Weighted MCRDR: Deriving Information about Relationships between Classifications in MCRDR.

Richard Dazeley and Byeong-Ho Kang

School of Computing, University of Tasmania, Hobart, Tasmania 7001, Australia[1].
Smart Internet Technology Cooperative Research Centre, Bay 8, Suite 9/G12 Australian Technology Park Eveleigh NSW 1430[1].
{rdazeley, bhkang}@utas.edu.au

Abstract. Multiple Classification Ripple Down Rules (MCRDR) is a knowledge acquisition technique that produces representations, or knowledge maps, of a human expert's knowledge of a particular domain. However, work on gaining an understanding of the knowledge acquired at a deeper meta-level or using the knowledge to derive new information is still in its infancy. This paper will introduce a technique called Weighted MCRDR (WM), which looks at deriving and learning information about the relationships between multiple classifications within MCRDR by calculating a meaningful rating for the task at hand. This is not intended to reduce the knowledge acquisition effort for the expert. Rather, it is attempting to use the knowledge received in the MCRDR knowledge map to derive additional information that can allow improvements in functionality of MCRDR in many problem domains. Preliminary testing shows that there exists a strong potential for WM to quickly and effectively learn meaningful weightings.

1. Introduction

Multiple Classification Ripple Down Rules (MCRDR) [1, 2] is an incremental Knowledge Acquisition (KA) methodology which allows the expert to perform both the KA process and the maintenance of a Knowledge Based System (KBS) over time [3]. MCRDR allows for multiple independent classifications and is an extension of its predecessor Ripple Down Rules (RDR), which only allowed for single classifications of each case presented. The underlying concept behind these approaches is to use the expert's knowledge within the context it is provided [4]. Thus, if the expert disagrees with a particular conclusion made by the system, knowledge can be added to improve future results.

While MCRDR works very effectively in a number of domains, there is implicit information contained within the structure itself that is not being extracted or used in its existing form. For example, when multiple classifications for a case occur, it

[1] Collaborative research project between both institutions

would be useful to know the relationship between those classes and how that relationship changes the meaning or importance of the case as a whole when compared to cases with differing class configurations. Potentially, such information could reveal important details that may not have been consciously realised by the user or that could have taken significant rule creation for the user to have been able to capture within the standard MCRDR structure.

This paper is going to present one means for addressing this issue through an extension to MCRDR, referred to as Weighted MCRDR (WM). WM provides the ability to extract and learn information about the interrelationships between the various classifications found in MCRDR, as well as derive a meaningful value for a given case through either direct or indirect means. This extension could provide the standard MCRDR algorithm with the ability to provide more functionality and usability to existing domains, as well as opening up additional application possibilities. Preliminary testing shows that there exists a strong potential for WM to quickly and effectively learn meaningful weightings.

2. Multiple Classification Ripple Down Rules (MCRDR)

MCRDR uses a collection of rules in an n-ary tree and evaluates each case by first evaluating the root and then moving down level by level. This continues until either a leaf node is reached or until none of the child rules evaluate to *true*. Due to the fact that any, or all, of a node's children have the potential to fire, the possibility exists for a number of conclusions or classifications to be reached by the system for each case presented [5]. The system then lists the collection of classifications found and the paths they followed. The path is given as the justification for the conclusion.

KA is achieved in the system through the expert inserting new rules when a misclassification has occurred. The new rule must allow for the incorrectly classified case to be distinguished from the existing stored cases that could reach the new rule [6]. This is accomplished by the user identifying key differences between the current case and one of the earlier cornerstone cases already stored. This is continued for all past cases that could reach the new rule in the tree structure, until there is a composite rule created that uniquely identifies the current case from all of the previous cases. The idea here is that the user will select differences that are representative of the new class they are trying to create. Stopper rules, which cancel a particular evaluation path, are added in the same way [6].

Previously, simulation studies using RDR illustrated, despite the random nature of rule creation and insertion, that the knowledge bases created are as compact and accurate as those produced by induction [7, 8]. Furthermore, Kang [1] showed that MCRDR produces somewhat better knowledge bases, even when used in single classification domains. It is also arguable that a multiple classification system, such as MCRDR, provides an approach to dealing with a number of problem types, such as configuration and scheduling [5, 9]. RDR and MCRDR, as well as variations to these methodologies, have been used in a number of application areas, such as the PEIRS pathology system [10], resource allocation [11] and document management [12].

3. Weighted MCRDR (WM)

However, MCRDR has a lack of cohesion between the individual classifications generated by a single case, due to each classification being uniquely derived with no consideration for what other classification paths may have also been followed. Therefore, in certain problem domains, a particular case's multiple classifications may all be individually correct but still not capture its essence or accurately represent the level of importance the case has to the expert.

For example, in an email classification system that sorts emails into various categories of varying importance to the user, such as that developed by Deards [13], there exists a class for work related email and one for advertising spam. The spam category catches advertising emails that are of little use to the user, while the work category holds material that requires the expert's immediate attention, before all the other emails received. However, if the system receives an email selling something directly relevant to the user's work then it should classify it as being both spam and work related. The problem with this classification is that neither category individually describes the case adequately, yet both are correct: while it is spam the user may wish to read it; and, then again, it is not as high a priority as the usual work material and should not be brought to the users' attention immediately.

To handle this in traditional MCRDR, the expert would need to create a new set of rules so that such mails could be categorised separately. This, however, can become a tedious and never ending task for emails that only appear infrequently. It also requires the user to be aware of the required classification, which can be difficult as the user generally has little or no knowledge of the MCRDR structure. Finally, it makes little sense for the user to create a new category when the existing classifications are already correct from the user's point of view. The intention of WM is to try to capture these relationships between various classifications that may exist, either consciously or otherwise. If we can identify a set of relative values for the various relationships, this information could be used to improve the functionality available to the user. For instance, in the above email example we could list the emails from each classification in their order of importance, using the relationship's value as a gauge.

3.1 WM Implementation

The most straight forward approach to calculating a rating based on the relationships in an MCRDR classification set is to include a weight at each rule node and simply find the weighted-sum of all the terminal rules found, thereby, giving a value for the case. However, the method used for acquiring each rule's value must take into consideration all the other concluding rules. In addition, the method would need to be able to change the value to cater for other possible relationships that are not evident when a new rule or class is initially created. Finally, it would be useful if such a system was able to provide a number of varying results in applications where dissimilar tasks may need to be rated differently.

Using the notion of finding a weighted-sum leads directly to the use of a single layer perceptron neural network, which is fundamentally a weighted-sum that has been thresholded, using the sigmoid function (equation 1) to lie between a lower and upper bound. Thus, each rule in the MCRDR tree would be given an associated input node in the neural network. Using such a network also has the added advantage of providing a means for each terminating rule to learn the optimal weight across all of the various relationships, by using the *generalized-Δ-rule* [14]. In addition, any number of outputs can be used (where each output is effectively a new perceptron network) for each different task that such values may be needed.

$$f(net) = 1 \Big/ \left(1 + e^{-k\ net}\right) \tag{1}$$

Thus, the full WM algorithm, given in pseudo code in Figure 1 and shown diagrammatically in Figure 2, consists of two primary components. Firstly, a document or case is pre-processed to identify all of the usable data elements, such as stemmed words or a patient's pulse. The data components are then presented to the standard MCRDR engine, which classifies them according to the rules previously provided by the user. Secondly, for each rule identified an associated input neuron in the neural network will fire. The network finally produces a number of outputs, $\bar{v}$, for the case presented. The system, therefore, essentially provides two separate outputs; the case's classifications and the associated set of values generated from those classifications.

1. Pre-process Case
Initialise Case *c*
c ← Identify all useful data elements.

2. Classification
Initialize *list l* to store classifications
Loop
If child's rule evaluates Case c to *true*
l ← goto step 2 (generate all classifications in child's branch).
Until no more children
If no children evaluated to true then
l ← Add this nodes classification.
Return *l*.

3. Rate Case
$\bar{i}$ ← Generate input vector from *l*.
NN ← $\bar{i}$
v ← *NN output value.*

4. Return RM evaluation
Return list *l* of classifications for case *c* and
Value *v* of case *c*.

Fig. 1. Algorithm for WM.

For example, in figure 2, the document {a b b a c f i} has been pre-processed to a set of unique tokens {a, b, c, f, i}. It is then presented to the MCRDR component of the WM system, which ripples the case down the rule tree finding three classifications *Z*, *Y*, and *U* from the terminating rules 1, 5, and 8. In this example, which is using the RA method (discussed below), the terminating rules then cause the three associated neurons to fire and feed forward through the neural network producing a single value of 0.126 to be outputted. Thus, this document has been allocated a set of classifications that can be used to store the document appropriately, plus a single rating measuring its overall level of importance to the user.

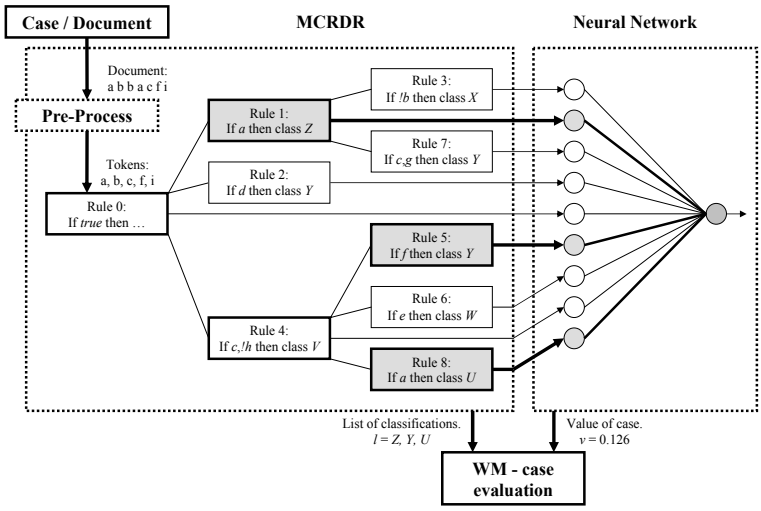

Fig. 2. WM illustrated diagrammatically.

3.2 Neuron Association Methods

There are three possible methods for the association of neurons to the MCRDR structure: the Class Association method (CA), the Rule Association method (RA) and the Rule Path Association method (RPA). The different methods arise from the possibility of many paths through the tree that result in the same class as the conclusion. The *class association* method, where each unique *class* has an associated neuron, can reduce the number of neurons in the network and potentially produce faster, but possibly less general, learning. The *rule association* method, where each *rule* has an associated neuron and only the terminating rule's neuron fires, allows for different results to be found for the same class depending on which path was used to generate that class as the conclusion. Therefore, it is more capable of finding variations in meaning and importance within a class than expected by the user that created the rules. The *rule path association* method, where all the *rules* in the *path* followed, including the terminating rule, cause their associated neuron to fire, would be expected to behave similarly but may find some more subtle results learnt through the paths rules, as well as being able to learn meaning hidden within the paths themselves.

3.3 Adding New Input Neurons

However, a perceptron network does not provide a standard way for calculating the initial weight for a rule node when it is first added to the MCRDR tree. This can be resolved, though by first adding new input nodes to the network each time a rule is added and secondly, calculating the initial weight for the new network node for each of the allocated outputs. The simplest approach is to give the new input connections created a random start up weight in the same fashion used when initialising the network. However, we already have the system's *correct rating* for the case that is causing the new rule to be created, which WM should use to foster faster learning. For this purpose the system's *correct rating* is assumed to have been determined for the case either through directly querying the user or through indirectly deriving it from observed user behaviour.

The approach taken in this implementation of WM captures this important information by directly calculating the required weight that provides us with the correctly weighted-sum for the given case without the need for a training period. This provides the system with an immediate understanding of the new case's general value. It can also then be used as the foundation for the new node's relationships with other terminating rules to be learnt.

3.4 The Single-Shot Δ-Rule in WM

In order to calculate the weight needed for the new input connection, w_{no}, the system must first calculate the error in the weighted-sum, δ_{ws}, and divide this by the input at the newly created input node, x_n, which is always 1 in this implementation, where there are $n>0$ input nodes and $o>0$ output nodes.

$$w_{no} = \frac{\delta_{ws}}{x_n} \tag{2}$$

δ_{ws} is calculated by first deriving the required weighted-sum, R_{ws}, from the known error, δ, and subtracting the actual weighted-sum, denoted by *net*.

$$\delta_{ws} = R_{ws} - net \tag{3}$$

The value for *net* for each output node, o, was previously calculated by the network during the feed forward operation, and is shown in Equation 4, where there are $n>1$ input nodes, where the n^{th} input node is our new input.

$$net_o = \sum_{i=0}^{n-1} x_i w_{io} \tag{4}$$

R_{ws}, can be found for each output node, by reversing the thresholding process that took place at the output node when initially feeding forward. This is calculated by finding the inverse of the sigmoid function, Equation 1, and is shown in Equation 5, where *f(net)* is the original thresholded value that was outputted from that neuron.

$$R_{ws_o} = \frac{\log\left[\frac{f(net)_o + \delta_o}{1 - \left(f(net)_o + \delta_o\right)}\right]}{k} \quad (5)$$

Thus, the full *single-shot Δ-rule*, used for each of the new input's connections to all of the output nodes is given in equation 6. Due to the use of the sigmoid function, it is clear that at no time can the system try and set the value of the output to be outside the range $0 > (f(net) + \delta) > 1$ as this will cause an error.

$$w_{no} = \frac{\left(\log\left[\frac{f(net)_o + \delta_o}{1 - \left(f(net)_o + \delta_o\right)}\right] \Big/ k\right) - \left(\sum_{i=0}^{n-1} x_i w_{io}\right)}{x_n} \quad (6)$$

4. Testing with a Simulated Expert

The problem with testing a system, such as WM, is the use of expert knowledge that cannot be easily gathered without the system first being applied in a real world application. This is a similar problem that has been encountered with the testing of any of the RDR methodologies [2, 8, 15]. Thus, these systems built their KB incrementally through the use of a simulated expert. The simulated expert essentially provided a rule trace for each case run through another KBS with a higher level of expertise in the domain than the RDR system being trained [2, 5]. WM, however, has the additional problem of needing to learn from results returned by the system, derived either directly or indirectly from the expert.

Thus, in order to perform preliminary testing of the rating component of the system, while still being able to create a KB in the MCRDR tree, a simulated expert was also developed for WM, with the ability to both classify cases, as well as form an opinion as to a case's overall importance. Basically, the simulated expert randomly generates weights, representing the level that each possible attribute in the environment contributes to each possible class, which is used to define rules for the MCRDR tree. Likewise, an additional weight is generated for each possible attribute, indicating what level of importance the case has overall to the simulated expert, allowing the simulated expert to form preferences for particular cases.

The environment then created many sets of documents consisting of randomly generated collections of attributes and passed each set to the WM system for classification and rating. The order allocated by the WM system for each set was then compared against the simulated expert's expected ordering. In testing the system, assumptions were made that the expert's interest in a case could be accurately measured and that the behaviour was constant, without noise or concept drift.

5. Results and Discussion

One useful application for a system such as WM is the rating of documents so they can be appropriately ordered according to the user's preference. The ability of WM to accomplish this is illustrated in the example cross sections shown in Figure 3. The first graph shows the order in which WM places the documents prior to any learning. The second graph, after only 10 document sets, clearly illustrates that it has been able to correctly order many of the documents.

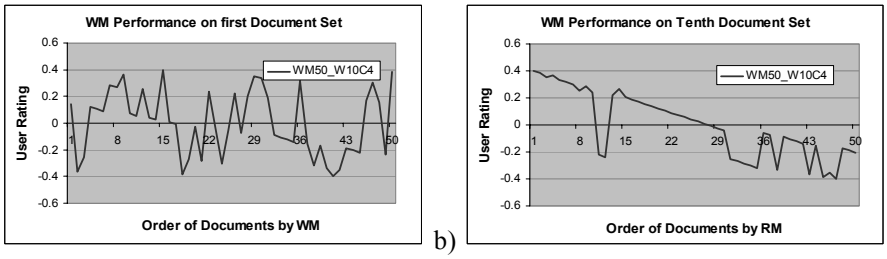

Fig. 3. Ability of WM to order cases according to user preference. a) Shows WM's performance prior to any training. b) Shows WM's performance after 10 document sets. Both graphs have the highest rated cases on the left and the lowest on the right according to RM.

To show how the system performs over time the difference was calculated between each documents' place in the WM ordering minus the place it should have been placed according to the simulated expert. These differences were then averaged over all the documents presented in that group and subtracted from the total number of documents in the set. Figure 4, shows how the system performed over the first 50 document sets. It can be seen in this example that the assignment of immediate weights to a position and the ongoing trainability of the network allows the system to immediately start to provide a reasonably accurate solution.

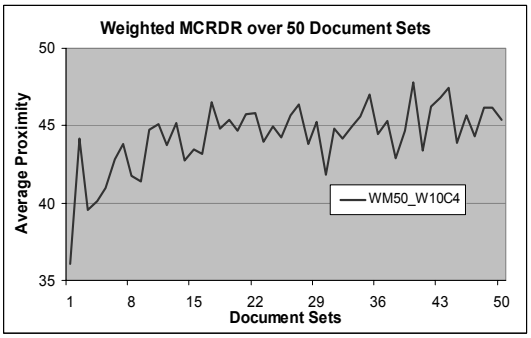

Fig. 4. Proximity of WM to simulated users ordering. The average proximity is the absolute difference between the place allocated by WM and the place given by the simulated expert, averaged over all the cases in each set and subtracted from the total number of documents in the set. Each document set contains 50 cases.

While these results are only preliminary, they do show that WM is capable of learning ratings that allow us to order documents into the users preferred order quickly. Additionally, due to the network learning appropriate ratings it shows that it has been able to identify patterns in the structure of the MCRDR tree and the way a case has been classified within that structure. Finally, it can be seen that a system that uses MCRDR to organise data items for long term storage and later retrieval, could easily include extra functionality by extracting this meta-knowledge from MCRDR.

6. Conclusion and Future Work

The system described in this paper was developed to provide extra functionality to the MCRDR system by finding values for representing the relationships between classifications generated by the knowledge base. It is designed to learn simple linear relationships quickly, so that the system can respond to new information without the need for a number of training examples. To facilitate this system the *single-step-Δ-rule* was developed for the input connection's initial weight, which provides the ability of the system to step immediately to the indicated value required by the system.

The system has undergone preliminary testing with a simulated expert using a randomly generated data set. These tests were done primarily to show that the system was able to learn quickly and to be used for parameter tuning purposes. Clearly a more rigorous testing regime needs to be used in order to fully justify the algorithm's ability to learn.

This paper was an introduction to one area where MCRDR can be used as a basis for deriving further information about a user's knowledge and understanding of a domain. It still holds the possibility of being refined further such as:

- Providing a means to learn non-linear relationships.
- Through the addition of a per-neuron reducing gain to allow the system to better cope with noise when creating new input nodes.
- Building in a windowing technique to allow the system to deal effectively with concept drift.
- Incorporating Temporal Difference Learning through the addition of an eligibility trace [16]. This would allow the system to predict the likelihood of future documents, linked to by the one being analyzed, being of importance.
- There exists the possibility of using the error information that is propagated back through the network to identify which rules in MCRDR are likely to be incorrect and only asking the user about those particular cases, thereby, reducing the workload of knowledge acquisition for the user.

Acknowledgements

The authors would like to thank Dr Ray Williams and Mr Robert Ollington for their helpful discussions.

References

1. Kang, B. H., Validating Knowledge Acquisition: Multiple Classification Ripple Down Rules, (1996)
2. Kang, B. H., Compton, P. and Preston, P. Multiple Classification Ripple Down Rules: Evaluation and Possibilities. in *The 9th Knowledge Acquisition for Knowledge Based Systems Workshop*. Department of Computer Science, University of Calgary, Banff, Canada: SRDG Publications (1995)
3. Martínez-Béjar, R., Ibáñez-Cruz, F., Le-Gia, T, Cao, T., and Compton, P., FMR: an incremental knowledge acquisition system for fuzzy domains., *Lecture Notes in Artificial Intelligence.* **1621** (1999) p. 349-364
4. Compton, P. and Jansen, R., A philosophical basis for knowledge acquisition., *Knowledge Acquisition*. **2** (1990) p. 241-257
5. Kang, B. H., Preston, P. and Compton, P. Simulated Expert Evaluation of Multiple Classification Ripple Down Rules. in *Eleventh Workshop on Knowledge Acquisition, Modeling and Management*. Voyager Inn, Banff, Alberta, Canada (1998)
6. Preston, P., Compton, P., Edwards, G. and Kang, B. H. An Implementation of Multiple Classification Ripple Down Rules. in *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*. Department of Computer Science, University of Calgary, Calgary, Canada UNSW, Banff, Canada: SRDG Publications (1996)
7. Compton, P., Preston, P. and Kang, B. H. The Use of Simulated Experts in Evaluating Knowledge Acquisition. in *The 9th Knowledge Acquisition for Knowledge Based Systems Workshop*. Department of Computer Science, University of Calgary, Calgary, Canada: SRDG Publications (1995)
8. Compton, P., Preston, P., Kang, B. H. and Yip, T., Local Patching Produces Compact Knowledge Bases, in A Future for Knowledge Acquisition, L. Steels, S. G. and W. V. d. Velde, Editors, Springer-Verlag: Berlin, Germany (1994) p. 104-117
9. Compton, P., Kang, B. H., Preston, P. and Mulholland, M. Knowledge Acquisition Without Analysis. in *Knowledge Acquisition for Knowledge Based Systems*. Berlin: Springer Verlag (1993)
10. Edwards, G., Compton, P., Malor, R., Srinivasan, A. and Lazarus, L., PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports., *Pathology*. **25** (1993) p. 27-34
11. Richards, D. and Compton, P. Knowledge Acquisition First, Modelling Later. in *Proceedings of the Tenth European Knowledge Acquisition Workshop* (1997)
12. Kang, B. H., Yoshida, K., Motoda, H. and Compton, P., A help desk system with intelligent interface, *Applied Artificial Intelligence*. **11**(7-8) (1997) p. 611-631
13. Deards, E. A., MCRDR Applied to Email Classification, in *Department of Computer Science* (2001)
14. Beale, R. and Jackson, T., *Neural Computing: An Introduction*, Bristol, Great Britain: IOP Publishing Ltd (1992)
15. Mansuri, Y., Kim, J. G., Compton, P. and Sammut, C. A comparison of a manual knowledge acquisition method and an inductive learning method. in *Australian workshop on knowledge acquisition for knowledge based systems*. Pokolbin, Australia (1991)
16. Sutton, R. and Barto, A., *Reinforcement Learning: An Introduction*, Cambridge, Massachusetts: A Bradford Book, The MIT Press (1998)