International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:4, 2007

Induction of Expressive Rules using the Binary Coding Method

Seyed R Mousavi

***Abstract*—**In most rule-induction algorithms, the only operator used against nominal attributes is the equality operator =. In this paper, we first propose the use of the inequality operator, ≠, in addition to the equality operator, to increase the expressiveness of induced rules. Then, we present a new method, *Binary Coding*, which can be used along with an arbitrary rule-induction algorithm to make use of the inequality operator without any need to change the algorithm. Experimental results suggest that the Binary Coding method is promising enough for further investigation, especially in cases where the minimum number of rules is desirable.

***Keywords*—**Data mining, Inequality operator, Number of rules, Rule-induction.

I. Introduction

IN general, a rule induction algorithm operates on a given dataset and generates a model based on a set of rules. One of the main types of attributes in such datasets is the *nominal* type. The value of a nominal attribute is usually a value in a predefined and finite set of discrete values. For example, in an application, we may define *Color* as an attribute which can only be *Red*, *Green*, or *Blue*. Other examples of nominal attributes include the Family, Product-type, Steel, and Formability in the Annealing dataset, all the attributes in the Car Evaluation, and all the attributes in the Vote datasets, which can all be found in the UCI Data Repository [1], at http://www.ics.uci.edu/~mlearn/MLRepository.html.

For the purpose of this paper, we introduce two types of nominal attributes. By a *binary* nominal attribute, or for short a binary attribute, we mean a nominal attribute which only has two possible values; e.g. Gender, which can only be Male or Female. On the other hand, by a *multi-value* nominal attribute, or for short a multi-value attribute, we mean an attribute which has more than two possible values. For example, if Color can be Red, Green, or Blue, then it will be a multi-value attribute.

Most well-known rule-induction algorithms only use the equality operator, =, against multi-value attributes [2]-[5]. In this paper, we propose the use of the inequality operator, ≠, in addition to the equality operator, against multi-value attributes. We first explain how this would increase the expressiveness of the induced rules, thereby giving the potential for reducing the number of rules. Obviously, a possible way to include the inequality operator in a rule-induction algorithm is to modify the algorithm. This would be desirable with respect to flexibility, i.e. the way the inequality operator is handled within the algorithm is completely under control and can be customized for that specific algorithm. However, in most cases, it would not be possible, or at least not desirable, to do so; e.g. the user of the algorithm might not be a computer programmer, or there might not be access to the code of the algorithm. To overcome this issue, in this paper we also present a new method called *Binary Coding*, which allows for a rule-induction algorithm to make use of the inequality operator, without any need to modify the algorithm.

The rest of the paper is organized as follows. Section II describes how the use of the inequality operator against multi-value attributes allows for more expressive rules. In Section III, we outline possible approaches to make use of the inequality operator. We describe the Binary Coding method in Section IV. Experimental Results are presented in Section V. Section VI summarizes the paper.

S. R. Mousavi is a Ph.D. student in the Computing Department., Imperial College London, SW7 2BZ UK (phone: +44 (0)20 8740 9377; e-mail: bsm99@doc.ic.ac.uk).

II. More Expressiveness by Inequality Operator

Let a be a multi-value attribute with n possible values of v_1, v_2, ..., v_n. The reason why the use of the inequality operator allows for higher expressiveness of rules is that (i) a single test $a \neq v_i$, $1 \leq i \leq n$, is equivalent to the disjunction of n-1 test of
$a = v_j$, where $j \neq i$ and $1 \leq j \leq n$, i.e.:

$$(a \neq v_i) \equiv \bigvee_{\substack{1 \leq j \leq n \\ j \neq i}} (a = v_j)$$

and (ii) the condition part of rules are normally conjunctions, as opposed to disjunctions, of tests; i.e. disjunctions must be represented by several rules. To clarify this, consider the

TABLE I
A SAMPLE DATASET

Theory	Project	Gender	Result
High	Good	F	Pass
Low	Good	M	Fail
High	Bad	M	Fail
Low	Bad	F	Fail
High	Good	M	Pass
Med	Good	M	Pass
High	Bad	F	Fail
Med	Bad	F	Fail

dataset shown in Table I.

In this dataset, there are three attributes: Theory, Project, and Gender. Project and Gender are both binary because each of them has only two possible values: Good and Bad for Project, and Female (F) and Male (M) for Gender. On the other hand, Theory is a multi-value attribute because it has three possible values: High, Medium (Med.), and Low.

The following ruleset, *R*, has been derived by applying ID3 [6] to this dataset:

1. IF Project=Good AND Theory=High → Pass
2. IF Project=Good AND Theory=Mid → Pass
3. IF Project=Good AND Theory=Low → Fail
4. IF Project=Bad → Fail

Using the ≠ operator, it is possible to obtain the following ruleset, say $R^{\neq}$, which has fewer rules (3 instead of 4) and fewer tests (5 instead of 7):

1. IF Project=Good AND Theory ≠ Low → Pass
2. IF Project=Good AND Theory=Low → Fail
3. IF Project=Bad → Fail

It is easy to see that $R^{\neq}$ is equivalent to *R* (rules 1 and 2 in *R* are reduced to rule 1 in $R^{\neq}$). In other words the use of the ≠ operator in the first rule of $R^{\neq}$ has increased its expressive power such that it is now equivalent to the first two rules in the ruleset *R*.

In the general case, let $a_1, a_2, \ldots, a_n$ be *n* distinct multi-value attributes with, respectively, $m_1, m_2, \ldots, m_n$ nominal values. If a rule consists of k_1 tests with '≠' against attribute a_1, k_2 tests with '≠' against attribute a_2, …, and k_n tests with '≠' against attribute a_n, then it will be equivalent to $\prod_{i=1}^{n}(m_i - k_i)$ rules in which only '=' has been used against the attributes $a_1, a_2, \ldots, a_n$. In the case of the first rule in $R^{\neq}$, for instance, we have n=1, m_1=3, and k_1=1. So the rule is equivalent to 2 rules with '=' operator, i.e. the first two rules in *R*.

III. How to Induce Rules with Inequlity Operator

Let us use $R^{\neq}$ to denote a ruleset in which the inequality operator, ≠, is allowed, and keep *R* to denote a ruleset with the ordinary syntax. In general, a rule-induction algorithm, *A*, operates on a given dataset *D* and generates a ruleset *R* as in Fig. 1.

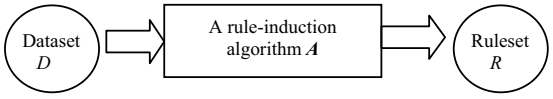

Fig. 1. A Generic Rule-induction Algorithm

In order to obtain a ruleset $R^{\neq}$, there are several possibilities, which include:

--Extend the algorithm *A* to *A′* such that *A′* considers ≠ as a possible operator, in addition to =, against multi-value attributes (Fig. 2).

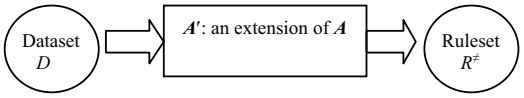

Fig. 2. Algorithm *A′* is an Extension of Algorithm A

--Devise a transformer algorithm that converts *R* to $R^{\neq}$, independently from the underlying algorithm *A* (Fig. 3). In other words, the original algorithm *A* is used as usual to generate *R,* and then a transformer algorithm converts *R* to $R^{\neq}$, irrespective from *A*.

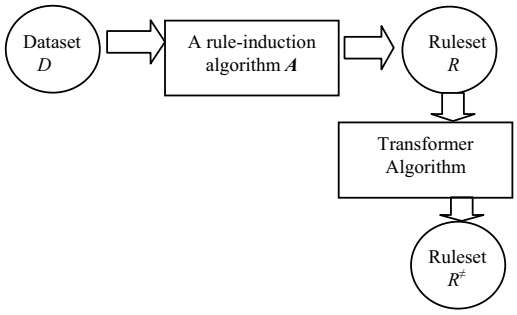

Fig. 3. Use a Transformer Algorithm to Convert *R* to $R^{\neq}$.

--Use the method of Binary Coding described in the next section.

The advantage of the first approach is that the way the inequality operator is used in an algorithm is entirely under the control and can be customized for that algorithm to achieve more desirable results. C4.5, for instance, is a sort of algorithms which has followed this approach [7]. Although it does not explicitly generate rules with the inequality operators, it uses the *belong to* operator, when run with –s switch, which is implicitly equivalent to using the inequality operator. However, the disadvantage with the first approach is that it has to be done for each individual algorithm separately. The second approach, on the other hand, works independently from the underlying algorithm. In other words, if a transformer algorithm is devised to convert a ruleset *R* to an equivalent ruleset $R^{\neq}$, then it will not be restricted to specific algorithms. However, no such algorithm has been proposed yet.

The method of Binary Coding, which is described in detail in the next section, has the advantage of the second approach, i.e. it is a generic method independent from the underlying algorithm. Moreover it is simple.

IV. The Method of Binary Coding

The method of Binary Coding (Fig. 4) adds two modules to a rule-induction system: (1) a pre processor called *Binary Encoder* and (2) a post processor called *Binary Decoder*. These two modules are co-related but entirely independent from the underlying rule-induction algorithm. Therefore, this

approach is a generic approach which may be applied to an arbitrary rule-induction algorithm.

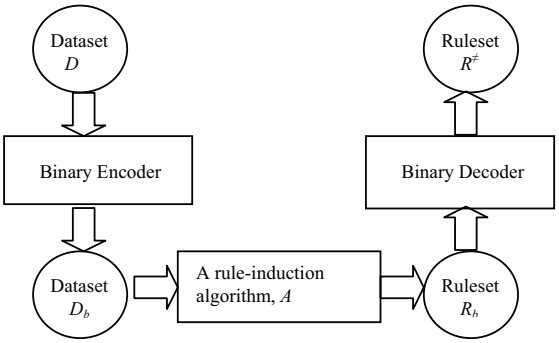

Fig. 4. The Binary Coding Method.

The Binary Encoder and the Binary Decoder are described below:

A. Binary Encoder

The Binary Encoder is used to encode the dataset D into another dataset D_b in the following way. Let a be a multi-value attribute with k possible values $v_1,...,v_k$. Then in D_b, a is replaced by k binary attributes $a_1,...,a_k$, such that for each sample x, the value of a_i will be 1 if and only if the value of a for x in D is v_i. Note that for each sample in D_b, at most, one of the attributes $a_1, a_2, ..., a_k$ can be 1, because a in D cannot have more than one value for each sample. If the value of a is missing, then the value of every a_i will also be considered as missing. We will call D_b a *binary dataset* because it is a dataset with no multi-value attribute.

Algorithm Binary_Encoder

$D_b := D$
For each multi-value attribute a in D with k possible values $v_1,v_2,..,v_k$ do
  Replace attribute a in D_b with k binary attributes $a_1, a_2, .., a_k$.
    For each sample x do
      If value of a in D is missing then
        set the value of each a_i, $1 \le i \le k$, in D_b as missing
      Else
        Let the value of a be v_i, $1 \le i \le k$
        set the value of a_i in D_b as 1
        set the value of a_j , $j \neq i$, in D_b as 0
End.

The Binary Encoder module can be represented by the following algorithm:

B. Binary Decoder

Let a be a multi value attribute in D with k possible values $v_1, v_2,...,v_k$ which has been replaced with k binary attributes $a_1, a_2, ..a_k$ in D_b. As the rule-induction algorithm operates on D_b, as opposed to D (Fig. 4), the resulting ruleset R_b would be expressed in terms of attributes $a_1, a_2, ..a_k$, as opposed to attribute a. However, since the original dataset D is based on attribute a, and not attributes $a_1, a_2, ..a_k$, it is desirable to convert each test based on a_i to the equivalent test based on the original attribute a. This conversion is the duty of the Binary Decoder module.

Let $a_j=b$, where b is either 0 or 1, be a test used in a rule in the ruleset R_b. This test evaluates to true if and only if:

b=1 and the value of the original attribute a is v_j, or

b=0 and the value of the original attribute is anything other than v_j.

In other words, the test $a_j=b$ can be replaced with $a=v_i$ when b=1 and with $a \neq v_j$ when b=0.

Algorithm Binary_Decoder

$R^{\neq} := R_b$
For each multi-value attribute a in D with k possible values $v_1,v_2,..,v_k$ which correspond respectively to attributes $a_1,a_2,...,a_k$ in D_b do
  For each rule r in $R^{\neq}$ do
    replace each test (a_i=1) in r with ($a=v_i$)
    replace each test (a_i=0) in r with ($a \neq v_i$)
End.

The Binary Decoder module can be represented by the following algorithm:

V. Experimental Results

In order to test the Binary Coding method we applied it to the TDIDT(entropy), Prism (standard), and J-Rule algorithms, using the *Inducer* software [8], [9]. All of the datasets are selected from the UCI repository [1].

The results of the experiments are shown in Table II. The first column shows the underlying dataset. The second column indicates the total number of classes in the datasets. The third column includes the number of attributes (binary, multi-value, and continuous, respectively). The last three columns show the results for TDIDT, Prism, and J-Rule algorithms, respectively. The experiments are based on the 10-fold test [9]. For each dataset for each algorithm, there are two sets of figures: the figures at the top are the result without using Binary Coding, and the ones at the bottom are the results when it has been used. For each set of data, there are three values. The first value is the average number of rules, the second value (inside parentheses) is the average number of tests per rule, and the last value is the average correct percentage. The correct percentage is the ratio of the number of samples for which the resulting ruleset provides the right class to the total number of samples.

As indicated in Table II, the Binary Coding method has improved considerably the TDIDT output for most of the cases. For Prism and J-rule, the improvement in the number of rules is roughly within 10% for about 66% of the cases. Among the other 34%, considerable improvements can be seen for the Nursery (about 50%) and Car Evaluation datasets (about 40%). In most of the cases, the correct percentage has not been affected considerably. However, in majority of cases,

TABLE II
EXPERIMENTAL RESULTS

Problem	Num. of Classes	Num. of Bin.,Mult.,Cont. Attributes	TDTIDT	Prism	J-Rule
Annealing	6	19,13,6	63.7 (4.97) 95.36	104.0 (3.05) 91.35	43.5 (3.12) 93.98
			26.4 (5.82) 99.37	62.4 (2.46) 97.37	32.3 (2.93) 96.61
Car Evaluation	4	0,6,0	267.4(5.46) 88.88	217.1(5.30) 89.23	149.1(4.78) 90.33
			83.3(9.68) 98.66	132.3 (5.38) 93.80	85.2 (4.75) 92.7
Chess	2	3,4,0	20.0 (5.25) 99.37	15.1 (3.12) 99.84	7.9 (4.27) 99.37
			9.9 (4.76) 99.84	9.8 (3.14) 99.84	5.9 (4.06) 99.37
Contraceptive Method Choice	3	3,4,2	664.7 (6.11) 46.57	678.7 (4.21) 42.97	404.4 (4.23) 44.46
			605.7 (8.05) 47.85	666.9 (4.37) 42.62	398.8 (4.44) 44.87
Contact Lenses	3	2,3,0	16.0 (3.8) 92.54	14.4 (3.09) 86.0	8.6 (3.36) 88.72
			11.9 (4.64) 94.36	12.5 (3.18) 87.81	7.4 (3.25) 92.54
Credit Screening	2	4,5,6	129.0(4.68) 78.84	151.3 (2.65) 76.52	75.3 (2.51) 80.0
			79.0 (6.55) 82.46	151.1 (2.63) 76.23	74.1 (2.54) 80.28
Dermatology	6	1,32,1	43.5(4.10) 91.53	55.5 (2.14) 84.93	31.4 (2.02) 87.15
			18.9(5.74) 93.99	52.3 (2.17) 86.85	30.5 (2.02) 86.88
Solar Flare 2- class 1	9	5,5,0	180.5 (5.00) 78.79	152.2 (5.70) 75.61	37.3 (4.17) 79.26
			170.6 (9.13) 79.54	149.8 (11.86) 75.98	37.2 (5.93) 79.17
Solar Flare 2- class 2	7	5,5,0	95.5 (4.52) 94.27	77.5 (4.07) 94.65	9.8 (3.39) 95.31
			75.8 (7.67) 95.58	72.5 (7.83) 94.65	9.8 (3.39) 95.31
Solar Flare 2- class 3	3	5,5,0	18.1 (3.37) 99.43	22.6 (2.26) 99.24	3.7 (3.94) 99.24
			13.7 (4.76) 99.34	21.1 (3.73) 99.24	3.7 (7.83) 99.24
Labor relations	2	3,5,8	4.8 (2.06) 85.0	7.7 (1.19) 80.0	5.0 (1.18) 85.0
			4.6 (2.10) 82.5	7.5 (1.18) 77.5	4.8 (1.16) 85.0
Monk 1st	2	2,4,0	38.2 (3.86) 83.91	25.0 (2.96) 73.33	14.0 (2.67) 87.56
			17.7 (6.06) 91.85	18.7 (2.90) 82.24	10.7 (2.68) 90.96
Monk 2nd	2	2,4,0	87.7 (4.69) 43.82	67.9 (3.83) 54.41	37.3 (3.99) 50.18
			56.8 (6.90) 73.97	60.0 (3.99) 52.57	32.9 (4.23) 50.14
Monk 3rd	2	2,4,0	26.5 (3.22) 86.92	24.8 (2.71) 84.55	15.2 (2.42) 85.32
			16.6 (5.43) 88.52	19.2 (2.80) 86.15	11.9 (2.48) 90.19
Nursery	5	1,7,0	811.0 (6.61)98.22	559.8 (5.42) 97.20	309.2 (5.33) 98.69
			227.6(10.61)99.75	296.4 (5.41) 98.51	154.6 (5.23) 99.33
Soybean-large	19	16,19,0	83.9 (4.52) 81.15	73.0 (2.72) 77.86	45.6 (1.75) 85.68
			45.6 (7.38) 86.64	72.4 (2.73) 76.25	45.5 (1.77) 84.39
Tic-Tac-Toe Endgame	2	0,9,0	190.9 (5.65) 84.45	46.1 (3.46) 96.66	23.8 (3.44) 96.65
			71.2 (7.52) 95.09	42.8 (3.56) 96.24	22.9 (3.62) 96.65
Congressional Voting	2	0,16,0	29.2 (4.34) 91.66	31.7 (2.49) 90.33	21.6 (2.26) 92.33
			21.4 (5.33) 91.66	30.0 (2.52) 90.66	20.1 (2.39) 93.0

it has slightly been improved by Binary Coding.

VI. CONCLUSION

In this paper, we firstly showed how the use of the inequality operator '$\neq$', in addition to the equality operator, could result in higher expressiveness of rules induced by rule-induction algorithms. We then introduced three possibilities to use the inequality operator. One of these possibilities is the Binary Coding method, which simply adds to a rule-induction system two extra layers of pre processing and post processing without actual changing the underlying rule-induction algorithm itself. This brings the remarkable advantage that we do not need to change the underlying rule-induction algorithm in order to make use of the inequality operator. In other words, the method of Binary Coding operates independently from the underlying rule-induction algorithm.

One of the main possibilities for future work could be to investigate the characteristics of datasets on which the Binary Coding method performs well; in other word to find the class of datasets where Binary Coding could be advantageous. Another possible work could be to extend the use of the inequality operator to continuous attributes in order to increase the likelihood of achieving fewer and more expressive rules.

REFERENCES

[1] C. L. Blake and C. J. Merz, "UCI repository of machine learning databases (http://www.ics.uci.edu/~mlearn/MLRepository.html)," Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[2] P. Smyth and R. M. Goodman, "An information theoretic approach to rule induction from databases," *IEEE Trans. Knowledge Data Engineering*, vol. 4, pp. 301-316, 1992.

[3] P. Clark and R. Boswell, "Rule induction with CN2: some recent improvements," in *Proc. 5th European Conf.(* EWSL-91*)* Ed.: Yves kodratoff Springer-Verlag, Berlin, pp. 151-163, 1991.

[4] J. Cendrowska, "PRISM: an algorithm for inducing modular rules," *Int. J. Man-Machine Studies*, vol. 27, pp. 349-370, 1987.

[5] M. A. Bramer, "Using J-pruning to reduce overfitting in classification trees" *Research and Development in Intelligent Systems*, vol. *XVIII*, Springer-Verlag, pp. 25–38, 2000.

[6] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81-106, 1986.

[7] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[8] M. A. Bramer, "The Inducer user guide and reference manual," University of Portsmouth, Faculty of Technology, UK, Tech. Rep., 1999.

[9] M. A. Bramer, "Inducer: a rule induction workbench for data mining," In Z. Shi, B. Faltings and M. Musen, editors, *Proc. 16th IFIP World Computer Congress Conf. Intelligent Information Processing*, Publishing House of Electronics Industry (Beijing), pp. 499-506, 2000.