Rated MCRDR:
Finding non-Linear 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 simple and effective knowledge acquisition technique that produces representations, or knowledge maps, of a human expert's knowledge of a particular domain. This knowledge map can then be used to automate and help the user perform classification and categorisation of cases while still being able to add more refined knowledge incrementally. While MCRDR has been applied in many domains, work on understanding the meta-knowledge acquired or using the knowledge to derive new information is still in its infancy. This paper will introduce a technique called Rated MCRDR (RM), which looks at deriving and learning information about both linear and non-linear relationships between the multiple classifications within MCRDR. This method uses the knowledge received in the MCRDR knowledge map to derive additional information that allows improvements in functionality within existing domains, to which MCRDR is currently applied, as well as opening up the possibility of new problem domains. Preliminary testing shows that there exists a strong potential for RM to quickly and effectively learn meaningful ratings.

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]. The basic concept behind MCRDR is to use the expert's knowledge within the context it is provided [4] to produce multiple classifications for an individual case. Therefore, if the expert disagrees with one or more of the conclusions found by the system, knowledge can be easily added to improve future results.

MCRDR has been found to work very effectively in a number of domains [5], however, there is implicit information contained within the knowledge map that is not being extracted or used in the existing methodology. For example, when multiple classifications for a case occur, it would be useful to have an understanding of the relationship between those classes found and how this association alters the meaning or importance of the case when compared to cases with different class configurations. This information, if extracted, could potentially reveal important details that may not have been consciously realised by

[1] Collaborative research project between both institutions

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 Rated MCRDR (RM). This method has the ability to extract and learn both linear and non-linear information about the relationships between the various classifications found in MCRDR and derive a meaningful value for a given case through either direct or indirect means. RM provides the standard MCRDR algorithm with more functionality and useability to existing domains, as well as opening up additional application possibilities especially where the multiple classifications tend to be related in a non-linear manner.

2 Multiple Classification Ripple Down Rules (MCRDR)

MCRDR uses an n-ary tree where each node contains a rule. Inference is performed by passing each case to the root node, which in turn feeds it on to any children with rules that evaluate it to *true*. Thus, the case continues to ripple down, level by level, until either a leaf node is reached or all of the child rules evaluate to false. 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 [6]. The system then lists the collection of classifications and the paths they followed.

KA is achieved in the system by inserting new rules into the MCRDR tree when a misclassification has occurred. The new rule must allow for the incorrectly classified case, identified by the expert, to be distinguished from the existing stored cases that could reach the new rule [7]. This is accomplished by the user identifying key differences between the current case and one of the earlier cornerstone cases. Where, a cornerstone case is any case that was used to create a rule and was also classified in the parent's node, or one of its child branches, of the new node being created. This is continued for all stored cornerstone cases, until there is a composite rule created that uniquely identifies the current case from all of the previous cases that could reach the new rule. The idea here is that the user will select differences that are representative of the new class they are trying to create [7].

3 Rated MCRDR (RM)

Individual classifications in MCRDR, however, are all uniquely derived with no consideration for what other classification paths may have also been followed. Thus, there is no cohesion between any of the classifications found, however, the fact that this case was classified in these classes; means there must be either a conscious or subconscious relationship between these cases in the experts mind. 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 [8], there may exist a class for work related mail 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 MCRDR will 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 may not have 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 categorized 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 RM 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 Implementation

Firstly, looking at what RM must accomplish mathematically, it can be seen that the output from the MCRDR methodology is essentially a set of classifications, denoted *C*, where $C \in \wp(C^*)$, and C^* is the set of all possible classifications. The output from the RM engine is a set of values, $\bar{v}$, to provide one or more varying results in applications where dissimilar tasks may need to be rated differently. For instance, v_0, may identify the desirability or importance of the case presented. Therefore, a mapping must be found from the set $C \rightarrow \bar{v}, \forall C \in \wp(C^*)$. Additionally, RM should be able to learn this mapping for both linear and non-linear sets of classifications quickly and be able to generalise effectively.

Thus, RM needs to identify patterns of classifications and then associate a value for each pattern. While there are a number of techniques used for pattern recognition, in this implementation of RM an Artificial Neural Network (ANN) was selected, primarily because of its adaptability, ease of application to the problem domain, and because pattern recognition is one of the dominating areas for the application of ANN's [9].

The neural network was integrated into MCRDR by linking each possible rule or class to an input neuron. Then, for each terminating rule or classification found by the MCRDR system, an associated neuron will fire. In this implementation of RM a simple multilayer perceptron network with a sigmoid thresholding function, see equation 1, using a *modified* generalised delta rule was used.

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

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,

```
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.
```

Figure 1: Algorithm for RM.

it is more capable of finding variations in meaning and importance within a class than may have been 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.

Thus, the full RM algorithm, given in pseudo code in Figure 1 and shown diagrammatically in Figure 2, consists of two primary components. Firstly, a 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 presented to the standard MCRDR engine, which classifies them according to the rules previously provided by the user. Secondly, for each rule or class identified an associated input neuron in the neural network will fire. The network finally produces a set 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 for those classifications.

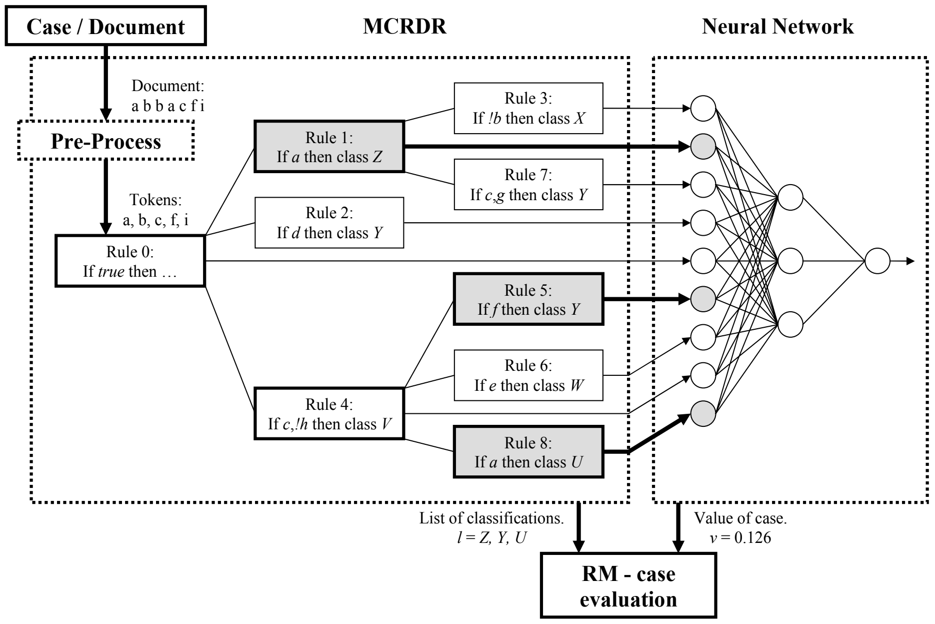

Case / Document
MCRDR
Neural Network
Document: a b b a c f i
Pre-Process
Tokens: a, b, c, f, i
Rule 0: If true then …
Rule 1: If a then class Z
Rule 2: If d then class Y
Rule 3: If !b then class X
Rule 7: If c,g then class Y
Rule 5: If f then class Y
Rule 6: If e then class W
Rule 4: If c,!h then class V
Rule 8: If a then class U
List of classifications. l = Z, Y, U
Value of case. v = 0.126
RM - case evaluation

Figure 2: RM illustrated diagrammatically.

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 RM 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, 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. Thus, this document has been allocated a set of classifications that can be used to store the document appropriately, plus a rating measuring its overall level of importance to the user.

3.2 Learning in RM

Learning in RM is achieved in two ways. Firstly, the rating component receives feedback from the environment concerning the accuracy of its predicted rating. Thus, a system using RM must provide some means of either directly gathering or indirectly estimating the correct rating. For example, in the email application, previously discussed, the amount of reward given to the network could be based on the order the articles are read by the user or whether the user prints, saves, replies, forwards or deletes the email. Secondly, the MCRDR component still acquires knowledge in the usual way; by the user identifying incorrect classifications and creating new rules and occasionally new classifications.

3.3 Neural Network Structure and Learning

This creation of additional rules and classifications, however, means the neural network will also require the capability to increase its number of input nodes to ensure one input node for each possible rule or class. Furthermore, as more inputs are added, the hidden layer will have an increased number of possible patterns that it would be required to learn; therefore, occasionally new hidden nodes may also need to be added. The amount of hidden nodes to be added is primarily determined by the application the algorithm is being applied. This implementation of RM used a system where hidden nodes where added, so that a predefined percentage of input to hidden nodes was maintained.

The simplest approach to adding additional neurons is to add the necessary neurons and give the new connections created a random start up weight in the same fashion used when initialising the network. However, these random weights would affect the results from already learnt classification relationships, slowing learning. In addition, we already have the system’s correct rating for the case that is causing the new rule creation, which RM should be able to use, in order to foster faster learning. For instance in the email example, if we know, from having observed the user, that a document was important, then we should give it a higher start up value, and vice versa if it was not significant.

The approach taken in this implementation of RM captures this important information by directly calculating the required weight that provides us with the correctly weighted sum without the need for a training period. When applying this weight, however, it must be done in such a way that does not affect any of the already learnt cases. Therefore, no changes can be made to weights connecting the hidden layer to the output layer. However, due to the thresholding that is performed at the hidden layer it is entirely possible that weights on the arcs from the input layer to hidden layer can not be increased enough to fully reduce the error and give the required result.

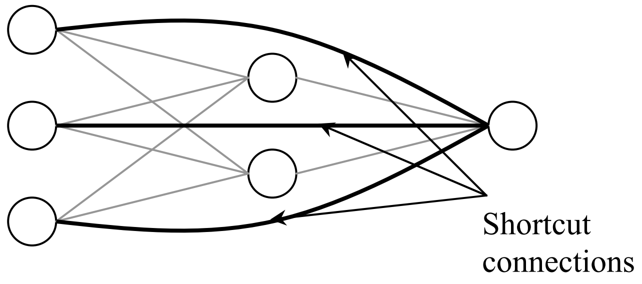


Figure 3: Network structure.

These restrictions meant that the network structure needed to be altered by adding shortcut connections from any newly created input node directly to each output node and using these connections to carry the entire weight adjustment required to gain the desired set of results. Figure 3 shows the network topology used. This system allows the network to adjust immediately when a new rule is created and yet still provide the ability to learn adjustments to the new node's relationships with other nodes through the underlying network.

One interesting result of using a topology such as this, is that we effectively now have two networks: the shortcut network is essentially a two layer network capable of learning linear functions, thus, linear relationships, very quickly; while the three layer network underneath allows the system to still be able to learn nonlinear relationships. Thus, the combination of these networks allows the system to learn a linear function quickly while still, over time, being able to derive the more complex non-linear functions.

When adding new input and, zero or more, hidden nodes, new connections must also be added in the following ways:

- from the new input node to all the old hidden nodes.
- from all input nodes, new and old, to each, if any, new hidden nodes.
- from each, if any, new hidden nodes to all the output nodes.
- the shortcut connections from the new input node to all output nodes.

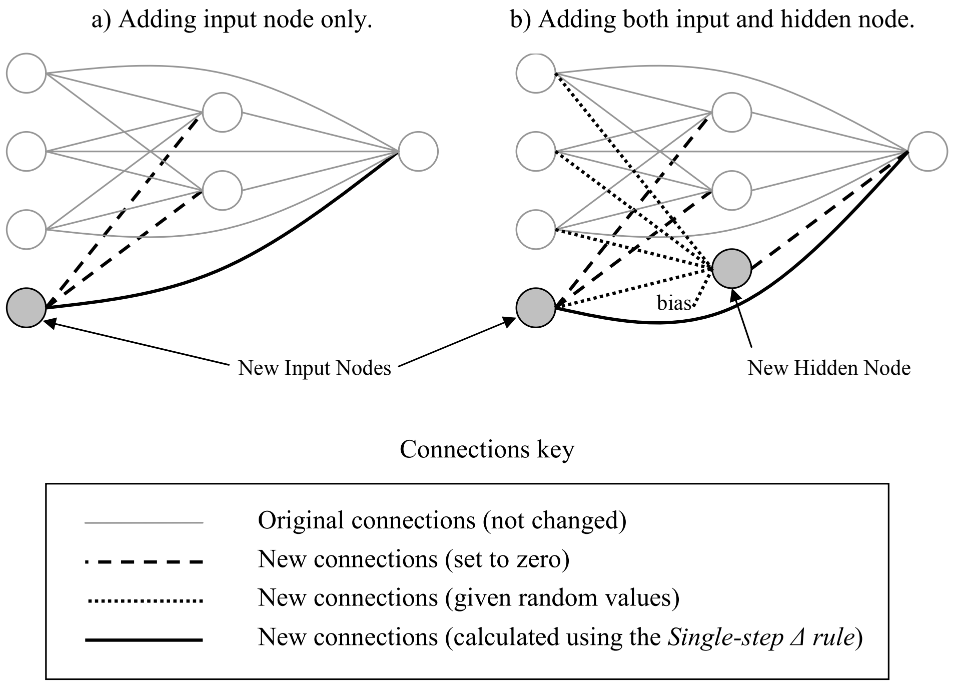

a) Adding input node only.
b) Adding both input and hidden node.
bias
New Input Nodes
New Hidden Node
Connections key
Original connections (not changed)
New connections (set to zero)
New connections (given random values)
New connections (calculated using the *Single-step Δ rule*)

Figure 4: Process used for adding new input and hidden nodes.

Each of these different groups of new connections requires particular start up values. Firstly, the new connections from the new input node to all the old hidden nodes should be set to zero so that they have no immediate effect on existing relationships. If the new input node should be included in particular patterns that already exist then this will be learnt over time. If new hidden nodes have also been added then the connections from them to the output nodes should also be set to zero for the same reason. However, so these connections can be trained, the output from the new hidden nodes must be non-zero. Therefore, the new connections from all input nodes to the new hidden nodes, plus the new hidden nodes' biases, are given random values rather than zero. Finally, the new shortcut connection is given a value that reduces the error, for the given case, to zero. The process for adding nodes and connections is illustrated in Figure 4.

In order to calculate the weight needed for a new shortcut 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 one 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, and the n^{th} input node is our new input node, and $m>0$ hidden nodes. Basically, this is the usual feed forward rule with the addition of the weighted sum of the shortcut connections.

$$net_o = \left(\sum_{i=0}^{n-1} x_i w_{io}\right) + \left(\sum_{h=0}^{m} x_h w_{ho}\right) \tag{4}$$

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.

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

Thus, the full *single-shot Δ-update-rule*, used for each new shortcut connection between the new input node and each output node 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.

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

4 Testing with a Simulated Expert

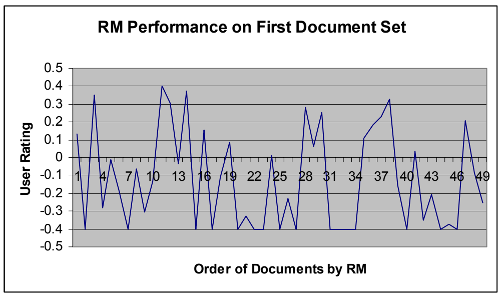
The problem with testing a system, such as RM, is the use of expert knowledge that cannot be easily gathered without the system first being applied in a real world application. This is a similar problem that has been encountered with the testing of any of the RDR methodologies [2, 10, 11]. Thus, these 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, 6]. RM, 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 test 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 RM, 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 RM system for classification and rating. The order allocated by the RM system for each set was then compared against the simulated expert's expected ordering. In the tests shown here the environment contained ten possible attributes and the simulated expert had four possible classes it could categorise cases. 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 RM is the rating of documents so they can be appropriately ordered according to a user's preference. The ability of RM to accomplish this is illustrated in the example cross sections shown in Figure 5. The first graph shows the order, most important on the left to the least important on the right, in which RM places the documents prior to any learning. The second graph, after only five document sets, clearly illustrates that it has been able to correctly order many of the documents.

a)


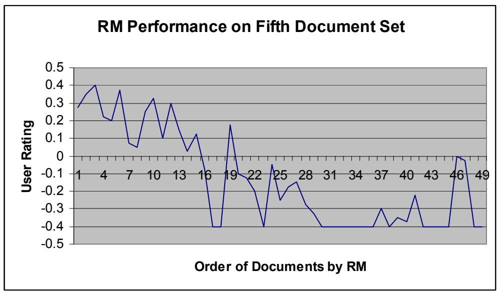
b)


Figure 5: Ability of RM to order cases according to user preference. a) Shows RM's performance prior to any training. b) Shows RM's performance after 5 document sets. Both graphs have the highest rated cases on the left and the lowest on the right according to RM.

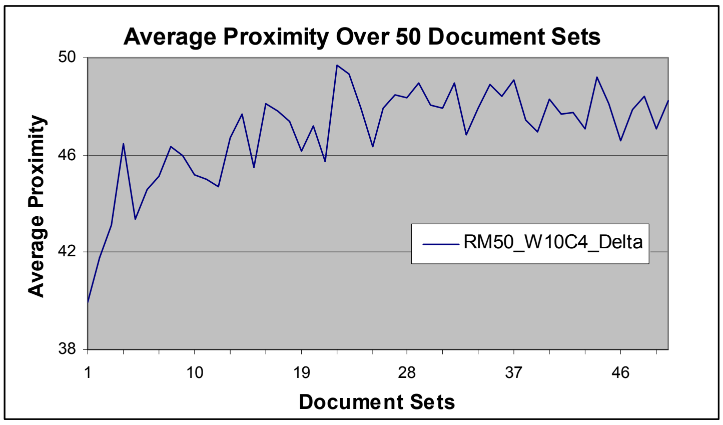


Figure 6: Proximity of RM to simulated users ordering. The average proximity is the absolute difference between the place allocated by RM and the place given by the simulated expert, averaged over all the cases in each set and subtracted from the total number of documents in the set. Each document set contains 50 cases.

To show how the system performs over time the difference was calculated between each case's place in the RM 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. Figure 6, 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 allows the system to immediately start to provide a reasonably accurate solution.

While these results are only preliminary, they do show that RM 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 various patterns of classifications according to either direct or indirectly derived values from the user. The system developed was designed to learn the simpler linear relationships quickly while still retaining the ability to learn the more complex non-linear functions, through the use of a neural network with the addition of short cut connections. Also developed was the *single-step-Δ-update-rule* for the short cut connections, which provides the ability of the system to step immediately to the indicated value required by the observations made of the user.

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: through the addition of a per-neuron reducing gain to allow the system to better cope with noise when creating new input nodes; and, through building in a windowing technique to allow the system to deal effectively with concept drift.

There also 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. Finally, through incorporating Temporal Difference Learning through the addition of an eligibility trace [12]. It may be possible to predict the likelihood of future cases, derived from the one being analysed, being of importance to the user.

Acknowledgements

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

References

[1] B. H. Kang, Validating Knowledge Acquisition: Multiple Classification Ripple Down Rules. 1996, University of New South Wales: Sydney.

[2] B. H. Kang, P. Compton and P. Preston. Multiple Classification Ripple Down Rules: Evaluation and Possibilities. in *The 9th Knowledge Acquisition for Knowledge Based Systems Workshop*. 1995. Department of Computer Science, University of Calgary, Banff, Canada: SRDG Publications.

[3] R. Martínez-Béjar, 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.*, (1999). **1621**: p. 349-364.

[4] P. Compton and R. Jansen, A philosophical basis for knowledge acquisition., *Knowledge Acquisition*, (1990). **2**: p. 241-257.

[5] H. Suryanto and P. Compton. Invented Predicates to Reduce Knowledge Acquisition Effort. 2003. School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.

[6] B. H. Kang, P. Preston and P. Compton. Simulated Expert Evaluation of Multiple Classification Ripple Down Rules. in *Eleventh Workshop on Knowledge Acquisition, Modeling and Management*. 1998. Voyager Inn, Banff, Alberta, Canada.

[7] P. Preston, P. Compton, G. Edwards and B. H. Kang. An Implementation of Multiple Classification Ripple Down Rules. in *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*. 1996. Department of Computer Science, University of Calgary, Calgary, Canada UNSW, Banff, Canada: SRDG Publications.

[8] E. A. Deards, MCRDR Applied to Email Classification, in *Department of Computer Science*. 2001, University of Tasmania: Hobart, Australia.

[9] R. Beale and T. Jackson, *Neural Computing: An Introduction*. 1992, Bristol, Great Britain: IOP Publishing Ltd.

[10] Y. Mansuri, J. G. Kim, P. Compton and C. Sammut. A comparison of a manual knowledge acquisition method and an inductive learning method. in *Australian workshop on knowledge acquisition for knowledge based systems*. 1991. Pokolbin, Australia.

[11] P. Compton, P. Preston, B. H. Kang and T. Yip, Local Patching Produces Compact Knowledge Bases, in A Future for Knowledge Acquisition, L. Steels, S. G. and W. V. d. Velde, Editors. 1994, Springer-Verlag: Berlin, Germany. p. 104-117.

[12] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. 1998, Cambridge, Massachusetts: A Bradford Book, The MIT Press.