258	16
D	216	14	*L*	270	17	*T*	83	4
E	126	7	*M*	68	2	*U*	194	12
F	82	3	*N*	133	8	*V*	391	19
G	430	22	*O*	82	3	*W*	228	15
H	106	6	*P*	67	1	*X*	178	11

Dataset 7:

We have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}325$; $m_1^- = 3{,}675$; $m_2 = 1{,}600$; $m_1^+ = 872$; $m_2^- = 728$;

$$f_7 = (x_1 \vee \overline{x_3} \vee \overline{x_7} \vee x_9 \vee \overline{x_{11}}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5} \vee x_6 \vee \overline{x_{10}} \vee x_{12} \vee \overline{x_{15}}) \wedge$$
$$(x_1 \vee \overline{x_8} \vee \overline{x_{10}} \vee x_{11} \vee \overline{x_{13}} \vee \overline{x_{14}}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4 \vee \overline{x_5} \vee \overline{x_7} \vee x_9 \vee \overline{x_{10}} \vee \overline{x_{12}} \vee x_{14} \vee x_{15}) \wedge$$
$$(x_2 \vee \overline{x_5} \vee \overline{x_6} \vee x_7 \vee x_8 \vee \overline{x_{10}} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee \overline{x_8} \vee x_{12}) \wedge$$
$$(\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_{13}) \wedge (x_3 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{12}) \wedge (\overline{x_3} \vee \overline{x_{11}} \vee x_{14}) \wedge$$
$$(\overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_{10} \vee \overline{x_{13}} \vee \overline{x_{15}}) \wedge (\overline{x_5} \vee x_9 \vee x_{11}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee x_{12} \vee \overline{x_{13}})$$

Table 6.7-1 Test Results of Weighting Schemes on Dataset 7.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	844	682	1526	28	46	74
B	840	646	1486	32	82	114
C	864	603	1467	8	125	133
D	808	603	1411	64	125	189
E	845	659	1504	27	69	96
F	846	683	1529	26	45	71
G	868	567	1435	4	161	165
H	855	679	1534	17	49	66
I	802	449	1251	70	279	349
J	791	447	1238	81	281	362
K	850	372	1222	22	356	378
L	800	559	1359	72	169	241
M	849	693	1542	23	35	58
N	840	649	1489	32	79	111
O	856	692	1548	16	36	52
P	851	694	1545	21	34	55
Q	823	606	1429	49	122	171
R	835	630	1465	37	98	135
S	814	593	1407	58	135	193
T	851	677	1528	21	51	72
U	818	622	1440	54	106	160
V	789	519	1308	83	209	292
W	831	614	1445	41	114	155
X	823	623	1446	49	105	154

Table 6.7-2 Ranks of Weighting Schemes on Dataset 7.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	74	7	*I*	349	22	*Q*	171	17
B	114	10	*J*	362	23	*R*	135	12
C	133	11	*K*	378	24	*S*	193	19
D	189	18	*L*	241	20	*T*	72	6
E	96	8	*M*	58	3	*U*	160	15

(table con'd)

F	71	5	*N*	111	9	*V*	292	21
G	165	16	*O*	52	1	*W*	155	14
H	66	4	*P*	55	2	*X*	154	13

Dataset 8:

We have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 3{,}768$; $m_1^- = 4{,}232$; $m_2 = 1{,}600$; $m_2^+ = 761$; $m_2^- = 839$;

$$f_8 = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_8 \vee \overline{x_{11}}) \wedge (\overline{x_1} \vee \overline{x_7} \vee x_{12} \vee \overline{x_{13}}) \wedge$$
$$(\overline{x_2} \vee x_4 \vee \overline{x_5} \vee \overline{x_8} \vee x_{10} \vee \overline{x_{12}} \vee x_{14}) \wedge (\overline{x_2} \vee \overline{x_6} \vee x_{13} \vee \overline{x_{14}}) \wedge$$
$$(\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_8 \vee x_{13}) \wedge (x_3 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{12}) \wedge (\overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_{10} \vee \overline{x_{13}} \vee \overline{x_{15}}) \wedge$$
$$(\overline{x_4} \vee \overline{x_{11}} \vee x_{14}) \wedge (\overline{x_5} \vee x_9 \vee x_{11}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee x_{12}) \wedge (\overline{x_8} \vee \overline{x_9} \vee x_{11} \vee x_{12} \vee \overline{x_{13}}) \wedge$$
$$(x_9 \vee \overline{x_{10}} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15})$$

Table 6.8-1 Test Results of Weighting Schemes on Dataset 8.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	698	807	1505	63	32	95
B	700	740	1440	61	99	160
C	446	815	1261	315	24	339
D	426	757	1183	335	82	417
E	696	781	1477	65	58	123
F	705	797	1502	56	42	98
G	454	809	1263	307	30	337
H	696	810	1506	65	29	94
I	647	584	1231	114	255	369
J	658	506	1164	103	333	436
K	434	754	1188	327	85	412
L	679	627	1306	82	212	294
M	704	806	1510	57	33	90
N	707	738	1445	54	101	155
O	699	817	1516	62	22	84
P	689	816	1505	72	23	95
Q	694	688	1382	67	151	218
R	708	718	1426	53	121	174
S	428	738	1166	333	101	434
T	688	811	1499	73	28	101
U	680	750	1430	81	89	170
V	410	698	1108	351	141	492
W	664	779	1443	97	60	157
X	659	758	1417	102	81	183

Table 6.8-2 Ranks of Weighting Schemes on Dataset 8.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	95	4	*I*	369	18	*Q*	218	14
B	160	10	*J*	436	22	*R*	174	12
C	339	17	*K*	412	19	*S*	434	21
D	417	20	*L*	294	15	*T*	101	6
E	123	7	*M*	90	2	*U*	170	11
F	98	5	*N*	155	8	*V*	492	23
G	337	16	*O*	84	1	*W*	157	9
H	94	3	*P*	95	4	*X*	183	13

Dataset 9:

We have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}611$; $m_1^- = 3{,}389$; $m_2 = 1{,}600$; $m_1^+ = 895$; $m_2^- = 705$;

$$f_9 = (x_1 \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_6 \vee x_7 \vee x_{11}) \wedge (\overline{x_1} \vee \overline{x_8} \vee x_9 \vee x_{14}) \wedge$$
$$(x_2 \vee \overline{x_4} \vee \overline{x_6} \vee x_{10} \vee \overline{x_{12}} \vee x_{13}) \wedge (\overline{x_2} \vee x_5 \vee \overline{x_8} \vee x_9 \vee x_{13} \vee \overline{x_{15}}) \wedge$$
$$(\overline{x_2} \vee \overline{x_7} \vee x_9 \vee x_{12} \vee \overline{x_{14}}) \wedge (\overline{x_3} \vee x_6 \vee \overline{x_{10}} \vee x_{11}) \wedge (x_3 \vee \overline{x_7} \vee \overline{x_8} \vee x_{12}) \wedge$$
$$(\overline{x_4} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (x_4 \vee \overline{x_9} \vee x_{10} \vee \overline{x_{12}}) \wedge$$
$$(\overline{x_4} \vee x_{10} \vee x_{11} \vee x_{12} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15}) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_7} \vee x_8 \vee \overline{x_{11}} \vee x_{12}) \wedge$$
$$(\overline{x_5} \vee x_7 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{13}) \wedge (\overline{x_6} \vee \overline{x_8} \vee x_9 \vee x_{10} \vee \overline{x_{12}} \vee \overline{x_{13}})$$

Table 6.9-1 Test Results of Weighting Schemes on Dataset 9.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	861	613	1474	34	92	126
B	865	514	1379	30	191	221
C	894	233	1127	1	472	473
D	849	387	1236	46	318	364
E	874	531	1405	21	174	195
F	874	606	1480	21	99	120
G	894	139	1033	1	566	567
H	885	567	1452	10	138	148
I	876	165	1041	19	540	559
J	853	201	1054	42	504	546
K	894	20	914	1	685	686
L	855	349	1204	40	356	396
M	866	630	1496	29	75	104
N	872	521	1393	23	184	207
O	880	600	1480	15	105	120

(table con'd)

P	871	615	1486	24	90	114
Q	865	420	1285	30	285	315
R	873	486	1359	22	219	241
S	857	322	1179	38	383	421
T	875	591	1466	20	114	134
U	873	464	1337	22	241	263
V	865	184	1049	30	521	551
W	881	415	1296	14	290	304
X	871	468	1339	24	237	261

Table 6.9-2 Ranks of Weighting Schemes on Dataset 9.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	126	4	*I*	559	21	*Q*	315	14
B	221	9	*J*	546	19	*R*	241	10
C	473	18	*K*	686	23	*S*	421	17
D	364	15	*L*	396	16	*T*	134	5
E	195	7	*M*	104	1	*U*	263	12
F	120	3	*N*	207	8	*V*	551	20
G	567	22	*O*	120	3	*W*	304	13
H	148	6	*P*	114	2	*X*	261	11

Dataset 10:

We have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}288$; $m_1^- = 3{,}712$; $m_2 = 1{,}600$; $m_2^+ = 852$; $m_2^- = 748$;

$$f_{10} = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_7) \wedge (x_1 \vee x_3 \vee \overline{x_9} \vee x_{11}) \wedge$$
$$(x_1 \vee \overline{x_4} \vee \overline{x_8} \vee x_{10} \vee \overline{x_{14}}) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_8 \vee \overline{x_9} \vee x_{12} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_{11}} \vee x_{15}) \wedge$$
$$(x_2 \vee \overline{x_4} \vee x_6 \vee x_{10} \vee x_{13} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee x_9 \vee \overline{x_{12}}) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_8} \vee \overline{x_9} \vee x_{10}) \wedge$$
$$(\overline{x_4} \vee x_5 \vee x_9 \vee \overline{x_{13}} \vee \overline{x_{14}}) \wedge (x_4 \vee \overline{x_6} \vee x_{12} \vee x_{14}) \wedge (\overline{x_4} \vee \overline{x_7} \vee x_{10} \vee x_{11} \vee x_{15}) \wedge$$
$$(\overline{x_5} \vee x_6 \vee \overline{x_7} \vee x_8 \vee \overline{x_{11}} \vee x_{12}) \wedge (\overline{x_5} \vee x_7 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{13}) \wedge$$
$$(\overline{x_6} \vee \overline{x_8} \vee x_9 \vee x_{10} \vee \overline{x_{12}} \vee \overline{x_{13}}) \wedge (\overline{x_6} \vee \overline{x_{10}} \vee x_{13} \vee x_{14} \vee \overline{x_{15}})$$

Table 6.10-1 Test Results of Weighting Schemes on Dataset 10.

Scheme	CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
A	822	660	1482	30	88	118
B	820	585	1405	32	163	195
C	842	580	1422	10	168	178
D	758	608	1366	94	140	234

(table con'd)

E	826	613	1439	26	135	161
F	828	653	1481	24	95	119
G	843	529	1372	9	219	228
H	834	648	1482	18	100	118
I	804	358	1162	48	390	438
J	793	363	1156	59	385	444
K	837	300	1137	15	448	463
L	800	474	1274	52	274	326
M	826	670	1496	26	78	104
N	826	600	1426	26	148	174
O	830	667	1497	22	81	103
P	824	678	1502	28	70	98
Q	816	529	1345	36	219	255
R	825	571	1396	27	177	204
S	761	589	1350	91	159	250
T	827	652	1479	25	96	121
U	815	561	1376	37	187	224
V	749	460	1209	103	288	391
W	821	554	1375	31	194	225
X	810	572	1382	42	176	218

Table 6.10-2 Ranks of Weighting Schemes on Dataset 10.

Scheme	Wrong	Rank	Scheme	Wrong	Rank	Scheme	Wrong	Rank
A	118	4	*I*	438	21	*Q*	255	18
B	195	10	*J*	444	22	*R*	204	11
C	178	9	*K*	463	23	*S*	250	17
D	234	16	*L*	326	19	*T*	121	6
E	161	7	*M*	104	3	*U*	224	13
F	119	5	*N*	174	8	*V*	391	20
G	228	15	*O*	103	2	*W*	225	14
H	118	4	*P*	98	1	*X*	218	12

6.3 Analysis of the Test Results

Now we have the ranks of each weighting schemes on 10 datasets separately, we want to put them together and see which scheme performs the best over all. What we do is we add up the ranks of the 10 datasets for each weighting schemes, then we can have the total rank according to the increasing order of the sum of the ranks.

Table 6.11 Total Rank of Weighting Schemes on Datasets 1 to 10.

Type	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	Sum	Total Rank
A	1	2	1	2	2	5	7	4	4	4	32	4
B	2	7	5	5	4	9	10	10	9	10	71	9

(table con'd)

C	9	12	6	2	5	18	11	17	18	9	107	14
D	8	9	10	7	6	14	18	20	15	16	123	16
E	3	4	4	4	3	7	8	7	7	7	54	6
F	1	3	3	2	2	3	5	5	3	5	32	4
G	10	13	6	5	7	22	16	16	22	15	132	17
H	5	4	3	2	4	6	4	3	6	4	41	5
I	6	13	10	9	13	21	22	18	21	21	154	20
J	10	15	12	12	14	20	23	22	19	22	169	22
K	11	16	12	10	10	23	24	19	23	23	171	23
L	9	14	8	11	12	17	20	15	16	19	141	19
M	1	1	1	1	2	2	3	2	1	3	17	1
N	3	5	4	4	2	8	9	8	8	8	59	8
O	4	4	2	1	1	3	1	1	3	2	22	2
P	1	4	6	3	1	1	2	4	2	1	25	3
Q	5	11	6	7	7	13	17	14	14	18	112	15
R	6	10	6	6	7	10	12	12	10	11	90	11
S	7	9	11	8	9	16	19	21	17	17	134	18
T	5	6	8	4	6	4	6	6	5	6	56	7
U	3	4	5	4	7	12	15	11	12	13	86	10
V	9	12	13	11	11	19	21	23	20	20	159	21
W	8	8	7	4	7	15	14	9	13	14	99	13
X	7	8	9	6	8	11	13	13	11	12	98	12

From the above table, we can see that, among the 24 weighting schemes, the *M*-th scheme performed the best. It ranks no higher than the third place on all datasets. Therefore the *M*-th scheme will become our final algorithm. From now on we will call it the *M** algorithm.

For an easy reference, we will present the table of weighting schemes here as follows:

Table 6.12 Twenty-Four Weighting Schemes.

Algorithm	Direct Relation	Indirect Relation	Examples on the Same Level	Factor
A	linear	0	0	(1, 1)
B	linear	0	0	(F_1, F_0)
C	linear	0	Linear	(1, 1)
D	linear	0	Linear	(F_1, F_0)
E	linear	1	0	(1, 1)
F	linear	1	0	(F_1, F_0)
G	linear	1	Linear	(1, 1)
H	linear	1	Linear	(F_1, F_0)
I	linear	linear	0	(1, 1)
J	linear	linear	0	(F_1, F_0)
K	linear	linear	Linear	(1, 1)

(table con'd)

L	linear	linear	Linear	(F_1, F_0)
M	exponential	0	0	(1, 1)
N	exponential	0	0	(F_1, F_0)
O	exponential	0	Exponential	(1, 1)
P	exponential	0	Exponential	(F_1, F_0)
Q	exponential	linear	0	(1, 1)
R	exponential	linear	0	(F_1, F_0)
S	exponential	linear	Exponential	(1, 1)
T	exponential	linear	Exponential	(F_1, F_0)
U	exponential	exponential	0	(1, 1)
V	exponential	exponential	0	(F_1, F_0)
W	exponential	exponential	Exponential	(1, 1)
X	exponential	exponential	Exponential	(F_1, F_0)

As it can be seen from this table, the *M** algorithm is the one where examples in direct relation with the target example have weights that change exponentially (refer to Figures 5.5, 5.7-1, and 5.7-2), and all the other examples have weights equal to 0. Moreover, this algorithm does not need to multiply the final voting scores by any factor to adjust the positive and negative votes.

CHAPTER 7. COMPUTATION EXPRESSION

7.1 Pseudocode

Input:

1) The number of attributes n.

2) Training data T_1: a set of examples $x_i = (A_{i,1}, A_{i,2}, ..., A_{i,n}, A_{i,n+1})$, where
$i = 1, 2, 3, ...$;
$A_{i,1}, A_{i,2}, ..., A_{i,n}$ are binary attributes, take value 1 or 0;
$A_{i,n+1}$ is the classification attribute, take value 1 or 0.

3) Testing data T_2: a set of examples $z_j = (A_{j,1}, A_{j,2}, ..., A_{j,n})$, where
$j = 1, 2, 3, \ldots$;
$A_{j,1}, A_{j,2}, ..., A_{j,n}$ are binary attributes, take value 1 or 0;

4) A pair of misclassification costs (c_1, c_0), where $c_1 + c_0 = 1$, and $c_1, c_0 \geq 0$

Output:

1) A pair of probabilities (p_1, p_0) for every example z_j in the testing dataset T_2.

2) Predicted class $A_{j,n+1}$ for every new example z_j in the testing dataset T_2.

Algorithm:

For every example z_j in testing data T_2,

1. Calculate PositiveVote and NegativeVote for z_j

```
Initialize PositiveVote = 0, NegativeVote = 0,
For every example x_i in training data T_1,
{
    If example x_i has direct relation with z_j
    {
        L = level of z_j;
        E = max {L, n – L };
        d = distance of x_i to z_j;
        Weight = 2^(E-d);
    }
```

```
        If x is positive,
        {
                PositiveVote = PositiveVote + Weight;
        }
        else,
        {
                NegativeVote = NegativeVote + Weight;
        }
}
```

2. Calculate the pair of probabilities (p_1, p_0) for z_j

If PositiveVote + Negative ≠ 0,
{

$$p_1 = \frac{PositiveVote}{PositiveVote + NegativeVote};$$

$$p_0 = \frac{NegativeVote}{PositiveVote + NegativeVote};$$

}
else,
{

$p_1 = p_0 = 0.5$;

}

3. Calculate the predicted class $A_{j,n+1}$ for z_j

Let $s_1 = p_0 \times c_1$; $s_0 = p_1 \times c_0$;
If $s_1 < s_0$,
{

$A_{j,n+1} = 1$;

}
else if $s_1 > s_0$,
{

$A_{j,n+1} = 0$;

}
else,
{

$A_{j,n+1}$ = rand(); // rand() randomly generates 0 or 1.

}

7.2 Computational Complexity

For every target example z_j, we have to take the following three steps:

1. Calculate PositiveVote and NegativeVote for z_j;
2. Calculate the pair of probabilities (p_1, p_0) for z_j;
3. Calculate the predicted class $A_{j,n+1}$ for z_j.

The second and third steps cost constant execution time, i.e., O(1).The first step needs to go through all the classified examples and find out the ones that are in direct relations with the target example z_j.

To find out all the examples in direct relations with z_j, in the most straightforward and brute force way, we can do the following: For every x_i in T_1 which is in an ordered relation with z_j, go through the rest of the examples besides x_i in T_1, if there is no other example y such that, y is in the opposite class of x_i, and $x_i \prec y \prec z_j$ or $x_i \succ y \succ z_j$ (i.e., y is in the path between x_i and z_j), then x_i is in a direct relation with z_j. We stop search as soon as we find one cutting example. Note that the number of x_i, examples in ordered relations with z_j, cannot exceed m_1. In fact, most cases this number will be much smaller than m_1. The number of y, examples from the opposite class of x_i and in ordered relations with z_j, is even smaller than the previous number. It has an upper bound $m_1 - 1$. Therefore the upper bound of the execution time will be $O(m_1 (m_1 - 1)) = O (m_1^2)$.

If we could achieve a small improvement to the above algorithm, before we start to process the training data, we can separate it into four groups. The first group K_1 is the positive examples that are greater than z_j; the second group K_2 is the positive examples that are less than z_j; the third group K_3 is the negative examples that are greater than z_j; the fourth group K_4 is the negative examples that are less than z_j. Suppose the number of examples of the four groups are k_1, k_2, k_3, and k_4, respectively. Then to check if any example in group K_1 is in a direct relation with z_j, we only need to go through examples in K_3. This is because only a negative example that is greater than z_j could serve as a cutting example for a positive example that is greater than z_j. Similarly, to check if any example in group K_2 is in a direct relation with z_j, we only need to go through examples in K_4; to check if any example in group K_3 is in a direct relation with z_j, we only need to go through examples in K_1; to check if any example in group K_4 is in a direct relation with z_j, we only need to go through examples in K_2. We also stop search as soon as we find one cutting example. Therefore, in this case, the upper bound of the execution time will be $\max\{O(k_1 \cdot (k_3 - 1)), O(k_2 \cdot (k_4 - 1)), O(k_3 \cdot (k_1 - 1)), O(k_4 \cdot (k_2 - 1))\} = \max\{O(k_1 \cdot k_3), O(k_2 \cdot k_4)\}$.

Furthermore, if we could make some change to the way we store the data, we could make the algorithm more efficient. Suppose we store the training data according to their levels. Then for each example x_i which is in an ordered relation, we only need to check the

examples which are from the opposite class of x_i and at levels between x_i and z_j. The reason is only examples at levels between x_i and z_j could serve as a cutting example. By doing this, we will be able to reduce the execution time even more.

Nevertheless, the upper bound for finding all the examples that are in direct relations with z_j will have the execution time at most quadratic to the number of training data, i.e., $O(m_1^2)$. In most cases, it does not cost nearly as much time.

Now if we will go through all the examples in the testing data, the execution time will be no more than $O(m_1^2 \cdot m_2)$, where m_1 is the number of examples in training data, and m_2 is the number of examples in testing data.

CHAPTER 8. AN EXTENSIVE ILLUSTRATIVE PROBLEM

Now we will apply the *M** algorithm on the illustrative problem. Please recall that we have the following binary system with dimension $n = 6$. The system has $2^6 = 64$ valid examples, and we are provided with the 32 classified examples as shown in the diagram below. Its hidden Boolean function is

$$f = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_4 \vee \overline{x_5} \vee x_6)$$

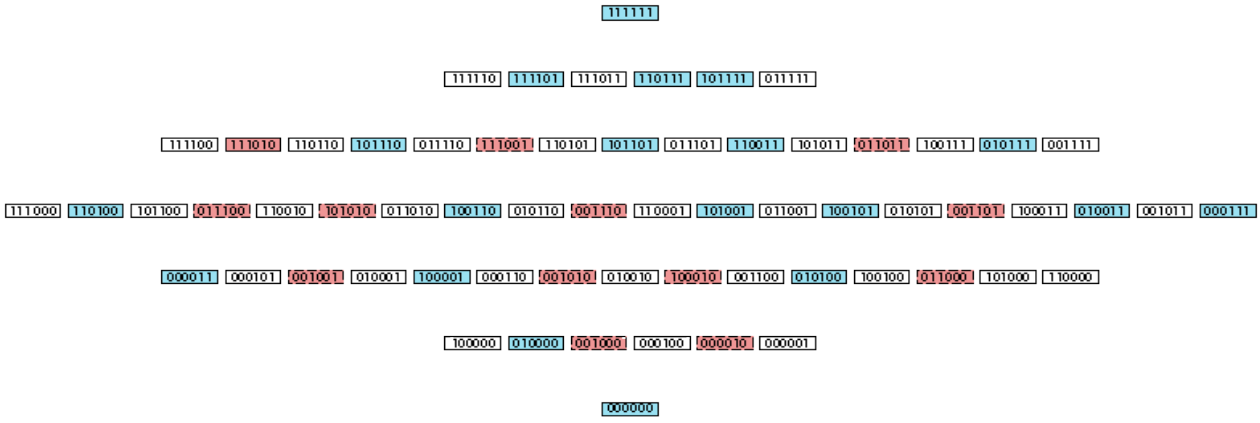

Figure 8.1 Poset Diagram of the Illustration Problem.

Our demo program returns a result table as follows:

Table 8.1 Result of the *M** Algorithm on the Illustrative Problem.

#	*z*	P-Vote	N-Vote	Actual	Predicted	Wrong	Probability Distribution
1	111110	48.000	28.000	1	1		(0.632, 0.368)
2	111011	32.000	64.000	0	1		(0.333, 0.667)
3	011111	40.000	54.000	0	0		(0.426, 0.574)
4	111100	20.000	14.000	1	1		(0.588, 0.412)
5	110110	34.000	0.000	1	1		(1.000, 0.000)
6	011110	4.000	28.000	0	0		(0.125, 0.875)
7	110101	47.000	0.000	1	1		(1.000, 0.000)
8	011101	12.000	26.000	0	0		(0.316, 0.684)
9	101011	28.000	16.000	1	1		(0.636, 0.364)
10	100111	52.000	0.000	1	1		(1.000, 0.000)
11	001111	24.000	26.000	0	0		(0.480, 0.520)
12	111000	0.000	14.000	0	0		(0.000, 1.000)
13	101100	13.000	2.000	1	1		(0.867, 0.133)
14	110010	8.000	10.000	0	1		(0.444, 0.556)
15	011010	0.000	20.000	0	0		(0.000, 1.000)
16	010110	13.000	2.000	1	1		(0.867, 0.133)

(table con'd)

17	110001	13.000	4.000	1	1		(0.765, 0.235)
18	011001	0.000	18.000	0	0		(0.000, 1.000)
19	010101	16.000	0.000	1	1		(1.000, 0.000)
20	100011	17.000	4.000	1	1		(0.810, 0.190)
21	001011	6.000	14.000	0	0		(0.300, 0.700)
22	000101	26.000	8.000	1	1		(0.765, 0.235)
23	010001	30.000	4.000	1	1		(0.882, 0.118)
24	000110	22.000	16.000	1	1		(0.579, 0.421)
25	010010	26.000	12.000	0	1	+	(0.684, 0.316)
26	001100	0.000	32.000	0	0		(0.000, 1.000)
27	100100	43.000	0.000	1	1		(1.000, 0.000)
28	101000	12.000	20.000	1	0	+	(0.375, 0.625)
29	110000	26.000	8.000	1	1		(0.765, 0.235)
30	100000	60.000	28.000	1	1		(0.682, 0.318)
31	000100	70.000	16.000	1	1		(0.814, 0.186)
32	000001	82.000	24.000	1	1		(0.774, 0.226)

On 32 testing examples, the *M** algorithm classifies 30 of them correctly, and 2 of them wrongly. To see how we get the above results, we will next examine the unknown example one by one.

For the 1st unknown example <111110>:

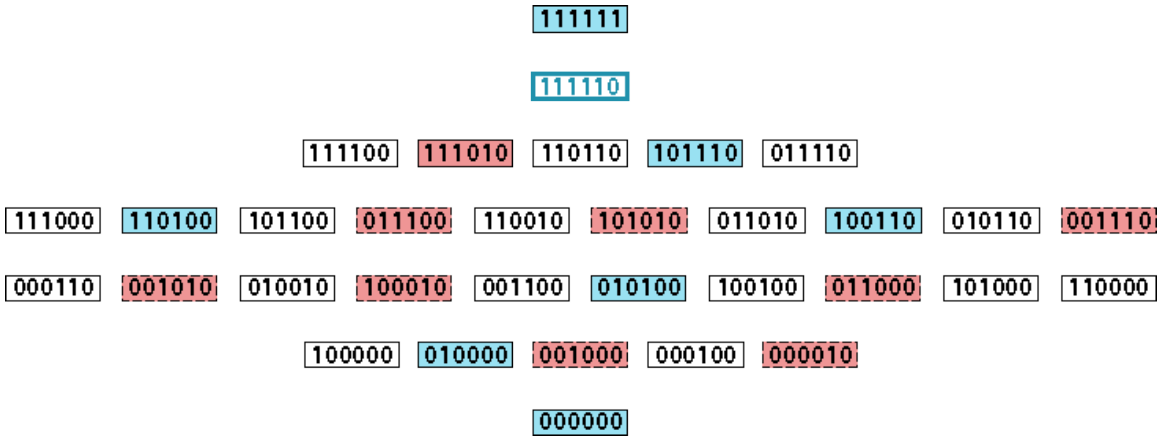

Figure 8.2 Examples in Ordered Relations with <111110>.

Again, 111110 is the unknown example *z* that we want to classify. The blue frame means its actual classification should be positive. All the rest examples are the examples that are in monotonic relation with *z*. At this time, we only need to consider the blue colored examples and the red colored examples, which means their classification are known to us.

111111 : "Direct relation" type, weight $2^4 = 16$;

101110 : "Direct relation" type, weight $2^4 = 16$;
110100 : "Direct relation" type, weight $2^3 = 8$;
100110 : "Direct relation" type, weight $2^4 = 8$;
010100 : "Indirect relation" type (cutting example 011100), weight 0;
010000 : "Indirect relation" type (cutting example 011100), weight 0;
000000 : "Indirect relation" type (cutting example 011100), weight 0;
Positive Vote = 16 + 16 + 8 + 8 = 48;
111010 : "Direct relation" type, weight $2^4 = 16$;
011100 : "Direct relation" type, weight $2^3 = 8$;
101010 : "Indirect relation" type (cutting example 111010), weight 0;
001110 : "Indirect relation" type (cutting example 101110), weight 0;
001010 : "Indirect relation" type (cutting example 111010), weight 0;
100010 : "Indirect relation" type (cutting example 111010), weight 0;
011000 : "Indirect relation" type (cutting example 111010), weight 0;
001000 : "Indirect relation" type (cutting example 111010), weight 0;
000010 : "Indirect relation" type (cutting example 111010), weight 0;
Negative Vote = 16 + 8 = 24;

Therefore, probability $p_1 = 48 / (48 + 24) = 0.632$, and $p_0 = 8 / (64 + 8) = 0.368$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.184$, and $s_0 = p_1 \times c_0 = 0.316$.
Since the s_1 is smaller, we will assign 111110 to be positive, which is correct.

Note that we only list one "cutting example" here, because as long as we know there exists one, the corresponding example will be decided to be in an indirect relation.

For the 2nd unknown example <111011>:

111111 : "Direct relation" type, weight $2^4 = 16$;
110011 : "Direct relation" type, weight $2^4 = 16$;
101001 : "Indirect relation" type (cutting example 111001), weight 0;
010011 : "Indirect relation" type (cutting example 011011), weight 0;
000011 : "Indirect relation" type (cutting example 011011), weight 0;
100001 : "Indirect relation" type (cutting example 111001), weight 0;
010000 : "Indirect relation" type (cutting example 111001), weight 0;
000000 : "Indirect relation" type (cutting example 111001), weight 0;

Positive Vote = 16 + 16 = 32;

111010: "Direct relation" type, weight $2^4 = 16$;

111001: "Direct relation" type, weight $2^4 = 16$;

011011: "Direct relation" type, weight $2^4 = 16$;

101010: "Direct relation" type, weight $2^3 = 8$;

001001: "Indirect relation" type (cutting example 101001), weight 0;

001010: "Direct relation" type, weight $2^2 = 4$;

100010: "Indirect relation" type (cutting example 110011), weight 0;

001000: "Indirect relation" type (cutting example 101001), weight 0;

000010: "Indirect relation" type (cutting example 110011), weight 0;

Negative Vote = 16 + 16 + 16 + 8 + 4 = 60;

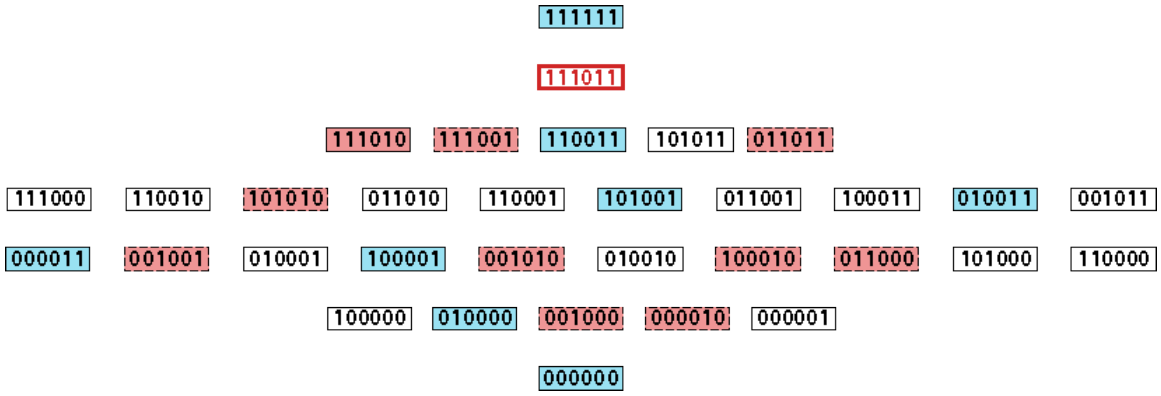

Figure 8.3 Examples in Ordered Relations with <111011>.

Therefore, probability $p_1 = 32 / (32 + 60) = 0.333$, and $p_0 = 60 / (32 + 60) = 0.667$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.3335$, and $s_0 = p_1 \times c_0 = 0.1665$.
Since s_0 is smaller, we will assign 111011 to be negative, which is correct.

For the 3rd unknown example <011111>:

111111: "Direct relation" type, weight $2^4 = 16$;

010111: "Direct relation" type, weight $2^4 = 16$;

010011: "Indirect relation" type (cutting example 011011), weight 0;

000111: "Direct relation" type, weight $2^3 = 8$;

000011: "Indirect relation" type (cutting example 011011), weight 0;

010100: "Indirect relation" type (cutting example 011100), weight 0;

010000: "Indirect relation" type (cutting example 011011), weight 0;

000000 : "Indirect relation" type (cutting example 011011), weight 0;
Positive Vote = 16 + 16 + 8 = 40;
011011 : "Direct relation" type, weight 2^4 = 16;
011100 : "Direct relation" type, weight 2^3 = 8;
001110 : "Direct relation" type, weight 2^3 = 8;
001101 : "Direct relation" type, weight 2^3 = 8;
001001 : "Direct relation" type, weight 2^2 = 4;
001010 : "Direct relation" type, weight 2^2 = 4;
011000 : "Direct relation" type, weight 2^2 = 4;
001000 : "Direct relation" type, weight 2^1 = 2;
000010 : "Indirect relation" type (cutting example 010111), weight 0;
Negative Vote = 16 + 8 + 8 + 8 + 4 + 4 + 4 + 2 = 54;

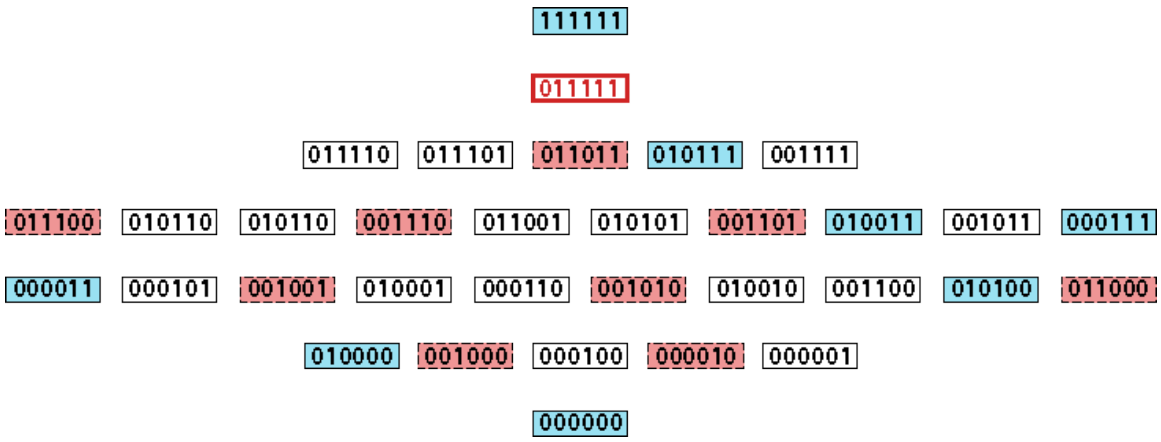

Figure 8.4 Examples in Ordered Relations with <011111>.

Therefore, probability p_1 = 40 / (40 + 54) = 0.426, and p_0 = 54 / (40 + 54) = 0.574.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.287$, and $s_0 = p_1 \times c_0 = 0.213$.
Since s_0 is smaller, we will assign 011111 to be negative, which is correct.

For the 4th unknown example <111100>:

111111 : "Direct relation" type, weight 2^2 = 4;
111101 : "Direct relation" type, weight 2^3 = 8;
110100 : "Direct relation" type, weight 2^3 = 8;
010100 : "Indirect relation" type (cutting example 011100), weight 0;
010000 : "Indirect relation" type (cutting example 011100), weight 0;

000000 : "Indirect relation" type (cutting example 011100), weight 0;
Positive Vote = 4 + 8 + 8 = 20;
011100 : "Direct relation" type, weight $2^3 = 8$;
011000 : "Direct relation" type, weight $2^2 = 4$;
001000 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 8 + 4 + 2 = 14;

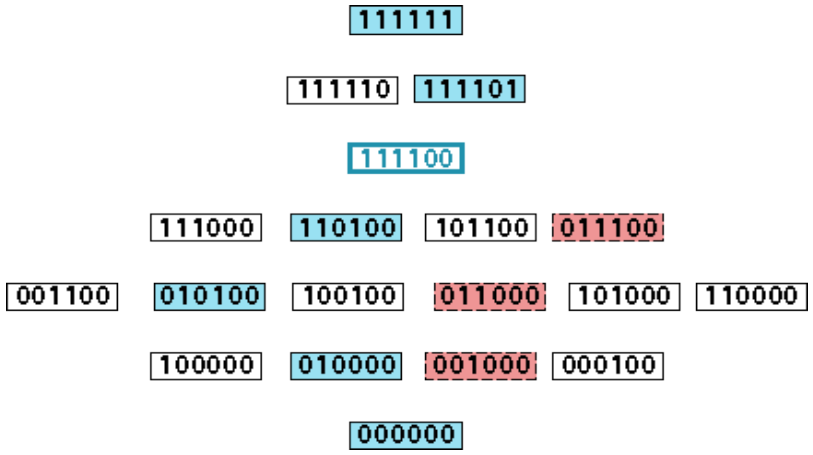

Figure 8.5 Examples in Ordered Relations with <111100>.

Therefore, probability $p_1 = 20 / (20 + 14) = 0.588$, and $p_0 = 14 / (20 + 14) = 0.412$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.206$, and $s_0 = p_1 \times c_0 = 0.294$.
Since s_1 is smaller, we will assign 111100 to be positive, which is correct.

For the 5th unknown example <110110>:

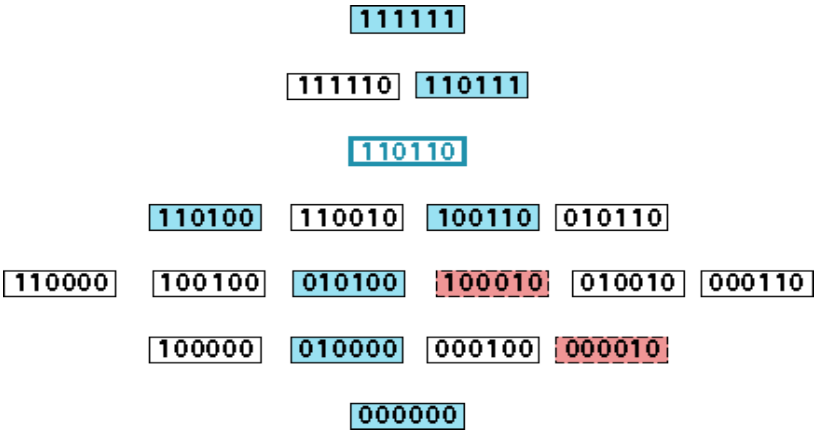

Figure 8.6 Examples in Ordered Relations with <110110>.

111111 : "Direct relation" type, weight $2^2 = 4$;

110111 : "Direct relation" type, weight $2^3 = 8$;
110100 : "Direct relation" type, weight $2^3 = 8$;
100110 : "Direct relation" type, weight $2^3 = 8$;
010100 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^1 = 2$;
000000 : "Indirect relation" type (cutting example 100010), weight 0;
Positive Vote = 4 + 8 + 8 + 8 +4 + 2 = 34;
100010 : "Indirect relation" type (cutting example 100110), weight 0;
000010 : "Indirect relation" type (cutting example 100110), weight 0;
Negative Vote = 0;

Therefore, probability $p_1 = 34 / (34 + 0) = 1.000$, and $p_0 = 0 / (34 + 0) = 0.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.000$, and $s_0 = p_1 \times c_0 = 0.500$.
Since s_1 is smaller, we will assign 110110 to be positive, which is correct.

For the 6th unknown example <011110>:

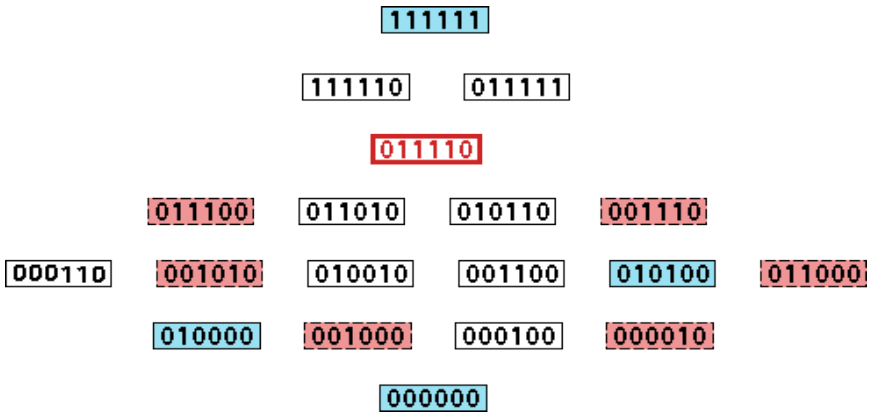

Figure 8.7 Examples in Ordered Relations with <011110>.

111111 : "Direct relation" type, weight $2^2 = 4$;
010100 : "Indirect relation" type (cutting example 011100), weight 0;
010000 : "Indirect relation" type (cutting example 011100), weight 0;
000000 : "Indirect relation" type (cutting example 011100), weight 0;
Positive Vote = 4;
011100 : "Direct relation" type, weight $2^3 = 8$;
001110 : "Direct relation" type, weight $2^3 = 8$;
001010 : "Direct relation" type, weight $2^2 = 4$;

011000 : "Direct relation" type, weight $2^2 = 4$;
001000 : "Direct relation" type, weight $2^1 = 2$;
000010 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 8 + 8 + 4 + 4 + 2 + 2 = 28;
Therefore, probability $p_1 = 4 / (4 + 28) = 0.125$, and $p_0 = 28 / (4 + 28) = 0.875$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.4375$, and $s_0 = p_1 \times c_0 = 0.0625$.
Since s_0 is smaller, we will assign 011110 to be negative, which is correct.

For the 7th unknown example <110101>:

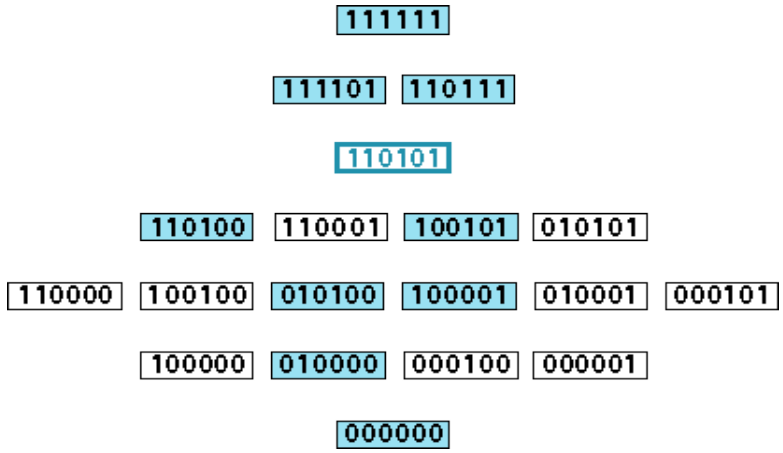

Figure 8.8 Examples in Ordered Relations with <110101>.

111111 : "Direct relation" type, weight $2^2 = 4$;
111101 : "Direct relation" type, weight $2^3 = 8$;
110111 : "Direct relation" type, weight $2^3 = 8$;
110100 : "Direct relation" type, weight $2^3 = 8$;
100101 : "Direct relation" type, weight $2^3 = 8$;
010100 : "Direct relation" type, weight $2^2 = 4$;
100001 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^1 = 2$;
000000 : "Direct relation" type, weight $2^0 = 1$;
Positive Vote = 4 + 8 + 8 + 8 + 8 + 4 + 4 + 2 + 1 = 47;
Negative Vote = 0;

Therefore, probability $p_1 = 47 / (47 + 0) = 1.000$, and $p_0 = 0 / (47 + 0) = 0.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.000$, and $s_0 = p_1 \times c_0 = 0.500$.
Since s_1 is smaller, we will assign 110101 to be positive, which is correct.

For the 8th unknown example <011101>:

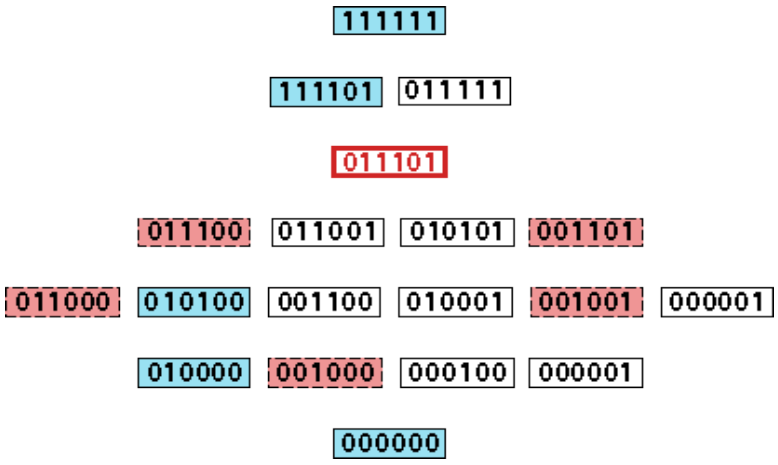

Figure 8.9 Examples in Ordered Relations with <011101>.

111111 : "Direct relation" type, weight $2^2 = 4$;
111101 : "Direct relation" type, weight $2^3 = 8$;
010100 : "Indirect relation" type (cutting example 011100), weight 0;
010000 : "Indirect relation" type (cutting example 011100), weight 0;
000000 : "Indirect relation" type (cutting example 011100), weight 0;
Positive Vote = 4 + 8 = 12;
011100 : "Direct relation" type, weight $2^3 = 8$;
001101 : "Direct relation" type, weight $2^3 = 8$;
011000 : "Direct relation" type, weight $2^2 = 4$;
001001 : "Direct relation" type, weight $2^2 = 4$;
001000 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 8 + 8 + 4 + 4 + 2 = 26;

Therefore, probability $p_1 = 12 / (12 + 26) = 0.316$, and $p_0 = 12 / (12 + 26) = 0.684$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.342$, and $s_0 = p_1 \times c_0 = 0.158$.
Since s_0 is smaller, we will assign 011101 to be negative, which is correct.

For the 9th unknown example <101011>:

111111 : "Direct relation" type, weight $2^2 = 4$;
101111 : "Direct relation" type, weight $2^3 = 8$;
101001 : "Direct relation" type, weight $2^3 = 8$;

100001 : "Direct relation" type, weight $2^2 = 4$;
000011 : "Direct relation" type, weight $2^2 = 4$;
000000 : "Indirect relation" type (cutting example 101010), weight 0;
Positive Vote = 4 + 8 + 8 + 4 + 4 = 28;
101010 : "Direct relation" type, weight $2^3 = 8$;
100010 : "Direct relation" type, weight $2^2 = 4$;
001010 : "Direct relation" type, weight $2^2 = 4$;
001001 : "Indirect relation" type (cutting example 101001), weight 0;
001000 : "Indirect relation" type (cutting example 101001), weight 0;
000010 : "Indirect relation" type (cutting example 000011), weight 0;
Negative Vote = 8 + 4 + 4 = 16;

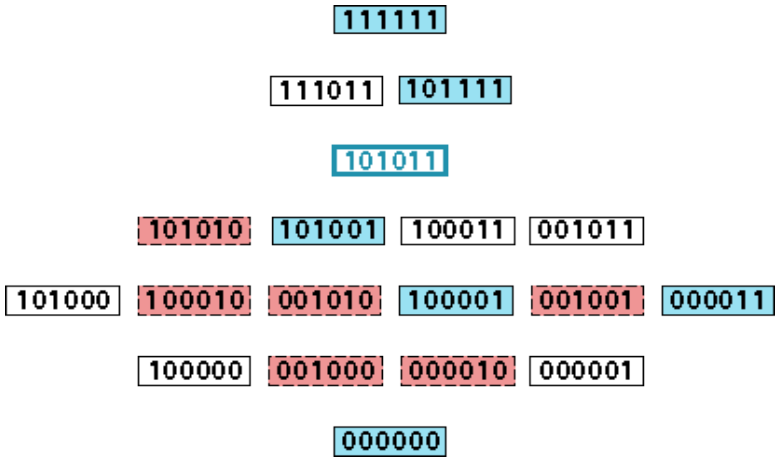

Figure 8.10 Examples in Ordered Relations with <101011>.

Therefore, probability $p_1 = 28 / (28 + 16) = 0.636$, and $p_0 = 16 / (28 + 16) = 0.364$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.182$, and $s_0 = p_1 \times c_0 = 0.318$.
Since s_1 is smaller, we will assign 101011 to be positive, which is correct.

For the 10th unknown example <100111>:

111111 : "Direct relation" type, weight $2^2 = 4$;
110111 : "Direct relation" type, weight $2^3 = 8$;
101111 : "Direct relation" type, weight $2^3 = 8$;
100110 : "Direct relation" type, weight $2^3 = 8$;
100101 : "Direct relation" type, weight $2^3 = 8$;
000111 : "Direct relation" type, weight $2^3 = 8$;

100001 : "Direct relation" type, weight $2^2 = 4$;
000011 : "Direct relation" type, weight $2^2 = 4$;
000000 : "Indirect relation" type (cutting example 100010), weight 0;
Positive Vote = 4 + 8 + 8 + 8 + 8 + 8 + 4 + 4 = 52;
100010 : "Indirect relation" type (cutting example 100110), weight 0;
000010 : "Indirect relation" type (cutting example 100110), weight 0;
Negative Vote = 0;

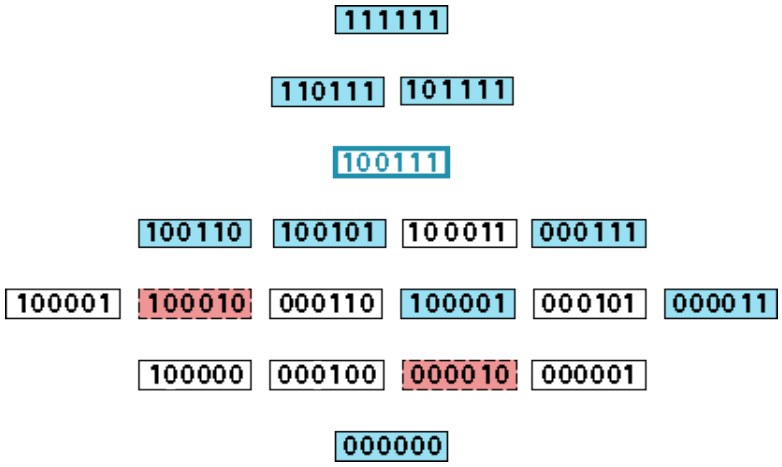

Figure 8.11 Examples in Ordered Relations with <100111>.

Therefore, probability $p_1 = 52 / (52 + 0) = 1.000$, and $p_0 = 0 / (52 + 0) = 0.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.000$, and $s_0 = p_1 \times c_0 = 0.500$.
Since s_1 is smaller, we will assign 100111 to be positive, which is correct.

For the 11th unknown example <001111>:

111111 : "Direct relation" type, weight $2^2 = 4$;
101111 : "Direct relation" type, weight $2^3 = 8$;
000111 : "Direct relation" type, weight $2^3 = 8$;
000011 : "Direct relation" type, weight $2^2 = 4$;
000000 : "Indirect relation" type (cutting example 001110), weight 0;
Positive Vote = 4 + 8 + 8 + 4 = 24;
001110 : "Direct relation" type, weight $2^3 = 8$;
001101 : "Direct relation" type, weight $2^3 = 8$;
001010 : "Direct relation" type, weight $2^2 = 4$;
001001 : "Direct relation" type, weight $2^2 = 4$;

001000 : "Direct relation" type, weight $2^1 = 2$;
000010 : "Indirect relation" type (cutting example 000111), weight 0;
Negative Vote = 8 + 8 + 4 + 4 + 2 = 26;

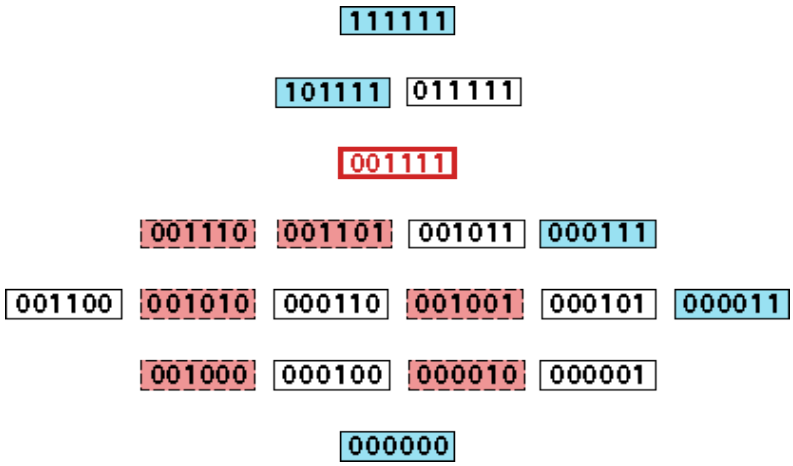

Figure 8.12 Examples in Ordered Relations with <001111>.

Therefore, probability $p_1 = 24 / (24 + 26) = 0.480$, and $p_0 = 26 / (24 + 26) = 0.520$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.26$, and $s_0 = p_1 \times c_0 = 0.24$.
Since s_0 is smaller, we will assign 001111 to be negative, which is correct.

For the 12th unknown example <111000>:

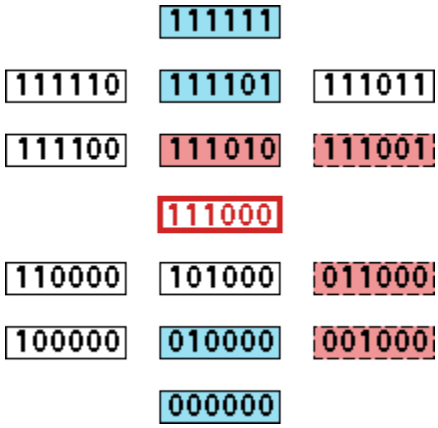

Figure 8.13 Examples in Ordered Relations with <111000>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
010000 : "Indirect relation" type (cutting example 011000), weight 0;
000000 : "Indirect relation" type (cutting example 011000), weight 0;

Positive Vote = 0;

111010: "Direct relation" type, weight $2^2 = 4$;

111001: "Direct relation" type, weight $2^2 = 4$;

011000: "Direct relation" type, weight $2^2 = 4$;

001000: "Direct relation" type, weight $2^1 = 2$;

Negative Vote = 4 + 4 + 4 + 2 = 14;

Therefore, probability $p_1 = 0 / (0 + 14) = 0.000$, and $p_0 = 14 / (0 + 14) = 1.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.500$, and $s_0 = p_1 \times c_0 = 0.000$.
Since s_0 is smaller, we will assign 111000 to be negative, which is correct.

For the 13th unknown example <101100>:

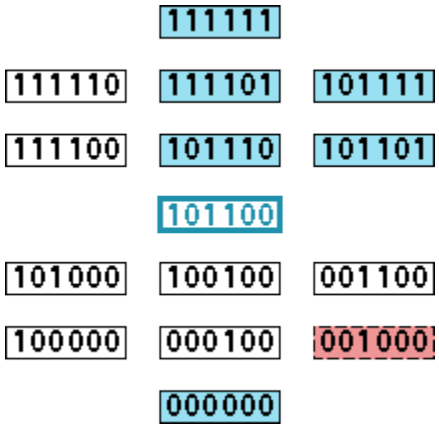

Figure 8.14 Examples in Ordered Relations with <101100>.

111111: "Direct relation" type, weight $2^0 = 1$;

111101: "Direct relation" type, weight $2^1 = 2$;

101111: "Direct relation" type, weight $2^1 = 2$;

101110: "Direct relation" type, weight $2^2 = 4$;

101101: "Direct relation" type, weight $2^2 = 4$;

000000: "Indirect relation" type (cutting example 001000), weight 0;

Positive Vote = 1 + 2 + 2 + 4 + 4 = 13;

001000: "Direct relation" type, weight $2^1 = 2$;

Negative Vote = 2;

Therefore, probability $p_1 = 13 / (13 + 2) = 0.867$, and $p_0 = 2 / (13 + 2) = 0.133$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.0665$, and $s_0 = p_1 \times c_0 = 0.4335$.
Since s_1 is smaller, we will assign 101100 to be positive, which is correct.

For the 14th unknown example <110010>:

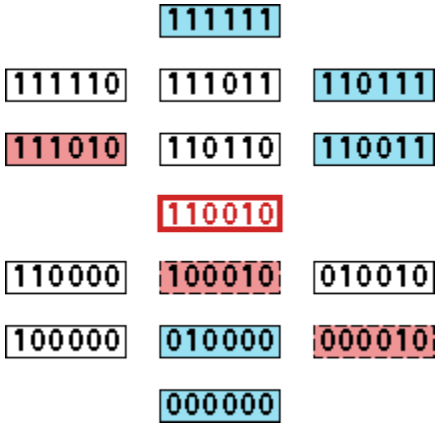

Figure 8.15 Examples in Ordered Relations with <110010>.

111111 : "Indirect relation" type (cutting example 111010), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
110011 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^1 = 2$;
000000 : "Indirect relation" type (cutting example 100010), weight 0;
Positive Vote = 2 + 4 + 2 = 8;
111010 : "Direct relation" type, weight $2^2 = 4$;
100010 : "Direct relation" type, weight $2^2 = 4$;
000010 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 4 + 4 + 2 = 10;

Therefore, probability $p_1 = 8 / (8 + 10) = 0.444$, and $p_0 = 10 / (8 + 10) = 0.556$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.278$, and $s_0 = p_1 \times c_0 = 0.222$.
Since s_0 is smaller, we will assign 110010 to be negative, which is correct.

For the 15th unknown example <011010>:

111111 : "Indirect relation" type (cutting example 011011), weight 0;
010000 : "Indirect relation" type (cutting example 011000), weight 0;
000000 : "Indirect relation" type (cutting example 011000), weight 0;
Positive Vote = 0;
111010 : "Direct relation" type, weight $2^2 = 4$;
011011 : "Direct relation" type, weight $2^2 = 4$;

011000 : "Direct relation" type, weight $2^2 = 4$;
001010 : "Direct relation" type, weight $2^2 = 4$;
001000 : "Direct relation" type, weight $2^1 = 2$;
000010 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 4 + 4 + 4 + 4 + 2 + 2 = 20;

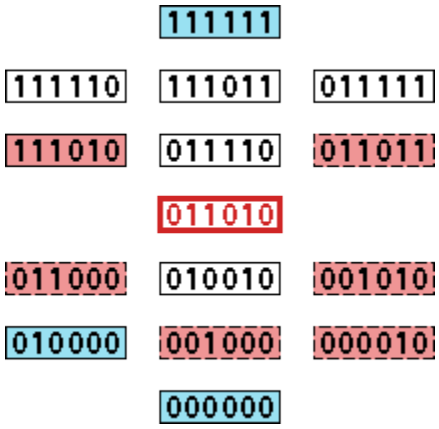

Figure 8.16 Examples in Ordered Relations with <011010>.

Therefore, probability $p_1 = 0 / (0 + 20) = 0.000$, and $p_0 = 20 / (0 + 20) = 1.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.500$, and $s_0 = p_1 \times c_0 = 0.000$.
Since s_0 is smaller, we will assign 011010 to be negative, which is correct.

For the 16th unknown example <010110>:

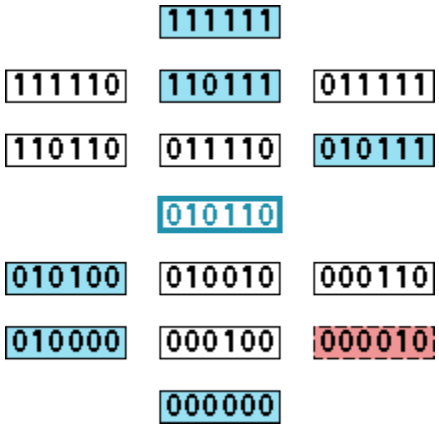

Figure 8.17 Examples in Ordered Relations with <010110>.

111111 : "Direct relation" type, weight $2^0 = 1$;
110111 : "Direct relation" type, weight $2^1 = 2$;

010111 : "Direct relation" type, weight $2^2 = 4$;
010100 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^1 = 2$;
000000 : "Indirect relation" type (cutting example 011000), weight 0;
Positive Vote = 1 + 2 + 4 + 4 + 2 = 13;
000010 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 2;

Therefore, probability $p_1 = 13 / (13 + 2) = 0.867$, and $p_0 = 2 / (13 + 2) = 0.133$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.0665$, and $s_0 = p_1 \times c_0 = 0.4335$.
Since s_1 is smaller, we will assign 010110 to be positive, which is correct.

For the 17th unknown example <110001>:

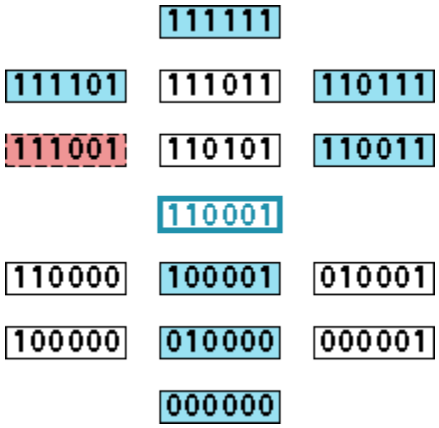

Figure 8.18 Examples in Ordered Relations with <110001>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
110011 : "Direct relation" type, weight $2^2 = 4$;
100001 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^1 = 2$;
000000 : "Direct relation" type, weight $2^0 = 1$;
Positive Vote = 2 + 4 + 4 + 2 + 1 = 13;
111001 : "Direct relation" type, weight $2^2 = 4$;
Negative Vote = 4;

Therefore, probability $p_1 = 13 / (13 + 4) = 0.765$, and $p_0 = 4 / (13 + 4) = 0.235$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.1175$, and $s_0 = p_1 \times c_0 = 0.3825$.
Since s_1 is smaller, we will assign 110001 to be positive, which is correct.

For the 18th unknown example <011001>:

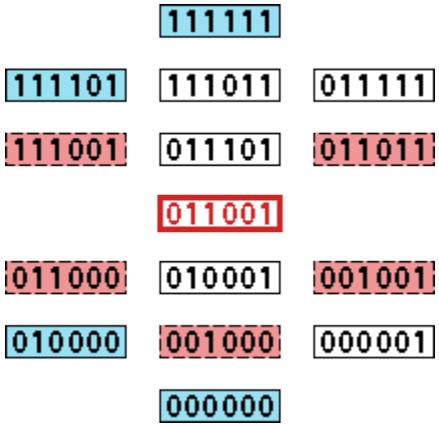

Figure 8.19 Examples in Ordered Relations with <011001>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
010000 : "Indirect relation" type (cutting example 011000), weight 0;
000000 : "Indirect relation" type (cutting example 011000), weight 0;
Positive Vote = 0;
111001 : "Direct relation" type, weight $2^2 = 4$;
011011 : "Direct relation" type, weight $2^2 = 4$;
011000 : "Direct relation" type, weight $2^2 = 4$;
001001 : "Direct relation" type, weight $2^2 = 4$;
001000 : "Direct relation" type, weight $2^1 = 2$;
Negative Vote = 4 + 4 + 4 + 4 + 2 = 18;

Therefore, probability $p_1 = 0 / (0 + 18) = 0.000$, and $p_0 = 18 / (0 + 18) = 1.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.500$, and $s_0 = p_1 \times c_0 = 0.000$.
Since s_0 is smaller, we will assign 011001 to be negative, which is correct.

For the 19th unknown example <010101>:

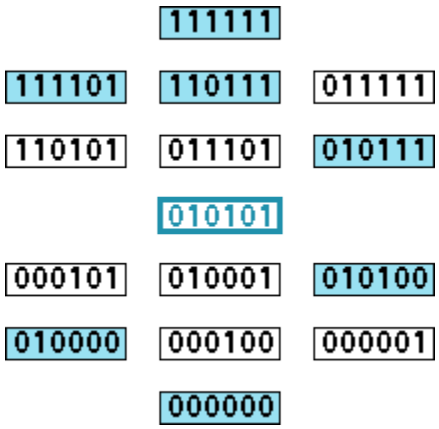

Figure 8.20 Examples in Ordered Relations with <010101>.

111111 : "Direct relation" type, weight $2^0 = 1$;
111101 : "Direct relation" type, weight $2^1 = 2$;
110111 : "Direct relation" type, weight $2^1 = 2$;
010111 : "Direct relation" type, weight $2^2 = 4$;
010100 : "Direct relation" type, weight $2^2 = 4$;
010000 : "Direct relation" type, weight $2^2 = 2$;
000000 : "Direct relation" type, weight $2^2 = 1$;
Positive Vote = 1+ 2 + 2 + 4 + 4 + 2 + 1 = 16;
Negative Vote = 0;

Therefore, probability $p_1 = 16 / (16 + 0) = 1.000$, and $p_0 = 0 / (16 + 0) = 0.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.000$, and $s_0 = p_1 \times c_0 = 0.500$.
Since s_1 is smaller, we will assign 010101 to be positive, which is correct.

For the 20th unknown example <100011>:

111111 : "Direct relation" type, weight $2^0 = 1$;
110111 : "Direct relation" type, weight $2^1 = 2$;
101111 : "Direct relation" type, weight $2^1 = 2$;
110011 : "Direct relation" type, weight $2^2 = 4$;
100001 : "Direct relation" type, weight $2^2 = 4$;
000011 : "Direct relation" type, weight $2^2 = 4$;
000000 : "Indirect relation" type (cutting example 100010), weight 0;

Positive Vote = 1 + 2 + 2 + 4 + 4 + 4 = 17;

100010 : "Direct relation" type, weight $2^2 = 4$;

000010 : "Indirect relation" type (cutting example 000011), weight 0;

Negative Vote = 4;

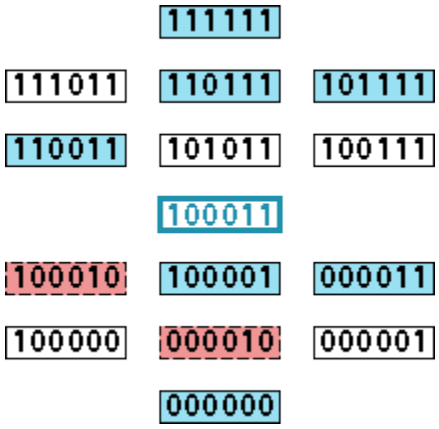

Figure 8.21 Examples in Ordered Relations with <100011>.

Therefore, probability $p_1 = 17 / (17 + 4) = 0.810$, and $p_0 = 4 / (17 + 4) = 0.190$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.095$, and $s_0 = p_1 \times c_0 = 0.405$.
Since s_1 is smaller, we will assign 100011 to be positive, which is correct.

For the 21st unknown example <001011>:

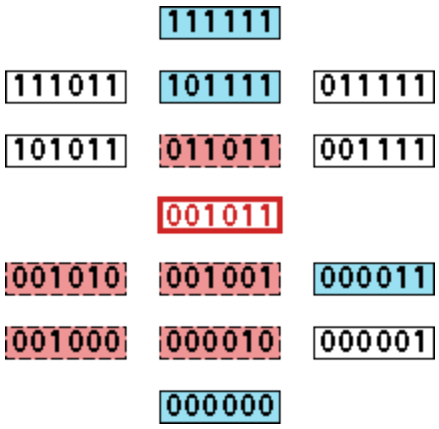

Figure 8.22 Examples in Ordered Relations with <001011>.

111111 : "Indirect relation" type (cutting example 011011), weight 0;

101111 : "Direct relation" type, weight $2^1 = 2$;

000011 : "Direct relation" type, weight $2^2 = 4$;

000000 : "Indirect relation" type (cutting example 001010), weight 0;

Positive Vote = 2 + 4 = 6;

011011 : "Direct relation" type, weight $2^2 = 4$;

001010 : "Direct relation" type, weight $2^2 = 4$;

001001 : "Direct relation" type, weight $2^2 = 4$;

001000 : "Direct relation" type, weight $2^1 = 2$;

000010 : "Indirect relation" type (cutting example 000011), weight 0;

Negative Vote = 4 + 4 + 4 + 2 = 14;

Therefore, probability $p_1 = 6 / (6 + 14) = 0.300$, and $p_0 = 14 / (6 + 14) = 0.700$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.350$, and $s_0 = p_1 \times c_0 = 0.150$.
Since s_0 is smaller, we will assign 001011 to be negative, which is correct.

For the 22nd unknown example <000101>:

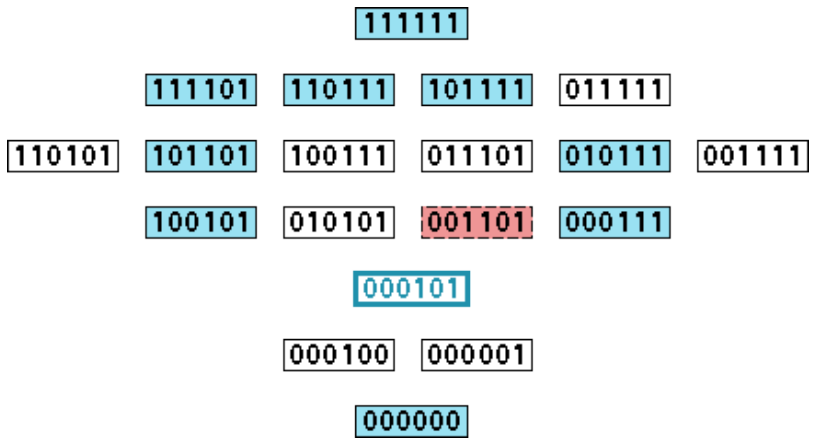

Figure 8.23 Examples in Ordered Relations with <000101>.

111111 : "Indirect relation" type (cutting example 001101), weight 0;

111101 : "Indirect relation" type (cutting example 001101), weight 0;

110111 : "Direct relation" type, weight $2^1 = 2$;

101111 : "Indirect relation" type (cutting example 001101), weight 0;

101101 : "Indirect relation" type (cutting example 001101), weight 0;

010111 : "Direct relation" type, weight $2^2 = 4$;

100101 : "Direct relation" type, weight $2^3 = 8$;

000111 : "Direct relation" type, weight $2^3 = 8$;

000000 : "Direct relation" type, weight $2^2 = 4$;

Positive Vote = 2 + 4 + 8 + 8 + 4 = 26;

001101 : "Direct relation" type, weight $2^3 = 8$;

Negative Vote = 8;

Therefore, probability $p_1 = 26 / (26 + 8) = 0.765$, and $p_0 = 8 / (26 + 8) = 0.235$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.1175$, and $s_0 = p_1 \times c_0 = 0.3825$.
Since s_1 is smaller, we will assign 000101 to be positive, which is correct.

For the 23rd unknown example <010001>:

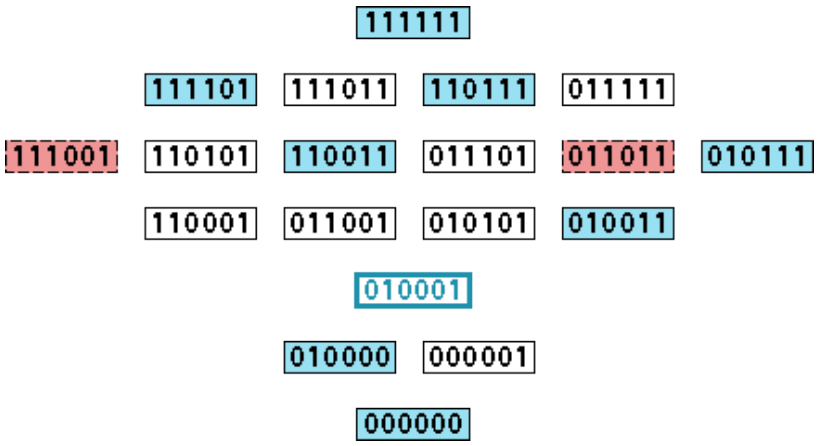

Figure 8.24 Examples in Ordered Relations with <010001>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
110011 : "Direct relation" type, weight $2^2 = 4$;
010111 : "Direct relation" type, weight $2^2 = 4$;
010011 : "Direct relation" type, weight $2^3 = 8$;
010000 : "Direct relation" type, weight $2^3 = 8$;
000000 : "Direct relation" type, weight $2^2 = 4$;
Positive Vote = 2 + 4 + 4 + 8 + 8 + 4 = 30;
111001 : "Direct relation" type, weight $2^2 = 4$;
011011 : "Indirect relation" type (cutting example 010011), weight 0;
Negative Vote = 4;

Therefore, probability $p_1 = 30 / (30 + 4) = 0.882$, and $p_0 = 4 / (30 + 4) = 0.118$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.059$, and $s_0 = p_1 \times c_0 = 0.441$.
Since s_1 is smaller, we will assign 010001 to be positive, which is correct.

For the 24th unknown example <000110>:

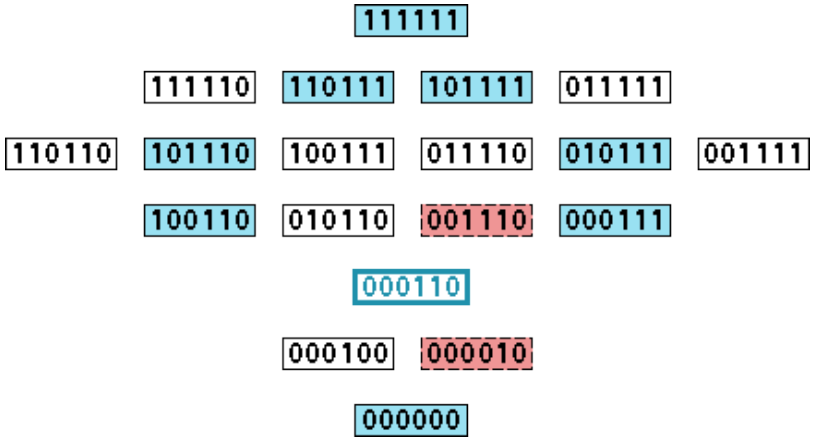

Figure 8.25 Examples in Ordered Relations with <000110>.

111111 : "Indirect relation" type (cutting example 001110), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
101111 : "Indirect relation" type (cutting example 001110), weight 0;
101110 : "Indirect relation" type (cutting example 001110), weight 0;
010111 : "Direct relation" type, weight $2^2 = 4$;
100110 : "Direct relation" type, weight $2^3 = 8$;
000111 : "Direct relation" type, weight $2^3 = 8$;
000000 : "Indirect relation" type (cutting example 000010), weight 0;
Positive Vote = 2 + 4 + 8 + 8 = 22;
001110 : "Direct relation" type, weight $2^3 = 8$;
000010 : "Direct relation" type, weight $2^3 = 8$;
Negative Vote = 8 + 8 = 16;

Therefore, probability $p_1 = 22 / (22 + 16) = 0.579$, and $p_0 = 16 / (22 + 16) = 0.421$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.2105$, and $s_0 = p_1 \times c_0 = 0.2895$.
Since s_1 is smaller, we will assign 000110 to be positive, which is correct.

For the 25th unknown example <010010>:

111111 : "Indirect relation" type (cutting example 011011), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
110011 : "Direct relation" type, weight $2^2 = 4$;
010111 : "Direct relation" type, weight $2^2 = 4$;

010011 : "Direct relation" type, weight $2^3 = 8$;
010000 : "Direct relation" type, weight $2^3 = 8$;
000000 : "Indirect relation" type (cutting example 000010), weight 0;
Positive Vote = 2 + 4 + 4 + 8 + 8 = 26;
111010 : "Direct relation" type, weight $2^2 = 4$;
011011 : "Indirect relation" type (cutting example 010011), weight 0;
000010 : "Direct relation" type, weight $2^3 = 8$;
Negative Vote = 4 + 8 = 12;

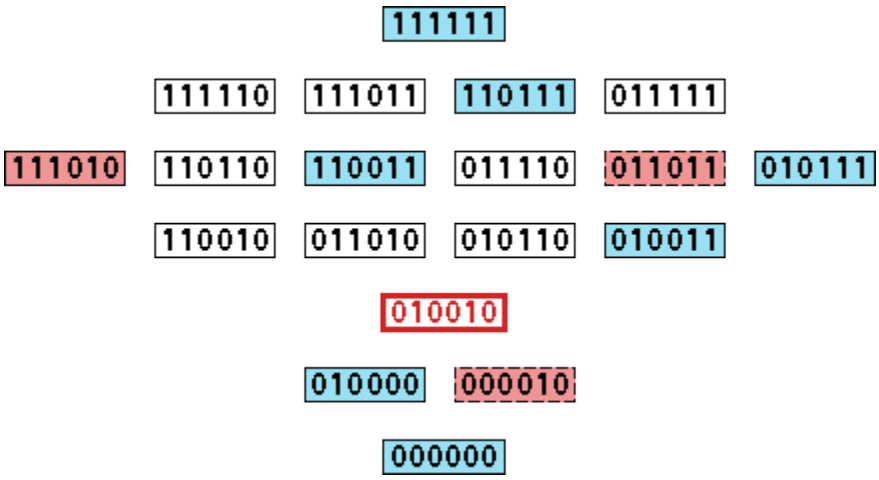

Figure 8.26 Examples in Ordered Relations with <010010>.

Therefore, probability $p_1 = 26 / (26 + 12) = 0.684$, and $p_0 = 12 / (26 + 12) = 0.316$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.158$, and $s_0 = p_1 \times c_0 = 0.342$.
Since s_1 is smaller, we will assign 010010 to be positive, which is wrong. Its actual class should be negative.

For the 26th unknown example <001100>:

111111 : "Indirect relation" type (cutting example 011100), weight 0;
111101 : "Indirect relation" type (cutting example 011100), weight 0;
101111 : "Indirect relation" type (cutting example 001110), weight 0;
101110 : "Indirect relation" type (cutting example 001110), weight 0;
101101 : "Indirect relation" type (cutting example 001101), weight 0;
000000 : "Indirect relation" type (cutting example 001000), weight 0;
Positive Vote = 0;

011100 : "Direct relation" type, weight $2^3 = 8$;
001110 : "Direct relation" type, weight $2^3 = 8$;
001101 : "Direct relation" type, weight $2^3 = 8$;
001000 : "Direct relation" type, weight $2^3 = 8$;
Negative Vote = 8 + 8 + 8 + 8 = 32;

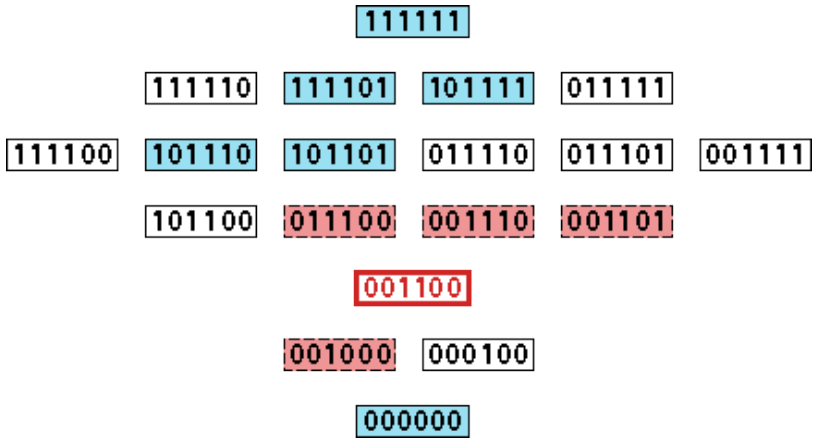

Figure 8.27 Examples in Ordered Relations with <001100>.

Therefore, probability $p_1 = 0 / (0 + 32) = 0.000$, and $p_0 = 32 / (0 + 32) = 1.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.500$, and $s_0 = p_1 \times c_0 = 0.000$.
Since s_0 is smaller, we will assign 001100 to be negative, which is correct.

For the 27th unknown example <100100>:

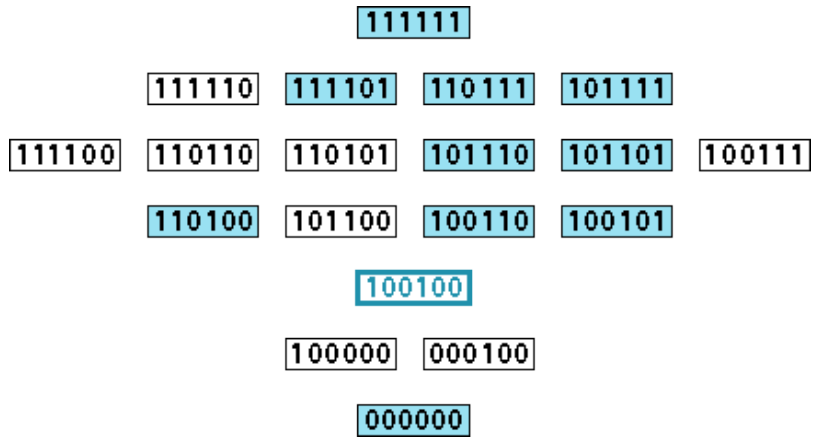

Figure 8.28 Examples in Ordered Relations with <100100>.

111111 : "Direct relation" type, weight $2^0 = 1$;
111101 : "Direct relation" type, weight $2^1 = 2$;

110111 : "Direct relation" type, weight $2^1 = 2$;
101111 : "Direct relation" type, weight $2^1 = 2$;
101110 : "Direct relation" type, weight $2^2 = 4$;
101101 : "Direct relation" type, weight $2^2 = 4$;
110100 : "Direct relation" type, weight $2^3 = 8$;
100110 : "Direct relation" type, weight $2^3 = 8$;
100101 : "Direct relation" type, weight $2^3 = 8$;
000000 : "Direct relation" type, weight $2^2 = 4$;
Positive Vote = 1 + 2 + 2 + 2 + 4 + 4 + 8 + 8 + 8 + 4 = 43;
Negative Vote = 0;

Therefore, probability $p_1 = 43 / (43 + 0) = 1.000$, and $p_0 = 0 / (43 + 0) = 0.000$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.000$, and $s_0 = p_1 \times c_0 = 0.500$.
Since s_1 is smaller, we will assign 100100 to be positive, which is correct.

For the 28th unknown example <101000>:

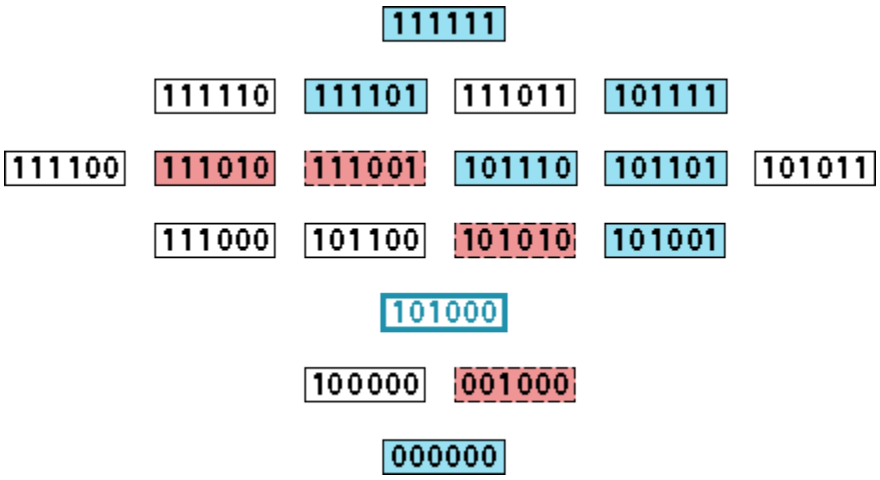

Figure 8.29 Examples in Ordered Relations with <101000>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
101111 : "Indirect relation" type (cutting example 101010), weight 0;
101110 : "Indirect relation" type (cutting example 101010), weight 0;
101101 : "Direct relation" type, weight $2^2 = 4$;
101001 : "Direct relation" type, weight $2^3 = 8$;
000000 : "Indirect relation" type (cutting example 001000), weight 0;

Positive Vote = 4 + 8 = 12;

111010 : "Direct relation" type, weight $2^2 = 4$;

111001 : "Indirect relation" type (cutting example 101001), weight 0;

101010 : "Direct relation" type, weight $2^3 = 8$;

001000 : "Direct relation" type, weight $2^3 = 8$;

Negative Vote = 4 + 8 + 8 = 20;

Therefore, probability $p_1 = 12 / (12 + 20) = 0.375$, and $p_0 = 20 / (12 + 20) = 0.625$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.3125$, and $s_0 = p_1 \times c_0 = 0.1875$.

Since s_0 is smaller, we will assign 101000 to be negative, which is wrong. Its actual class should be positive.

For the 29th unknown example <110000>:

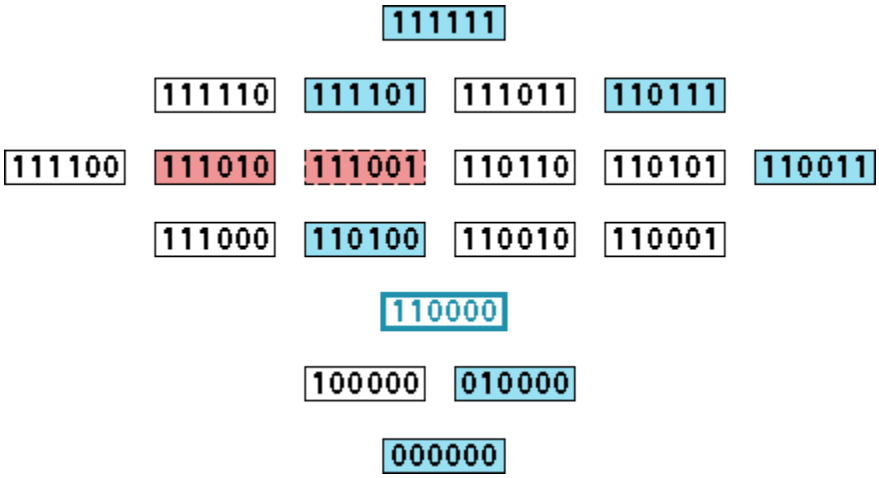

Figure 8.30 Examples in Ordered Relations with <110000>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;

111101 : "Indirect relation" type (cutting example 111001), weight 0;

110111 : "Direct relation" type, weight $2^1 = 2$;

110011 : "Direct relation" type, weight $2^2 = 4$;

110100 : "Direct relation" type, weight $2^3 = 8$;

010000 : "Direct relation" type, weight $2^3 = 8$;

000000 : "Direct relation" type, weight $2^2 = 4$;

Positive Vote = 2 + 4 + 8 + 8 + 4 = 26;

111010 : "Direct relation" type, weight $2^2 = 4$;

111001 : "Direct relation" type, weight $2^2 = 4$;

Negative Vote = 4 + 4 = 8;

Therefore, probability p_1 = 26 / (26 + 8) = 0.765, and p_0 = 8 / (26 + 8) = 0.235.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.1175$, and $s_0 = p_1 \times c_0 = 0.3825$.
Since s_1 is smaller, we will assign 110000 to be positive, which is correct.

For the 30th unknown example <100000>:

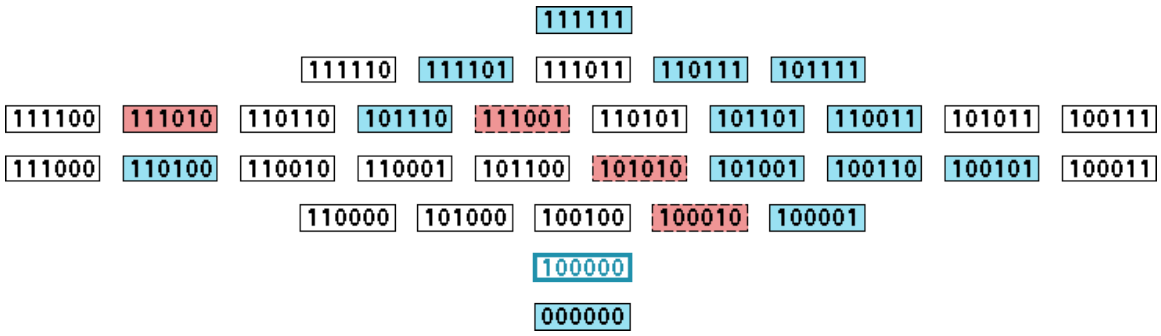

Figure 8.31 Examples in Ordered Relations with <100000>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
110111 : "Indirect relation" type (cutting example 100010), weight 0;
101111 : "Indirect relation" type (cutting example 101010), weight 0;
101110 : "Indirect relation" type (cutting example 101010), weight 0;
101101 : "Direct relation" type, weight 2^2 = 4;
110011 : "Indirect relation" type (cutting example 100010), weight 0;
110100 : "Direct relation" type, weight 2^3 = 8;
101001 : "Direct relation" type, weight 2^3 = 8;
100110 : "Indirect relation" type (cutting example 100010), weight 0;
100101 : "Direct relation" type, weight 2^3 = 8;
100001 : "Direct relation" type, weight 2^4 = 16;
000000 : "Direct relation" type, weight 2^4 = 16;
Positive Vote = 4 + 8 + 8 + 8 + 16 + 16 = 60;
111010 : "Direct relation" type, weight 2^2 = 4;
111001 : "Indirect relation" type (cutting example 101001), weight 0;
101010 : "Direct relation" type, weight 2^3 = 8;
100010 : "Direct relation" type, weight 2^4 = 16;

Negative Vote = 4 + 8 + 16 = 28;

Therefore, probability p_1 = 60 / (60 + 28) = 0.682, and p_0 = 28 / (60 + 28) = 0.318.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.159$, and $s_0 = p_1 \times c_0 = 0.341$.
Since s_1 is smaller, we will assign 100000 to be positive, which is correct.

For the 31st unknown example <000100>:

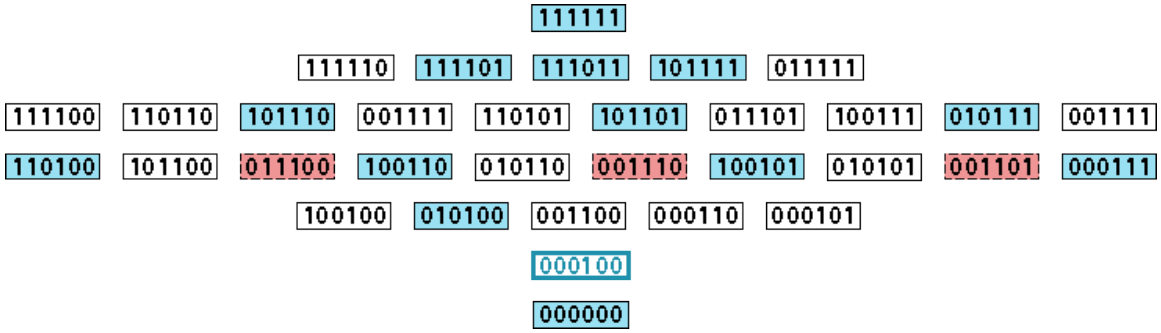

Figure 8.32 Examples in Ordered Relations with <000100>.

111111 : "Indirect relation" type (cutting example 011100), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
110111 : "Direct relation" type, weight 2^1 = 2;
101111 : "Indirect relation" type (cutting example 001110), weight 0;
101110 : "Indirect relation" type (cutting example 001110), weight 0;
101101 : "Indirect relation" type (cutting example 001101), weight 0;
010111 : "Direct relation" type, weight 2^2 = 4;
110100 : "Direct relation" type, weight 2^3 = 8;
100110 : "Direct relation" type, weight 2^3 = 8;
100101 : "Direct relation" type, weight 2^3 = 8;
000111 : "Direct relation" type, weight 2^3 = 8;
010100 : "Direct relation" type, weight 2^4 = 16;
000000 : "Direct relation" type, weight 2^4 = 16;
Positive Vote = 2 + 4 + 8 + 8 + 8 + 8 + 16 + 16 = 70;
011100 : "Indirect relation" type (cutting example 010100), weight 0;
001110 : "Direct relation" type, weight 2^3 = 8;
001101 : "Direct relation" type, weight 2^3 = 8;
Negative Vote = 8 + 8 = 16;

Therefore, probability p_1 = 70 / (70 + 16) = 0.814, and p_0 = 16 / (70 + 16) = 0.186.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.093$, and $s_0 = p_1 \times c_0 = 0.407$.
Since s_1 is smaller, we will assign 000100 to be positive, which is correct.

For the 32nd unknown example <000001>:

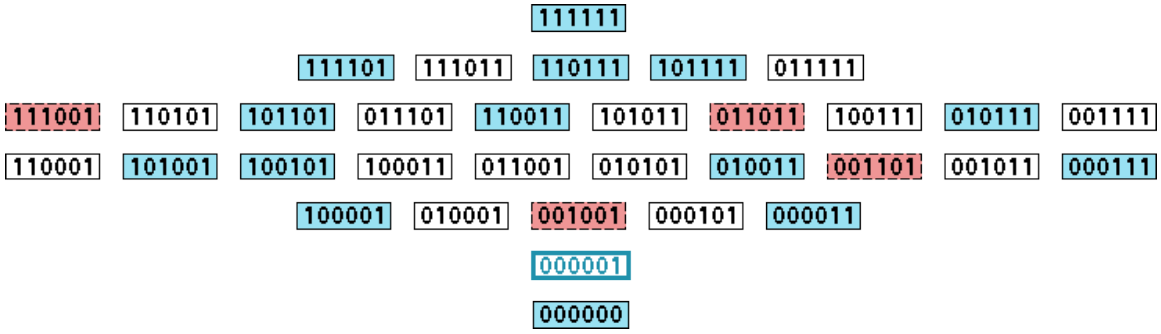

Figure 8.33 Examples in Ordered Relations with <000001>.

111111 : "Indirect relation" type (cutting example 111001), weight 0;
111101 : "Indirect relation" type (cutting example 111001), weight 0;
110111 : "Direct relation" type, weight $2^1 = 2$;
101111 : "Indirect relation" type (cutting example 001101), weight 0;
101101 : "Indirect relation" type (cutting example 001101), weight 0;
110011 : "Direct relation" type, weight $2^2 = 4$;
010111 : "Direct relation" type, weight $2^2 = 4$;
101001 : "Indirect relation" type (cutting example 001001), weight 0;
100101 : "Direct relation" type, weight $2^3 = 8$;
010011 : "Direct relation" type, weight $2^3 = 8$;
000111 : "Direct relation" type, weight $2^3 = 8$;
100001 : "Direct relation" type, weight $2^4 = 16$;
000011 : "Direct relation" type, weight $2^4 = 16$;
000000 : "Direct relation" type, weight $2^4 = 16$;
Positive Vote = 2+ 4 + 4 + 8 + 8 + 8 + 16 + 16 + 16 = 82;
111001 : "Indirect relation" type (cutting example 101001), weight 0;
011011 : "Indirect relation" type (cutting example 010011), weight 0;
001101 : "Direct relation" type, weight $2^3 = 8$;
001001 : "Direct relation" type, weight $2^4 = 16$;
Negative Vote = 8 + 16 = 24;

Therefore, probability $p_1 = 82 / (82 + 24) = 0.774$, and $p_0 = 24 / (82 + 24) = 0.226$.
Expected misclassification cost $s_1 = p_0 \times c_1 = 0.113$, and $s_0 = p_1 \times c_0 = 0.387$.
Since s_1 is smaller, we will assign 

to be positive, which is correct.

CHAPTER 9. COMPARISON WITH EXISTING ALGORITHMS

9.1 Results of Comparison

Now we will run the 10 synthetic datasets we generated in Chapter 6 on Weka, and compare the results with the *M** algorithm. There are, in total, 75 classifiers in Weka that can be applied to binary systems, including 8 Bayesian methods, 7 Function methods, 5 Lazy methods, 29 Meta methods, 5 Miscellaneous methods, 12 Decision Tree methods, and 9 Rule Induction methods. We will compare the *M** algorithm result to all of these 75 existing classifiers. All the classifiers that need parameters were run on their default parameters.

As mentioned earlier, because not all classifiers have the ability to adjust to the change of misclassification costs, here we assume the misclassification costs take the same value, i.e., $(c_1, c_0) = (0.5, 0.5)$.

For Dataset 1:

Please recall that we have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 391$; $m_1^- = 109$; $m_2 = 100$; $m_2^+ = 78$; $m_2^- = 23$.

$$f_1 = (\overline{x_1} \vee \overline{x_2} \vee x_5 \vee x_8 \vee \overline{x_9}) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4} \vee x_7 \vee \overline{x_{10}}) \wedge (x_2 \vee x_3 \vee x_6 \vee x_{10}) \wedge (x_3 \vee \overline{x_5} \vee x_7 \vee \overline{x_9}) \wedge (x_4 \vee \overline{x_6} \vee \overline{x_8} \vee \overline{x_9} \vee x_{10})$$

Table 9.1-1 Comparison with Weka Results on Dataset 1.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		77	15	92	1	7	8
Bayes	AODE	74	8	82	4	14	18
	AODEsr	74	8	82	4	14	18
	BayesNet	74	9	83	4	13	17
	HNB	75	9	84	3	13	16
	NaiveBayes	74	9	83	4	13	17
	NaiveBayesSimple	74	9	83	4	13	17
	NaiveBayesUpdateable	74	9	83	4	13	17
	WAODE	74	11	85	4	11	15
Function	Logistic	70	11	81	8	11	19
	MultiLayerPerceptron	68	12	80	10	10	20
	RBFNetwork	72	10	82	6	12	18
	SimpleLogistic	77	5	82	1	17	18
	SMO	77	5	82	1	17	18
	VotedPerceptron	70	7	77	8	15	23
	Winnow	42	19	61	36	3	39

(table con'd)

Lazy	IB1	69	17	86	9	5	14
	IBk	78	15	93	0	7	7
	KStar	78	15	93	0	7	7
	LBR	75	12	87	3	10	13
	LWL	78	0	78	0	22	22
Meta	AdaBoostM1	67	13	80	11	9	20
	AttributeSelectedClassifier	78	9	87	0	13	13
	Bagging	77	10	87	1	12	13
	ClassificationViaClustering	62	12	74	16	10	26
	ClassificationViaRegression	77	5	82	1	17	18
	CostSensitiveClassifier	78	0	78	0	22	22
	CVParameterSelection	78	0	78	0	22	22
	Dagging	78	2	80	0	20	20
	Decorate	76	16	92	2	6	8
	END	75	11	86	3	11	14
	EnsembleSelection	78	9	87	0	13	13
	FilteredClassifier	75	11	86	3	11	14
	Grading	78	0	78	0	22	22
	LogitBoost	73	11	84	5	11	16
	MetaCost	78	0	78	0	22	22
	MultiBoostAB	78	0	78	0	22	22
	MultiClassClassifier	70	11	81	8	11	19
	MultiScheme	78	0	78	0	22	22
	OrdinalClassClassifier	75	11	86	3	11	14
	RacedIncrementalLogitBoost	78	0	78	0	22	22
	RandomCommittee	77	15	92	1	7	8
	RandomSubSpace	78	0	78	0	22	22
	Stacking	78	0	78	0	22	22
	StackingC	78	0	78	0	22	22
	ThresholdSelector	70	11	81	8	11	19
	Vote	78	0	78	0	22	22
	ClassBalancedND	75	11	86	3	11	14
	DataNearBalancedND	75	11	86	3	11	14
	ND	75	11	86	3	11	14
Misc	HyperPipes	78	0	78	0	22	22
	MinMaxExtension	49	20	69	29	2	31
	OLM	63	15	78	15	7	22
	OSDL	49	21	70	29	1	30
	VFI	56	15	71	22	7	29
Trees	ADTree	60	13	73	18	9	27
	BFTree	76	13	89	2	9	11
	DecisionStump	78	0	78	0	22	22
	Id3	74	16	90	4	6	10
	J48	75	11	86	3	11	14
	J48graft	75	11	86	3	11	14
	LMT	73	14	87	5	8	13
	NBTree	78	12	90	0	10	10

(table con'd)

	RandomForest	77	14	91	1	8	9
	RandomTree	69	11	80	9	11	20
	REPTree	75	11	86	3	11	14
	SimpleCart	76	11	87	2	11	13
Rules	ConjunctiveRule	78	0	78	0	22	22
	DecisionTable	78	0	78	0	22	22
	JRip	78	22	100	0	0	0
	NNge	74	17	91	4	5	9
	OneR	78	0	78	0	22	22
	PART	78	17	95	0	5	5
	Prism	72	18	90	0	4	10*
	Ridor	72	11	83	6	11	17
	ZeroR	78	0	78	0	22	22

*Note: Classifier Rules.Prism has got 4 of unknowns wrong, and 6 of them were assigned as "unclassified" type. Since other classifiers do not have an "unclassified" class, for uniformity reason, 10 instead of 4 is put in the "wrong" column.

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 39 of all unknown examples wrong. The *M** algorithm gets 8 of all unknown examples wrong, and it ranks at the 5th place. Of all the 4 classifiers that outperform *M**, the distribution is:

Table 9.1-2 Outperforming classifiers on Dataset 1.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	2	0	0	0	2

For Dataset 2:

Recall that we have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 325$; $m_1^- = 175$; $m_2 = 100$; $m_2^+ = 60$; $m_2^- = 40$.

$$f_2 = (\overline{x_1} \vee x_8 \vee \overline{x_9}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_7 \vee \overline{x_8}) \wedge (x_2 \vee x_5 \vee x_6 \vee x_{10}) \wedge (\overline{x_3} \vee \overline{x_5} \vee \overline{x_7}) \wedge (x_4 \vee \overline{x_6} \vee \overline{x_7} \vee \overline{x_9} \vee x_{10}) \wedge (\overline{x_5} \vee \overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}})$$

Table 9.2-1 Comparison with Weka Results on Dataset 2.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		58	33	91	2	7	9
Bayes	AODE	57	22	79	3	18	21
	AODEsr	57	22	79	3	18	21
	BayesNet	55	16	71	5	24	29
	HNB	59	25	84	1	15	16
	NaiveBayes	55	16	71	5	24	29

(table con'd)

	NaiveBayesSimple	55	16	71	5	24	29
	NaiveBayesUpdateable	55	16	71	5	24	29
	WAODE	56	24	80	4	16	20
Function	Logistic	52	20	72	8	20	28
	MultiLayerPerceptron	59	32	91	1	8	9
	RBFNetwork	55	22	77	5	18	23
	SimpleLogistic	51	20	71	9	20	29
	SMO	52	19	71	8	21	29
	VotedPerceptron	54	16	70	6	24	30
	Winnow	43	20	63	17	20	37
Lazy	IB1	57	26	83	3	14	17
	IBk	60	29	89	0	11	11
	KStar	60	30	90	0	10	10
	LBR	56	19	75	4	21	25
	LWL	60	9	69	0	31	31
Meta	AdaBoostM1	53	19	72	7	21	28
	AttributeSelectedClassifier	58	27	85	2	13	15
	Bagging	58	30	88	2	10	12
	ClassificationViaClustering	35	24	59	25	16	41
	ClassificationViaRegression	60	29	89	0	11	11
	CostSensitiveClassifier	60	0	60	0	40	40
	CVParameterSelection	60	0	60	0	40	40
	Dagging	56	12	68	4	28	32
	Decorate	59	30	89	1	10	11
	END	60	30	90	0	10	10
	EnsembleSelection	60	29	89	0	11	11
	FilteredClassifier	60	30	90	0	10	10
	Grading	60	0	60	0	40	40
	LogitBoost	53	19	72	7	21	28
	MetaCost	60	0	60	0	40	40
	MultiBoostAB	56	9	65	4	31	35
	MultiClassClassifier	52	20	72	8	20	28
	MultiScheme	60	0	60	0	40	40
	OrdinalClassClassifier	60	30	90	0	10	10
	RacedIncrementalLogitBoost	60	0	60	0	40	40
	RandomCommittee	57	27	84	3	13	16
	RandomSubSpace	60	23	83	0	17	17
	Stacking	60	0	60	0	40	40
	StackingC	60	0	60	0	40	40
	ThresholdSelector	54	14	68	6	26	32
	Vote	60	0	60	0	40	40
	ClassBalancedND	60	30	90	0	10	10
	DataNearBalancedND	60	30	90	0	10	10
	ND	60	30	90	0	10	10
Misc	HyperPipes	60	0	60	0	40	40
	MinMaxExtension	0	40	40	60	0	60
	OLM	34	24	58	26	16	42
	OSDL	0	40	40	60	0	60

(table con'd)

	VFI	46	30	76	14	10	24
Trees	ADTree	57	26	83	3	14	17
	BFTree	60	28	88	0	12	12
	DecisionStump	60	0	60	0	40	40
	Id3	57	34	91	3	6	9
	J48	60	30	90	0	10	10
	J48graft	60	30	90	0	10	10
	LMT	56	30	86	4	10	14
	NBTree	60	34	94	0	6	6
	RandomForest	59	31	90	1	9	10
	RandomTree	46	24	70	14	16	30
	REPTree	60	29	89	0	11	11
	SimpleCart	60	30	90	0	10	10
Rules	ConjunctiveRule	60	0	60	0	40	40
	DecisionTable	56	28	84	4	12	16
	JRip	60	40	100	0	0	0
	NNge	54	33	87	6	7	13
	OneR	60	0	60	0	40	40
	PART	60	37	97	0	3	3
	Prism	58	35	93	0	5	7*
	Ridor	59	36	95	1	4	5
	ZeroR	60	0	60	0	40	40

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 60 of all unknown examples wrong. The *M** algorithm gets 9 of all unknown examples wrong, and it ranks at the 6th place. Of all the 5 classifiers that outperform *M**, the distribution is:

Table 9.2-2 Outperforming classifiers on Dataset 2.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	0	0	1	4

For Dataset 3:

Recall that we have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 228$; $m_1^- = 272$; $m_2 = 100$; $m_2^+ = 43$; $m_2^- = 57$.

$$f_3 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (x_1 \vee x_7 \vee \overline{x_8}) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_8 \vee x_9 \vee x_{10}) \wedge$$
$$(\overline{x_2} \vee \overline{x_9}) \wedge (\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_8 \vee \overline{x_9}) \wedge (x_3 \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee \overline{x_{10}})$$

Table 9.3-1 Comparison with Weka Results on Dataset 3.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		40	56	96	3	1	4
Bayes	AODE	35	45	80	8	12	20

(table con'd)

	AODEsr	35	45	80	8	12	20
	BayesNet	32	45	77	11	12	23
	HNB	40	50	90	3	7	10
	NaiveBayes	32	45	77	11	12	23
	NaiveBayesSimple	32	45	77	11	12	23
	NaiveBayesUpdateable	32	45	77	11	12	23
	WAODE	36	42	78	7	15	22
Function	Logistic	34	44	78	9	13	22
	MultiLayerPerceptron	41	54	95	2	3	5
	RBFNetwork	34	41	75	9	16	25
	SimpleLogistic	33	45	78	10	12	22
	SMO	34	40	74	9	17	26
	VotedPerceptron	32	41	73	11	16	27
	Winnow	43	0	43	0	57	57
Lazy	IB1	32	49	81	11	8	19
	IBk	41	51	92	2	6	8
	KStar	40	54	94	3	3	6
	LBR	35	51	86	8	6	14
	LWL	43	30	73	0	27	27
Meta	AdaBoostM1	32	44	76	11	13	24
	AttributeSelectedClassifier	39	42	81	4	15	19
	Bagging	40	53	93	3	4	7
	ClassificationViaClustering	12	38	50	31	19	50
	ClassificationViaRegression	38	46	84	5	11	16
	CostSensitiveClassifier	0	57	57	43	0	43
	CVParameterSelection	0	57	57	43	0	43
	Dagging	33	42	75	10	15	25
	Decorate	39	50	89	4	7	11
	END	40	56	96	3	1	4
	EnsembleSelection	39	48	87	4	9	13
	FilteredClassifier	40	56	96	3	1	4
	Grading	0	57	57	43	0	43
	LogitBoost	33	43	76	10	14	24
	MetaCost	0	57	57	43	0	43
	MultiBoostAB	32	42	74	11	15	26
	MultiClassClassifier	34	44	78	9	13	22
	MultiScheme	0	57	57	43	0	43
	OrdinalClassClassifier	40	56	96	3	1	4
	RacedIncrementalLogitBoost	0	57	57	43	0	43
	RandomCommittee	41	48	89	2	9	11
	RandomSubSpace	29	48	77	14	9	23
	Stacking	0	57	57	43	0	43
	StackingC	0	57	57	43	0	43
	ThresholdSelector	37	30	67	6	27	33
	Vote	0	57	57	43	0	43
	ClassBalancedND	40	56	96	3	1	4
	DataNearBalancedND	40	56	96	3	1	4
	ND	40	56	96	3	1	4

(table con'd)

Misc	HyperPipes	43	0	43	0	57	57
	MinMaxExtension	2	57	59	41	0	41
	OLM	18	35	53	25	22	47
	OSDL	2	57	59	41	0	41
	VFI	33	43	76	10	14	24
Trees	ADTree	36	49	85	7	8	15
	BFTree	40	54	94	3	3	6
	DecisionStump	26	41	67	17	16	33
	Id3	37	56	93	6	1	7
	J48	40	56	96	3	1	4
	J48graft	40	56	96	3	1	4
	LMT	39	53	92	4	4	8
	NBTree	41	56	97	2	1	3
	RandomForest	39	51	90	4	6	10
	RandomTree	30	42	72	13	15	28
	REPTree	39	48	87	4	9	13
	SimpleCart	40	55	95	3	2	5
Rules	ConjunctiveRule	26	41	67	17	16	33
	DecisionTable	38	44	82	5	13	18
	JRip	41	45	86	2	12	14
	NNge	37	50	87	6	7	13
	OneR	26	41	67	17	16	33
	PART	42	56	98	1	1	2
	Prism	42	49	91	0	8	9*
	Ridor	43	57	100	0	0	0
	ZeroR	0	57	57	43	0	43

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 57 of all unknown examples wrong. The *M** algorithm gets 4 of all unknown examples wrong, and it ranks at the 4th place. Of all the 3 classifiers that outperform *M**, the distribution is:

Table 9.3-2 Outperforming classifiers on Dataset 3.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	0	0	1	2

For Dataset 4:

Recall that we have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 206$; $m_1^- = 294$; $m_2 = 100$; $m_2^+ = 36$; $m_2^- = 64$.

$$f_4 = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_6 \vee \overline{x_7} \vee x_8 \vee x_9 \vee x_{10}) \wedge (x_1 \vee \overline{x_4} \vee \overline{x_5} \vee x_8) \wedge (x_2 \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_2} \vee \overline{x_9}) \wedge (\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_8 \vee \overline{x_9}) \wedge (x_3 \vee \overline{x_6} \vee \overline{x_7} \vee x_{10}) \wedge (\overline{x_4} \vee x_8)$$

Table 9.4-1 Comparison with Weka Results on Dataset 4.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		36	64	100	0	0	0
Bayes	AODE	32	62	94	4	2	6
	AODEsr	32	62	94	4	2	6
	BayesNet	28	58	86	8	6	14
	HNB	36	61	97	0	3	3
	NaiveBayes	28	58	86	8	6	14
	NaiveBayesSimple	28	58	86	8	6	14
	NaiveBayesUpdateable	28	58	86	8	6	14
	WAODE	35	62	97	1	2	3
Function	Logistic	29	54	83	7	10	17
	MultiLayerPerceptron	36	63	99	0	1	1
	RBFNetwork	27	56	83	9	8	17
	SimpleLogistic	29	54	81	7	12	19
	SMO	29	57	86	7	7	14
	VotedPerceptron	30	57	87	6	7	13
	Winnow	15	51	66	21	13	34
Lazy	IB1	30	51	81	6	13	19
	IBk	36	59	95	0	5	5
	KStar	36	62	98	0	2	2
	LBR	35	60	95	1	4	5
	LWL	34	45	79	2	19	21
Meta	AdaBoostM1	28	58	86	8	6	14
	AttributeSelectedClassifier	27	62	89	9	2	11
	Bagging	36	62	98	0	2	2
	ClassificationViaClustering	27	27	54	9	37	46
	ClassificationViaRegression	36	64	100	0	0	0
	CostSensitiveClassifier	0	64	64	36	0	36
	CVParameterSelection	0	64	64	36	0	36
	Dagging	31	57	88	5	7	12
	Decorate	35	63	98	1	1	2
	END	35	64	99	1	0	1
	EnsembleSelection	36	63	99	0	1	1
	FilteredClassifier	35	64	99	1	0	1
	Grading	0	64	64	36	0	36
	LogitBoost	29	55	84	7	9	16
	MetaCost	0	64	64	36	0	36
	MultiBoostAB	29	58	87	7	6	13
	MultiClassClassifier	29	54	83	7	10	17
	MultiScheme	0	64	64	36	0	36
	OrdinalClassClassifier	35	64	99	1	0	1
	RacedIncrementalLogitBoost	0	64	64	36	0	36
	RandomCommittee	34	55	89	2	9	11
	RandomSubSpace	28	60	88	8	4	12
	Stacking	0	64	64	36	0	36
	StackingC	0	64	64	36	0	36

(table con'd)

	ThresholdSelector	33	49	82	3	15	18
	Vote	0	64	64	36	0	36
	ClassBalancedND	35	64	99	1	0	1
	DataNearBalancedND	35	64	99	1	0	1
	ND	35	64	99	1	0	1
Misc	HyperPipes	36	0	36	0	64	64
	MinMaxExtension	0	62	62	36	2	38
	OLM	15	47	62	21	17	38
	OSDL	0	62	62	36	2	38
	VFI	30	55	85	6	9	15
Trees	ADTree	36	63	99	0	1	1
	BFTree	35	64	99	1	0	1
	DecisionStump	28	48	76	8	16	24
	Id3	34	63	97	2	1	3
	J48	35	64	99	1	0	1
	J48graft	35	64	99	1	0	1
	LMT	36	63	99	0	1	1
	NBTree	36	64	100	0	0	0
	RandomForest	36	63	99	0	1	1
	RandomTree	30	54	84	6	10	16
	REPTree	36	61	97	0	3	3
	SimpleCart	36	60	96	0	4	4
Rules	ConjunctiveRule	28	48	76	8	16	24
	DecisionTable	36	60	96	0	4	4
	JRip	36	64	100	0	0	0
	NNge	34	63	97	2	1	3
	OneR	28	48	76	8	16	24
	PART	36	64	100	0	0	0
	Prism	36	63	99	0	1	1
	Ridor	36	63	99	0	1	1
	ZeroR	0	64	64	36	0	36

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 64 of all unknown examples wrong. The *M** algorithm gets 0 of all unknown examples wrong, and it ranks at the 1st place. There are no classifier outperforms *M**, the distribution is:

Table 9.4-2 Outperforming classifiers on Dataset 4.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	0	0	0	0

For Dataset 5:

Recall that we have the following parameters for this dataset:

$n = 10$; $2^n = 1{,}024$; $m_1 = 400$; $m_1^+ = 300$; $m_1^- = 200$; $m_2 = 100$; $m_2^+ = 70$; $m_2^- = 30$.

$$f_5 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_8 \vee x_{10}) \wedge (x_2 \vee \overline{x_4} \vee \overline{x_7} \vee \overline{x_9}) \wedge$$
$$(\overline{x_2} \vee x_5 \vee \overline{x_9}) \wedge (\overline{x_2} \vee \overline{x_6} \vee x_7 \vee x_8 \vee \overline{x_{10}}) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_5} \vee x_7 \vee x_9) \wedge$$
$$(\overline{x_3} \vee \overline{x_5} \vee x_8 \vee \overline{x_{10}}) \wedge (x_4 \vee x_5 \vee \overline{x_7} \vee x_9 \vee x_{10}) \wedge (\overline{x_5} \vee \overline{x_6} \vee x_9 \vee \overline{x_{10}})$$

Table 9.5-1 Comparison with Weka Results on Dataset 5.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		65	27	92	5	3	8
Bayes	AODE	55	23	78	15	7	22
	AODEsr	55	23	78	15	7	22
	BayesNet	50	18	68	20	12	32
	HNB	62	21	83	8	9	17
	NaiveBayes	50	18	68	20	12	32
	NaiveBayesSimple	50	18	68	20	12	32
	NaiveBayesUpdateable	50	18	68	20	12	32
	WAODE	54	22	76	16	8	24
Function	Logistic	48	19	67	22	11	33
	MultiLayerPerceptron	64	21	85	6	9	15
	RBFNetwork	55	20	75	15	10	25
	SimpleLogistic	50	18	68	20	12	32
	SMO	48	20	68	22	10	32
	VotedPerceptron	53	17	70	17	13	30
	Winnow	65	5	70	5	25	30
Lazy	IB1	53	22	75	17	8	25
	IBk	67	25	92	3	5	8
	KStar	66	27	93	4	3	7
	LBR	59	19	78	11	11	22
	LWL	43	24	67	27	6	33
Meta	AdaBoostM1	46	21	67	24	9	33
	AttributeSelectedClassifier	65	23	88	5	7	12
	Bagging	64	22	86	6	8	14
	ClassificationViaClustering	44	15	59	26	15	41
	ClassificationViaRegression	67	23	90	3	7	10
	CostSensitiveClassifier	70	0	70	0	30	30
	CVParameterSelection	70	0	70	0	30	30
	Dagging	56	11	67	14	19	33
	Decorate	64	24	88	6	6	12
	END	67	23	90	3	7	10
	EnsembleSelection	61	21	82	9	9	18
	FilteredClassifier	67	23	90	3	7	10
	Grading	70	0	70	0	30	30
	LogitBoost	50	18	68	20	12	32
	MetaCost	70	0	70	0	30	30
	MultiBoostAB	55	11	66	15	19	34
	MultiClassClassifier	48	19	67	22	11	33
	MultiScheme	70	0	70	0	30	30

(table con'd)

	OrdinalClassClassifier	67	23	90	3	7	10
	RacedIncrementalLogitBoost	70	0	70	0	30	30
	RandomCommittee	66	24	90	4	6	10
	RandomSubSpace	66	10	76	4	20	24
	Stacking	70	0	70	0	30	30
	StackingC	70	0	70	0	30	30
	ThresholdSelector	68	3	71	2	27	29
	Vote	70	0	70	0	30	30
	ClassBalancedND	67	23	90	3	7	10
	DataNearBalancedND	67	23	90	3	7	10
	ND	67	23	90	3	7	10
Misc	HyperPipes	70	0	70	0	30	30
	MinMaxExtension	8	29	37	62	1	63
	OLM	19	21	40	51	9	60
	OSDL	8	29	37	62	1	63
	VFI	41	24	65	29	6	35
Trees	ADTree	50	16	66	20	14	34
	BFTree	66	24	90	4	6	10
	DecisionStump	38	22	60	32	8	40
	Id3	68	27	95	2	3	5
	J48	67	23	90	3	7	10
	J48graft	67	23	90	3	7	10
	LMT	67	26	93	3	4	7
	NBTree	66	22	88	4	8	12
	RandomForest	66	22	88	4	8	12
	RandomTree	52	18	70	18	12	30
	REPTree	53	22	75	17	8	25
	SimpleCart	67	24	91	3	6	9
Rules	ConjunctiveRule	70	0	70	0	30	30
	DecisionTable	64	16	80	6	14	20
	JRip	68	30	98	2	0	2
	NNge	62	23	85	8	7	15
	OneR	38	22	60	32	8	40
	PART	68	26	94	2	4	6
	Prism	69	25	94	0	5	6*
	Ridor	66	25	91	4	5	9
	ZeroR	70	0	70	0	30	30

In summary, among all 76 classifiers, the best result gets 2 of all unknown examples wrong. The worst result gets 63 of all unknown examples wrong. The *M** algorithm gets 8 of all unknown examples wrong, and it ranks at the 7th place. Of all the 3 classifiers that outperform *M**, the distribution is:

Table 9.5-2 Outperforming classifiers on Dataset 5.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	1	0	0	2	3

For Dataset 6:

Recall that we have the following parameters for this dataset:

n = 15; 2^n = 32,768; m_1 = 6,400; m_1^+ = 4,763; m_1^- = 3,237; m_2 = 1,600; m_2^+ = 999; m_2^- = 601.

$$f_6 = (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee \overline{x_4} \vee x_7 \vee x_{11}) \wedge (\overline{x_1} \vee \overline{x_8} \vee x_9) \wedge$$
$$(x_2 \vee \overline{x_4} \vee \overline{x_6} \vee \overline{x_{12}} \vee x_{13}) \wedge (\overline{x_2} \vee x_5 \vee \overline{x_9} \vee x_{10} \vee x_{13} \vee \overline{x_{15}}) \wedge$$
$$(\overline{x_3} \vee \overline{x_7} \vee x_9 \vee x_{12} \vee \overline{x_{14}}) \wedge (\overline{x_4} \vee x_6 \vee \overline{x_{10}} \vee x_{11}) \wedge (x_6 \vee \overline{x_7} \vee \overline{x_8} \vee x_{12}) \wedge$$
$$(\overline{x_7} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (x_8 \vee \overline{x_9} \vee x_{10} \vee \overline{x_{12}}) \wedge$$
$$(\overline{x_9} \vee x_{10} \vee x_{11} \vee x_{12} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15})$$

Table 9.6-1 Comparison with Weka Results on Dataset 6.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		976	556	1532	23	45	68
Bayes	AODE	865	361	1226	134	240	374
	AODEsr	865	361	1226	134	240	374
	BayesNet	831	331	1162	168	270	438
	HNB	905	428	1333	94	173	267
	NaiveBayes	831	331	1162	168	270	438
	NaiveBayesSimple	831	331	1162	168	270	438
	NaiveBayesUpdateable	831	331	1162	168	270	438
	WAODE	862	365	1227	137	236	373
Function	Logistic	808	346	1154	191	255	446
	MultiLayerPerceptron	985	527	1512	14	74	88
	RBFNetwork	806	366	1172	193	235	428
	SimpleLogistic	814	344	1158	185	257	442
	SMO	818	340	1158	181	261	442
	VotedPerceptron	813	333	1146	186	268	454
	Winnow	553	337	890	446	264	710
Lazy	IB1	841	483	1324	158	118	276
	IBk	956	499	1455	43	102	145
	KStar	974	541	1515	25	60	85
	LBR	976	502	1478	23	99	122
	LWL	713	383	1096	286	218	504
Meta	AdaBoostM1	803	337	1140	196	264	460
	AttributeSelectedClassifier	797	445	1242	202	156	358
	Bagging	986	547	1533	13	54	67
	ClassificationViaClustering	405	403	808	594	198	792
	ClassificationViaRegression	970	537	1507	29	64	93
	CostSensitiveClassifier	999	0	999	0	601	601
	CVParameterSelection	999	0	999	0	601	601
	Dagging	840	323	1163	159	278	437
	Decorate	961	554	1515	38	47	85

(table con'd)

	END	976	542	1518	23	59	82
	EnsembleSelection	985	540	1525	14	61	75
	FilteredClassifier	976	542	1518	23	59	82
	Grading	999	0	999	0	601	601
	LogitBoost	826	331	1157	173	270	443
	MetaCost	999	0	999	0	601	601
	MultiBoostAB	747	326	1073	252	275	527
	MultiClassClassifier	808	346	1154	191	255	446
	MultiScheme	999	0	999	0	601	601
	OrdinalClassClassifier	976	542	1518	23	59	82
	RacedIncrementalLogitBoost	839	298	1137	160	303	463
	RandomCommittee	963	473	1436	36	128	164
	RandomSubSpace	991	371	1362	8	230	238
	Stacking	999	0	999	0	601	601
	StackingC	999	0	999	0	601	601
	ThresholdSelector	931	228	1159	68	373	441
	Vote	999	0	999	0	601	601
	ClassBalancedND	976	542	1518	23	59	82
	DataNearBalancedND	976	542	1518	23	59	82
	ND	976	542	1518	23	59	82
Misc	HyperPipes	999	0	999	0	601	601
	MinMaxExtension	220	581	801	779	20	799
	OLM	468	470	938	531	131	662
	OSDL	220	581	801	779	20	799
	VFI	677	416	1093	322	185	507
Trees	ADTree	939	343	1282	60	258	318
	BFTree	962	557	1519	37	44	81
	DecisionStump	602	398	1000	397	203	600
	Id3	957	581	1538	42	20	62
	J48	976	542	1518	23	59	82
	J48graft	977	542	1519	22	59	81
	LMT	958	561	1519	41	40	81
	NBTree	976	559	1535	23	42	65
	RandomForest	988	562	1550	11	39	50
	RandomTree	822	436	1258	177	165	342
	REPTree	963	538	1501	36	63	99
	SimpleCart	978	537	1515	21	64	85
Rules	ConjunctiveRule	602	398	1000	397	203	600
	DecisionTable	956	492	1448	43	109	152
	JRip	999	601	1600	0	0	0
	NNge	-	-	-	-	-	-
	OneR	602	398	1000	397	203	600
	PART	998	599	1597	1	2	3
	Prism	979	577	1556	0	24	44*
	Ridor	999	591	1590	0	10	10
	ZeroR	999	0	999	0	601	601

Note: Classifier Rules.NNge could not generate a result due to the size of the dataset.

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 799 of all unknown examples wrong. The *M** algorithm gets 68 of all unknown examples wrong, and it ranks at the 9th place. Of all the 8 classifiers that outperform *M**, the distribution is:

Table 9.6-2 Outperforming classifiers on Dataset 6.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	1	0	3	4

For Dataset 7:

Recall that we have the following parameters for this dataset:

n = 15; 2^n = 32,768; m_1 = 6,400; m_1^+ = 4,325; m_1^- = 3,675; m_2 = 1,600; m_1^+ = 872; m_2^- = 728.

$$f_7 = (x_1 \vee \overline{x_3} \vee \overline{x_7} \vee x_9 \vee \overline{x_{11}}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5} \vee x_6 \vee \overline{x_{10}} \vee x_{12} \vee \overline{x_{15}}) \wedge$$
$$(x_1 \vee \overline{x_8} \vee \overline{x_{10}} \vee x_{11} \vee \overline{x_{13}} \vee \overline{x_{14}}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4 \vee \overline{x_5} \vee \overline{x_7} \vee x_9 \vee \overline{x_{10}} \vee \overline{x_{12}} \vee x_{14} \vee x_{15}) \wedge$$
$$(x_2 \vee \overline{x_5} \vee \overline{x_6} \vee x_7 \vee x_8 \vee \overline{x_{10}} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee \overline{x_8} \vee x_{12}) \wedge$$
$$(\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_{13}) \wedge (x_3 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{12}) \wedge (\overline{x_3} \vee \overline{x_{11}} \vee x_{14}) \wedge$$
$$(\overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_{10} \vee \overline{x_{13}} \vee \overline{x_{15}}) \wedge (\overline{x_5} \vee x_9 \vee x_{11}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee x_{12} \vee \overline{x_{13}})$$

Table 9.7-1 Comparison with Weka Results on Dataset 7.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		849	693	1542	23	35	58
Bayes	AODE	685	515	1200	187	213	400
	AODEsr	685	516	1201	187	212	399
	BayesNet	654	475	1129	218	253	471
	HNB	767	594	1361	105	134	239
	NaiveBayes	654	475	1129	218	253	471
	NaiveBayesSimple	654	475	1129	218	253	471
	NaiveBayesUpdateable	654	475	1129	218	253	471
	WAODE	691	537	1228	181	191	372
Function	Logistic	641	485	1126	231	243	474
	MultiLayerPerceptron	872	714	1586	0	14	14
	RBFNetwork	675	517	1192	197	211	408
	SimpleLogistic	643	481	1124	229	247	476
	SMO	642	478	1120	230	250	480
	VotedPerceptron	643	482	1125	229	246	475
	Winnow	693	325	1018	179	403	582
Lazy	IB1	755	597	1352	117	131	248
	Ibk	836	628	1464	36	100	136
	Kstar	852	690	1542	20	38	58
	LBR	864	663	1527	8	65	73

(table con'd)

	LWL	578	490	1068	294	238	532
Meta	AdaBoostM1	614	501	1115	258	227	485
	AttributeSelectedClassifier	616	634	1250	256	94	350
	Bagging	858	685	1543	14	43	57
	ClassificationViaClustering	498	403	901	374	325	699
	ClassificationViaRegression	846	683	1529	26	45	71
	CostSensitiveClassifier	872	0	872	0	728	728
	CVParameterSelection	872	0	872	0	728	728
	Dagging	660	473	1133	212	255	467
	Decorate	832	690	1522	40	38	78
	END	852	685	1537	20	43	63
	EnsembleSelection	857	680	1537	15	48	63
	FilteredClassifier	852	685	1537	20	43	63
	Grading	872	0	872	0	728	728
	LogitBoost	640	467	1107	232	261	493
	MetaCost	872	0	872	0	728	728
	MultiBoostAB	578	484	1062	294	244	538
	MultiClassClassifier	641	485	1126	231	243	474
	MultiScheme	872	0	872	0	728	728
	OrdinalClassClassifier	852	685	1537	20	43	63
	RacedIncrementalLogitBoost	624	487	1111	248	241	489
	RandomCommittee	840	624	1464	32	104	136
	RandomSubSpace	811	552	1363	61	176	237
	Stacking	872	0	872	0	728	728
	StackingC	872	0	872	0	728	728
	ThresholdSelector	794	290	1084	78	438	516
	Vote	872	0	872	0	728	728
	ClassBalancedND	852	685	1537	20	43	63
	DataNearBalancedND	852	685	1537	20	43	63
	ND	852	685	1537	20	43	63
Misc	HyperPipes	872	0	872	0	728	728
	MinMaxExtension	204	719	923	668	9	677
	OLM	398	627	1025	474	101	575
	OSDL	204	719	923	668	9	677
	VFI	602	507	1109	270	221	491
Trees	ADTree	785	493	1278	87	235	322
	BFTree	849	684	1533	23	44	67
	DecisionStump	578	484	1062	294	244	538
	Id3	833	703	1536	39	25	64
	J48	852	685	1537	20	43	63
	J48graft	852	685	1537	20	43	63
	LMT	843	704	1547	29	24	53
	NBTree	867	673	1540	5	55	60
	RandomForest	866	690	1556	6	38	44
	RandomTree	664	545	1209	208	183	391
	REPTree	849	667	1516	23	61	84
	SimpleCart	847	684	1531	25	44	69

(table con'd)

Rules	ConjunctiveRule	578	484	1062	294	244	538
	DecisionTable	859	625	1484	13	103	116
	JRip	871	728	1599	1	0	1
	NNge	-	-	-	-	-	-
	OneR	578	484	1062	294	244	538
	PART	870	721	1591	2	7	9
	Prism	856	712	1568	0	16	32*
	Ridor	869	661	1530	3	67	70
	ZeroR	872	0	872	0	728	728

In summary, among all 76 classifiers, the best result gets 1 of all unknown examples wrong. The worst result gets 728 of all unknown examples wrong. The *M** algorithm gets 58 of all unknown examples wrong, and it ranks at the 8th place. Of all the 7 classifiers that outperform *M**, the distribution is:

Table 9.7-2 Outperforming classifiers on Dataset 7.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	1	0	1	0	2	3

For Dataset 8:

Recall that we have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 3{,}768$; $m_1^- = 4{,}232$; $m_2 = 1{,}600$; $m_2^+ = 761$; $m_2^- = 839$.

$$f_8 = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_8 \vee \overline{x_{11}}) \wedge (\overline{x_1} \vee \overline{x_7} \vee x_{12} \vee \overline{x_{13}}) \wedge$$
$$(\overline{x_2} \vee x_4 \vee \overline{x_5} \vee \overline{x_8} \vee x_{10} \vee \overline{x_{12}} \vee x_{14}) \wedge (\overline{x_2} \vee \overline{x_6} \vee x_{13} \vee \overline{x_{14}}) \wedge$$
$$(\overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_8 \vee x_{13}) \wedge (x_3 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{12}) \wedge (\overline{x_4} \vee x_5 \vee \overline{x_6} \vee x_{10} \vee \overline{x_{13}} \vee \overline{x_{15}}) \wedge$$
$$(\overline{x_4} \vee \overline{x_{11}} \vee x_{14}) \wedge (\overline{x_5} \vee x_9 \vee x_{11}) \wedge (\overline{x_6} \vee \overline{x_7} \vee x_8 \vee x_{12}) \wedge (\overline{x_8} \vee \overline{x_9} \vee x_{11} \vee x_{12} \vee \overline{x_{13}}) \wedge$$
$$(x_9 \vee \overline{x_{10}} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15})$$

Table 9.8-1 Comparison with Weka Results on Dataset 8.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		704	806	1510	57	33	90
Bayes	AODE	524	654	1178	237	185	422
	AODEsr	524	654	1178	237	185	422
	BayesNet	491	629	1120	270	210	480
	HNB	599	690	1289	162	149	311
	NaiveBayes	491	629	1120	270	210	480
	NaiveBayesSimple	491	629	1120	270	210	480
	NaiveBayesUpdateable	491	629	1120	270	210	480
	WAODE	523	660	1183	238	179	417

(table con'd)

Function	Logistic	490	624	1114	271	215	486
	MultiLayerPerceptron	748	738	1486	13	101	114
	RBFNetwork	519	620	1139	242	219	461
	SimpleLogistic	487	624	1111	274	215	489
	SMO	490	618	1108	271	221	492
	VotedPerceptron	493	619	1112	268	220	488
	Winnow	430	569	999	331	270	601
Lazy	IB1	628	692	1320	133	147	280
	IBk	708	728	1436	53	111	164
	KStar	701	807	1508	60	32	92
	LBR	708	718	1426	53	121	174
	LWL	541	608	1149	220	231	451
Meta	AdaBoostM1	537	580	1117	224	259	483
	AttributeSelectedClassifier	500	612	1112	261	227	488
	Bagging	707	796	1503	54	43	97
	ClassificationViaClustering	507	390	897	254	449	703
	ClassificationViaRegression	687	770	1457	74	69	143
	CostSensitiveClassifier	0	839	839	761	0	761
	CVParameterSelection	0	839	839	761	0	761
	Dagging	499	614	1113	262	225	487
	Decorate	695	799	1494	66	40	106
	END	691	770	1461	70	69	139
	EnsembleSelection	689	775	1464	72	64	136
	FilteredClassifier	691	770	1461	70	69	139
	Grading	0	839	839	761	0	761
	LogitBoost	499	612	1111	262	227	489
	MetaCost	0	839	839	761	0	761
	MultiBoostAB	513	536	1049	248	303	551
	MultiClassClassifier	490	624	1114	271	215	486
	MultiScheme	0	839	839	761	0	761
	OrdinalClassClassifier	691	770	1461	70	69	139
	RacedIncrementalLogitBoost	492	623	1115	269	216	485
	RandomCommittee	676	746	1422	85	93	178
	RandomSubSpace	633	676	1309	128	163	291
	Stacking	0	839	839	761	0	761
	StackingC	0	839	839	761	0	761
	ThresholdSelector	659	375	1034	102	464	566
	Vote	0	839	839	761	0	761
	ClassBalancedND	691	770	1461	70	69	139
	DataNearBalancedND	691	770	1461	70	69	139
	ND	691	770	1461	70	69	139
Misc	HyperPipes	761	0	761	0	839	839
	MinMaxExtension	84	822	906	677	17	694
	OLM	279	703	982	482	136	618
	OSDL	84	822	906	677	17	694
	VFI	516	594	1110	245	245	490

(table con'd)

Trees	ADTree	559	644	1203	202	195	397
	BFTree	691	766	1457	70	73	143
	DecisionStump	454	482	936	307	357	664
	Id3	702	781	1483	59	58	117
	J48	691	770	1461	70	69	139
	J48graft	691	770	1461	70	69	139
	LMT	703	781	1484	58	58	116
	NBTree	737	780	1517	24	59	83
	RandomForest	734	787	1521	27	52	79
	RandomTree	566	655	1221	195	184	379
	REPTree	685	769	1454	76	70	146
	SimpleCart	689	774	1463	72	65	137
Rules	ConjunctiveRule	454	482	936	307	357	664
	DecisionTable	636	661	1297	125	178	303
	JRip	704	770	1474	57	69	126
	NNge	-	-	-	-	-	-
	OneR	454	482	936	307	357	664
	PART	750	836	1586	11	3	14
	Prism	737	800	1537	0	39	63*
	Ridor	754	810	1564	7	29	36
	ZeroR	0	839	839	761	0	761

In summary, among all 76 classifiers, the best result gets 14 of all unknown examples wrong. The worst result gets 839 of all unknown examples wrong. The *M** algorithm gets 90 of all unknown examples wrong, and it ranks at the 6th place. Of all the 5 classifiers that outperform *M**, the distribution is:

Table 9.8-2 Outperforming classifiers on Dataset 8.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	0	0	2	3

For Dataset 9:

Recall that we have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}611$; $m_1^- = 3{,}389$; $m_2 = 1{,}600$; $m_2^+ = 895$; $m_2^- = 705$.

$$
\begin{aligned}
f_9 = {} & (x_1 \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_6 \vee x_7 \vee x_{11}) \wedge (\overline{x_1} \vee \overline{x_8} \vee x_9 \vee x_{14}) \wedge \\
& (x_2 \vee \overline{x_4} \vee \overline{x_6} \vee x_{10} \vee \overline{x_{12}} \vee x_{13}) \wedge (\overline{x_2} \vee x_5 \vee \overline{x_8} \vee x_9 \vee x_{13} \vee \overline{x_{15}}) \wedge \\
& (\overline{x_2} \vee \overline{x_7} \vee x_9 \vee x_{12} \vee \overline{x_{14}}) \wedge (\overline{x_3} \vee x_6 \vee \overline{x_{10}} \vee x_{11}) \wedge (x_3 \vee \overline{x_7} \vee \overline{x_8} \vee x_{12}) \wedge \\
& (\overline{x_4} \vee x_{11} \vee \overline{x_{13}} \vee x_{14} \vee \overline{x_{15}}) \wedge (x_4 \vee \overline{x_9} \vee x_{10} \vee \overline{x_{12}}) \wedge \\
& (\overline{x_4} \vee x_{10} \vee x_{11} \vee x_{12} \vee \overline{x_{13}} \vee \overline{x_{14}} \vee x_{15}) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_7} \vee x_8 \vee \overline{x_{11}} \vee x_{12}) \wedge \\
& (\overline{x_5} \vee x_7 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{13}) \wedge (\overline{x_6} \vee \overline{x_8} \vee x_9 \vee x_{10} \vee \overline{x_{12}} \vee \overline{x_{13}})
\end{aligned}
$$

Table 9.9-1 Comparison with Weka Results on Dataset 9.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		866	630	1496	29	75	104
Bayes	AODE	754	395	1149	141	310	451
	AODEsr	754	395	1149	141	310	451
	BayesNet	721	337	1058	174	368	542
	HNB	756	443	1199	139	262	401
	NaiveBayes	721	337	1058	174	368	542
	NaiveBayesSimple	721	337	1058	174	368	542
	NaiveBayesUpdateable	721	337	1058	174	368	542
	WAODE	752	395	1147	143	310	453
Function	Logistic	710	349	1059	185	356	541
	MultiLayerPerceptron	865	527	1392	30	178	208
	RBFNetwork	713	409	1122	182	296	478
	SimpleLogistic	719	328	1047	176	377	553
	SMO	707	350	1057	188	355	543
	VotedPerceptron	692	360	1052	203	345	548
	Winnow	675	285	960	220	420	640
Lazy	IB1	741	561	1302	154	144	298
	IBk	853	574	1427	42	131	173
	KStar	866	621	1487	29	84	113
	LBR	854	543	1397	41	162	203
	LWL	708	405	1113	187	300	487
Meta	AdaBoostM1	721	318	1039	174	387	561
	AttributeSelectedClassifier	769	443	1212	126	262	388
	Bagging	884	597	1481	11	108	119
	ClassificationViaClustering	509	405	914	386	300	686
	ClassificationViaRegression	862	593	1455	33	112	145
	CostSensitiveClassifier	895	0	895	0	705	705
	CVParameterSelection	895	0	895	0	705	705
	Dagging	728	328	1056	167	377	544
	Decorate	839	620	1459	56	85	141
	END	862	611	1473	33	94	127
	EnsembleSelection	877	577	1454	18	128	146
	FilteredClassifier	862	611	1473	33	94	127
	Grading	895	0	895	0	705	705
	LogitBoost	721	313	1034	174	392	566
	MetaCost	895	0	895	0	705	705
	MultiBoostAB	750	256	1006	145	449	594
	MultiClassClassifier	710	349	1059	185	356	541
	MultiScheme	895	0	895	0	705	705
	OrdinalClassClassifier	862	611	1473	33	94	127
	RacedIncrementalLogitBoost	688	357	1045	207	348	555
	RandomCommittee	851	547	1398	44	158	202
	RandomSubSpace	823	377	1200	72	328	400
	Stacking	895	0	895	0	705	705
	StackingC	895	0	895	0	705	705

(table con'd)

	ThresholdSelector	837	139	976	58	566	624
	Vote	895	0	895	0	705	705
	ClassBalancedND	862	611	1473	33	94	127
	DataNearBalancedND	862	611	1473	33	94	127
	ND	862	611	1473	33	94	127
Misc	HyperPipes	895	0	895	0	705	705
	MinMaxExtension	268	657	925	627	48	675
	OLM	449	540	989	446	165	611
	OSDL	268	657	925	627	48	675
	VFI	574	491	1065	321	214	535
Trees	ADTree	780	271	1051	115	434	549
	BFTree	843	620	1463	52	85	137
	DecisionStump	532	446	978	363	259	622
	Id3	830	647	1477	65	58	123
	J48	862	611	1473	33	94	127
	J48graft	863	613	1476	32	92	124
	LMT	836	641	1477	59	64	123
	NBTree	869	633	1502	26	72	98
	RandomForest	878	608	1486	17	97	114
	RandomTree	686	472	1158	209	233	442
	REPTree	847	555	1402	48	150	198
	SimpleCart	868	602	1470	27	103	130
Rules	ConjunctiveRule	532	446	978	363	259	622
	DecisionTable	786	495	1281	109	210	319
	JRip	895	705	1600	0	0	0
	NNge	-	-	-	-	-	-
	OneR	532	446	978	363	259	622
	PART	886	691	1577	9	14	23
	Prism	859	661	1520	0	44	80*
	Ridor	894	676	1570	1	29	30
	ZeroR	895	0	895	0	705	705

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 705 of all unknown examples wrong. The *M** algorithm gets 104 of all unknown examples wrong, and it ranks at the 6th place. Of all the 5 classifiers that outperform *M**, the distribution is:

Table 9.9-2 Outperforming classifiers on Dataset 9.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	0	0	0	1	4

For Dataset 10:

Recall that we have the following parameters for this dataset:

$n = 15$; $2^n = 32{,}768$; $m_1 = 6{,}400$; $m_1^+ = 4{,}288$; $m_1^- = 3{,}712$; $m_2 = 1{,}600$; $m_2^+ = 852$; $m_2^- = 748$;

$$f_{10} = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee x_7) \wedge (x_1 \vee x_3 \vee \overline{x_9} \vee x_{11}) \wedge$$
$$(x_1 \vee \overline{x_4} \vee \overline{x_8} \vee x_{10} \vee \overline{x_{14}}) \wedge (\overline{x_1} \vee \overline{x_6} \vee x_8 \vee \overline{x_9} \vee x_{12} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_{11}} \vee x_{15}) \wedge$$
$$(x_2 \vee \overline{x_4} \vee x_6 \vee x_{10} \vee x_{13} \vee \overline{x_{15}}) \wedge (\overline{x_2} \vee x_9 \vee \overline{x_{12}}) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_8} \vee \overline{x_9} \vee x_{10}) \wedge$$
$$(\overline{x_4} \vee x_5 \vee x_9 \vee \overline{x_{13}} \vee \overline{x_{14}}) \wedge (x_4 \vee \overline{x_6} \vee x_{12} \vee x_{14}) \wedge (\overline{x_4} \vee \overline{x_7} \vee x_{10} \vee x_{11} \vee x_{15}) \wedge$$
$$(\overline{x_5} \vee x_6 \vee \overline{x_7} \vee x_8 \vee \overline{x_{11}} \vee x_{12}) \wedge (\overline{x_5} \vee x_7 \vee \overline{x_9} \vee \overline{x_{10}} \vee x_{13}) \wedge$$
$$(\overline{x_6} \vee \overline{x_8} \vee x_9 \vee x_{10} \vee \overline{x_{12}} \vee \overline{x_{13}}) \wedge (\overline{x_6} \vee \overline{x_{10}} \vee x_{13} \vee x_{14} \vee \overline{x_{15}})$$

Table 9.10-1 Comparison with Weka Results on Dataset 10.

Algorithm		CorrectPos	CorrectNeg	Correct	WrongPos	WrongNeg	Wrong
*M**		826	670	1496	26	78	104
Bayes	AODE	643	463	1106	209	285	494
	AODEsr	643	463	1106	209	285	494
	BayesNet	601	440	1041	251	308	559
	HNB	732	513	1245	120	235	355
	NaiveBayes	601	440	1041	251	308	559
	NaiveBayesSimple	601	440	1041	251	308	559
	NaiveBayesUpdateable	601	440	1041	251	308	559
	WAODE	654	444	1098	198	304	502
Function	Logistic	600	437	1037	252	311	563
	MultiLayerPerceptron	839	622	1461	13	126	139
	RBFNetwork	638	443	1081	214	305	519
	SimpleLogistic	608	439	1047	244	309	553
	SMO	535	483	1018	317	265	582
	VotedPerceptron	605	431	1036	247	317	564
	Winnow	409	446	855	443	302	745
Lazy	IB1	694	588	1282	158	160	318
	IBk	805	618	1423	47	130	177
	KStar	818	683	1501	34	65	99
	LBR	806	628	1434	46	120	166
	LWL	535	483	1018	317	265	582
Meta	AdaBoostM1	599	444	1043	253	304	557
	AttributeSelectedClassifier	609	571	1180	243	177	420
	Bagging	843	655	1498	9	93	102
	ClassificationViaClustering	523	299	822	329	449	778
	ClassificationViaRegression	810	631	1441	42	117	159
	CostSensitiveClassifier	852	0	852	0	748	748
	CVParameterSelection	852	0	852	0	748	748
	Dagging	535	483	1018	317	265	582
	Decorate	804	690	1494	48	58	106
	END	813	644	1457	39	104	143
	EnsembleSelection	839	642	1481	13	106	119
	FilteredClassifier	813	644	1457	39	104	143
	Grading	852	0	852	0	748	748

(table con'd)

	LogitBoost	600	431	1031	252	317	569
	MetaCost	852	0	852	0	748	748
	MultiBoostAB	535	483	1018	317	265	582
	MultiClassClassifier	600	437	1037	252	311	563
	MultiScheme	852	0	852	0	748	748
	OrdinalClassClassifier	813	644	1457	39	104	143
	RacedIncrementalLogitBoost	620	373	993	232	375	607
	RandomCommittee	797	597	1394	55	151	206
	RandomSubSpace	772	506	1278	80	242	322
	Stacking	852	0	852	0	748	748
	StackingC	852	0	852	0	748	748
	ThresholdSelector	744	281	1025	108	467	575
	Vote	852	0	852	0	748	748
	ClassBalancedND	813	644	1457	39	104	143
	DataNearBalancedND	813	644	1457	39	104	143
	ND	813	644	1457	39	104	143
Misc	HyperPipes	852	0	852	0	748	748
	MinMaxExtension	148	730	878	704	18	722
	OLM	345	574	919	507	174	681
	OSDL	148	730	878	704	18	722
	VFI	542	482	1024	310	266	576
Trees	ADTree	852	318	1170	0	430	430
	BFTree	802	648	1450	50	100	150
	DecisionStump	535	483	1018	317	265	582
	Id3	789	681	1470	63	67	130
	J48	813	644	1457	39	104	143
	J48graft	813	643	1456	39	105	144
	LMT	788	679	1467	64	69	133
	NBTree	833	673	1506	19	75	94
	RandomForest	833	654	1487	19	94	113
	RandomTree	659	553	1212	193	195	388
	REPTree	796	627	1423	56	121	177
	SimpleCart	808	655	1463	44	93	137
Rules	ConjunctiveRule	535	483	1018	317	265	582
	DecisionTable	777	505	1282	75	243	318
	JRip	852	748	1600	0	0	0
	NNge	-	-	-	-	-	-
	OneR	535	483	1018	317	265	582
	PART	845	732	1577	7	16	23
	Prism	823	703	1526	0	45	74*
	Ridor	852	704	1556	0	44	44
	ZeroR	852	0	852	0	748	748

In summary, among all 76 classifiers, the best result gets 0 of all unknown examples wrong. The worst result gets 778 of all unknown examples wrong. The *M** algorithm gets 104 of all unknown examples wrong, and it ranks at the 8th place. Of all the 7 classifiers that outperform *M**, the distribution is:

Table 9.10-2 Outperforming classifiers on Dataset 10.

Bayes	Function	Lazy	Meta	Misc	Tree	Rules
0	0	1	1	0	1	4

9.2 Analysis of Comparison Results

If we summarize the ten Outperforming Classifiers Tables, we will get the following table:

Table 9.11 Summary of Outperforming Classifiers.

Dataset	Bayes	Function	Lazy	Meta	Misc	Trees	Rules
D1	0	0	2	0	0	0	2
D2	0	0	0	0	0	1	4
D3	0	0	0	0	0	1	2
D4	0	0	0	0	0	0	0
D5	0	0	1	0	0	2	3
D6	0	0	0	0	0	3	4
D7	0	1	0	1	0	2	3
D8	0	0	0	0	0	2	3
D9	0	0	0	0	0	1	4
D10	0	0	1	1	0	1	4

From the above table, the easiest conclusion we can draw upon is, the *M** algorithm has performed better than all the classifiers in Bayesian and Miscellaneous categories on all 10 datasets.

For the rest of the categories, there are at least one classifier which has performed better than the *M** algorithm for at least one time. Now we will take a closer look at these classifiers in each category.

Table 9.12 Details of Outperforming Classifiers.

Algorithm		D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	Times
Function	MultiLayerPerceptron							X				1
Lazy	IBk	X										1
	KStar	X				X					X	3
Meta	Bagging							X			X	2
Trees	Id3					X	X					2
	LMT					X		X				2
	NBTree		X	X			X		X	X	X	6
	RandomForest						X	X	X			3
Rules	JRip	X	X			X	X	X		X	X	7
	PART	X	X	X		X	X	X	X	X	X	9
	Prism		X			X	X	X	X	X	X	7
	Ridor		X	X			X		X	X	X	6

The above table means, there are in total 12 classifiers of 5 categories which have outperformed *M** algorithm at least once. Among them, Classifier "MultiLayer Perceptron" of category "Function" has outperformed *M** algorithm once on Dataset 7; Classifier "IBk" of category "Lazy" has outperformed *M** algorithm once on Dataset 1; and so on.

Among the 12 classifiers in Table 9.12, 5 classifiers have outperformed the *M** algorithm for more than five times, i.e., half of the times. These are classifier "NBTree" from Decision Tree methods, classifiers "JRip", "PART", "Prism", and "Ridor" from Rule Induction methods. We will now examine the detailed results of these five classifiers more closely, and compare them with the detailed result of the *M** algorithm. We will use Dataset 1 as an illustrative example. Though the analyses on the rest datasets have not been shown in this thesis, they all exhibit the same properties.

What we want to see is, if the examples which are misclassified by the *M** algorithm are also misclassified by other classifiers; and if the probabilities generated by these classifiers are distributed in a wide range.

For the *M algorithm:**
A total of 8 examples were classified wrongly. They are:

Table 9.13 Partial Detailed Result of the *M** Algorithm on Dataset 1

	#	z	(p_1, p_0)
Wrong Positive	12	<0111000010>	(0.450, 0.550)
Wrong Negative	5	<1000100100>	(0.571, 0.429)
	18	<0010011110>	(0.806, 0.194)
	32	<1101010011>	(0.671, 0.329)
	34	<0110011110>	(0.677, 0.323)
	39	<1100001010>	(0.735, 0.265)
	49	<1010011110>	(0.684, 0.316)
	59	<1101011010>	(0.765, 0.235)

A complete detailed result can be drawn in a diagram. In the following figure the horizontal axis is the number of examples. The vertical axis is the probability p_1. The blue points depict the examples which have actual class "1", while the red points depict the examples which have actual class "0". The horizontal line $p_1 = 0.5$ separates all the examples into two sections. The examples in the above section are predicted to be positive, while the examples in the below section are predicted to be negative. In other words, the blue examples in the above section and the red examples in the below section are correctly classified. While the blue examples in the below section and the red examples in the above section are incorrectly classified. The blue examples in the below

section correspond to the "Wrong Positive" rows in the table. The red examples in the above section correspond to the "Wrong Negative" rows in the table.

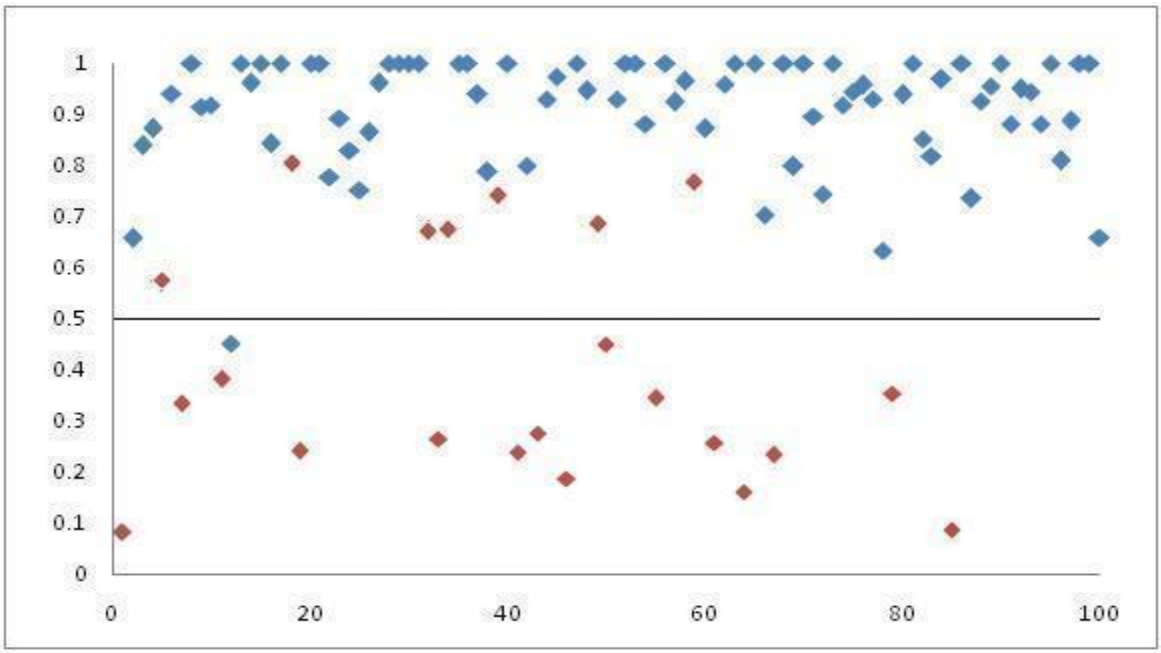

Figure 9.1 The Distribution of probability p_1 for the M^* Algorithm on Dataset 1.

For Classifier NBTree:

Weka has generated assessments of probabilities for every classifier. Therefore we will take a look at these probabilities for the other classifiers. For classifier NBTree, 10 examples were classified wrongly. There are no wrong positive examples, and 10 wrong negative examples. They are:

Table 9.14 Partial Detailed Result of Classifier NBTree on Dataset 1.

	#	z	(p_1, p_0)
Wrong Positive	-	-	-
Wrong Negative	7	<0001110111>	(0.520, 0.480)
	18	<0010011110>	(0.598, 0.402)
	19	<0001100111>	(0.809, 0.191)
	32	<1101010011>	(0.779, 0.221)
	39	<1100001010>	(0.653, 0.347)
	41	<0000010110>	(0.674, 0.326)
	49	<1010011110>	(0.544, 0.456)
	55	<1101010010>	(0.644, 0.356)
	59	<1101011010>	(0.718, 0.282)
	67	<0001100011>	(0.853, 0.147)

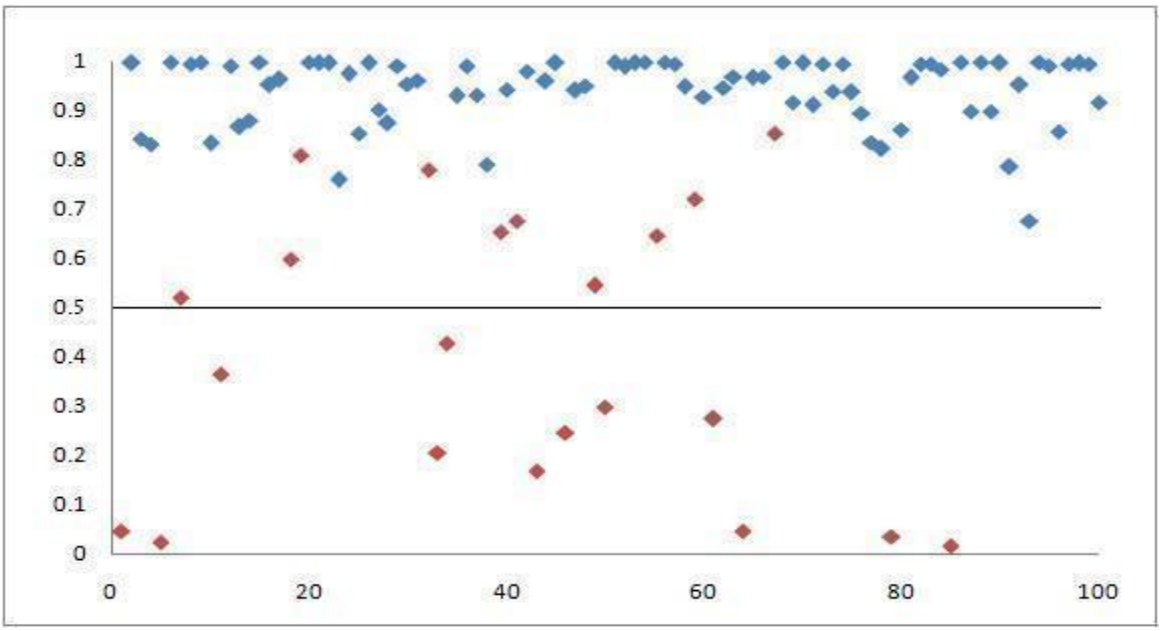

Figure 9.2 The Distribution of probability p_1 for Classifier NBTree on Dataset 1.

For Classifier JRip:

None of the examples were classified wrongly.

Table 9.15 Partial Detailed Result of Classifier JRip on Dataset 1.

	#	z	(p_1, p_0)
Wrong Positive	-	-	-
Wrong Negative	-	-	-

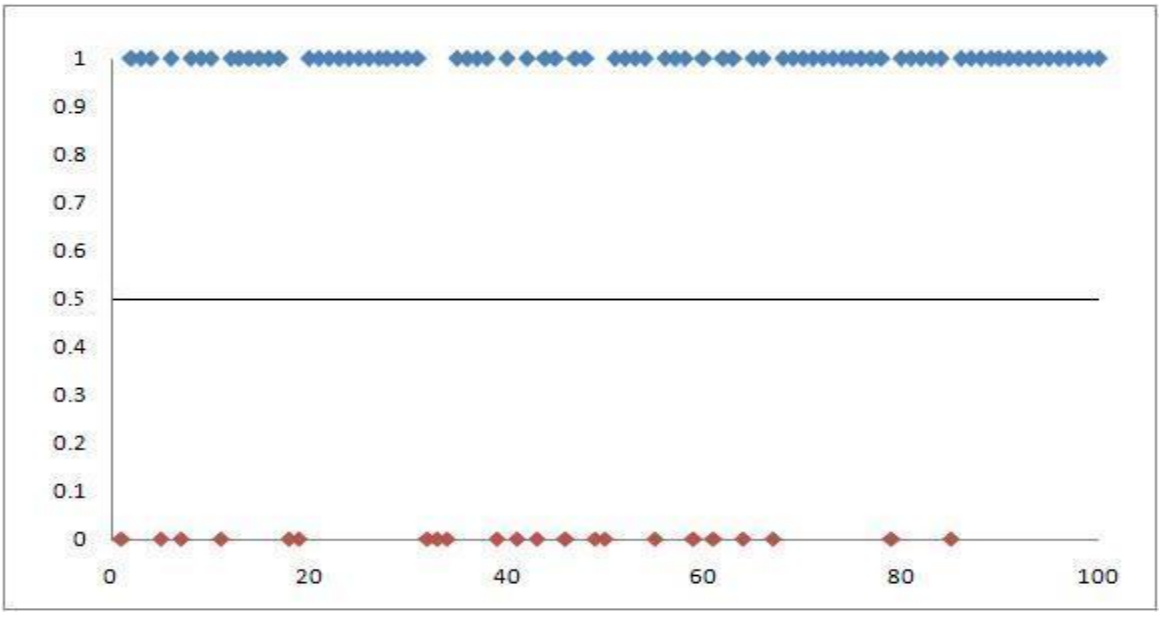

Figure 9.3 The Distribution of probability p_1 for classifier JRip on Dataset 1.

For Classifier PART:

A total of 5 examples were classified wrongly. There are no wrong positive examples, and 5 wrong negative examples. They are:

Table 9.16 Partial Detailed Result of Classifier PART on Dataset 1.

	#	z	(p_1, p_0)
Wrong Positive	-	-	-
Wrong Negative	18	<0010011110>	(1, 0)
	39	<1100001010>	(0.933, 0.067)
	41	<0000010110>	(1, 0)
	49	<1010011110>	(1, 0)
	59	<1101011010>	(0.933, 0.067)

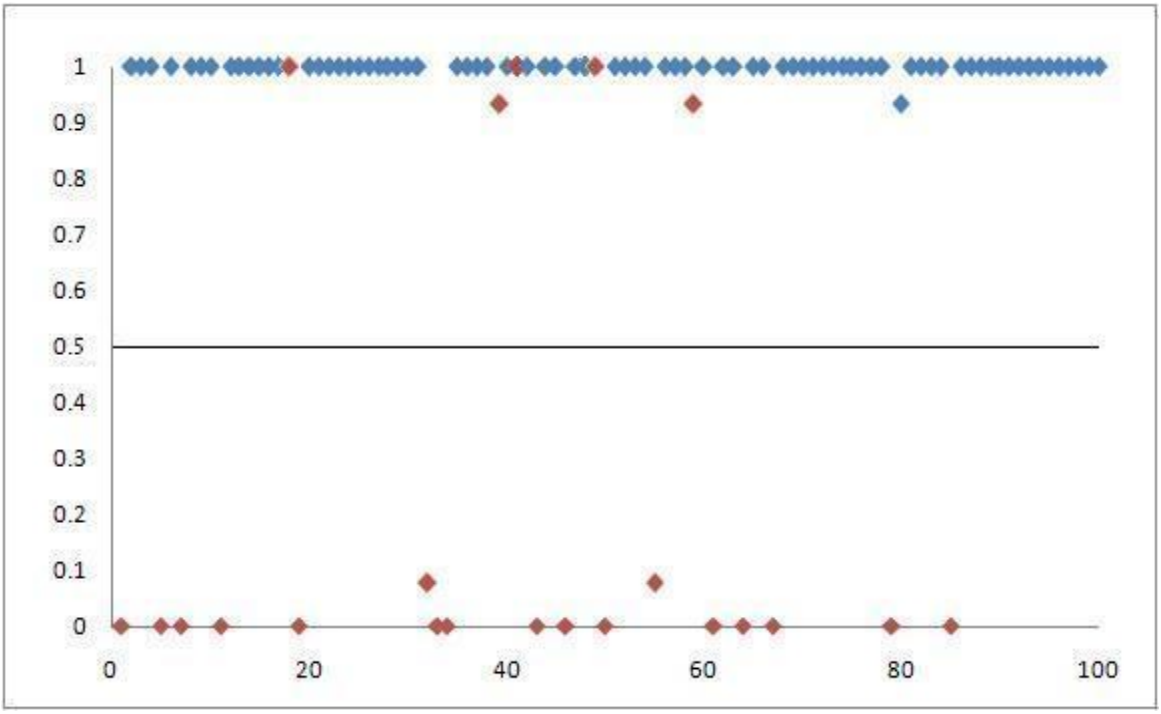

Figure 9.4 The Distribution of probability p_1 for classifier PART on Dataset 1.

For Classifier Prism:

A total of 4 examples were classified wrongly. There are no wrong positive examples, and 4 wrong negative examples. In addition, there are 6 examples classified as "undecidable" ones. (i.e., the blank ones in the middle). They are:

Table 9.17 Partial Detailed Result of Classifier Prism on Dataset 1.

	#	z	(p_1, p_0)
Wrong Positive	-	-	-
Wrong Negative	18	<0010011110>	(1, 0)
	39	<1100001010>	(1, 0)
	49	<1010011110>	(1, 0)
	59	<1101011010>	(1, 0)
Undecidable	2	<0111110110>	(0.5, 0.5)
	24	<1111111110>	(0.5, 0.5)
	38	<0101111110>	(0.5, 0.5)
	42	<1111100111>	(0.5, 0.5)
	66	<1111110111>	(0.5, 0.5)
	72	<0010010010>	(0.5, 0.5)

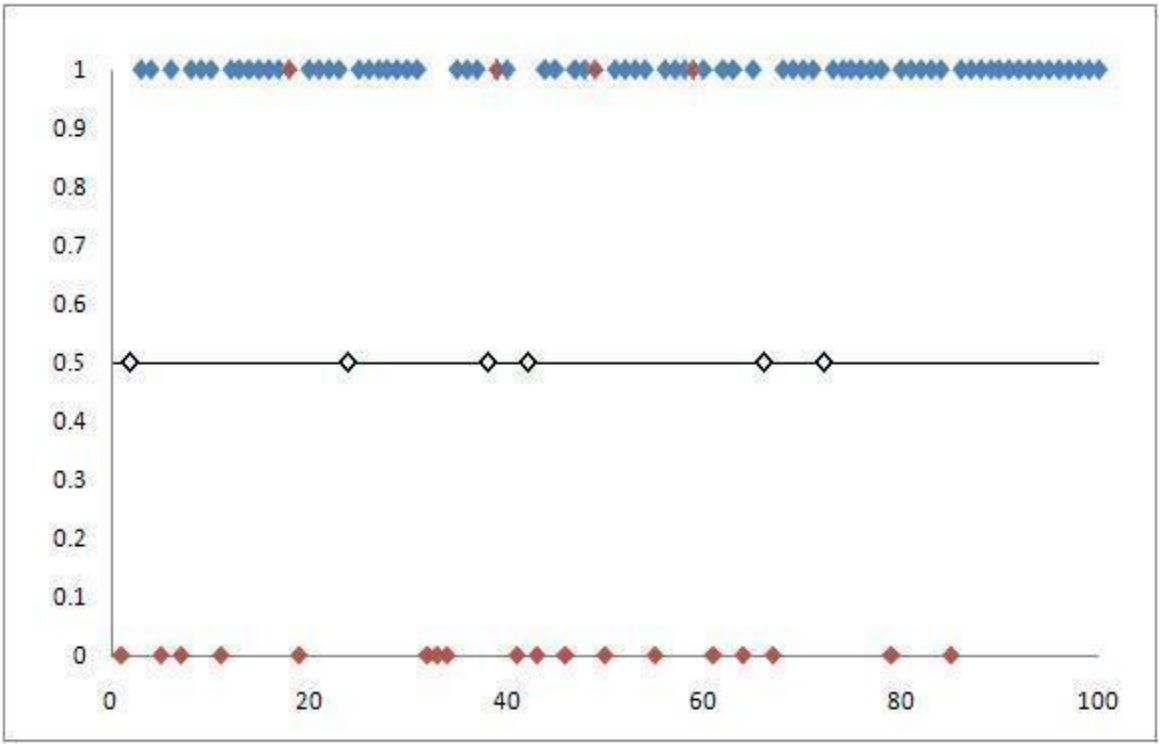

Figure 9.5 The Distribution of probability p_1 for classifier Prism on Dataset 1.

For Classifier Ridor:

A total of 17 examples were classified wrongly. There are 6 wrong positive examples, and 11 wrong negative examples. They are:

Table 9.18 Partial Detailed Result of Classifier Ridor on Dataset 1.

	#	z	(p_1, p_0)
Wrong Positive	2	<0111110110>	(0, 1)
	21	<1011000101>	(0, 1)
	23	<1010010011>	(0, 1)
	26	<0010110011>	(0, 1)
	54	<1011010010>	(0, 1)
	94	<0011010100>	(0, 1)
Wrong Negative	18	<0010011110>	(1, 0)
	32	<1101010011>	(1, 0)
	34	<0110011110>	(1, 0)
	39	<1100001010>	(1, 0)
	41	<0000010110>	(1, 0)
	49	<1010011110>	(1, 0)
	50	<0010010110>	(1, 0)
	55	<1101010010>	(1, 0)
	59	<1101011010>	(1, 0)
	64	<1001101110>	(1, 0)
	79	<0000000100>	(1, 0)

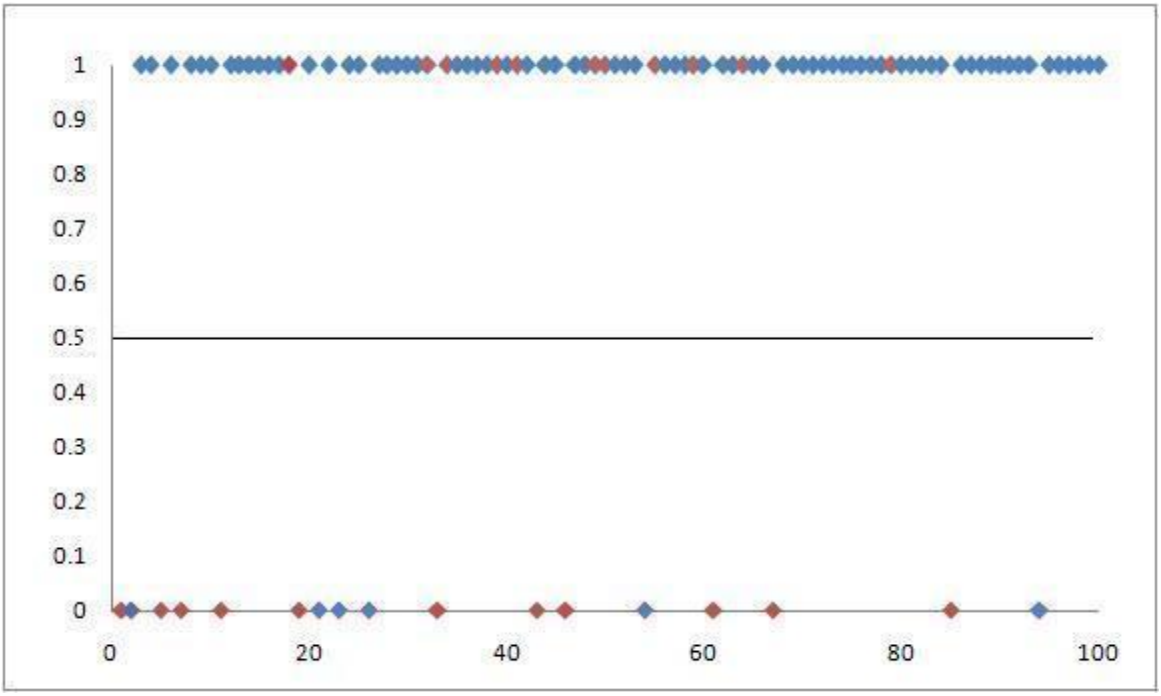

Figure 9.6 The Distribution of probability p_1 for classifier Ridor on Dataset 1.

9.3 Conclusions

In summary,

- On datasets 1 to 10, the *M** algorithm ranks from the 1st to the 8th places.

- Most of the misclassified examples by the *M** algorithm are also misclassified by other fine classifiers (i.e., the classifiers that have outperformed the *M** algorithm).
- The *M** algorithm generates probabilities that are distributed in a wide range. It has the ability to adjust to the change of misclassification costs.

CHAPTER 10. RATIONALE

10.1 Analysis of the Rationale

Though the *M** algorithm gives good results, we still need to know the rationale. Is it just coincidental or there is some reason behind it?

To answer this question, at first we need to transform all the Boolean functions into disjunction normal form. We already know that every Boolean function can be converted into conjunction normal form (CNF) or disjunction normal form (DNF). Moreover, CNF and DNF can be converted to each other by applying the distribution rule. For instance, the hidden function *f* for our illustrative problem can be written in DNF as *f* ':

$$f = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_4 \vee \overline{x_5} \vee x_6)$$

$$\Updownarrow$$

$$f' = (x_1 \wedge \overline{x_2} \vee x_5) \vee (x_1 \wedge \overline{x_2} \vee x_6) \vee (x_1 \wedge x_4) \vee (\overline{x_3} \wedge x_4) \vee (\overline{x_3} \wedge \overline{x_5}) \vee (\overline{x_3} \wedge x_6)$$

The above DNF function means as long an examples satisfies one of the six clauses, the Boolean function will have value "1", i.e., the example will become positive. To satisfy a clause, an example must satisfy all the values of the attributes in that clause. Take the first clause for instance. Satisfying clause $(x_1 \wedge \overline{x_2} \vee x_5)$ means an example must have "1" for the value of x_1, and "0" for the values of x_2 and x_5. Now we keep the values of attributes x_1, x_2, and x_5 as 1, 0, and 0, respectively. Then by filling in the values of the rest of the attributes, we will have all the examples that satisfy the clause $(x_1 \wedge \overline{x_2} \vee x_5)$. If we arrange these examples according to their levels, we will have the following diagram:

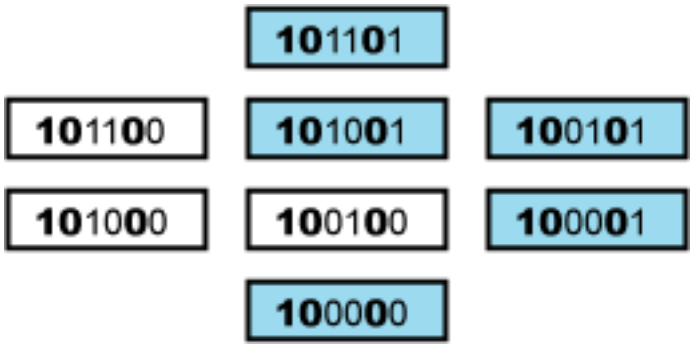

Figure 10.1 All the Examples Satisfying the Clause $(x_1 \wedge \overline{x_2} \vee x_5)$.

There are three important observations based on the above figure:

- All the examples are positive or unclassified examples.
- All the examples form a monotone subset.

- All the unclassified examples are actually positive too, according to the hidden function.

Similarly, we can find out all the examples that satisfy the rest five clauses.

For clause $(x_1 \wedge \overline{x_2} \vee x_6)$:

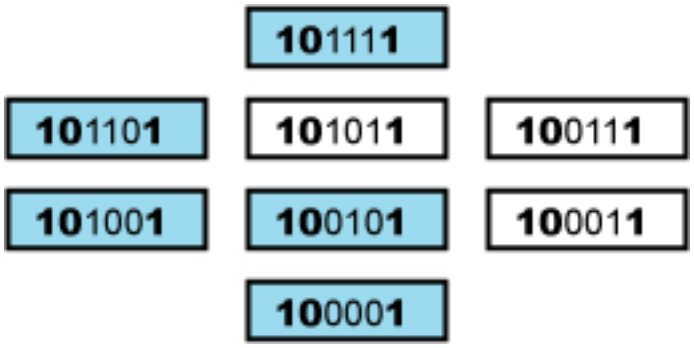

Figure 10.2 All the Examples Satisfying the Clause $(x_1 \wedge \overline{x_2} \vee x_6)$.

For clause $(x_1 \wedge x_4)$:

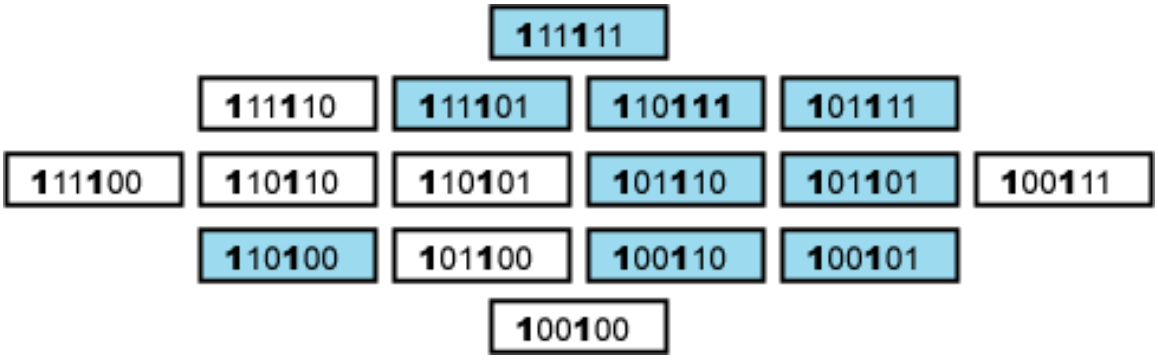

Figure 10.3 All the Examples Satisfying the Clause $(x_1 \wedge x_4)$.

For clause $(\overline{x_3} \wedge x_4)$:

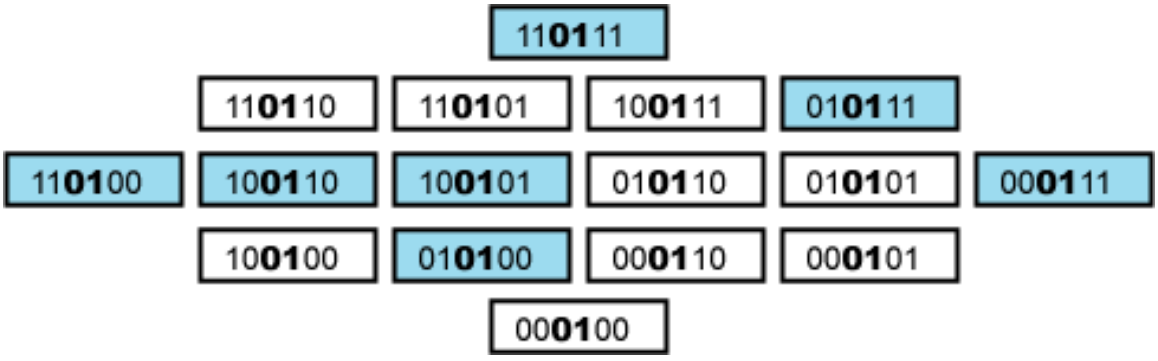

Figure 10.4 All the Examples Satisfying the Clause $(\overline{x_3} \wedge x_4)$.

For clause $(\overline{x_3} \wedge \overline{x_5})$:

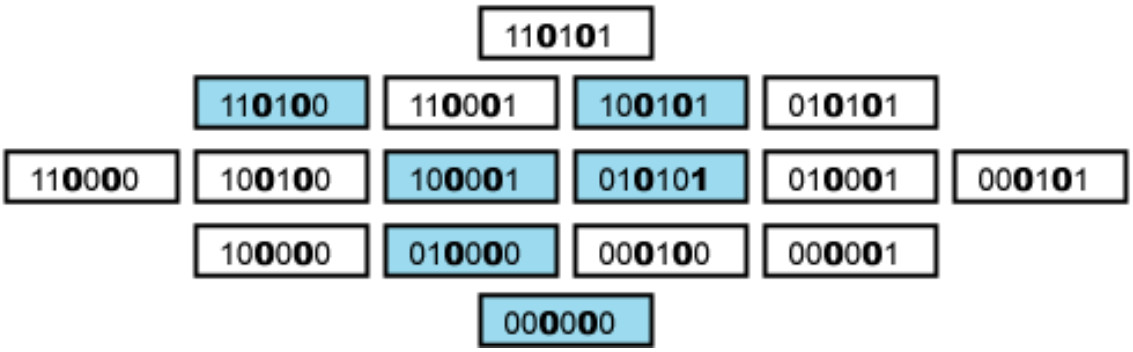

Figure 10.5 All the Examples Satisfying the Clause $(\overline{x_3} \wedge \overline{x_5})$.

For clause $(\overline{x_3} \wedge x_6)$:

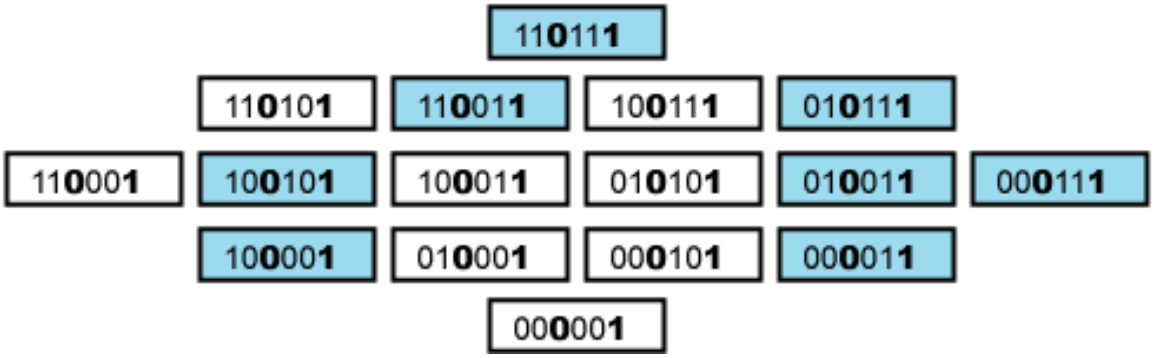

Figure 10.6 All the Examples Satisfying the Clause $(\overline{x_3} \wedge x_6)$.

Based on Figures 10.1 to 10.6, there are some important observations to make:

- All the actual positive examples (including classified and unclassified ones) must be included in one of the above monotone subsets.
- All the examples which are not included in one of the above monotone subsets must be non- positive examples.
- These monotone subsets may overlap with each other, or may be isolated.

To explain the third observation, we can see that the monotone subsets in Figures 10.1 to 10.6 all overlap with each other. For instance, example <100101> is included in all these monotone subsets. Our example above does not have isolated subsets. However, imagine a hidden function as the following:

$$f = (x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_3)$$

Then the subsets implied by the first and second clauses must not have any example in common. That is because the examples in the first subset must have value "1" for

attribute x_1, while the examples in the second subset must have value "0" for attribute x_1. Therefore they cannot be the same sets of examples.

Now if we draw all these positive subsets together in one diagram, we get the following:

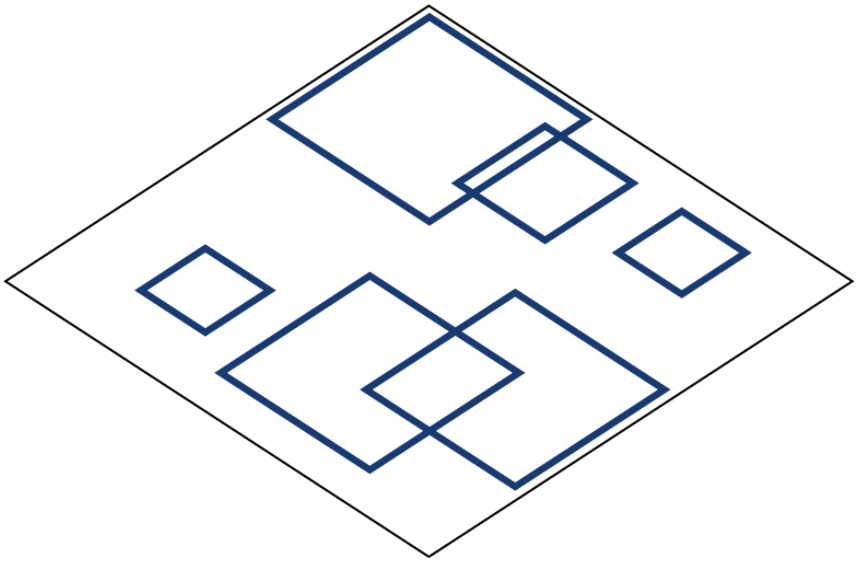

Figure 10.7 The Group of Positive Monotone Subsets.

Similarly, we can have the group of negative subsets. First we negate the original function *f*, then every example that satisfies $\bar{f}$ must have class value "0", or, the example will be negative.

$$\bar{f} = (\overline{x_1} \wedge x_3) \vee (x_2 \wedge x_3 \wedge \overline{x_4}) \vee (\overline{x_4} \wedge x_5 \wedge \overline{x_6})$$

Same as before, all the examples satisfying one of the three clauses must also satisfy the whole function. All the examples satisfying one of the clauses must form a monotone subset. All these monotone subsets must only contain positive examples or unclassified examples which are actually positive.

Therefore, for clause $(\overline{x_1} \wedge x_3)$:

Figure 10.8 All the Examples Satisfying the Clause $(\overline{x_1} \wedge x_3)$.

For clause $(x_2 \wedge x_3 \wedge \overline{x_4})$:

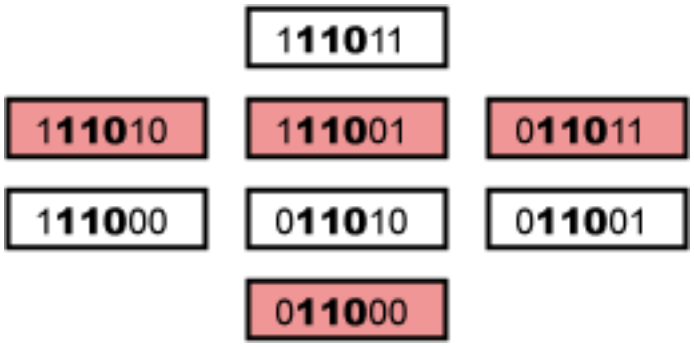

Figure 10.9 All the Examples Satisfying the Clause $(x_2 \wedge x_3 \wedge \overline{x_4})$.

For clause $(\overline{x_4} \wedge x_5 \wedge \overline{x_6})$:

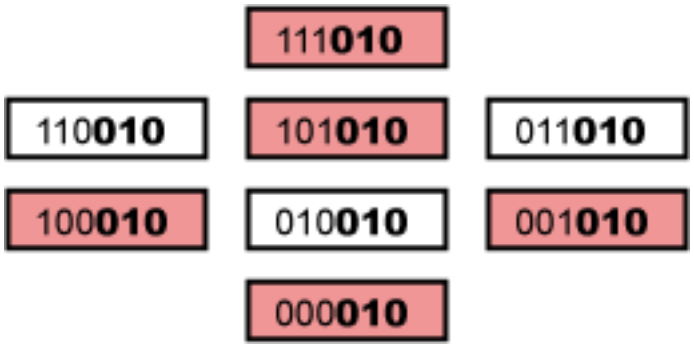

Figure 10.10 All the Examples Satisfying the Clause $(\overline{x_4} \wedge x_5 \wedge \overline{x_6})$.

Similar to positive subsets, all the negative examples must fall in one of the negative subsets. Any example which is not included in the group of the negative subsets must not be in the negative class. The group of negative subsets can be drawn as follows:

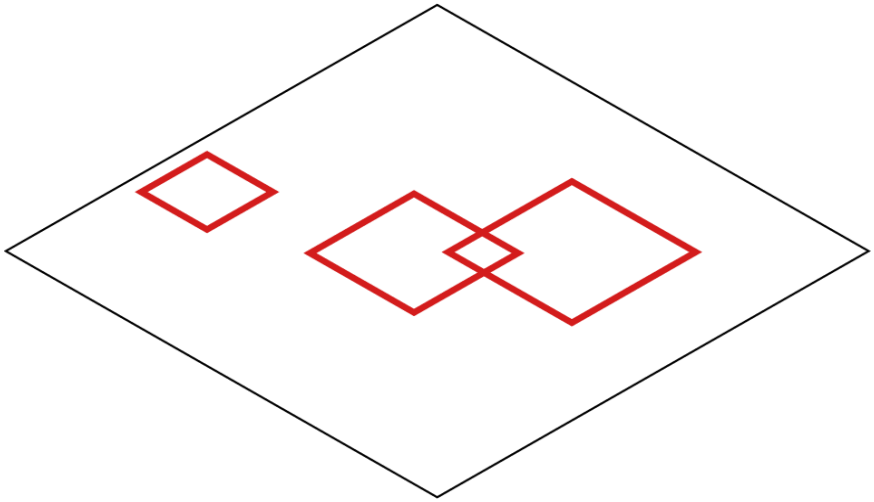

Figure 10.11 The Group of Negative Monotone Subsets.

Now we know all classified or unclassified examples in the positive class must be included in the group of positive subsets, and all classified or unclassified examples in the negative class must be included in the group of negative subsets. Since there are only two classes in binary systems, then the two groups must complement each other. In other words, in Figure 10.7, what is not included in one of the blue regions must be part of the negative subsets. Similarly, in Figure 10.11, what is not included in one of the red regions must be part of the positive subsets.

According to the above reasoning, in the following figure, v_1, an unclassified example which falls in one of the blue region, must have actual class value "1". While v_2, a positive example, must fall in one of the blue regions. v_3, a negative example, must fall out of the blue region; and v_4, an unclassified example which falls out of the blue regions, must have actual class value "0".

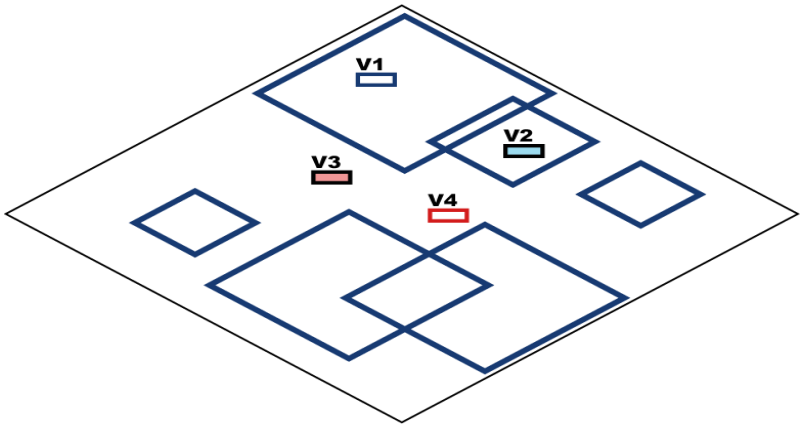

Figure 10.12 The Complement Property of Groups of Positive and Negative Subsets.

Now we know there exist two groups of monotone subsets in every binary system. Suppose we have a target example *z*. Then the ordered examples of *z* will be in the area of the dashed lines as follows:

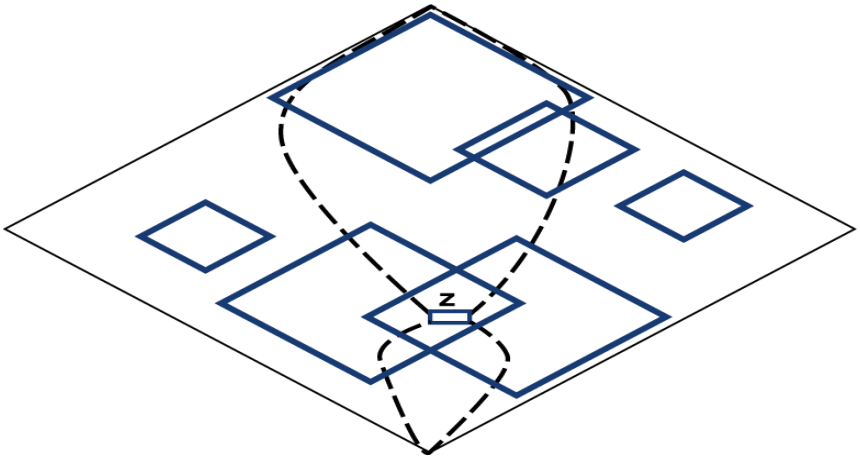

Figure 10.13 A Target Example *z* and the Ordered Examples with Relation to the Group of Monotone Positive Subsets.

When we are trying to determine the class of *z*. An ideal case would be if we knew the classes of all the examples besides *z*. Therefore the ideal case would be like follows:

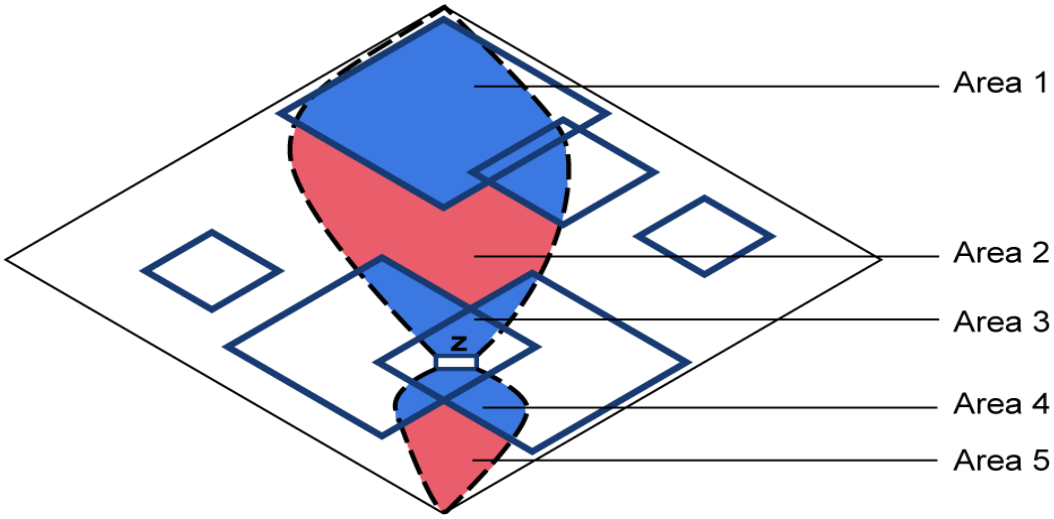

Figure 10.14 The Ideal Case for Determining the Class of the Target Example *z*.

If all the information needed for the ideal case is given, then in the above figure:

Area 1 will be the positive examples in indirect relations with *z*;
Area 2 will be the negative examples in indirect relations with *z*;
Area 3 will be the positive examples in direct relations with *z*;
Area 4 will be the positive examples in direct relations with *z*;
Area 5 will be the negative examples in indirect relations with *z*.

It is obvious that in the ideal case, the class of the target example *z* can be decided by the examples which are in direct relations with *z*. This explains why the "direct relation" type has become the basis of our voting system.

As for the indirect relation type, they could be in the same class as the target example *z*, as examples in area 1; and they also could be in the opposite class as the target example *z*, as examples in areas 2 and 5. This explains why the "indirect relation" type has not played any part in our voting system.

Now suppose we are in a not-so-ideal-case, which means some information is not given to us. In that situation, if we already have found out that a certain example is in indirect relation with the target example *z*, then this example must truly be in indirect relation with *z*. Therefore, this example will be assigned to weight 0. On the other hand, if we have found out that a certain example is in direct relation with z, due to the lack of information, this example might be truly in a direct relation type or it still could be in an indirect relation type. See Figure 10.15 for an illustration.

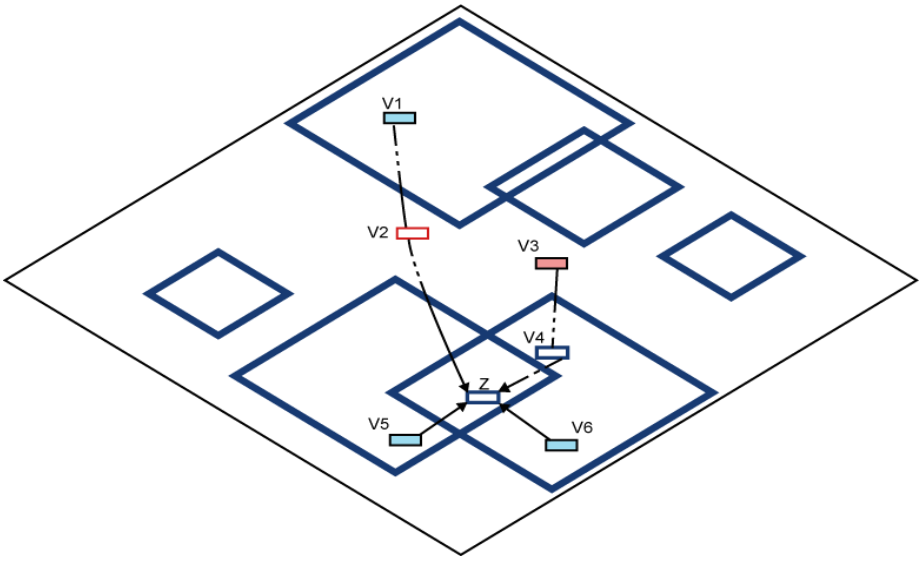

Figure 10.15 The "Direct Relation" Type in a Not-so-ideal-case.

As shown in the above figure, because the classes of v_2 and v_4 are unknown to us right now, v_1 and v_3 are both decided to be in direct relations with the target example z. However, neither v_1 nor v_3 are in fact in direct relation with z. Meanwhile, v_5 and v_6 are truly of the "direct relation" type. Therefore, our assumption here is, the subset(s) that include z might be very small, hence the closer an example is to z, the more likely it is in the same subset as z. This explains why the examples that are closer to the target example should have higher weights.

10.2 Conclusions

From above analysis, we can see that the accuracy of the *M** algorithm is decided by the "depth" of the monotone subsets. By "depth", we mean the number of levels in a subset. For instance, for clause $(x_1 \wedge \overline{x_2} \vee x_5)$, we have a monotone subset with depth 3:

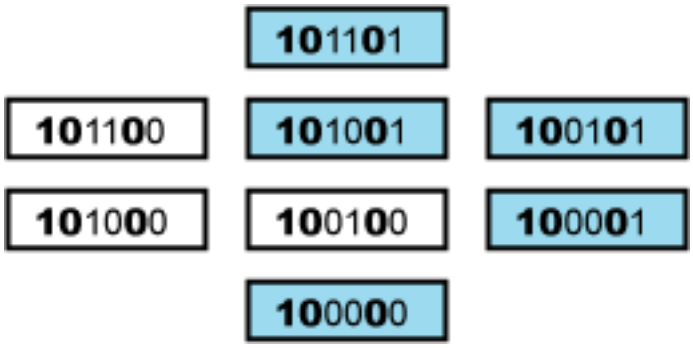

Figure 10. 16 A Monotone Subset with Depth 3.

Meanwhile, for another clause $(x_1 \wedge x_4)$, we have a monotone subset with depth 4:

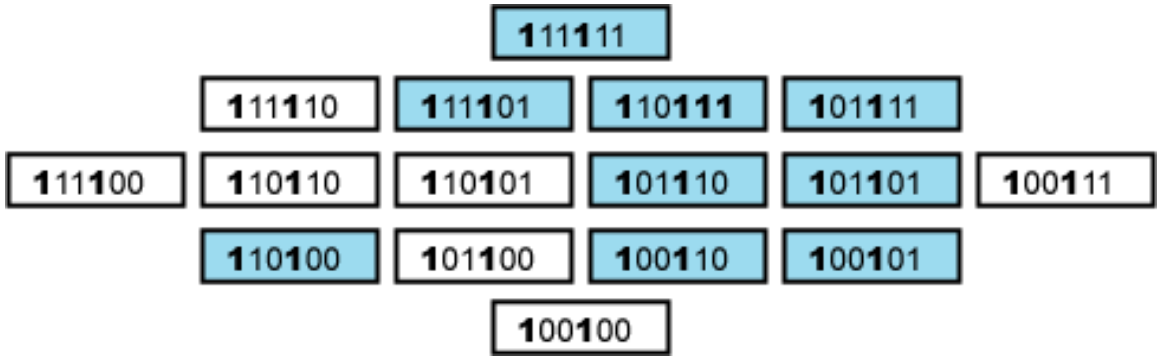

Figure 10.17 A Monotone Subset with Depth 4.

In other words, the depth of a subset is the difference of the number of attribute subtracting the number of attributes specified in the corresponding clause. The deeper the monotone subset where the target example is in, the more examples there are in direct relations with the target example, hence the more accurate the result of the *M** algorithm will be.

Suppose a Boolean function takes the simplest form. If the number of attributes contained in a clause is relatively small, then the corresponding monotone subset will have greater depth. This means the target example that fall in this monotone subset tend to have more examples in direct relations with it, which helps to determine the class of the target example. If the number of attributes contained in a clause is relatively small, then we have the opposite case.

The number of clauses in a Boolean function also affects the accuracy of the *M** algorithm. This is the case because, if we negate the Boolean function to get the Boolean expression of the negative subsets, then the number of attributes contained in each clause is decided by the number of clauses in the original Boolean function.

Suppose there are two binary systems along with two Boolean functions in their simplest form. We have the same size of training datasets and testing datasets for both systems. Besides that, all the examples are randomly selected from both systems. Then the performance of the *M** algorithm on both systems is completely decided by how the hidden Boolean functions are like. Suppose one of the Boolean functions have a lot of clauses and each clause contain a large number of attributes, and the other Boolean function have a relatively small number of clauses and each clause contain a fewer number of attributes. Then, the performance of the *M** algorithm on the first system is definitely worse than the second case.

Led by the above reasoning, we can see what the worst case would be for the *M** algorithm. That is when all the examples at odd levels belong to one class, and all the

examples at the even levels have the other class. In this case, each monotone subset will contain only 1 example, and hence all the subsets will have depth 1. However, this will only happen when the classification attribute is generally irrelevant with the descriptive attributes. This would imply that the selection of the descriptive attributes is wrong. Therefore we would not worry about this situation happening in a well pre-processed system.

CHAPTER 11. CONCLUDING REMARKS

We will summarize what we have concluded so far as follows:

- An upper bound of the execution time for the *M** algorithm is $O(m_1^2 \cdot m_2)$, where m_1 is the number of examples in training data, and m_2 is the number of examples in testing data. This is relatively a loose upper bound. In most cases, the algorithm does not cost that much time.

- On datasets 1 to 10, the *M** algorithm ranks from the 1st to the 8th places compared the other 75 classifiers provided by Weka 3.5.7.

- There are 12 classifiers of 5 categories which have outperformed *M** algorithm at least once. Among them, 5 classifiers have outperformed the *M** algorithm for more than half of the times. These are classifier "NBTree" from Decision Tree methods, classifiers "JRip", "PART", "Prism", and "Ridor" from Rule Induction methods.

- Most of the misclassified examples by the *M** algorithm are also misclassified by other fine classifiers (i.e., the classifiers that have outperformed the *M** algorithm).

- While all the rule induction classifiers that have outperformed the *M** algorithm mostly generate probability pairs (1, 0) or (0, 1), the *M** algorithm generates probability pairs that are distributed in a wide range, which implies the *M** algorithm has the ability to adjust to the change of misclassification costs.

- The performance of the *M** algorithm is decided by the Boolean function which is unknown to us. When the Boolean function take the simplest form, the smaller number of attributes each clause contains, and the smaller number of clauses the Boolean function has, the better performance the *M** algorithm will have.

- The worst case for the *M** algorithm is when all the examples at odd levels belong to one class, and all the examples at the even levels have the other class. In this case, each monotone subset will contain only 1 example, and hence all the subsets will have depth 1. However, this situation should not happen in a well pre-processed system.

CHAPTER 12. SOME POSSIBLE FUTURE RESEARCH DIRECTIONS

12.1 Modification of the Weights

The author still does not think the current weighting scheme is the optimal one. For instance, we only know that the closer certain example is to the target example, the more impact that example has on determining the class of the target example. But we are not sure if increasing weights exponentially according to the distance to the target example is the best way. Or for the examples at the same level as the target example, maybe they have some other options of weight schemes too.

12.2 How to Iterate

Iterations often generate better results, and many good algorithms do have iterates. For now, the *M** algorithm does not iterate at all. Maybe it could run once, and based on the probability pair it gets from the previous run, make adjustment to the results. For instance, suppose at first we do not have much information of the classes of the closely related examples of a target example *z*, then after we run the first iteration, we find out that all the closely related examples have a high probability of belonging to the same class, therefore at the second iteration, we might want to raise the probability for *z* of belonging to that class.

12.3 Effect of Irrelevant and Marked Attributes

An irrelevant attribute is an attribute that does not play any role in the hidden function. For instance, suppose in a binary system of dimension 4 we have the following hidden function:

$$f = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3)$$

Then in this case x_4 is an irrelevant attribute. Many classifiers degrade their performance when there is an irrelevant attribute present. We need to find out what exactly is the impact of an irrelevant attribute on the *M** algorithm.

A marked attribute is just the opposite. Suppose this time we have a binary system of dimension 2, and we have the same hidden function as above. Then under this situation, x_3 is a marked attribute. It does play a role in the hidden function, but it is not recognized in the binary system. A marked attribute will also affect the performance of the classifiers. We want to be able to identify such cases and study them in depth.

12.4 A More General Application

So far, we have seen the *M** algorithm as it has been applied to binary systems where either descriptive attributes or classification attribute can only take values 0 or 1. However, in reality, such binary systems are relatively few. Real systems might have descriptive attributes and classification attribute that can take any data type. So, can the *M** algorithm be applied to these systems? A complete answer is yet to be found. Next

we can show the application of the *M** algorithm on the Iris dataset where the descriptive attributes can take real values and the classification attribute could be nominal and have more than two values.

12.4.1 The Iris Dataset

The dataset we are going to use next is a famous one in the data mining world. It was created by R.A. Fisher and donated by M. Marshall in July of 1988. There have been many notable analyses on this dataset. To name a few, (Fisher, 1950), (Duda, 1973), and (Gates, 1972), their publications have been frequently referenced to this day. This dataset contains 3 classes, and each class refers to a type of iris plant, namely, "Iris Setosa", "Iris Versicolour", and "Iris Virginica". Class "Iris Setosa" is linearly separable from the other two. But "Iris Versicolour" and "Iris Virginica" are not linearly separable from each other. The Iris dataset has four continuous descriptive attributes, i.e., "sepal length", "sepal width", "petal length", and "petal width", all measured in centimeters. It has 150 classified examples, 50 in each of three classes. Part of the Iris dataset is shown as follows:

Table 12.1 Part of the Iris Dataset.

Ex #	Sepal length	Sepal width	Petal length	Petal width	Class
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	7.0	3.2	4.7	1.4	Iris-versicolor
8	6.4	3.2	4.5	1.5	Iris-versicolor
9	6.9	3.1	4.9	1.5	Iris-versicolor
10	5.5	2.3	4.0	1.3	Iris-versicolor
11	6.5	2.8	4.6	1.5	Iris-versicolor
12	5.7	2.8	4.5	1.3	Iris-versicolor
13	6.3	3.3	6.0	2.5	Iris-virginica
14	5.8	2.7	5.1	1.9	Iris-virginica
15	7.1	3.0	5.9	2.1	Iris-virginica
16	6.3	2.9	5.6	1.8	Iris-virginica
17	6.5	3.0	5.8	2.2	Iris-virginica
18	7.6	3.0	6.6	2.1	Iris-virginica
…	…	…	…	…	…

12.4.2 Data Preparation on the Iris Dataset

As it was mentioned before, attributes in any data type can be transformed into binary ones. This process is called Data Binarization. Therefore, our first intuition is to binarize the Iris dataset, and then apply the *M** algorithm on it.

Here, we will use the method describe in (Triantaphyllou, 2008) to binarize the dataset. We will not describe the whole method thoroughly, but we will use the Iris dataset as an illustration to show this binarization process.

First step: find $Val(A_j)$ for all continuous descriptive attributes A_j, where $Val(A_j)$ is an ordered set with all the different values of attribute A_j in an increasing order.

Val("sepal length") = {V_i("sepal length"), for i = 1, 2, 3, …, 35} =
= {4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.6, 7.7, 7.9}

Here, the cardinality of Val("sepal length") is 35. Similarly, we get

Val("sepal width") = {V_i("sepal width"), for i = 1, 2, 3, …, 23} =
= {2.0, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.4}

Val("petal length") = {V_i("petal length"), for i = 1, 2, 3, …, 43} =
= {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.9, 3.0, 3.3, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.3, 6.4, 6.6, 6.7, 6.9}

Val("petal width") = {V_i("petal width"), for i = 1, 2, 3, …, 22} =
= {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5}

Second step: for each continuous descriptive attributes A_j, introduce i binary attributes $A_{j,i}'$, i = 1, 2, 3, …, K, where K is the cardinality of the set $Val(A_j)$, such that

$$A'_{j,i} = \begin{cases} 1, \text{if and only if } A_{j,l} \geq V_i(A_j), \text{for } i = 1,2,3,\ldots,K \\ 0, \text{otherwise} \end{cases}$$

where $A_{j,i}'$ is the i-th binary attribute for A_j, $A_{j,l}$ is the value of attribute A_j for the l-th example, $V_i(A_j)$ is the i-th value of $Val(A_j)$. Therefore, for every continuous number $A_{j,l}$, we can find an i-digit series of 0/1's to replace it.

Table 12.2-1 The Binary Representation of the First Attribute "Sepal length".

Ex #	$A_{1,l}$	Series of 0/1 to replace $A_{1,l}$
1	5.1	11111111100000000000000000000000000
2	4.9	11111110000000000000000000000000000

(table con'd)

3	4.7	11111000000000000000000000000000
4	4.6	11110000000000000000000000000000
5	5.0	11111111000000000000000000000000
6	5.4	11111111111100000000000000000000
7	7.0	11111111111111111111111111110000
8	6.4	11111111111111111111100000000000
9	6.9	11111111111111111111111111100000
10	5.5	11111111111110000000000000000000
11	6.5	11111111111111111111111000000000
12	5.7	11111111111111100000000000000000
13	6.3	11111111111111111111100000000000
14	5.8	11111111111111110000000000000000
15	7.1	11111111111111111111111111111000
16	6.3	11111111111111111111100000000000
17	6.5	11111111111111111111111000000000
18	7.6	11111111111111111111111111111100
…	…	…

Table 12.2-2 The Binary Representation of the Second Attribute "Sepal width".

Ex #	$A_{2,l}$	Series of 0/1 to replace $A_{2,l}$
1	3.5	11111111111111100000
2	3.0	11111111110000000000
3	3.2	11111111111100000000
4	3.1	11111111111000000000
5	3.6	11111111111111110000
6	3.9	11111111111111111110
7	3.2	11111111111100000000
8	3.2	11111111111100000000
9	3.1	11111111111000000000
10	2.3	11000000000000000000
11	2.8	11111111000000000000
12	2.8	11111111000000000000
13	3.3	11111111111110000000
14	2.7	11111110000000000000
15	3.0	11111111110000000000
16	2.9	11111111100000000000
17	3.0	11111111110000000000
18	3.0	11111111110000000000
…	…	…

Table 12.2-3 The Binary Representation of the Third Attribute "Petal length".

Ex #	$A_{3,l}$	Series of 0/1 to replace $A_{3,l}$
1	1.4	11111000000000000000000000000000000000000000
2	1.4	11111000000000000000000000000000000000000000
3	1.3	11110000000000000000000000000000000000000000
4	1.5	11111100000000000000000000000000000000000000
5	1.4	11111000000000000000000000000000000000000000
6	1.7	11111111000000000000000000000000000000000000
7	4.7	11111111111111111111111000000000000000000000
8	4.5	11111111111111111111110000000000000000000000
9	4.9	11111111111111111111111100000000000000000000
10	4.0	11111111111111110000000000000000000000000000
11	4.6	11111111111111111111110000000000000000000000
12	4.5	11111111111111111111110000000000000000000000
13	6.0	11111111111111111111111111111111111111000000
14	5.1	11111111111111111111111111000000000000000000
15	5.9	11111111111111111111111111111111111110000000
16	5.6	11111111111111111111111111111111110000000000
17	5.8	11111111111111111111111111111111100000000000
18	6.6	11111111111111111111111111111111111111111100
…	…	…

Table 12.2-4 The Binary Representation of the Fourth Attribute "Petal width".

Ex #	$A_{4,l}$	Series of 0/1 to replace $A_{4,l}$
1	0.2	11000000000000000000
2	0.2	11000000000000000000
3	0.2	11000000000000000000
4	0.2	11000000000000000000
5	0.2	11000000000000000000
6	0.4	11110000000000000000
7	1.4	11111111110000000000
8	1.5	11111111111000000000
9	1.5	11111111111000000000
10	1.3	11111111110000000000
11	1.5	11111111111000000000
12	1.3	11111111110000000000
13	2.5	11111111111111111111
14	1.9	11111111111111000000
15	2.1	11111111111111110000
16	1.8	11111111111111000000
17	2.2	11111111111111111000
18	2.1	11111111111111110000
…	…	…

Now, we can just concatenate the four sequences of 0/1 and call them representations of the original examples. For instance, the first three examples will become,

<5.1, 3.5, 1.4, 0.2> =>
<11111111100000000000000000000000011111111111111100000000111110000000000000000000000000000000000000000000000011000000000000000000000>

<4.9, 3.0, 1.4, 0.2> =>
<11111110000000000000000000000000011111111110000000000000111110000000000000000000000000000000000000000000000011000000000000000000000>

<4.7, 3.2, 1.3, 0.2> =>
<11111000000000000000000000000000011111111111000000000000111100000000000000000000000000000000000000000000000011000000000000000000000>

However, there are some drawbacks in using this representation. First of all, the four descriptive attributes have become (35+23+43+22) = 123 descriptive attributes. With the growing number of different values of each attributes, the binary representations will become extraordinarily long. Secondly, these examples will form a very sparse poset. Notice that the binary representations are concatenated by four sequences of 0/1's, and each sequence is formed by a sub-sequence of 1's followed by a sub-sequence of 0's. This means a lot of 0/1 series that do not follow this pattern will never become a valid example. For instance, the example of all 0's will never be used, nor the example formed by repeating the string 01.

So, how do we fix these problems? Well, a little revision shall be made. We can take advantage of the fact that these 0/1 sequences have patterns as we mentioned before, i.e., the binary representations are concatenated by four sequences of 0/1's, and each sequence is formed by a sub-sequence of 1's followed by a sub-sequence of 0's. We can replace <11111111100000000000000000000000011111111111111100000000111110000000000000000000000000000000000000000000000011000000000000000000000> with <9, 15, 5, 2>, and will lose no information. The value 9 means in the binary sequence that represent the first continuous attribute there are 9 1's. 9 also means the original continuous number 5.1 takes the 9-th place in the ordered set $Val(A_1)$. Recall that $Val(A_1) = \{V_i(A_1)$, for $i = 1, 2, 3, \ldots, 35\} = \{4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.6, 7.7, 7.9\}$. Of course, similarly with above, the second number 15 means in the representation of A_2 that there are 15 1's, or the original continuous number 3.5 takes the 15-th place in the ordered set $Val(A_2)$.

In this way, we have

<5.1, 3.5, 1.4, 0.2> => <9, 15, 5, 2>

<4.9, 3.0, 1.4, 0.2> => <7, 10, 5, 2>

<4.7, 3.2, 1.3, 0.2> => <5, 12, 4, 2>

We will use this new representation $<A_1', A_2', A_3', A_4'>$. Attribute A_1' can take natural numbers from 1 to the cardinality of $Val(A_1) = 35$; A_2' can take natural numbers from 1 to the cardinality of $Val(A_2) = 23$; A_3' can take natural numbers from 1 to the cardinality of $Val(A_3) = 43$; A_4' can take natural numbers from 1 to the cardinality of $Val(A_4) = 22$.

The ordered relation "$\succ$" is defined as follows:

> $<a_1', a_2', a_3', a_4'> \succ <b_1', b_2', b_3', b_4'>$, iff at least one of the attribute(s) is greater that the corresponding attribute(s), and the rest attribute(s) are greater than and equal to the corresponding attribute(s).
>
> That is, $<a_1', a_2', a_3', a_4'> \succ <b_1', b_2', b_3', b_4'>$, iff one of the following conditions holds:
>
> $$a_1' > b_1', a_2' \geq b_2', a_3' \geq b_3', a_4' \geq b_4'$$
> $$a_1' \geq b_1', a_2' > b_2', a_3' \geq b_3', a_4' \geq b_4'$$
> $$a_1' \geq b_1', a_2' \geq b_2', a_3' > b_3', a_4' \geq b_4'$$
> $$a_1' \geq b_1', a_2' \geq b_2', a_3' \geq b_3', a_4' > b_4'$$

According to the definition of the ordered relation "$\succ$", we have

<35, 23, 43, 22> $\succ$ <1, 1, 1, 22>

<15, 15, 15, 15> $\succ$ <14, 15, 15, 15>

<15, 15, 15, 15> $\succ$ <10, 10, 10, 10>

This diagram has <35, 23, 43, 22> as the maximal example, <1, 1, 1, 1> as the minimal example, and every example in between is valid.

Now, we are ready to apply the *M** algorithm method! The performance of the *M** algorithm on such real datasets needs to be found out next.

REFERENCES

Aha, D. W., D. Kibler, and M. K. Albert (1991): Instance-based learning algorithms. Machine Learning, 6(1): 37–66.

Alpaydın, E. (2004): Introduction to Machine Learning (Adaptive Computation and Machine Learning), MIT Press, ISBN 0262012111.

Bartnikowski, S., M. Grandberry, J. Mugan, and K. Truemper (2006): "Transformation of Rational Data and Set Data to Logic Data," in Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques, Triantaphyllou, E., and G. Felici, (editors), Massive Computing Series, Springer, Heidelberg, Germany, pp. 253-278.

Berger, J. O. (1999): Statistical Decision Theory and Bayesian Analysis. Second Edition. Springer Verlag, New York. ISBN 0-387-96098-8 and also ISBN 3-540-96098-8.

Bhagat, P. M. (2005): Pattern Recognition in Industry, Elsevier. ISBN 0-08-044538-1.

Bishop, C. M. (1995): Neural Networks for Pattern Recognition, Oxford: Oxford University Press. ISBN 0-19-853849-9.

Bishop, C. M. (2007): Pattern Recognition and Machine Learning, Springer ISBN 0-387-31073-8.

Bolstad, W. M. (2004): Introduction to Bayesian Statistics, John Wiley ISBN 0-471-27020-2.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984): Classification and Regression Trees. Wadsworth International Group, Belmont, California.

Breiman, L. (2001): Random Forests. Machine Learning. 45(1): 5-32.

Buhmann, M. D., and M. J. Ablowitz (2003): Radial Basis Functions : Theory and Implementations. Cambridge University. ISBN 0-521-63338-9.

Cendrowska, J. (1987): PRISM: An algorithm for inducing modular rules. International Journal of Man-Machine Studies. 27(4): 349-370.

Cleary, J. G., and L. E. Trigg (1995): K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 108-114.

Cohen, W. W. (1995): Fast Effective Rule Induction. In: Twelfth International Conference on Machine Learning, 115-123.

Dasarathy, B. V. editor (1991): Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, ISBN 0-8186-8930-7.

Davies, E. R. (1988): Training sets and a priori probabilities with the nearest neighbor method of pattern recognition. Pattern Recognition Letters, 8: 11–13.

Devijver, P. A. and J. V. Kittler (1982): Pattern Recognition. A Statistical Approach. Prentice Hall, Englewood Cliffs.

Devilly, G. J. (2004): Random Number Generator: Version 1.0 [computer program]. Centre for Neuropsychology, Swinburne University, Australia.

Duda, R., and P. Hart (1973): Pattern Classification and Scene Analysis. Wiley, New York.

Duda, R. O., P. E. Hart, and D. G. Stork (2001): Pattern classification (2nd edition), Wiley, ISBN 0-471-05669-3.

Egmont-Petersen, M., D. de Ridder, and H. Handels (2002): "Image processing with neural networks - a review". Pattern Recognition 35: 2279–2301.

Frank, E., and I. H. Witten (1998): Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, 144-151.

Frawley, W., G. Piatetsky-Shapiro and C. Matheus (1992): "Knowledge Discovery in Databases: An Overview". AI Magazine: pp. 213-228. ISSN 0738-4602.

Freund, Y., and L. Mason (1999): The alternating decision tree learning algorithm. In: Proceeding of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, 124-133.

Friedman, J., T. Hastie, and R. Tibshirani (2000): Additive logistic regression : A statistical view of boosting. Annals of statistics. 28(2): 337-407.

Fukunaga, K., and P. M. Narendra (1975): A branch and bound algorithm for computing nearest neighbors. IEEE Trans. Comput., C-25: 917–922.

Gaines, B. R., and P. Compton (1995): Induction of Ripple-Down Rules Applied to Modeling Large Databases. J. Intell. Inf. Syst.. 5(3): 211-228.

Gates, G. W. (1972): Thereduced nearest neighbour rule. IEEETransactions on Information Theory, IT-18:431.

Gelman, A., J. Carlin, H. Stern, and D. B. Rubin (2003): Bayesian Data Analysis. Second Edition. Chapman & Hall/CRD, Boca Raton, Florida. ISBN 1-58488-388-X.

Hand, D. J., and B. G. Batchelor (1978): An edited nearest neighbour rule. Information Sciences, 14:171–180.

Hand, D., H. Mannila, and P. Smyth (2001): Principles of Data Mining. MIT Press, Cambridge, MA,. ISBN 0-262-08290-X.

Haykin, S. (1998): Neural Networks: A Comprehensive Foundation, 2, Prentice Hall. ISBN 0132733501.

Hart, P. E. (1968): Thecondensed nearest neighbour rule. IEEE Transactions on Information Theory, IT-14:515–516.

Hertz, J., A. Krogh, and R. Palmer (1991): Introduction to the Theory of Neural Computation. Addison-Wesley.

Holte, R. C. (1993): Very simple classification rules perform well on most commonly used datasets. Machine Learning. 11:63-91.

Jensen, F. V. (1996): An introduction to Bayesian networks. New York: Springer Verlag.

Jiang, L., and H. Zhang (2006): Weightily Averaged One-Dependence Estimators. In: Proceedings of the 9th Biennial Pacific Rim International Conference on Artificial Intelligence, PRICAI 2006, 970-974.

John, G. H., and P. Langley (1995): Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345.

Kohavi, R. (1995): The Power of Decision Tables. In: 8th European Conference on Machine Learning, 174-189.

Kohavi, R. (1996): Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In: Second International Conference on Knoledge Discovery and Data Mining, 202-207.

Koppelberg, S. (1989): "General Theory of Boolean Algebras", Handbook of Boolean Algebras, Vol. 1 (ed. J. Donald Monk with Robert Bonnet). Amsterdam: North Holland. ISBN 978-0-444-70261-6.

Kovalerchuk, B., E. Triantaphyllou, and E. Vityaev (1995): Monotone Boolean Function Learning Techniques Integrated with User Interaction. Proceedings of Workshop "Learning from Examples vs. Programming by Demonstration", 12th International Conference on Machine Learning, Lake Tahoa, CA, USA, 41-48.

Kovalerchuk, B., E. Triantaphyllou, and A. S. Deshpande (1996): Interactive Learning of Monotone Boolean Functions. Information Sciences 94 87-118.

Landwehr, N., M. Hall, and E. Frank (2005): Logistic Model Trees. Machine Learning. 95(1-2): 161-205.

Lee, P. M. (1997): Bayesian Statistics: An Introduction. Second Edition. ISBN 0-340-67785-6.

Martin, B. (1995): Instance-Based learning: Nearest Neighbor With Generalization. Hamilton, New Zealand.

McLachlan, G. J. (1992): Discriminant Analysis and Statistical Pattern Recognition. John Wiley, New York.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (1994): Machine Learning, Neural and Statistical Classification. Ellis Horwood.

Mitchell, T. (1997): Machine Learning, McGraw Hill. ISBN 0-07-042807-7.

Nieto Sanchez, S., E. Triantaphyllou, J. Chen, and T. W. Liao (2001): An Incremental Learning Algorithm for Constructing Boolean Functions From Positive and Negative Examples. Computers and Operations Research.

O'Hagan, A. and J. Forster (2003): Kendall's Advanced Theory of Statistics, Volume 2B: Bayesian Inference. Arnold, New York. ISBN 0-340-52922-9.

Quinlan, R. (1986): Induction of decision trees. Machine Learning. 1(1): 81-106.
Quinlan, R. (1993): C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

Roy, S. (2002): Nearest Neighbor With Generalization. Christchurch, New Zealand.

Schalkoff, R. J. (1992): Pattern Recognotion: Statistical, Structural and Neural Approaches. Wiley, Singapore.

Scott, D. W. (1992): Multivariate Density Estimation: Theory, Practice, and Visualization. JohnWiley, New York.

Shakhnarovish, Darrell, and Indyk, editors (2005): Nearest-Neighbor Methods in Learning and Vision, edited by The MIT Press, ISBN 0-262-19547-X.

Shi, H. (2007): Best-first decision tree learning. Hamilton, NZ.

Sumner, M., E. Frank, and M. Hall (2005): Speeding up Logistic Model Tree Induction. In: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, 675-683.

Todeschini, R. (1989): k-nearest neighbor method: the influence of data transformations and metrics. Chemometrics Intell. Labor. Syst., 6: 213–220.

Triantaphyllou, E. (1994): Inference of a Minimum Size Boolean Function by Using a New Efficient Branch-and-Bound Approach from Examples. Journal of Global Optimization 5 69-84.

Triantaphyllou, E. and A. L. Soyster, (1996a): An Approach to Guided Learning of Boolean Functions.

Triantaphyllou, E. and A. L. Soyster, (1996b): On the Minimum Number of Logical Clauses Which Can be Inferred From Examples. Computers and Operations Research 23 (8) 783-799.

Triantaphyllou, E. (2008): Data Mining and Knowledge Discovery via a Logic-Based Approach. Springer.

Webb, G. (1999): Decision Tree Grafting From the All-Tests-But-One Partition, San Francisco, CA.

Webb, G., J. Boughton, and Z. Wang (2005): Not So Naive Bayes: Aggregating One-Dependence Estimators. Machine Learning. 58(1): 5-24.

Weiss, S. M. and C. A. Kulikowski, (1991): Computer systems that learn: classification and prediction methods from statistics, neural networks, machine learning and expert systems. Morgan Kaufmann, San Mateo, CA.

Winkler, R. L. (2003): Introduction to Bayesian Inference and Decision, 2nd Edition Probabilistic. ISBN 0-9647938-4-9.

Witten, I. H., and E. Frank (2005): "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

Yee, P. V. and S. Haykin (2001): Regularized Radial Basis Function Networks: Theory and Applications. John Wiley. ISBN 0-471-35349-3.

Zhang, H., L. Jiang, and J. Su (2005): Hidden Naive Bayes. In: Twentieth National Conference on Artificial Intelligence, 919-924.

Zheng, F., G. I. Webb (2006): Efficient lazy elimination for averaged-one dependence estimators. In: Proc. 23th Int. Conf. Machine Learning (ICML 2006), 1113-1120.

VITA

Hongyi Chen was born in Chengdu, China. She received the degree of Bachelor of Engineering in Computer Science from Sichuan University, China, in 2001. She then worked as a C/C++ programmer in China for three years before she joined the graduate program at Department of Computer Science of Louisiana State University, Baton Rouge (LSU) in fall, 2004. Her research interests are data mining, graph theory, and combinatorics. She is a candidate for the degree of Master of Science in computer science to be awarded at the commencement of Summer 2008.